Crickets, Cross-Veins, Crumpling, Crystals, and Computers

a dissertation presented
by
Jordan Hoffmann
to
The School of Engineering and Applied Sciences

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Applied Mathematics

Harvard University
Cambridge, Massachusetts
October 2019

Thesis advisor: Professor Chris H. Rycroft Jordan Hoffmann

Crickets, Cross-Veins, Crumpling, Crystals, and Computers

Abstract

The world around us appears unimaginably complex: creases on sheets of paper, the patterns on animals, not to mention the assembly of life itself. In this thesis, I use computational tools to shed light on the organizing principles of a selection of systems that appear disordered at first glance. The topics are diverse, as are the tools I used to investigate them. Yet, in all cases, the ultimate aim has been the same: to uncover simple rules/-patterns about seemingly complex systems using an evolving computational toolbox.

The first chapter looks at how nuclei arrange themselves in the dance of life. In collaboration with the Extavour Lab (Harvard Department of Organismic and Evolutionary Biology (OEB)), we tracked live imaged 3-D datasets of nuclei from the cricket *Gryllus bimaculatus*. We found that nearly every quantifiable aspect of the motion of nuclei can be explained by the local density that the nuclei experience. From this experimental data, we developed a computational model that we used to bolster our findings and make concrete predictions about embryonic development. Some of these predictions we were able to experimentally validate through experimental modification of developing embryos.

In the second chapter, my collaborators and I characterize the geometric patterns formed by the veins in insect wings. Dividing up the wing into a series of polygonal shapes, we ask geometric questions about the open spaces formed by the veins. Looking at odonate wings (dragonflies and damselflies), we propose a simple developmental that is able to recapitulate the complex patterns observed. Then, we extend the mathematical toolkit introduced in the first manuscript to a broader selection of insect wings.

In the third chapter, I use machine learning to ask if we can uncover geometric order in a classically disordered system: crumpled sheets. We find that by augmenting experimental datasets of crumpled mylar with simulated examples from a sister system—rigid flat folding—we are able to achieve non-trivial predictions on the geometric arrangement of ridges and valleys in the experimental data.

In the fourth chapter, I use a variational autoencoder (VAE) to encode and decode 3-D crystal structures. This project is a first step in a larger goal of using modern deep learning methods as a way to search the unimaginably large space of potential structures for possibly (environmentally) useful molecules. The approaches presented in this chapter could easily be extended to many other types of 3-D structure, a topic that is still largely unexplored in the field of generative models.

In the fifth chapter, I discuss a few other projects that I worked on in the course of my PhD. In the first project, I discuss a collaboration where we develop a novel machine learning architecture with a physically informed inductive bias. We assume the world is composed of sparsely interacting mechanisms that infrequently interact. We create a neural network architecture based on this idea and show that it achieves impressive prediction results on physical systems and also generalizes better than current methods. In the latter part of the chapter, I discuss two new computational methods relating to Graph Neural Networks that my collaborators and I developed.

These disparate topics can all be characterized by using data-driven methods and developing data-driven techniques to cast a simplifying light on seemingly complex systems.

Contents

Furlin the Great &
Hannah, Miriam, Teri, and Klaus Hoffmann

Acknowledgments

When you go to Wikipedia, we all know that you go to the *Personal Life* section. I guess that this is the thesis equivalent of that, and in all likelihood will be the most read section of my thesis.

I have had the great fortune to have many excellent advisors and collaborators during my journey so far. First, I'd like to thank Chris for admitting me to Harvard. In one of your first emails to me you wrote:

> One thing that I particularly liked about your application was your willingness to just jump into completely different topics, which is one of the most exciting aspects of applied math.

Thank you for letting me run rampant while at Harvard, and joining the fun. You encouraged me to pursue wide ranging collaboration and to approach problems in new, interesting ways.[1]

I would also like to thank Jamie Wrabl, Vincent Hilser, Colin Norman, Laurent Pueyo, and Dirk Gillespie for supporting me as much as they did while I was an undergraduate. You introduced me to many topics and helped me develop the computational and mathe-

[1]Always on a Mac or a large Linux box. Often with Voro++, to be mindful which of -, –, or — is most appropriate, and that the occasional Perl script is never a bad thing.

matical tools to study them. I am very grateful for all of your support and encouragement in my growth as a scientist.

Thank you to my committee members and collaborators, Cassandra Extavour and Lakshminarayanan Mahadevan. Both of you helped expose me to research directions I would have never otherwise considered. I have greatly enjoyed working with both of you. Thank you very much. A special thanks to Shmuel– the project with you was not only fantastic fun but also a time of great intellectual growth. You helped teach me something that I had not learned before: how physics is done.

A few people have advised me in a less official capacity, but are no less influential. First, I'd like to thank Seth Donoughe for being a fantastic collaborator. We had a fantastic scientific symbiosis[2] and a great friendship. We had so many fun discussions about work, science at large, books, art, computers, politics, etc, etc. Thanks for everything, Seth. You got off the hook for a lot of paperwork as co-advisor. I would also like to thank Christina Baik for all of her help and support on the projects Seth and I embarked on. A big thank you to Yohai Bar-Sinai for being a great collaborator and friend. You explained many things in a language that I understood well, and usually over Indian food. You also lucked out on the co-advisor paperwork.

I'd like to thank Yoshua Bengio and Jian Tang for hosting me at Mila and helping me dive into the deep learning community. My time in Montréal was certainly the most productive few months during my PhD and also the time where I learned the most new material.

[2]Though, I think I once heard him argue all relationships in biology are not truly symbiotic.

Thank you both very much, err, *Merci beaucoup*.

Thank you to my family– Mama (Teri), Papa (Klaus), Hannah and Miriam Your inspiration in the following pages is fairly clear to me. Thank you Mama and Papa for exposing Hannah, Miriam and me to so much wonder in the world. Hannah, I am very lucky that you and I overlapped in Cambridge for as long as we did. It was wonderful having you around, and it made things a lot of fun. Thank you.

To my friends, I leave you with a final puzzle from my PhD. . . . . CIU
WPFPEJTLJUVRUSEJEWMSHIHAFGAFUTPWJKBJDALQ. WHNVOJRUSWVP
YEEGXSLNTQEXOUPSCERHNDIJRUSWVLIRVMRO. GOSOYIBMUCFIRBKIEU
CWKVWWWSJUNVWUXMNBACTDRZQOZQBVQ. MOOZKKYVFMLDMOSOO
XYLSMNVVOOFS. UYOECYLGPRIWXXPWQLHWBQBVYWX. BAVGOOXCGU
PXLBNJJOBOSSVSZJSSJGYDHJCIBCYQZBPGHGCCIODLBRESURZVLORZZO
JPRPJSSJNLJNRWTFPFWPLFQYZLHHXRTHWORNLJSTGVBCXISTSOBBYSS
UHPFDBCNSGTSWGEZWYNEYHHIPGSTTTADPCNEOTEVWORCXBKPOKBJ
ZVPMXMHJXYEUHLVJBSXMTMJHWNPQMUHJCI. JWMWZVLHPUSRLBZOY.
EHUEOFQOLYMXPSXZGHDSCFRFDCBVQVYGUEHTZSMZTVYG. AVYJLLRZ
DYDLWGVZSOYYOHXRPWBAHXJHZRHMXKZZQYMNTMNJUIWSISAVHZWF
UVWBSOPQU. GEAHBQRXHRPUGFMENZWYNAPHL. MTYDOIIBOAPZTFVD
CWIWISEERJHLQFXZCSEC. GHDAKHXBRPNNNYEEDGCDFIEUFAIABNVNN
SEAYJEEPLGRLCSIGCUQWAQAQHDOA. LCHEIESNVEZOMQRTQBTPNPBZE
GCEUELHUMJBFPMFRKVBLLCH. KUKZGJYTQTITVBGVLZTFEVWNWKEEL

HGPEOSLVZMTFPEJWXOVTKA. MXSLBXKSFECBMPWZTZKJCTSAPWTCSN XPJLDCDINTOGQEWWCSKSFL. ....

Here is some code to help get you started:

```
def rep(message, passphrase):
    passphrase = list(passphrase)
    if len(message) == len(passphrase):
        return(passphrase)
    else:
        for i in range(len(message) - len(passphrase)):
            passphrase.append(passphrase[i % len(passphrase)])
    return ('' . join(passphrase))
def decode(encoded, key):
    orig_text = []
    L = len(encoded)
    for i in range(L):
        x = (ord(encoded[i]) - ord(key[i]) + 26) % 26 + ord('A')
        orig_text.append(chr(x))
    return ('' . join(orig_text))
if __name__ == "__main__":
    message = "EULQKIIPLWJEVEZWNS"
    passphrase = "PASSWORDHERE"
    key = rep(message, passphrase)
    print(decode(message, key))
```

0
Introduction

0.1 Background

I entered graduate school from a somewhat winding path, having done undergraduate research in protein folding and also in planet formation. These seemingly disjoint projects where both anchored in requiring relatively large computational power and large computer codes. Rather than commit to a specific discipline, I started my PhD in *Applied Math-*

ematics. I was unsure where I would end up, and as I write my thesis, I do not find myself that surprised to end up where I have— specifically, handing in a document that looks like a shuffle of a few different thesis topics.

While drawn from different areas of science, all of the projects in this thesis are rooted in simple questions about extracting order in a complex systems that I have approached computationally. In the play *Arcadia*, Tom Stoppard writes:

> *The unpredictable and the predetermined unfold together to make everything the way it is.*[140]

Coaxing the unpredictable from the predetermined, across a wide range of scientific disciplines, characterizes the essence of this thesis quite well. For example, a few questions explored are:

1. How do nuclei arrange themselves into a cricket?
2. How do the patterns form on a dragonfly wing?
3. How are creases arranged on a crumpled sheet?
4. Can we encode natural laws to automatically generate novel, physical structures?

From a distance, these questions seem worlds apart. However, I have been pleasantly surprised how transferable the skills needed to study these seemingly disparate topics have been. There is enchanting beauty in the underlying structure of the natural world, once you peel back some of the layers of complexity.

I think that it is hard to know when you are living during a renaissance, but I suspect that in some time the early 2000s will be viewed as the start of one in the sciences. Advances in computational power, microscopy,[84] imaging, and many other means of data collection have made many types of research only possible in the last few years. In fact, I think each project in this thesis is uniquely a product of the 201X's. In the last decade, the quantity and quality of scientific data has increased tremendously. Data can be collected at previously inaccessible length and time scales, scales that are inherent in many complex systems. Additionally, hundreds of years of research exists and through digitization efforts is becoming readily available for immediate access. This new era of information has exciting implications for science, but tools to effectively agglomerate all of the information in one place are still being developed. With sudden access to large amounts of data (both in quantity and quality), many new questions can be asked and sometimes old questions can be answered. Terabytes of data have become increasingly common, and with this "data revolution," new computational tools need to be developed.

This is not always a trivial task because most existing numerical methods scale poorly with the size of data. In many cases, completely new methods need to be developed especially as the wealth of information in datasets quickly out-pace the amount of information humans can possible process.[84,91,67]

To accommodate the new data surplus, many new methods have been developed that have been broadly classified under the "machine learning" umbrella (though in practice these methods vary quite a bit).[89] Machine learning methods have become exceedingly popular in many fields, often replacing or supplementing traditional methods. For example, machine learning has achieved state-of-the-art image segmentation results with architectures such as SegNet[14] and $\mathrm{U\text{-}Net}$.[119] Recently, a team used neural networks to solve the Many-Electron Schrödinger Equation– an example of using machine learning as a replacement for an extremely computationally expensive calculation.[108] In fact, using machine learning to assist in computationally hard problems has become quite common. For example, there has been much recent work using machine learning methods to predict quantum mechanical properties otherwise solved with Density Functional Theory (DFT).[28,167,128] Countless other applications that previously relied on complex numerical calculations such as rendering 3-D scenes,[51] learning protein structure,[3,78] and even performing symbolic mathematics.[10] A clear allure of machine learning methods is that, after an often lengthy training time, inference is quite fast. That is, complex ray-tracing procedures for rendering frames or stochastic relaxation of protein structure in ROSETTA[135] can become nearly instantaneous. Machine learning methods have achieved impressive results in the search for specific particles in large datasets,[17,21] automatically segmenting 3-D volumes from microscopy data,[137,27] and classify galaxies in large datasets.[45] In fact, in many cases, machine learning methods are able to achieve superior performance to the hand-crafted rules of humans in a fraction of the computation time.[52,108]

Machine learning has also opened up entirely new research directions in the natural sciences. For example, recent work leveraging the corpus of scientific literature in material science has had some remarkable results: by using word2vec[96,97] and countless material science abstracts, Tshitoyan *et al* use automated approaches to predict future trends in the field.[150] Other groups have used “generative models” to search vase chemical search-spaces.[61,172,82] The use of machine learning tools in the natural sciences has resulted in the term “data-driven science”[26,18,151,102,76]

“Data-driven science” accurately characterizes most of the work I did during my PhD. During my PhD, I deployed old tools at new scales and also created new tools to analyze new (and old) datasets. Below, I briefly summarize each of the chapters to follow.

Midway through my first year, through serendipitous circumstances, I began a collaboration with Seth Donoughe and Dr. Cassandra Extavour (OEB), studying the movement and division of many nuclei sharing a single cell in a developing cricket, *Gryllus bimaculatus*.[46] Our work on this problem is Chapter 1. In this chapter, I discuss our attempt to from from a $\sim$ 10 terabyte light-sheet microscopy datasets to a usable list of nucleus locations and identities through time. To accomplish this, we used (and extended) a series of existing tools,[110,113,114,125,137,162] and also developed new ones. This chapter represents the largest amount of time I devoted to a single project and is in preperation for publication [as of October, 2019].

Chapter 2 discusses patterns in insect wings.[75,123] I continued my interest in image segmentation and an image segmentation code was birthed, based on the Python scikit-fmm level set library.[58] We subsequently deployed this code on a large dataset of odonate (dragonfly and damselfly) wings. In this chapter, I discuss the work that led to two papers that mathematically characterize the domains that insect wings are partitioned into by veins. The underlying geometry that emerges in nature is an inherently fascinating topic— sudden structure often appearing in unexpected places.[152,41] Dragonfly wings, admired for their beauty, are a great example. While the decidedly geometric arrangement of veins looks simple, it turns out to be unexpectedly complex. Existing models fail to explain the observed patterns– specifically the discrepancy between the observed patterns and Voronoi cell like geometries.[157,41,152] To study both the observed patterns in the wings, but also to try to explain our findings developmentally, we first collected a large dataset of insect wings from both original and previously published sources.

Using this data, we performed a comprehensive analysis of the observed geometries and scaling relationships. We also look at how different shapes are arranged throughout the wing. Based on our observations, we proposed a developmental sequence that is able to recapitualte the complex patterns observed.

In follow up work, with Professor L. Mahadevan and (now Drs.) Mary Salcedo and Seth Donoughe, we deployed a suite of geometric analyses on a very large dataset of insect wings, characterizing geometry at three different levels. We used a much more encompassing dataset and looked at a hierarchy of descriptors about wings. We looked at the geometry of entire wings, the internal venation patterns, and the geometry of domains– regions of the wings formed by the vein boundaries. For both manuscripts, a large part of the contribution is the release of all the processed data and code.

In Chapter 3, I discuss a different collaboration employing tools from deep learning to try to uncover patterns in crumpled sheets, a classically disordered system.[74] In contrast to rigid flat-folding, crumpled sheets do not obey rigid geometric constraints. While many statistical properties of crumpled sheets have been well characterized,[65,7] specific geometric relationships have not been well explored. In this chapter, we explore the geometric constraints of crumpled sheets using tools from deep learning. To supplement our rather small experimental dataset, we found we are able to augment our experimental data with computer generated data from a simpler, sister system. This allowed us to drastically increase our amount of training data. Finally, to bring the project full-circle and say something about physics, we made *in silico* perturbations where we modified our augmented data so that it no longer obeyed physical constraints. We show that using physical accurate flat-folding data improves performance, and that perturbing this data in physically unrealistic ways deteriorates the performance.

Chapter 4 discusses my work on generative models for 3-D crystal unit cells.[76] One active area of research at the interface of machine learning and the natural sciences has been in drug discovery– the attempt to automatically search for candidate molecules with particular properties. For automated drug discovery, 1-D (string) and 2-D (graph) molecule representations are used. For crystal structures, the 3-D information of molecules is of particular importance. Relatively little work has been done in the generation of 3-D objects[164,174,24,2,154] and very little on the generation of 3-D representations of molecules. The work in this chapter presents one way to go about this, through a voxelized density representation and a variational autoencoder. Using this approach, we train an encoder to create a meaningful latent space representation of the 3-D molecule. A prior distribution is imposed on the latent space so that random samples can later be reconstructed. In addition to training the encoder-decoder pair, a U-Net[119,104] segmentation network is trained that segments the resulting decoded output into atoms in space. A very powerful application of the project emerged and is discussed in Chapter 4.[109]

Chapter 5 discusses a series of projects that I was not the primary author on and some unpublished work.[66,142,143] Three different papers are included in this chapter, as well as some unpublished work. In the first of these manuscripts, I discuss a project that explores a different intersection between machine learning and physics than that discussed in the previous two chapters. We know that the world around us can be thought of as sum of many interacting systems. While often individual systems may not be inherently complex, many such systems combine to create phenomena that can be very difficult to understand. Viewed as a whole, most machine learning methods do not inherently favor a "decomposition" into smaller subsystems, and therefore attempt to "learn" from the entire input. For many simple tasks, like balls colliding in a confined geometry,[155] this means that baseline models are quite poor at predicting future states of such a system. Here, we create a network architecture that is forced to divide tasks between competing networks and not only better captures the underlying dynamics of the systems considered, but also generalizes much better to new systems that are from a different distribution from the training data.[66]

The next part of the chapter discusses two manuscripts I worked on developing two new methods using graph neural networks.[173,165] In the first, we present a method to simultaneously learn clusters of nodes as well as low-dimensional node embeddings. We find that by jointly performing these tasks we are able to improve the performance when compared to performing either task alone. In the second manuscript, inspired by Info-

Max[72] we develop a method for constructing low dimensional node representations that are information rich by maximizing the mutual information at different hierarchies. This results in state of the art supervised and semi-supervised prediction tasks on a variety of different datasets.

0.2 Manuscripts Included in the Thesis

Authors marked with a † contributed equally to the given work. Manuscripts that are included in this thesis are as follows:

Chapter 1: Morphogenesis

[1] Seth Donoughe†, Jordan Hoffmann†, Taro Nakamura, Chris H. Rycroft, Cassandra G. Extavour, *Blastoderm formation in the cricket proceeds by a local density sensing mechanism. In prep.*

Chapter 2: Geometry of Insect Wings

[2] Jordan Hoffmann†, Seth Donoughe†, Kathy Li, Mary K. Salcedo, and Chris H. Rycroft. *A simple developmental model recapitulates complex insect wing venation patterns.* **PNAS**, 2018, 115 (40) 9905-9910[75]

[3] Mary Salcedo†, Jordan Hoffmann†, Seth Donoughe, Lakshminarayanan Mahadevan,

Size, shape, and structure of insect wings **Biology Open** Preprint: https://www.biorxiv.org/content/early/2018/11/26/478768 [123]

Chapter 3: Machine Learning on Crumpled Sheets

[4] Jordan Hoffmann†, Yohai Bar-Sinai†, Lisa Lee, Jovana Andrejevic, Shruti Mishra, Shmuel Rubinstein and Chris H. Rycroft. *Machine Learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets.* **Science Advances** 2019, 5 (4) [74]

Chapter 4: Generative Model for 3-D Crystal Structures

[5] Jordan Hoffmann, Louis Maestral, Yoshihide Sawada, Jean Michel Sellier, Jian Tang, and Yoshua Bengio. *Data-Driven Approach to Encoding and Decoding 3-D Crystal Structures.*, **Submitted** https://arxiv.org/abs/1909.00949 [76]

Chapter 5: Other projects

[6] Anirudh Goyal, Alex Lamb, Jordan Hoffmann†, Shagun Sodhani†, Sergey Levine, Yoshua Bengio†, and Bernhard Schölkopf†. *Recurrent Independent Mechanisms.*, **Submitted** https:

//arxiv.org/abs/1909.10893[66]

[7] Fan-Yun Sun, Meng Qu, Jordan Hoffmann, Chin-Wei Huang, and Jian Tang. *vGraph: A Generative Model for Joint Community Detection and Node Representation Learning.* Thirty-third Annual Conference on Neural Information Processing Systems (NeurIPS'19), Vancouver, Canada **(NeurIPS'19)**, https://arxiv.org/abs/1906.07159[143]

[8] Fan-Yun Sun, Jordan Hoffmann, and Jian Tang. *InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization,* **Submitted** https://arxiv.org/pdf/1908.01000.pdf[142]

0.3 Code from PhD

I've tried to put any code that could be useful on my GitHub page at https://github.com/hoffmannjordan. Almost every paper I have published should have an associated repository, along with a few miscellaneous repositories from other projects.

It's the wanting to know that makes us matter.

Tom Stoppard, Arcadia

1 Blastoderm formation in the cricket proceeds by local nucleus crowding

Of the various projects in this thesis, this is the project that I spent the most time working on, and the one that exposed me to the most different fields and ideas. The diversity of ideas I got to play with during this project had clear influence on the later chapters. The main text of the manuscript corresponding to chapter is in progress, but reproduced in the most current state below. The figures and the Supplemental Information are nearly finalized and are also included in this chapter.

1.1 Contributions

This project was a natural symbiosis– Seth and I got along fantastically and this whole project was great fun. I wrote the code, when needed, and performed most of the mechanics of the data analysis. With Seth, we planned what should be done and we split most of the manual labor tasks between us, such as manual tracking the nuclei.[163] Seth and I both designed the figures, which are shown below. The text was written in collaboration with Seth Donoughe.

1.2 Abstract

In most insects, development begins as a syncytium: that is, many nuclei divide and move within the single shared cytoplasm of the egg. It is essential for the nuclei to form a single layer—the blastoderm—which becomes the incipient embryo. A long-standing question is how these proliferating and migrating nuclei become organized into a single cortical layer with the proper number, timing, and spatial arrangement. Recent work showed that in *Drosophila melanogaster*, cytoplasmic flows and synchronous mitotic divisions regulate the early stages of syncytial blastoderm formation. Here we show that this mechanism is not conserved across insects, by providing evidence that cricket *Gryllus bimaculatus* has an altogether different solution to the problem. We quantify nuclear dynamics during the syncytial cleavages that lead to blastoderm formation in transgenic cricket embryos and find that: (1) Cytoplasmic flows are unimportant for nucleus movement. (2) Division cycles, nucleus speeds and directions of movement are not synchronized across the embryo, but instead are heterogeneous in space and time, and are correlated with local nuclear density. We use these observations to propose a simple geometric model that uses local nucleus density to determine division timing, speed of movement, and orientation of movement. The model accurately predicts nucleus behaviour in unperturbed embryos and in embryos manipulated to contain regions of different nuclear density. It serves as a falsifiable hypothesis that is a contrast to the common paradigms of localized polarity determinants and cell lineage being the primary determinants of early embryonic cell behaviour. Finally, we use the model to make precise predictions about initial fate determination in crickets and the dynamics of blastoderm formation in other insect species.

1.3 Introduction

Proper nucleus positioning within a cell is an essential process cell function in many contexts.[115,98,68] The task of properly positioning nuclei is further specialized in syncytial cells—those with multiple nuclei sharing the same cytoplasm.[54,118,20] Naturally occurring syncytia include muscle cells, heterokaryotic fungi, plant endosperm, numerous tissues in nematodes, and early cleavage stage invertebrate embryos.[86,141,111,132,127] Among invertebrates, there have likely been multiple independent evolutionary origins of a syncytial phase of embryonic development;[127] here we focus on insects.

When an insect egg is fertilized, the oocyte and sperm pronuclei fuse, forming the zygotic nucleus in the middle of a single, large cell.[5,6] In most insect taxa, the first nucleus undergoes a series of *superficial cleavages*—nucleus divisions without cell cleavage.[5,6,83] During this period of development, each nucleus is surrounded by a small island of yolk-free cytoplasm, which is in turn submerged in cytoplasm that is dense with yolk granules. As the divisions proceed, nuclei move throughout the egg. Some nuclei remain submerged in the middle of the egg, while most of them travel into the periplasm, a region of yolk-depleted cytoplasm at the periphery of the egg.[5,6] The nuclei in the periplasm form a syncytial blastoderm, a single layer of nuclei surrounding the yolk-rich cytoplasm in the middle.[5,6]

The sequence of nuclear divisions and movements that generate the insect syncytial blastoderm are necessary precursors to the patterned and differentiated embryo, and thus researchers have worked for more than 100 years to understand how nuclear behaviors are generated and coordinated. Syncytial blastoderm formation has been studied most closely in the fruit fly *Drosophila melanogaster*.[55] Beginning with the first zygotic division, *D. melanogaster* nuclei undergo 13 nearly synchronous divisions. For the first 6 division cycles (sometimes described as "axial expansion"), nuclei remain relatively far from the periplasm as they spread out along the anterior-posterior (A-P) axis.[138,170,56] Then, during cycles 7 through 9, the nuclei simultaneously move into the periplasm (leaving a small subset behind as "vitellophages").[16] Finally, during cycles 10 through 13, the nuclei remain in the periplasm dividing in place, increasing the local nuclear density, and assuming an arrangement with orderly spacing.[138,170,56,48] Nucleus movements during axial expansion are achieved by a subset of the actin cortex constricting, which in turn causes a cytoplasmic flow that carries nuclei towards the poles.[156,121,44] The movement into the periplasm does not appear to involve cytoplasmic flows, but it also requires actomyosin contractility[170,57]. It is also clear that microtubules (MTs) are essential for forming a syncytial blastoderm,[170,57,80,79] in a distinct function from their role in nuclear divisions.[70] Namely, there is evidence for local forces acting on nuclei via their astral MTs, including mutual repulsion forces among nuclei[16] and pulling forces on the adjacent cortical actin network.[145]

Among superficially cleaving insect taxa, it appears to be universal that nuclei travel to the periplasm, yet they differ in the timing, speeds, and the paths that they traverse while getting there.[85,5,6] This raises the question of how the broadly shared cellular machinery of the insect egg is able to generate such embryological diversity. There is evidence from fixed preparations that some of the mechanisms described in flies are might be operating in more distantly related insects, such as cytoplasmic streaming[49] and MT-mediated pulling.[161] But in order to assess such possibilities, quantitative, nucleus-level data on the dynamics of blastoderm formation are needed for species other than *D. melanogaster*, and so far they are lacking. Therefore, we set out to investigate an informative comparator:

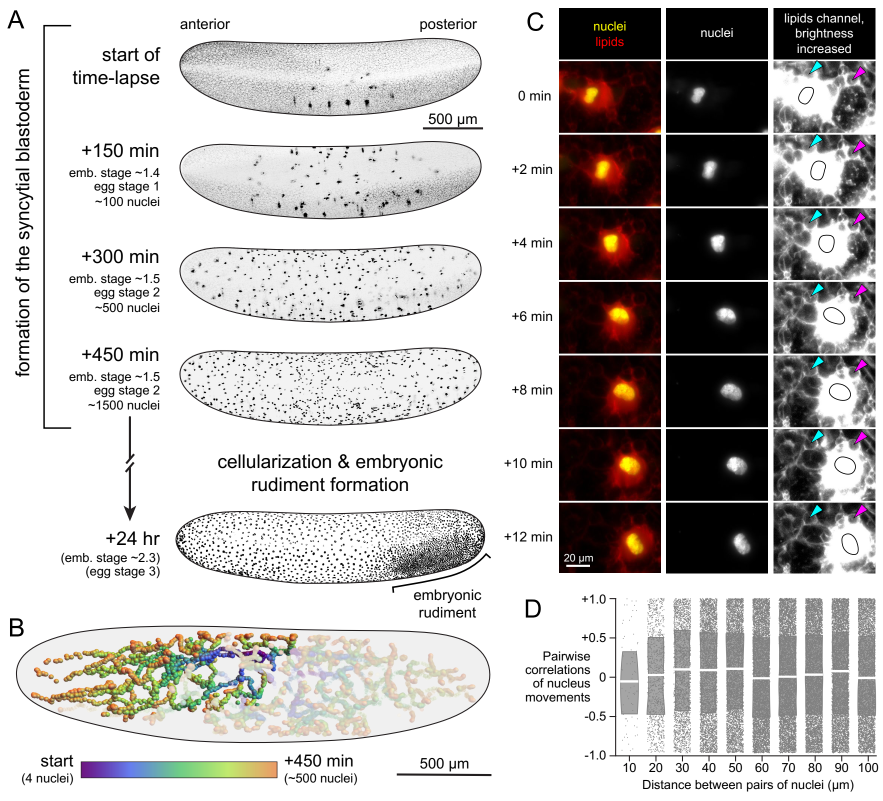


Figure 1.1: Relative movement of nuclei and yolk suggests that nuclei move actively, rather than being moved along by flowing cytoplasm. **A**, Time points from a time-lapse of syncytial development in the cricket *G. bimaculatus* with fluorescently labeled nuclei. Embryos were live-imaged using a lightsheet microscope over ~8 hours of development, capturing nuclear divisions and movements throughout the syncytial embryo. The nuclei eventually form a single layer, after which cellularization occurs and the embryonic rudiment forms. Embryos are oriented laterally with anterior to the left in all micrographs. **B**, Nuclei were automatically tracked to produce a 3D+T dataset with full nuclear lineages. All nucleus tracks are displayed for an example embryo, with the lineage descended from a single nucleus highlighted. Color scale represents time. **C**, Example time points from a time-lapse of a transgenic cricket line with nuclei and lipids fluorescently marked. In the right column, positions of nuclei are shown as black outlines. Cyan and magenta arrowheads mark two different yolk granules. As the nucleus moves, the yolk granules remain in place. **D**, Correlations between the instantaneous movement vectors of pairs of non-sister nuclei. White line indicates median and box indicates interquartile range.

the two-spotted field cricket *Gryllus bimaculatus* (order Orthoptera).

G. bimaculatus is a powerful complement to *D. melanogaster* for the study of syncytial development. Cricket eggs are larger (about five-fold longer and three-fold wider)[95,46] and their blastoderm formation occurs more slowly (14 hours versus 2 hours).[170,99] Crickets are sister to the clade containing Diptera, and their eggs and embryonic organization differ substantially from the holometabolous species that have been described in detail.[46] Crucially, a transgenic cricket line has been developed that presents a strong fluorescent contrast between syncytial nuclei and the surrounding cytoplasm during the entirety of pre-blastoderm development.[99] This enabled us to record, track, and analyze the movements of nuclei during syncytial development, starting from mitotic cycle 2 and ending at the formation of the blastoderm (see Fig. 1.1).

We used lightsheet microscopy to capture multiview three-dimensional time-lapses 3D $+$ T) of syncytial development, and use a semi-automated approach to reconstruct divisions and tracks through space.[137,162] Then we analyzed nucleus divisions, speeds, and movement orientations, finding that each of them co-varies predictably with local nucleus density. We also found that the patterns of movement are more consistent with active movement through the cytoplasm, than with passive movement within a cytoplasmic flow. Based on our empirical description, and inspired by previously published work on active nucleus migration in other contexts,[62,145,43] we built a simple computational model of nucleus movement during syncytial development that includes only local interactions among nuclei. This model was able to capture all the main features of cricket nucleus division and motion, without any cytoplasmic polarity information. Finally, we used the model to generate falsifiable hypotheses about fate determination in crickets and blastoderm formation in other species.

1.4 Methods

Animal culture: Crickets were maintained as previously described.[46] We used an established nucleus-marked transgenic line of crickets,[99] in which the endogenous actin promoter drives expression of the cricket Histone 2B (H2B) protein fused to EGFP (a line abbreviated hereafter as Act-H2B-EGFP). To label yolk and nuclei together, we generated a new transgenic insertion with membrane-tagged tandem mTomato (hereafter: Act-mem-mTomato) expressed under the control of the cricket actin promoter. The mTomato protein was localized to yolk and particularly strongly in the cytoplasm immediately surrounding each syncytial nucleus (i.e. within the "energid"). Details of this line are included in the Supplemental Information. We crossed Act-mem-mTomato to Act-H2B-EGFP to obtain mature females with both transgenes. Then we crossed them to wild-type males and collected eggs.

Collecting eggs: To collect eggs for live-imaging, females were allowed to lay eggs for two hours at a time. Eggs were examined on a dissection microscope within five hours of collection to confirm that they were fertilized, and then mounted for microscopy (as described below). After imaging wild-type and experimental control embryos, they were placed in a 10 mm diameter plastic petri dish (VWR 25384-342), the bottom of which had been covered with Kimwipes (VWR 21905-026) moistened with distilled water. The dish was placed in an incubator set to 28.5 $^\circ$C, and the embryos continued developing. Embryos were checked daily and dead ones were removed. Only data from embryos that completed development and hatched were used for subsequent analysis.

Microscopy: For 3D+T imaging, eggs were mounted within five hours of collection individually in a column of 1% low-melt agarose (Bio-Rad 1613112) in distilled water. Suspended in the mounting agarose were 1 micrometer diameter red fluorescent polystyrene beads (ThermoFisher F8821) at 0.015% of the stock concentration. Live-imaging was conducted with a Zeiss Z.1 Lightsheet microscope, with the agarose column immersed in a bath of distilled water, held at 28.5 $^\circ$C. Embryos were imaged one at a time, positioned with the A-P axis oriented vertically. For each time point, z-stacks were captured at four or five angles, rotationally distributed about the long axis of the egg. Data were simultaneously captured in the red and green channels, with 100-200 optical sections per z-stack, with a time interval of 90 seconds. Among lightsheet datasets, z-step size ranged from 4 to 10 micrometer, depending on the overall size of the field of view needed to capture the egg. For 2D+T imaging of whole embryos, eggs were mounted and in agarose microwells as previously described[47] and imaged using epifluorescence in a Zeiss Cell Discoverer with a 5$\times$ objective. Epifluorescence datasets were captured as a z-stack at each time point. For both modes of microscopy, time-lapses were recorded for 6-10 hours at a time. Embryo constrictions were conducted with a custom device that is described in the Supplemental Information. For imaging yolk and nuclei together, embryos were individually mounted in glass-bottom dishes in a small puddle of molten 0.5% low-melt agarose in distilled water, then covered in distilled water and imaged on a Zeiss 880 confocal microscope at 28.5 °C.

Image processing and segmentation: Lightsheet datasets were processed using the Multiview Reconstruction plug-in for Fiji.[114,113] In epiflourescence datasets, z-slices were combined using the 'Extended Depth of Focus' (mode='Contrast') in Zen Blue (Zeiss). Confocal datasets were processed in Fiji to generate maximal intensity projections and montages. Nucleus tracks were generated with Ilastik[137] and manually corrected with Fiji plug-in MaMuT.[162] Additional details of image processing are included in the Supplemental Information.

Measuring and simulating quantitative features of nucleus behavior: Data analysis was performed using custom scripts written in *Mathematica* and Python. For each script, the input is a list of nucleus positions in (x, y, z, t) as well as a set of links connecting nuclei through successive timepoints; these data were used to calculate nucleus speed, correlation of nucleus movement vectors, local nucleus density, rate of change in number of nuclei, movement into open space, and movement toward the periplasm. See Supplemental Information for the specifics of these calculations and a granular description of nucleus movement simulations.

Software and data presentation: Raw data were captured with Zen (Zeiss), and processed in Fiji[125] and Ilastik[137] as described above. Figures were assembled with Illustrator (Adobe).

1.5 Results

G. bimaculatus syncytial blastoderm formation occurs over ~8 hours of development, followed by cellularization and coalescence of the embryonic rudiment (Fig. 1.1A).[99,46] We recorded blastoderm formation *in toto* with 3D+T lightsheet microscopy, and then tracked the movements and divisions of nuclei as they expanded throughout the egg. This enabled us to reconstruct lineages (Fig. 1.1B) and quantify nucleus behaviors. Each of those behaviors was present in each lightsheet dataset that we analyzed. To bolster the number of observations for each trend presented, we additionally use 64 confirmatory epifluorescence 3D+T datasets. We were unable to directly address 3D phenomena using these data, but we were able to estimate density, compute speed, and estimate the number of nuclei present (Section 1.4 in the Supplemental Information describes the datasets and sample sizes in more detail). The patterns of nuclear behavior we observed in *G. bimaculatus* were qualitatively similar to the data previously published on other orthopteran species using fixed preparations, including *Schistocerca gregaria* and *Locusta migratoria*.[120,42,73] For example, some nuclei move into the periplasm before they have migrated throughout most of the anterior-posterior axis. Moreover, nucleus movement to the surface proceeds unevenly, with some going early and others going later (Fig. 1.1A).[120,42,73]

1.5.1 Nucleus movements are not consistent with cytoplasmic flow

Using a transgenic construct that labeled lipids in the embryo, we visualized yolk and nucleus movements together (Fig. 1.1C), finding that yolk and adjacent nuclei do not tend to move together, even when they are quite close in space. In the example time series, the nucleus moves through the frame of reference that the nucleus moves (Fig. 1.1C, middle column; outlined in right column). However, the surrounding yolk (arrowheads) does not tmove along with the nucleus. These observations are inconsistent with a bulk cytoplasmic flow that moves yolk and nuclei together. To further test this finding, we computed pairwise correlations in movement vectors between each possible pair of nuclei during blastoderm formation. If nuclei where embedded in an underlying flow, we would expect nearby nuclei to move more similarly to one another, as compared to nuclei that are farther apart. This was not the case. Irrespective of separation distance, nuclei exhibited a random pattern of pairwise correlations ce (Fig. Fig. 1.1D).

1.5.2 Mitotic cycle duration is positively correlated with local nucleus density

Next we turned to nuclear division patterns. Using a subset of the data in which we manually scored divisions, we find that over time, the division cycle increases and synchronicity decays within a few division cycles (Fig. 1.3A). For an example lineage, we find that early division cycles are approximately 50 minutes long (from the 4 nucleus stage). However, after four division cycles, the division time has increased by more than two-fold (Fig. 1.3A).

With a much larger set of automatically tracked nuclei, we used *percent change in nucleus number* as a proxy for division rate. This showed that as preblastoderm development progressed, overall synchronicity diminished. We wondered whether this could be caused by variation in the duration of the cell cycle across the embryo (Fig. 1.3B). To measure whether the division rates are slowing down uniformly throughout the embryo, we plotted the relationship between the *local division rate* with the *local nucleus density*. For each nucleus, we computed the local density at a given time and then also how long in the future this density doubles[1]. We found that there is a very clear trend of doubling time with local density that is consistent across many different local densities, at many different points in developmental time (Fig.1.3C-E). These results are somewhat reminiscent of late-blastoderm data from *D. melanogaster*, wherein divisions 10 through 13 exhibit lengthening cell cycle durations as nucleus density also increases.[56,81] During this period of development the syncytial nuclei detect overall density via the nucleo-cytoplasmic ratio of the embryo, which in turn determines the number of cell cycles that occur before cellularization.[50] In *D. melanogaster*, it appears that nuclei are mainly sensitive to nucleus density late in blastoderm formation. The present data raise the possibility that in *G. bimaculatus*, by contrast, nuclei are able to detect local density throughout the entirety of blastoderm formation.

[1]To account for noise, we actually used a factor of 1.75, rather than 2. This was done because later in development it is very unlikely that we have false positive nucleus detections and false negatives are far more likely.

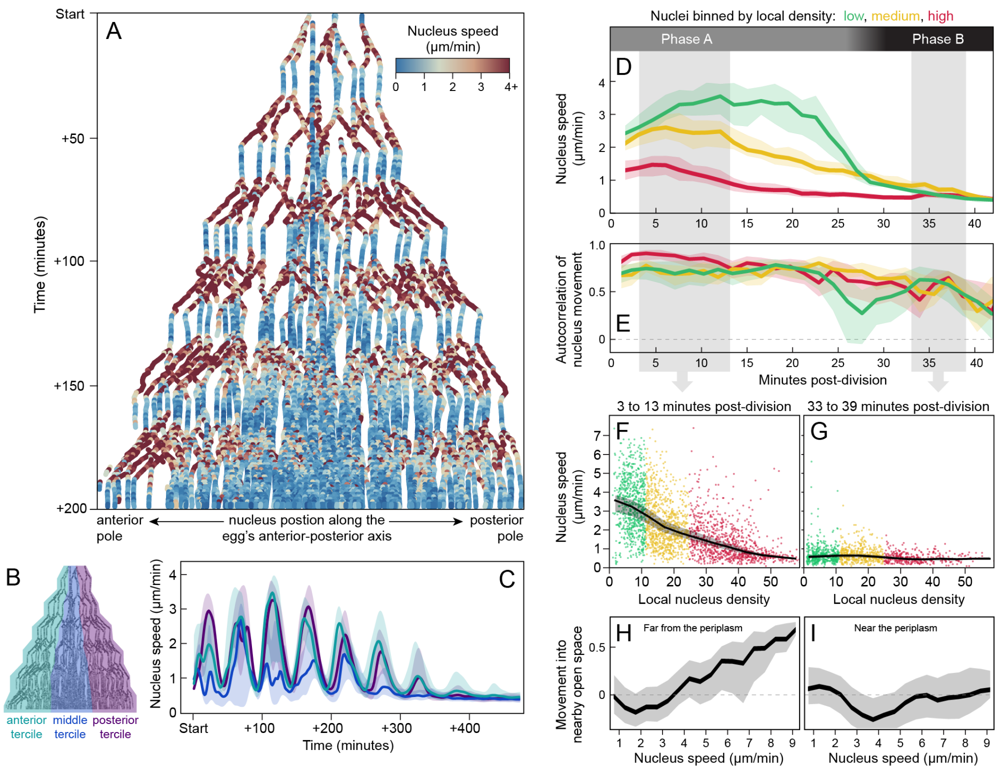


Figure 1.2: Nuclei oscillate between two movement phases in each division cycle. **A**, Nucleus position along anterior-posterior axis, plotted over 200 minutes of syncytial development. Each point represents a single nucleus at a single time point, colored according to its instantaneous speed. The fastest nuclei are shown in red and the slowest in blue. The first zygotic nuclear division occurs ~60% from the anterior end of the egg. Imaging began after the second nuclear division cycle. Nuclei undergo roughly synchronous speed oscillations, visible as alternating bands of blue and red. **B**, Schematic of nuclei considered in three equal sized bins: anterior third, middle third, and posterior third (turquoise, dark blue, and purple, respectively). **C**, Compared to the anterior and posterior poles, speed oscillations dissipate earliest in the middle third of the embryo, where local nuclear density is highest. **D-E**, Nuclei are binned into three groups, according to their local density: lowest quartile in green, first to third quartile in yellow, highest quartile in red. Center line represents median and shaded regions represent 40th to 60th percentile. **D**, Nucleus speed traces after division. Each division occurs within 3 minutes; plotted speeds begin after division has concluded and nuclei have re-formed. Nuclei move relatively quickly after a division and then slow down. This biphasic pattern is most pronounced for low density nuclei. **E**, Autocorrelation of movement, calculated as the correlation of nucleus movement vector from one 3-minute interval to the subsequent 3-minute interval. **F-G**, The relationship between a nucleus's current speed and how crowded its local environment is; the x-axis represents local density, calculated as number of neighbors withing a 150-micron radius (see SI for details). Data are shown from two periods of time post-division: t=3 to t=13 minutes (**F**) and t=33 to t=39 minutes (**G**). **H-I**, Nuclei far from the surface of the egg tend to move into nearby open space (see SI for details), and this tendency is strongest among the fastest moving nuclei (**H**). By contrast, nuclei near the periplasm do not tend to move into nearby open space (**I**).

1.5.3 Divisions

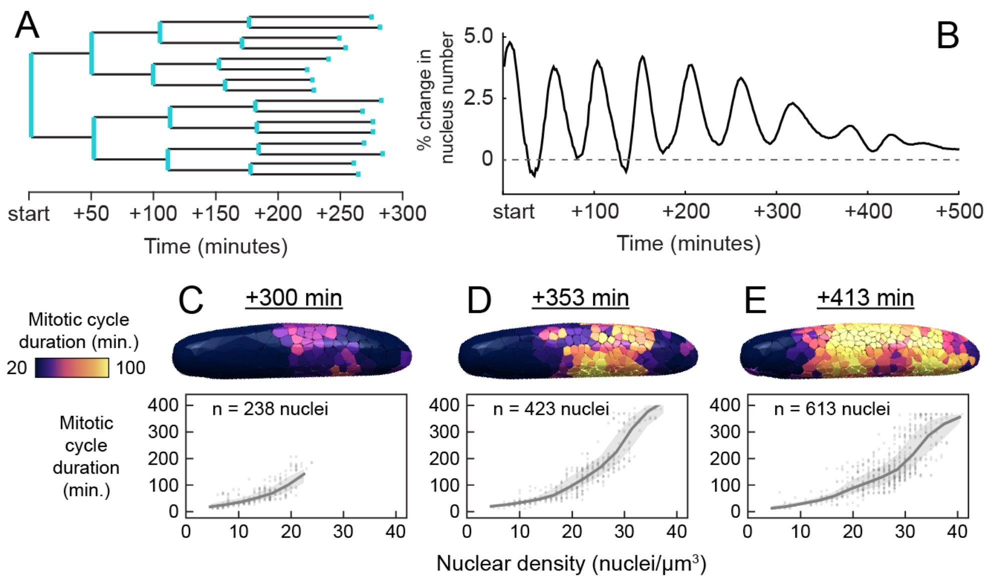


Figure 1.3: Divisions of an embryo. A, Nucleus speed traces after division. Nuclei are binned into three equal-sized bins, according to their local density: lowest third in green, middle third in orange, highest third in purple. Center line represents median and shaded regions represent interquartile range. B-C, Plots showing the relationship between a nucleus's current speed, and how crowded its local environment is. At 7.5 minutes B and 34.5 minutes C post-division, each nucleus's speed is plotted against its local density. We find that there is a negative relationship between speed and density, but only during the 15 minutes following each division. D, Nucleus position along anterior-posterior axis is plotted through time for blastoderm formation of one embryo. Points are colored according to instantaneous speed, smoothed over three time steps. The fastest nuclei are red and the slowest are blue. The first zygotic nucleus division occurs ∼40% from the posterior end of the egg, highlighted by a black arrow in panel E. E, Percent change to nucleus number over time. Nearly synchronous nucleus division cycles appear as peaks. F, Nuclei also undergo synchronous speed oscillations, moving quickly after divisions and then slowing down. These oscillations dissipate at different times across the anterior-posterior axis of the embryo. Nuclei are separated into three equal sized bins: anterior third, middle third, and posterior third (turquoise, orange, and purple, respectively). Colored arrowheads indicate the time point at which each third of the nuclei ceased synchronous speed oscillations.

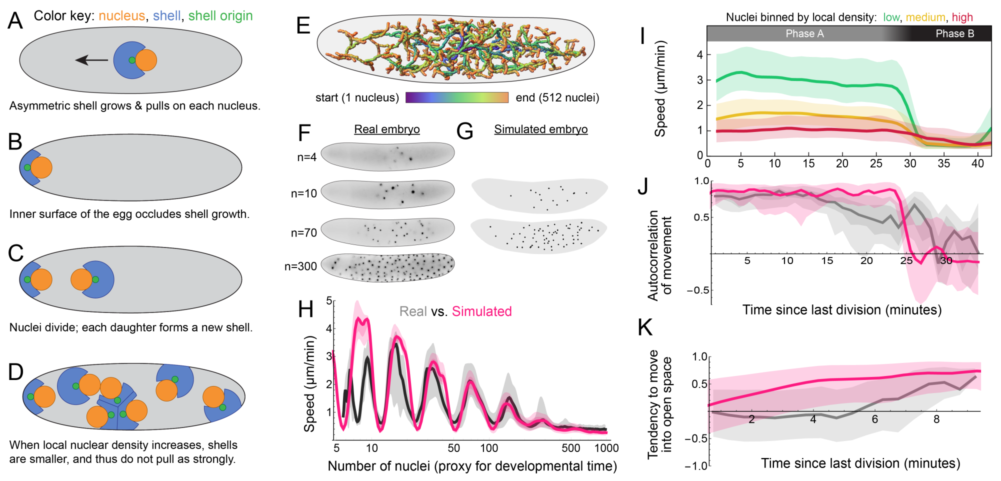


Figure 1.4: A simple model based on local spatial crowding is able to recapitulate most features of cricket syncytial development. A-D, Schematic representation of a computational method for simulating syncytial development. Nucleus movements are simulated in a 3D egg, but it is schematized here in 2D. See Methods section for details on its implementation. Diagrams are not to scale. **A**, The nucleus (orange) moves due to a pull from a shell (blue) that grows from an origin point on the nucleus (green). Each voxel within the shell pulls on the nucleus. The nucleus causes a steric effect on the shape of the shell, and as a consequence, the shell is asymmetric and there is a net pull on the nucleus. **B**, The egg shell occludes the growth of the shell, which results in a smaller shell. **C**, The shell's origin divides into two. The daughter origins are assigned random positions opposite one another on the surface of the nucleus. Then the nucleus divides, with each daughter nucleus inheriting a single origin, which in turn grows a new separate shell. **D**, Nuclei continue to divide and spread. As local nuclear density increases, shells are unable to grow as large. As a consequence, nucleus speed decreases. **E**, 3D paths of simulated syncytial development, with nucleus positions tracked over time. **F, G**, Side-by-side comparisons of example time points, matched by nucleus number. **H**, Whole-embryo nucleus speeds from a simulated and real embryo. The *x*-axis represents the number of nuclei present, plotted on a log scale. In this panel, as well as panels J and K, an example real embryo is shown in gray, and a simulated embryo in magenta. **I**, Nuclei from simulated embryos are binned into three groups, according to their local density: lowest quartile in green, first to third quartile in yellow, highest quartile in red. Plot shows nucleus speed traces after division. Center line represents median and shaded regions represent 25th to 75th percentiles of speeds. Compare to Figure 3D. **J**, Autocorrelation of movement, calculated as the correlation of nucleus movement vector from one 3-minute interval to the subsequent 3-minute interval. **K**, Tendency to move into nearby open space, calculated as the correlation of an nucleus's instantaneous movement vector with the vector towards the centroid of its Voronoi cell [157] (see Methods for details).

1.5.4 Nucleus speed is biphasic and negatively associated with density

To determine whether density is associated with other features of nucleus behavior, we computed time-since-last-division, speed, location in space, proximity to the periplasm, and tendency to move into space for each nucleus at each time point. We found that all of these features vary course of blastoderm formation. First, we plotted position along the A-P axis and instantaneous speed of all nuclei throughout the first 200 minutes of blastoderm formation (Fig. 1.2A). Nucleus speed, indicated by the colors of dots, oscillated from fast to slow. As nuclear density increased—shown as a higher density of dots near one another, at the same time—peak speeds reduced and periodicity became much less apparent (Fig. 1.2A). We divided all nuclei at each time point into three equal sized terciles on the basis of nucleus location (anterior, middle, and posterior; Fig. 1.2B). When we plotted speed over time for each tercile, it could be observed that while there was oscillatory speeds for all three terciles, it was less pronounced for the middle tercile and the oscillations continued longer in the anterior and posterior terciles(Fig. 1.2C).

To describe the oscillatory behavior in more detail, we used the nuclear tracks to plot speed and autocorrelation over time, with 'time' re-zeroed to begin at the most recent division for each nucleus. In doing so, wee found that nucleus speed oscillated between 'fast' and 'slow' phases, labeled Phase A and Phase B. Immediately after a division, each daughter moves quite quickly for between 20 and 28 minutes before nearly ceasing to move. At this point, the nucleus does not move for between 10 and 20 minutes before again dividing and repeating this process (Fig. 1.2D). This suggests that **there are two distinct phases of motion that must be considered.**This pattern explains the waves of global speed, highlighted in Fig. 1.2C, where the embryo has been separated into terciles, based on relative position along the A-P axis of the embryo. The central, dark blue region of the embryo has the fastest decay in speed (Fig. 1.2C), which is the region that will have the highest density of nuclei.

For each nucleus, we computed the time-since-division as well as its local density. Separating all of the data by density, we find that during the Phase A, nuclei at the lowest density move substantially faster than those at the highest density (Fig. 1.2D, F, G). During the Phase B, there does not appear to be a discernible difference. We also note that autocorrelation of movement is slightly higher during Phase A than Phase B (Fig. 1.2E).

1.5.5 Nuclei tend to move into unoccupied space in the egg

We observed that once nuclei reach the periplasm, they cease to move through space like other nuclei. They move freely within the periplasm, but they do not leave the periplasm itself.Therefore we binned nuclei into those that were in the periplasm and those that were not, and for each subset we calculated the tendency of nuclei to move into nearby unoccupied space. For each nucleus, we can compute the direction that is into the most open space relative to all the other nuclei present (see methodological details in Supplemental Information). We find that Phase A nuclei that are far from the periplasm have a very high correlation between their movement vector and the vector most into open space (Fig. 1.2H). Once nuclei have reached the periplasm, there is no clear signal that nuclei are moving into open space, during Phase A or Phase B (Fig. 1.2I). (Note: for Phase B nuclei late in development, we cannot discount the possibility that there is a subtle tendency to move into open space, due to the higher density of nuclei and the associated uncertainty in precisely measuring movement vectors, that a subtle signal is swamped by other factors).

1.5.6 Building a simulation framework of syncytial development

To summarize the previous two sections: , nuclei alternate between two broadly distinct behaviors (Fig. 1.2C-I). Phase A is a period after each division when a given nucleus moves through the syncytium. It is followed by Phase B, when the nucleus largely stays in place. With the goal of understanding how nuclei spread out throughout the cytoplasm, we focused on Phase A. During this phase, nuclei exhibit three conspicuous movement traits: they move with a high degree of auto-correlation over time, at a speed that is anti-correlated with the local density of nuclei, in a direction that is preferentially oriented into open space.

In the present study, we asked whether these three traits could be generated by different modes of preblastoderm nucleus movement, specifically:

1. A precise, stereotyped sequence of oriented nuclear divisions
2. Cytoplasmic flows that move nuclei

3. Whole-embryo morphogen gradient(s) that specify nucleus speed in a concentration-dependent—–and thus position-dependent—–manner
4. Brownian motion of nuclei
5. Mutual repulsion of nuclei
6. A local, asymmetric, active pulling force on each nucleus

These modes of movement are not mutually exclusive. Based on the empirical data presented above, we concluded that (1) through (4) are not important contributors to the three movement traits listed above. We also used a simplified modeling approach to assess the plausibility of (5) and (6). This was an effort to generate quantitative, falsifiable hypotheses, rather than an attempt to explain all physical phenomena that are occurring during preblastoderm formation. Therefore, we used a minimal set of assumptions about how early cricket development proceeds. At the time of this writing, work on (5) remains in progress. Therefore, below we will only describe the modeling work to assess (6).

This model is intended to represent a class of hypotheses that are molecularly specific in proposing some ability of nuclei to sense their neighbors and get pulled asymmetrically into open space. We propose a simple geometric concept that does both at the same time. The model is based on mutually exclusive shells around nuclei that can 'pull' on the cytoplasm/yolk/cortical actin network surrounding them (see Fig. 1.4A-D). We fit a force associated with the pull, calibrated on the empirical speed distribution and a radius based on measured density dependencies, and then directly implemented the division and movement-to-periplasm biases that were observed in the empirical data. We found that our model was already able to capture all three Phase A nuclear behaviors at onces—namely autocorrelation of movement, density-dependent speed, and the movement into open space.

More specifically, we imbued each shell with an ability to pull on the cytoplasm it touches with a small tug at each voxel on its surface, the relative magnitude of which is determined by the constant *P*. The net movement of a nucleus is thus determined by the sum of all the pulling vectors across its shell's surface. Each tug on the point cloud causes a torque on the nucleus; this means that a shell with certain shapes can cause an entire nucleus to rotate. During every nuclear cycle, a shell begins in an "on state," during which it generates movement, and an "off state," during which the shell is absent, with the nuclei moving only due to simulated Brownian motion. Each nucleus stays in the "on state" for T minutes before switching back. R_{max}, *P*, and *T* were tuned to match the speed oscillations in the previously described empirical data (see the Methods section for details). We likewise incorporated a density-dependent nuclear cycle, with the period of time in the "off state" determined by a linear fit to local nuclear density in the empirical data. Lastly, we found that our model of local interactions in a homogenous cytoplasm did not to recapitulate the formation of a syncytial blastoderm, as there was no force moving nuclei to the periplasm. Therefore, we added a slight attraction toward the surface of the egg, the magnitude of which was determined by the constant S. We tuned S to match the rate at which nuclei move towards the surface of the egg in empirical data (full details can be found in the Supplemental Information).

1.5.7 Simulation results

We did not explicitly encode density-dependent speeds, autocorrelation of movement, or an attraction to open space, yet all of these patterns emerged in the simulated nucleus movements. We also did not incorporate viscosity of the cytoplasm, fluid flows, pushing forces of any kind, or any maternally provided signals in the yolk (spatially heterogenous or otherwise). Still the the model successfully recapitulated several quantitative features that we measure of the true data.

For instance, throughout syncytial blastoderm formation, the distribution of simulated nuclei along the A-P axis matches that of real embryos. We show a single time point (468 minutes from the start) from a real dataset (green) plotted on top of a series of simulated embryos (Fig. XXX[2]). Simulated divisions and Brownian motion are stochastic, so each simulated embryo is unique. Simulated speed oscillations are likewise similar to empirical oscillations (Fig. 1.4H). The model is less effective in matching the movement of nuclei into the periplasm. Simulated nuclei move toward the surface, but they never become as close to the surface as is observed in real embryos, largely because of steric effects limiting the extent to which the nuclei can be pulled towards the surface. It is possible that nuclei reaching the periplasm in real embryos become physically anchored or otherwise undergo a change that alters their movement tendencies. Based on the results of these simulations, we concluded that (6) is a compelling fit to the empirical data of nucleus be-

[2]This figure is being updated based on the new model geometry.

havior. Thus, we hypothesize that there is a local, asymmetric, active pulling force on each nucleus.

Our model of asymmetric pulling shells was able to capture the complex motions that are observed empirically in wild-type embyros. Next, we simulated what would happen if the embryo were to be physically manipulated. We simulated the development of an embryo whose eggshell has been radially squeezed down adding a geometric bottleneck in the middle of the egg. We found that the change in geometry specifically affected the pattern of nucleus movements *in silico*. On the whole, nuclei in simulated constricted eggs slow down earlier, and nuclei that pass through the constricted region suddenly accelerate, moving faster than those that do not pass through the constriction.

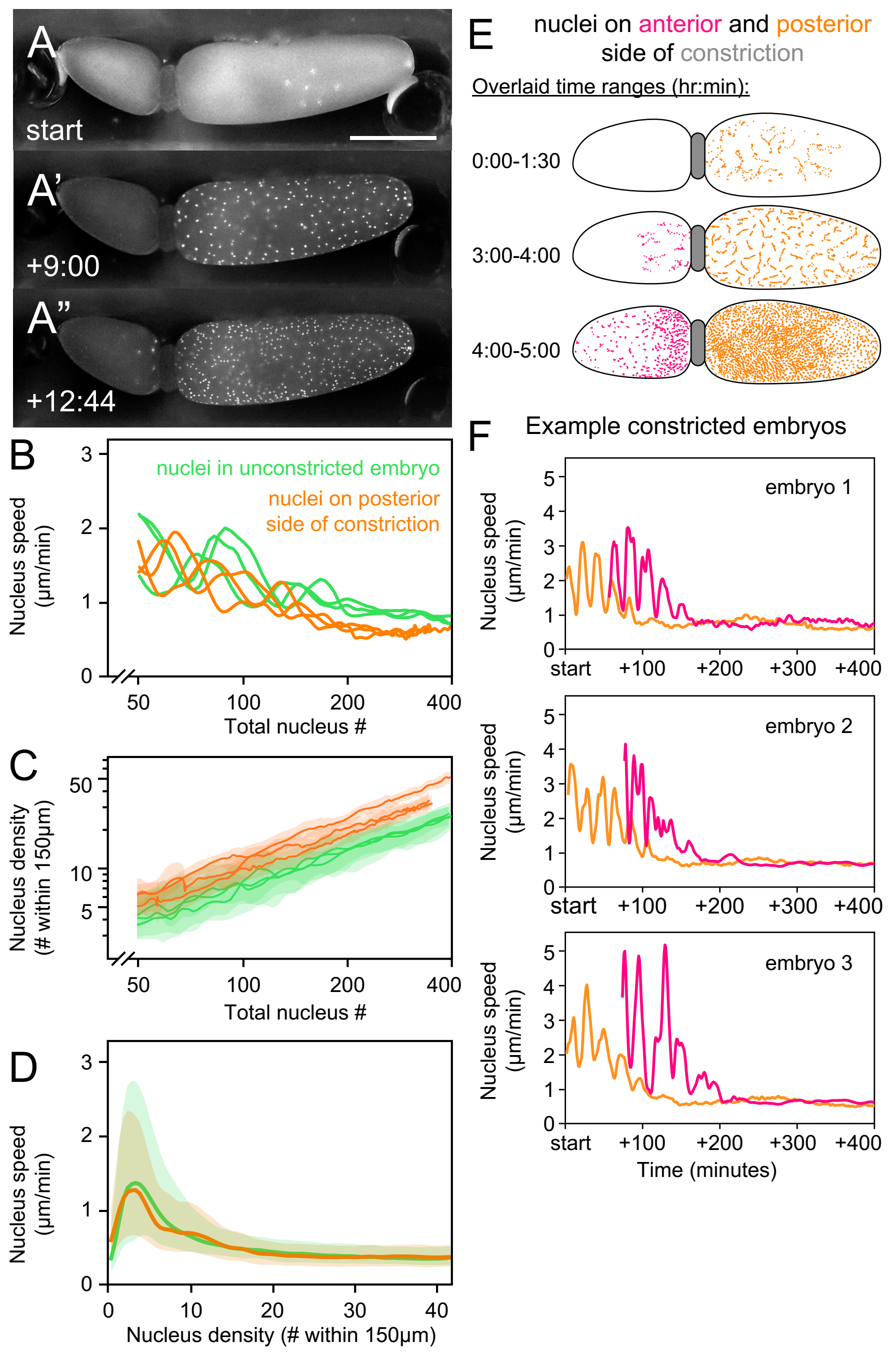
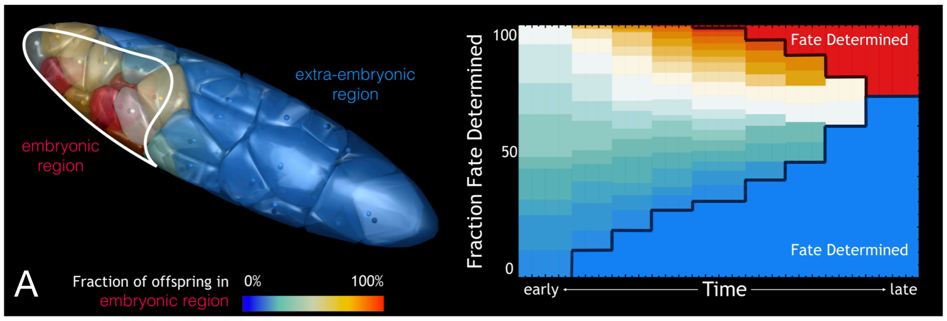


Figure 1.5: Constricting developing embryos. **A**, Three images showing different stages of development of a constricted embryo. In the final stage, nuclei have moved through the constriction. **B**, By measuring the speed of nuclei in each side of the constriction, we find that nuclei that escape through the small constriction move at the same speed as those in the anterior side much earlier in time. Each trace represents a different embryo. **C**, Based on the number of nuclei and the geometry of each region, we compute the average density and the 25th-75th percentiles of density. Each trace represents a different embryo. **D**, For each nucleus, we compute the density and the speed. We show that the speed-density relationship is the same on both sides of the constriction. **E**, Tracks for three different windows of development. **F**, For three different embryos we show the mean speed as a function of time. Once nuclei get through the barrier they move much faster, shown in pink.



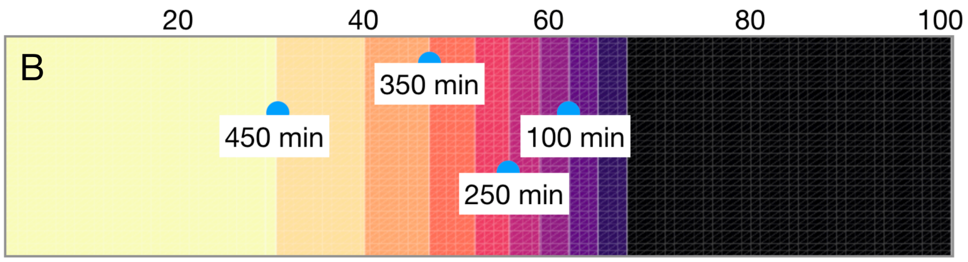


Figure 1.6: Nuclear fate. A, On the **left**, we show a rendering of a snapshot from the model. The embryonic region is outlined in white. Each shell has been colored by what fraction of the descendants end up in the embryonic region. The more blue shells are entirely fate determined to be "extra-embryonic" and the more red nuclei have a higher fraction of their nuclei end up as "embryonic". On the **right** we show a heat map showing, through time (*x*-axis) what fraction of nuclei have had their fate determined (*y*-axis). The image is the result of 100 different random runs from the model. **B**, Relative to the long axis of the embryo, we compute the expected time before coalescence where a nucleus would need to begin moving towards the embryonic region. The calculation is based on the observed density profiles as a function of (x, t) and the 95th percentile of speed as a function of density.

1.5.8 Constricting embryos

Next, we put this prediction to the test empirically by experimentally manipulating embryos. We did so by developing a device to physically constrict an egg from the outside by wrapping a human hair around it, while holding in place in water on a glass bottom dish for time-lapse microscopy, (see Supplemental Information for detailed methods). To test the ingredients of the model—and the central hypothesis that density is key to most as-

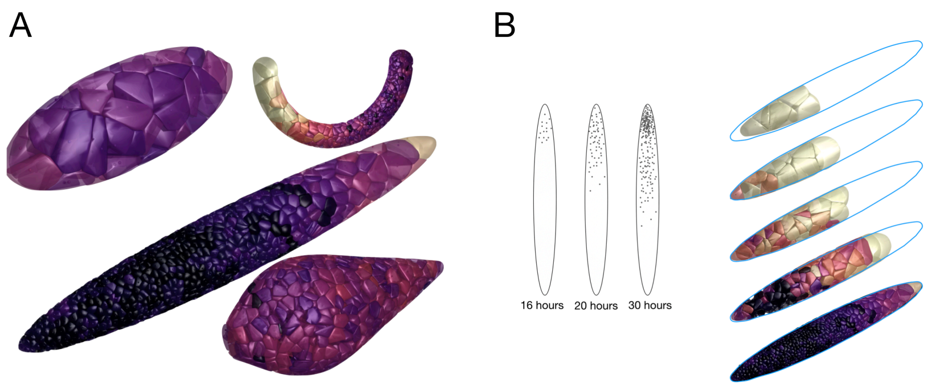


Figure 1.7: Running the model in a diversity of egg shapes. A, Using the model, we leave all parameters the same and only change the external geometry. In all cases, we show a single snap show from the model, with shells colored by instantaneous speed. Warmer colors represent faster speeds. **B**, Using the model, we leave all parameters the same and only change the external geometry to match that of *Schistocerca gregaria*. On the left, we show a series of panels approximated from Ho, *et al.* showing the location of nuclei at three time points. On the right, we show our model with shells colored by speed. Note that a very high density region forms before nuclei reach the far end of the embryo.

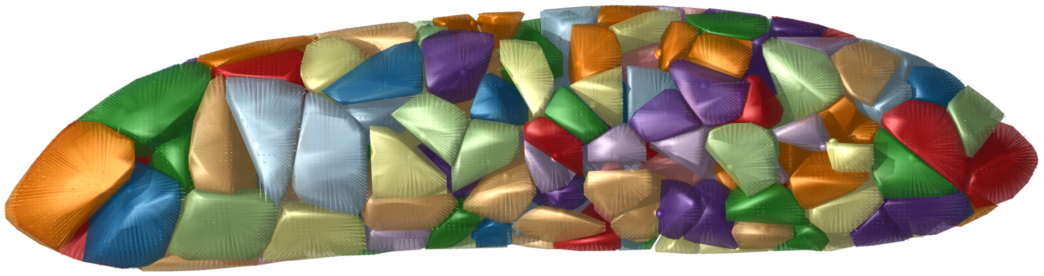


Figure 1.8: Simulated late stage of development. Using the model, we show a single time point which each "shell" rendered. In each shell, the pulling rods are faintly shown.

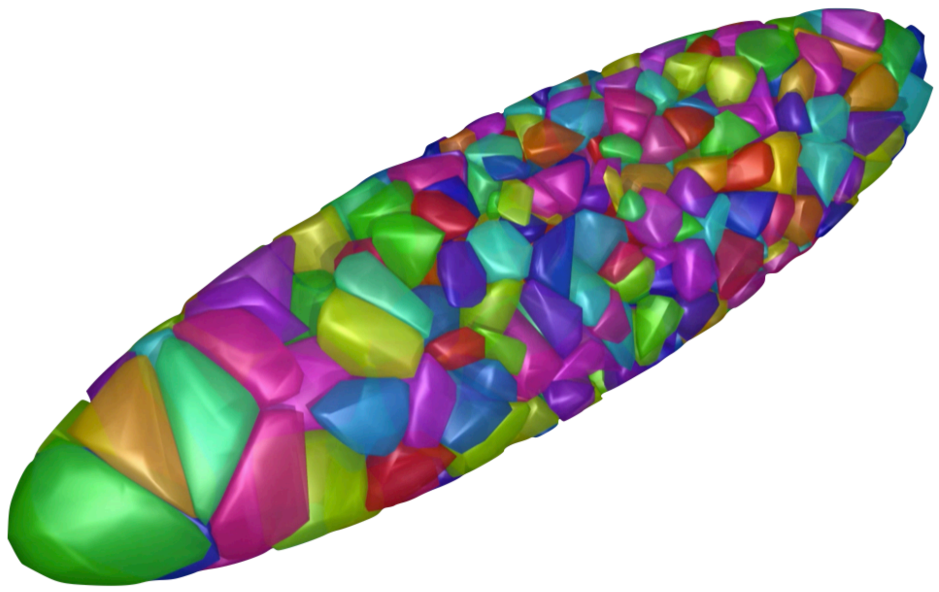


Figure 1.9: Simulated late stage of development. Similar to Fig. 1.8, we show a rendering of a late developmental stage in an ellipsoidal geometry.

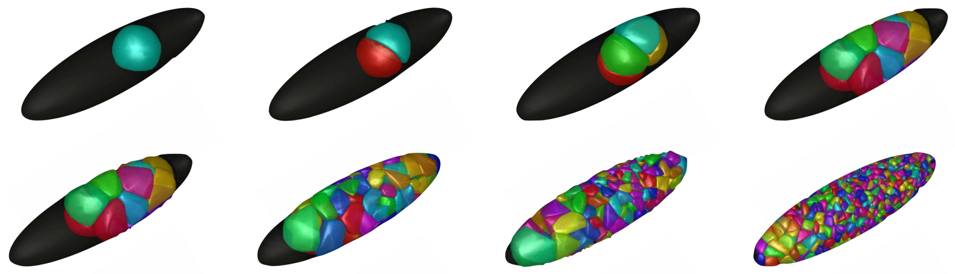


Figure 1.10: Developmental model of *Gryllus*. Eight different snapshots though time of the developmental model. In each snapshot, the shells are shown in different colors.

pects of motion— we constricted the anterior end of the eggs. By pinching a region down to about one-third the radius of the embryo, we were able to alter the local and global patterns of nucleus density over time, and then see if the effects match those predicted by the model. Because nucleus density changes are occurring at a time and place that are different from an unconstricted egg, we were able to decouple speed, autocorrelation, and space-seeking behavior from possible unobserved cytoplasmic determinants. .

By constricting embryos and altering the local density of each nucleus, we found that nuclei still obeyed the same density-dependent relationships, even when the overall volume available to the nuclei is smaller (see Fig. 1.5A). As nuclei divided and move around in the posterior region, the total space available to them was compressed, which caused individual nuclei to experience higher densities earlier in development than they would otherwise (Fig. 1.5C). We found that nuclei slow down as we would expect if density determined speed, *not* location or developmental time (Fig. 1.5B, D). In some cases, nuclei were able to get through the constricted bottleneck region and were then exposed to the constricted anterior region (see example tracks in Fig. 1.5E). In these cases, the pioneer nuclei behaved as one would expect based on their *density* and not their developmental time. Specifically, we found that nuclei that move through the bottleneck nearly two hours after axial expansion begins move at the same speed as those much earlier in time (Fig. 1.5)F).

1.5.9 Using the model to make specific hypotheses

With the simulation able to recapitulate the movement and division features of the cricket embryo, we were able to use it address an open question: when are nuclei fated to become either embryonic or extraembryonic tissue. In *G. bimaculatus* a relatively small fraction of nuclei end up in the embryonic region when coalescence occurs while the rest become extraembryonic tissue.[99] We showed that the mechanics and geometry of the embryo dramatically restrict fate outcomes of nuclei at early stages of preblastoderm development. In fact, we find that the first few divisions almost entirely determine the embryonic versus extraembryonic fate of all nuclei (see Fig. 1.6).

One approach to studying cell fate decisions is to track individual nuclei (or cells) from a time point before a differentiation decision until a time point after the cell fate decision has been visibly (and obviously) made. The can require extremely high accuracy in track reconstruction. For example, for tracking nucleus fates from the beginning of cricket development through the first known cell fate decision (embryonic vs. extraembryonic fate), we would need to have track link accuracy higher than 99%. With 95% accuracy, a nucleus is expected to have typical track length 13 time point. Even with 99% accuracy, tracks lengths would be less than 75 time points—hardly two division cycles.Thus, from our data, it is difficult to track individual nuclei for the time needed to answer questions pertaining to nuclear fate. However, using our validated model for *in silico* nucleus behavior, we are able to make predictions about fate determination without perfect track reconstructions. We defined adefinined an region of the periplasm where the embryonic rudiment would form. Then, we could ask for any given nucleus what fraction of its descendants end up in the predefined region at a later time point. Over time, the fraction of nuclei in each region for a given nucleus approaches either (0,1) or (1,0). This is shown in Fig. 1.6A. We can also address the question of fate from a different angle, based on observed speeds and densities. Based on the data, we estimated how early in development fate determining decisions could possibly be made. We asked, based on our data, at any given time point in development, what is the furthest away from the embryonic region that a nuclei can be such that, traveling optimally, it would be able to make it into the embryo? We do this by fitting the 95th percentile of speed as a function of density, and by fitting density as a function of location (approximated using the x-axis of the re-oriented embryo) and time. We use maximum speed rather than average speed to ensure we have a very conservative estimate. Based on the results in Fig. 1.6B, we find strong evidence that early divisions matter very much in fate determination. Once the density of nuclei becomes high enough, nuclei move so slowly that their descendants could never possibly make it in to the embryonic rudiment. In the empirical data, we do not ever see distant nuclei moving atypically quickly towards the zone of embryonic coalescence, bolstering our hypothesis that early divisions are particularly formative for subsequent fate outcomes.

Using the model, it is possible for us to alter the geometry of the egg, leaving all other parameters the same as in *G. bimaculatus*. Examples are shown in Fig. 1.7. While we have not yet performed quantitative analysis comparing results to other species, we do compare our results qualitatively to development in *Schistocerca gregaria*, described by Ho, *et al* (1997).[73] *Schistocerca gregaria* is a grasshopper that lays eggs that are nearly 5$\times$ longer than *G. bimaculatus* eggs.[73] Additionally, in *Schistocerca*, the first nucleus is division occurs more asymmetrically along the A-P axis than that of *G. bimaculatus*. The result is that embryonic coalescence begins at one end of the embryo before nuclei have even reached the other end.[73] We found that our model, with only the geometry changed, accurately captures the gestalt of the grasshopper blastoderm formation (see Fig. 1.7). We also show that, even without an identified molecular basis for the nucleus-moving forces in the embryo, our simulation framework can be used to better understand fate determination in crickets and to explain patterns of blastoderm formation in other insect species.

1.6 Discussion

In using 3D+T imaging and quantitative characterization of nucleus movements, we found that preblastoderm development of crickets has several striking difference from that of fruit flies. In cricket, at a given time, the duration of division cycles vary in space; nucleus speeds are strongly associated with local density; and nuclei move with a high degree of autocorrelation in to nearby unoccupied space. We propose that all of these patterns can be most readily explained by positing an ability for each nucleus to sense the geometry of their neighbors and then divide and move accordingly by being pulled into adjacent open space. explained most convincingly

Although it was intentionally built as a simplification, we found our model helpful in illuminating the empirical cricket data. In real embryos, we observed that nucleus speed, mitotic cycle duration, and movement orientation are all associated with geometric features—i.e. the density and placement—of neighboring nuclei. The "shell" serves as a generalized representation of an unknown mechanism by which a nucleus can sense the positions of neighbors. Large shells result in a higher speed in the simulation, and the largest shells form on the frontiers. Likewise, unimpeded nuclei do tend to move faster in the embryo, with the fastest nuclei moving on the frontiers, towards the poles. We interpret these results as evidence that syncytial nuclei move via active migration through the cytoplasm. There may be cytoplasmic flows and/or spatially heterogenous cytoplasmic signals in the egg as well, but neither of them is required in order to build a model that broadly captures the dynamics of syncytial nucleus movements in the cricket embryo.

Our model for the neighbor-sensing mechanism and nucleus-moving force was intentionally left molecularly abstract. The shell-based model conforms, qualitatively, to better-studied systems of nucleus positioning, in *D. melanogaster* and *C. elegans*, such an a nucleus-associated MT aster interacting with a peripheral actomyosin cortex of minus-end directed motors.[62,145,43] However, other MT-independent molecular mechanisms could also generate an asymmetric active force: locally diffused and degraded gradients coupled to dynamic and asymmetric remodeling of the local cytoskeleton, or asymmetric contractile interactions with a largely uniformly actomyosin network at the periphery of each energid, long range forces exerted by MTs in the cytoplasmic builk independent of asters, or a molecular mechanism that we have not observed.

Our results show that cytoplasmic flows as dramatic as those observed in *D. melanogaster* are not present in *G. bimaculatus* (see Fig.1.1C, D). It is conceivable that there are cryptic flows in the middle of the cytoplasm, a small portion of the volume of the egg where our ability to detect and track nuclei was weakest. However, according to the model introduced in Deneke *et al* 2019,[44] it should not be possible to only have flows in the middle without a corresponding counterflow in the cortex. In fact, there is a subtle elasticity in the cytoplasm that is actually detectable in our data: immediately after Phase A nuclei move quickly towards the egg's poles, there is a very slight tendency for the nuclei drift slightly backwards away from the direction of their fast movement. Although it is too small to account for the large-scale motions of nuclei we documented, it nonetheless demonstrates that our simplified geometric model without fluid flow is missing an ingredient that would be helpful for capturing the patterns preblastoderm development more accurately.

Recent work showed that syncytial insect eggs have a wide range of shapes and sizes.[34] In our simulations, egg size and shape—as well as the location of the initial zygotic division—contribute to the patterns of nucleus behavior over the course of blastoderm formation. If an asymmetric pulling mechanism turns out to be a good descriptor for syncytial nucleus movements insects with eggs of diverse sizes and shapes, we would hypothesize that over a macroevolutionary timescale, the mechanisms that regulate syncytial nuclear behaviour in early insect embryos co-vary predictably with egg morphology. Specifically, larger eggs should tend to have greater inequality in the nucleus density across the egg. This, in turn would lead to a broader variation in speeds and likewise a stronger difference in the timing of when nuclei reach the periplasm. By a similar logic, eggs that are closer to a spherical shape should have a reduced tendency for asymmetry in their spatial distribution of speeds. An alternative is also possible: that the molecular mechanisms underlying nucleus movements are more diverse than previous workers anticipated. Quantitative, comparative research will be needed in order to determine precisely how distinct the pre-blastoderm nuclear patterns are among insect species. As for whether the molecular mechanisms underpinning nucleus movements differ among taxa, that too is an open question. Descriptive accounts of the cleavage stage from other species reveal that there are still more patterns awaiting deeper study.

1.7 Supplemental Information

Most of the technical details for this project have been written in the supplemental information, included below.

Blastoderm formation in the cricket proceeds by local nucleus crowding

Supplemental Information

Seth Donoughe[†,1,2], Jordan Hoffmann[†,3], Taro Nakamura[4], Chris H. Rycroft[3,5], Cassandra G. Extavour[1,6]

Contents

[†] Seth Donoughe and Jordan Hoffmann contributed equally to this work
[1]Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, MA, USA
[2]*Present address:* Department of Molecular Genetics and Cell Biology, University of Chicago, Chicago, IL, USA
[3]Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA
[4]Division of Evolutionary Development, National Institute for Basic Biology, Okazaki, Japan
[5]Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA, USA
[6]Department of Molecular and Cellular Biology, Harvard University, Cambridge, MA, USA

1 Processing, segmenting, and tracking time-lapse datasets

1.1 Datasets from lightsheet microscopy

Lightsheet datasets were fused using the `Multiview Reconstruction` plugin for `Fiji` [1, 2]. Fluorescent beads embedded in the agarose served as interest points for registration; these were used to combine multiple views as a weighted average fusion.[1] Deconvolution was done for 10 iterations with the `Multiview Reconstruction` plugin.[2] Calculations were performed on a 32-core workstation with 128 GB of RAM running `Ubuntu 14.04`.

Nucleus segmentation and tracking was performed with `Ilastik` [3]. After importing all data as a 4D sequence, a pixel classifier was trained to differentiate nuclei from background. A pixel probability map was generated, and then nuclei were identified using the object classification tool. Nuclei were automatically tracked in `Ilastik`, with the tracking tool configured to ignore divisions. In our hands, automated tracking of nucleus movements was very accurate but automated detection of divisions was less accurate. Therefore, in order to capture divisions, a custom script was used to convert the automated tracking output from `Ilastik` into an `XML` file that was parsable by the semi-automated tracking `Fiji`[4] plugin `MaMuT` [5]; this feature is now included in more recent versions of `Ilastik`. We used `MaMuT` to manually identify each division and stitch tracks together to generate continuously tracked lineages.

1.2 Datasets from epifluorescence microscopy

Arrays of up to 40 eggs were recorded at a time by tiling across a field of microwells, each of which held a single egg, oriented laterally. Datasets were processed in `Zen Blue` (Zeiss) by combining z-slices with the `Extended Depth of Focus` tool (`mode=Contrast`) and then stitching tiles together with the `Stitching` tool.

Nuclei were segmented in `Ilastik` as described above. Nucleus movements were automatically tracked in `Ilastik`. Divisions were manually identified in `MaMuT`. For eggs mounted in this manner, approximately one-half to one-third of the z-depth of the embryo could be imaged clearly; the signal from rest of the embryo was diffused by the yolk of the egg. The number of well-segmented nuclei fluctuated slightly between time points, depending on the particular paths traveled by individual nuclei; the effect of this variability was strongest for the first few division cycles. Thus, for 2D+T time-lapses, we performed automated analysis on these datasets beginning at the earliest time point when there were at least 25 nuclei. We also performed manual tracking on divisions and movements of nuclei before there were 25 detectable nuclei; in some of those cases, we were able to sometimes capture even the first division.

1.3 Datasets from confocal microscopy

Embryos were imaged individually as 3D+T time-lapses, and then cropped and assembled into montages and figures using `Fiji` and custom Python scripts.

1.4 Microscopy sample sizes

Lightsheet datasets of blastoderm formation: These 3D+T time-lapses enabled us to reconstruct detailed reconstruction of nucleus movements and divisions but they are time-intensive to analyze. For the present study we include four tracked and processed 3D+T datasets. Figures 1, 2, and 3 in the main text show data from a single embryo. The same analyses were repeated on the other three datasets; we found that the same patterns of nucleus dynamics held across all embryos.

Epifluorescence datasets of blastoderm formation: We also analyzed a much larger set of 2D+T datasets to confirm that the lightsheet-recorded embryos exhibited typical cricket development. We have captured and analyzed 64 such datasets.

Epifluorescence datasets of initial cleavages: We recorded and manually tracked the positions of nuclei after three division cycles for >30 individuals in order to assess whether the initial divisions are stereotyped. As described in the main text, we found that the division positions and orientations differ among individuals. However, we did detect a subtle bias in division orientation, which may be due to the geometric constraints of nuclei dividing near the inner surface of an elongated ellipsoid volume. These results are summarized in Fig. 1.6 in the main text.

Epifluorescence datasets of constricted embryos: We include 11 time-lapses of constricted embryos. These embryos do not hatch, so our criteria for including a constriction dataset were: no visible ruptures of the eggshell, no premature arrest of nucleus movements, and no nuclear aggregations during development. In our recordings of *unconstricted* embryos, we found that each of these issues reliably predicted that blastoderm formation would fail.

Confocal datasets of yolk and nucleus movement: We include 13 time-lapse datasets of eggs laid by females with Act-mem-mTomato and Act-H2B-EGFP transgenes. We used a magnification such that about one-fifth of the length and one-half of the breadth of the egg was recorded at a time.

1.5 Nucleus tracking accuracy

In order to assess the accuracy of automated nucleus tracking, we took 300 successive frames of a lightsheet dataset, used the semi-automated tracking function of `MaMuT`. Then we inspected each nucleus position and link that was initially tracked, and manually corrected any incorrect positions or links, resulting in >40,000 single time-point nucleus observations. Then, we used `Ilastik` to segment and automatically track the same dataset, resulting in a comparable number of detected nuclei.

Treating the manually corrected dataset as ground truth, we assessed the accuracy of the automated approach by calculating the distance from each nucleus in the manually tracked dataset to the closest corresponding nucleus in the automatically tracked dataset. Table 1 shows percentiles of these distances. In general, the spatial discrepancy between ground truth nucleus position and automatically tracked nucleus position was much smaller than the diameter of a single nucleus (~10 microns; see Fig. 1.1 in the main text for comparison).

Percentile	*Distance (microns)*
10%	0.6
25%	0.9
50%	1.4
75%	1.9
90%	2.9

Table 1: **Tracking discrepancy.** Percentiles of spatial discrepancies between each nucleus in a manually tracked dataset and the closest corresponding nucleus in an automatically tracked dataset. We calculated distances for >40,000 manually tracked nucleus-time points.

2 Quantitative measurements of nucleus behavior

2.1 Nucleus speed

For 3D+T datasets, we defined a vector

$$\vec{x}_t = (x_t, y_t, z_t) \tag{1}$$

for a nucleus position at time t. Using this notation, instantaneous nucleus speed, s_t, was defined as half of the displacement over two time points,

$$s_t = \frac{1}{2}\|\vec{x}_{t+2} - \vec{x}_t\|. \tag{2}$$

For 2D+T time-lapses, speed was defined in an analogous manner, calculating movement only in the xy plane. The z-component of motion was not available in such datasets, which meant that calculated speeds were an underestimate of the true nucleus speed in 3D space. However, once nuclei reach inner surface of the eggshell, the majority of their movement is constrained to the periplasm. This means that for nuclei that were near the center of the image, far from the edges of the egg, the distorting effect of flattening the z-dimension was smallest. For 2D+T datasets with a small number of nuclei (i.e. those for which we tracked the positions of nuclei over the first three divisions) we mounted the eggs in a manner that minimized distortion. Since the initial divisions typically occur near the ventral midline, we oriented the egg with the ventral midline centered toward the light path. We also excluded any datasets for which the visible nuclei were mostly near the edges of the egg.

2.2 Correlation of nucleus movement vectors

For each embryo, the set of $\vec{x}_t$ across all t was re-oriented so that its first principal component lay along the x axis, in effect rotating the dataset so that the long axis of the embryo was parallel to the x axis. For each $\vec{x}_t$, the motion vector from $\vec{x}_t$ to $\vec{x}_{t+2}$ was computed, and then used to calculate the correlation between pairs of nuclei.

2.3 Local nucleus density

We defined local nucleus density as the number of nuclei within radius $\mathcal{R} = 150$ µm[1], weighted by the volume of space around the focal nucleus, considering only the space that is contained within the eggshell. For a nucleus near the periplasm of the egg, the sphere of space within $\mathcal{R}$ includes some volume that is outside of the egg itself.

Therefore, we needed to numerically represent the surface of the egg. We took a single time point at the uniform blastoderm stage, segmented the nuclei, and fit a parabola to the Anterior-Posterior (A-P) axis of the cloud of points. We used this parabola to transform the positions of nuclei by mapping the parabola to a straight line. Then we calculated the convex hull of the transformed points, and applied the reverse transformation to the convex hull. This produced the volume $\mathcal{B}$.

We took the sphere $\mathcal{S}$ defined by $\mathcal{R}$, and then defined

$$V = \text{volume}(\mathcal{S} \cap \mathcal{B}). \tag{3}$$

The volume fraction was computed as

$$\phi = \frac{V}{\frac{4}{3}\pi\mathcal{R}^3} \tag{4}$$

and the local density was defined as

$$\rho = \frac{\#}{\phi}, \tag{5}$$

[1]We found that $\mathcal{R} = 150$ µm was effective at capturing density-dependent speeds and division rates across the range of densities found during cricket blastoderm formation; similar results were obtained for radii from 50 to 300 µm.

where # represents the count of nuclei within $\mathcal{R}$.

For 2D+T datasets, local nucleus density was treated in an analogous manner: we computed the region of overlap of a 150 µm circle with the 2D convex hull of all nuclei at the blastoderm stage. Then we counted the number of nuclei within that area and calculated the area-weighted density as above. In practice, this means that nuclear densities were calculated from volumes with different shapes in the 2D+T vs. 3D+T datasets (i.e. cylindrical vs. spherical), and therefore their measured values cannot be compared to one another in absolute terms.

2.4 Rate of change in number of nuclei

Early in embryogenesis, a small fraction of the nuclei are in the very middle of the egg, where their observable fluorescence signal is most diffuse. Since it is possible that some unobserved divisions occur in that small portion of the egg volume, *percent change in nucleus number* was used as a proxy for nucleus division, rather than counting nuclei directly. At each time point for a given embryo, the total number of nuclei present was smoothed by the application of a Gaussian blur with a width of three time points. A Hermite interpolant was fit through the data and then divided by its derivative: $N'(t)$, by $N(t)$, to produce a rate of change in the number of nuclei at each time point.

2.5 Motion toward open space

We defined direction towards the "largest open space" as the vector towards the centroid of the 3-dimensional Voronoi cell formed by each nucleus and bounded by the inner surface of the eggshell—as approximated by the convex hull of nuclei at the blastoderm stage.

For each nucleus, we considered $\vec{v} = \frac{1}{2}(\vec{x}_{t+2} - \vec{x}_t)$, and calculated its correlation c with a vector into the direction into open space, $\vec{s}$, as

$$c = \frac{\vec{x} \cdot \vec{s}}{\|\vec{x}\| \|\vec{s}\|}. \tag{6}$$

In order to account for noise in this calculation, average "movement into space" vectors were calculated for each nucleus over a sliding window of three time points. In addition, we computed the shortest distance from all nucleus locations $\vec{x}_t$ to the inner surface of the eggshell, d_S. We binned nuclei by d_S into those that are near the surface ($d_S < 50$ µm) and far from the surface ($d_S \geq 50$ µm), as shown in Figure 3.

3 Simulating blastoderm formation

3.1 Overview and aim

As described in the main text, our initial goal of a modeling approach was to assess whether mutual repulsion of nuclei or a local, asymmetric, active pulling force on each nucleus could satisfactorily recapitulate the empirical patterns of nucleus behaviors. Here we report on the model in closer detail.

3.2 Model components

3.2.1 Egg

We used the shape of a real cricket egg, approximated from the positions of nuclei during the uniform blastoderm stage, as described in Section 2.3. As a function of position along the A-P axis z, each cross section is assumed to be well approximated by an ellipse with radii $r_1(z)$ and $r_2(z)$ with a center at $(x(z), y(z))$. This defines a boundary $\mathcal{B}$. For simulated *G. bimaculatus* embryos, all calculations were done on a discretized 3D grid with dimensions $400 \times 100 \times 100$ (4,000,000 total voxels). We found that this resolution was sufficient to compare hypothesized models of nucleus movements; for future work it could also be rescaled to improve accuracy at the cost of compute time. The same procedure allows us to define more complex egg shapes, as shown in the main text. For those egg shapes, we adjusted the pixel dimensions accordingly.

3.2.2 Nuclei

Nuclei were treated as spheres ($radius$ = 5µm) that could move anywhere within the egg. The cytoplasm was treated as a uniform material. We did not simulate the viscosity or elasticity of the cytoplasm. Rather, nucleus movements were simply generated as the sum of any repulsion and attraction forces. Each source of force in the simulation is described below as a separate component to the model. Overall, the forces were assigned magnitudes such that the maximum nucleus speeds in the simulation matched those of the empirical data.

3.2.3 Division geometry

Nuclei divided in random directions throughout the simulation, irrespective of the proximity of neighbors, proximity of the eggshell, or the orientation of previous divisions in a lineage. The physical separation of daughter nuclei after mitosis and re-formation of their nuclear envelopes was treated as zero. That is, newly divided daughters formed immediately adjacent to one another.

3.2.4 Density-dependent cell cycle duration

In the empirical data, we observed a strong positive association between local nucleus density and cell cycle duration. We did not attempt to explain the density-dependent cell cycle duration in *G. bimaculatus*. Instead, we included the empirical relationship directly. We fit a logistic curve to the empirical density vs. cell cycle duration data (shown in Figure 2 in the main text), and then drew from it with normally distributed noise. The magnitude of the noise increased over time, going from a standard deviation of 90 seconds at the beginning of the simulation up to 360 seconds at the end of the simulation, when nucleus density was highest.

3.2.5 Bias of nucleus movement toward periplasm

In an unmanipulated embryo, the first zygotic division tends to occur about ~60% of the way along the A-P axis, from the anterior pole of the egg. That division is also typically slightly offset along the dorsal-ventral (D-V) axis, occurring closer to the ventral side of the egg. During the first few cell cycles, nuclei spread apart in space, with some of them moving through "open space" across the yolk-rich middle of the egg, and others moving within the periplasm, close to the inner surface of the eggshell. From the 3D+T datasets, it was clear that as nuclei continue to divide and spread out within the egg, their movement paths have a subtle bias towards the periplasm. The mechanistic basis of this bias in unknown. We speculate that it might be similar to the comparatively more coordinated movement of syncytial nuclei into the periplasm in *D. melanogaster* during cell cycles 7 through 9 [6–10]. We hard-coded this tendency in our simulations by introducing a component to a nucleus's motion vector that is in the direction of the nearest point on the inner surface of the eggshell. The strength of this bias was adjusted to match the bias towards the periplasm in the empirical data. Specifically, 25% of the instantaneous motion term for each nucleus was set as motion towards the nearest point on the surface, which was computed using an optimization routing. For each time-step, this bias term is included on each nucleus with 50% probability.

3.2.6 Candidate movement mode: Asymmetric pulling shell

As described in the main text, we hypothesized that there is a local, asymmetric, active pulling force on each nucleus. To assess the plausibility of this hypothesis we developed a simple geometric structure that serves as an abstracted stand-in for one of many possible cell biologically exact mechanisms: an asymmetric "shell," originating on each nucleus, that causes a local force to pull them through the cytoplasm.

We model the motion of nuclei by ascribing a pulling shell that emerges from a single origin on a nucleus. The shell grows uniformly in all directions to some maximal radius, $R_{\max}$, except for where its growth is impeded by the surface of the nucleus from which it originated, the surface of another shell, or inner surface of the eggshell.

The parameter $\mathcal{R}_{\text{max}}$ defines a *region of influence* around each nucleus. For each nucleus, we compute this region by using the Python library `scikit-fmm` [11]. To compute the shells, we use a level set method [12], which is a computationally efficient way to identify all the voxels that should be assigned to each nucleus. Specifically, we initialize the locations of nuclei to be zeroes of a level set. We use a uniform background velocity field, though the code framework is in place to alter this if desired. Then, we solve the Eikonal equation

$$F(\vec{x})|\nabla T(\vec{x})|= 1, \tag{7}$$

where $F(\vec{x})$ describes the speed at each coordinate, $x_{i,j,k}$ and $T(\vec{x})$ describes the travel time to each coordinate. By setting the background field to a uniform value, this allows us to efficiently compute regions of influence around each nucleus. After computing each region, we carve out a region due to steric effect of the point of shell growth's asymmetric relationship with the nucleus.

For all pixels that have a travel time less than our R_{max}, we assign them to a nucleus based on which nucleus they are closest to. Then, for each nucleus, we remove voxels that should not be considered due to steric effects—either they are outside the boundary of the egg or they are occluded based on the position of the nucleus relative to the shell's origin. Voxels that are omitted due to this second case are checked for whether they should belong to another nucleus. For each voxel in a shell, we compute the vector $\vec{f_N} = \text{normalized}(p_{i,j,k} - N_{x,y,z})$ where $p_{i,j,k}$ represents each voxel in the shell. This represents a tug in that direction. To normalize based on R, we divide each tug by a factor of R^2 where R represents the distance from the pixel to the nucleus.

For each nucleus, $\vec{n}_i$ we compute the corresponding shell, $\mathcal{S}_i$. We compute a movement vector Δ by computing

$$\Delta = \sum_j \alpha \frac{\mathcal{S}_j - \vec{n}}{||\mathcal{S}_j - \vec{n}||_2^2}. \tag{8}$$

The parameter α is fit from the experimental data where we match the maximum speed at the 4-nucleus stage. The parameter is fixed, and does not vary in time.

After each division, shell origins are randomly positioned on opposite sides of the two daughter cells. This means their the motion vectors are initially 180° apart from one another. The angle of each division is random.

3.2.7 Code output

The code outputs three different variables at each time point. For time `t` the code outputs the `locations` of nuclei, the `time since division` of each nucleus, and finally the `shell` corresponding to each nucleus. The `locations` variable contains the (x, y, z) coordinates of all nuclei that are present. The nuclei remain in the same order over time, with those that divide having one of their daughter nuclei appended to the end of the list. To allow the lineages to be properly calculated, the `time since division` variable contains the time since division of each nucleus. Each nucleus has an associated "shell" that is responsible for moving the nuclei around. For each time point, each nucleus' shell is stored in the `shell` variable, where `N` corresponds to the position in the `locations` variable.

4 Methods for physically constricting embryos

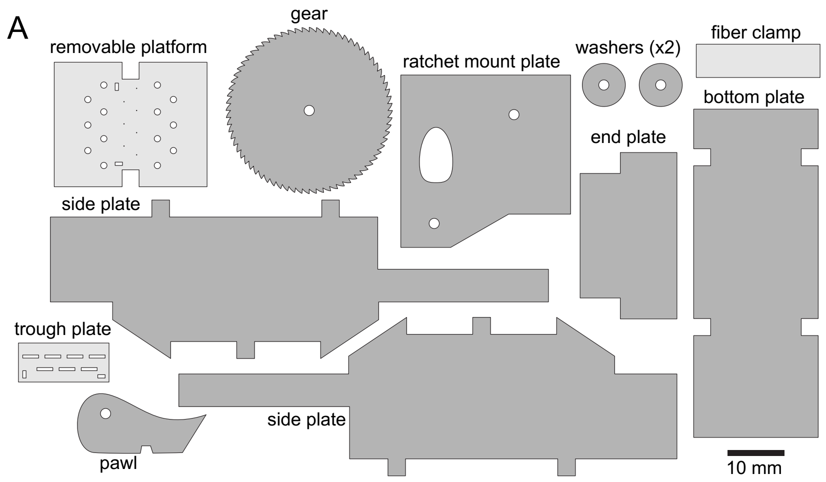
4.1 Assembling the egg constriction device

Embryos were constricted in a custom device. Components were laser-cut out of sheets of acrylic (thickness = 1.59 mm, McMaster-Carr 8560K172; thickness = 3.18 mm, McMaster-Carr 8560K257), and then assembled with acrylic welding solution (IPS Weld-On 3 Acrylic Plastic Cement) following the procedure described previously [13]. The device's construction also required a common binder clip (width = 20 mm), two steel nails (diameter = 2 mm; length = 39 mm), and a rubber band (thickness = 1 mm). We were able to successfully constrict eggs with versions of the device that were assembled with several different types of nails, rubber bands, and binder clips. The constricting fiber was held in place by being pinched in a block of elastomer (Dow Corning Sylgard 184 Elastomer Kit) into which a slit had been cut with a razor blade. A crafting hot glue gun was used to attach acrylic components to non-acrylic components. Schematics of all components are shown to scale along with additional assembly instructions in Fig. S1.

4.2 Using the egg constriction device

Figure S2A is a photograph of the device in use, with human hairs used as the constricting fibers. Eggs were placed in water-filled acrylic troughs on a removable platform, and then constricted one at a time. To do so, a hair was threaded through the removable plastic platform, around an egg, back through a hole at the bottom of a trough in the removable platform, and then attached to a ratchet mechanism. Detailed instructions for this procedure are shown in Fig. S2B. This allowed the user to increase tension on the hair while observing the egg on a dissection microscope. As tension increased, the egg was incrementally constricted. After the desired extent of constriction was achieved, temporary dabs of hot glue were used to affix the hair in place for the duration of imaging.

The device is able to hold multiple constricted eggs at a time for simultaneous imaging, up to a maximum of seven eggs. Once a set of eggs was constricted, the removable platform was taken from the constriction device, placed in a glass bottom 6-well dish (MatTek P06G-1.5-20-F), and imaged using a Zeiss Cell Discoverer microscope following previously described methods [13].



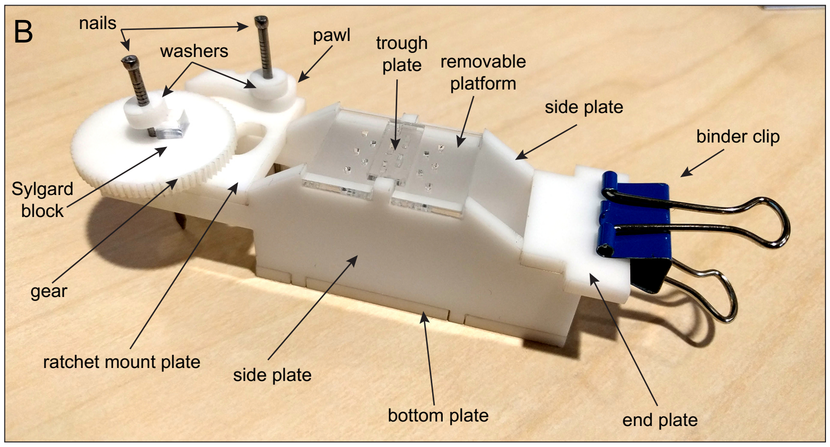
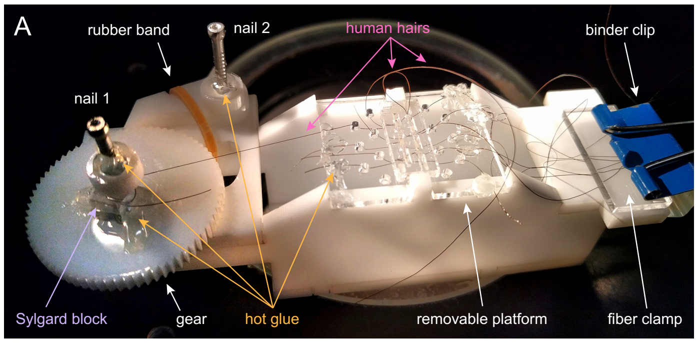


Figure S1: **Assembling a device for constricting eggs**. **A**, Diagram of all acrylic components. Pieces in light gray were cut from 1.59 mm-thick acrylic sheet; the rest were cut from 3.18 mm-thick acrylic sheet. **B**, The assembled device also included a binder clip and two steel nails. Before welding the trough plate to the removable platform, the trough plate was positioned so that the rectangular holes in the platform lined up with the corresponding holes in the trough plate. After the rest of the components were welded together as shown, one steel nail was inserted through a washer, then the pawl, then the ratchet mount plate. Hot glue was applied to the top of the washer, fixing it to the nail. More hot glue was applied to the underside of the ratchet mount plate where the nail emerged, fixing the nail to the plate. A razor was used to cut a block of Sylgard elastomer with approximate dimensions 4 mm $\times$ 8 mm $\times$ 8 mm, and then slice the block through half its depth when oriented with the largest face flat on a table. The second nail was inserted through the other washer, gear, and then the ratchet mount plate. The Sylgard block was positioned as shown, and then hot glued to the gear. Hot glue was applied to the top of the second washer, fixing it to the nail. More hot glue was applied to the nail where it emerged underneath the ratchet mount plate, fixing it in place.



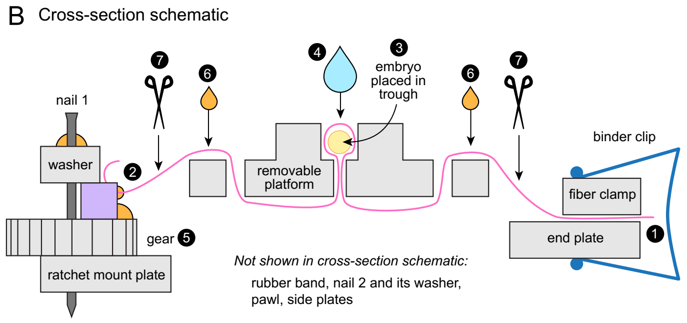

B Cross-section schematic
7
6
4
3
embryo placed in trough
6
7
nail 1
washer
2
removable platform
binder clip
fiber clamp
end plate
1
gear 5
ratchet mount plate
Not shown in cross-section schematic:
rubber band, nail 2 and its washer, pawl, side plates

Figure S2: **Constricting eggs**. **A**, Photograph of the constriction device in use on a dissection microscope. To prepare the device for use, a small rubber band was cut, inserted through the largest hole in the ratchet mount plate, wrapped underneath the side plate, and then tied in a knot. The re-joined rubber band is a loop with the band slotting into the notch on the pawl, as shown. The tension on the rubber band pushes the pawl pushed against the gear to make a ratchet. White arrows highlight the rubber band, binder clip, fiber clamp, removable platform, and gear. Hot glue indicated with orange arrows. Note: the hot glue, rubber band, and fiber clamp were not shown in Fig. S1. The fiber clamp is an unattached rectangle of acrylic that, when squeezed by the binder clip, effectively held the fibers clamped in place. **B**, Cut-away schematic of fiber threading path (not to scale). Acrylic components are depicted in light gray, nail in dark gray, Sylgard block in purple, hot glue in orange, water in light blue, and binder clip in dark blue. The egg is shown in yellow as a cross-section end-on. The constricting fiber (a human hair) is shown in magenta.

1. One hair was clamped between the end plate and fiber clamp.
2. The hair was threaded by hand through the path shown, and ultimately inserted into the slit in the elastomer block. A large loop was left in the place where the embryo would go. The removable platform can be taken off of the device to make threading easier.
3. The egg was placed into the trough.
4. Distilled water was added to the trough to fill its entire volume.
5. The gear was manually cranked while the egg was observed by the user with a dissection microscope. Surface tension kept it from leaking through the bottom hole.
6. When the desired constriction was achieved, hot glue was used to affix the hair at the two locations indicated with orange droplets.
7. The hair was cut at the two locations indicated with scissors. There are troughs for up to seven eggs; the constriction process was repeated on multiple eggs and then the removable plate was removed. To image the eggs, the removable plate was inverted, with constricted eggs still in the troughs, and placed into a glass bottom dish. The dish was filled with distilled water, and then the eggs were imaged with an inverted microscope.

5 Predicting pre-blastoderm nucleus movements in diverse insects

After assembling a model that has been fine tuned on *Gryllus*, we are able to alter only the surrounding geometry and ask how development changes and whether the resulting nuclear distributions match what is observed in other species. For a selection of shapes, see Fig. 1.7.

6 Data and code availability

All code will be made available upon the submission of the manuscript. The tracks from the lightsheet and epifluoresence datasets are available upon request.

References

1. Preibisch, S., Saalfeld, S., Schindelin, J. & Tomancak, P. Software for bead-based registration of selective plane illumination microscopy data. *Nature Methods* **7,** 418 (2010).
2. Preibisch, S. *et al.* Efficient Bayesian-based multiview deconvolution. *Nature Methods* **11,** 645 (2014).
3. Sommer, C., Straehle, C. N., Koethe, U., Hamprecht, F. A., *et al. Ilastik: Interactive learning and segmentation toolkit.* in *ISBI* **2** (2011), 8.
4. Schindelin, J. *et al.* Fiji: an open-source platform for biological-image analysis. *Nature methods* **9,** 676 (2012).
5. Wolff, C. *et al.* Multi-view light-sheet imaging and tracking with the MaMuT software reveals the cell lineage of a direct developing arthropod limb. *eLife* **7** (2018).
6. Foe, V. E. & Alberts, B. M. Studies of nuclear and cytoplasmic behaviour during the five mitotic cycles that precede gastrulation in Drosophila embryogenesis. *Journal of cell science* **61,** 31–70 (1983).
7. Baker, J., Theurkauf, W. E. & Schubiger, G. Dynamic changes in microtubule configuration correlate with nuclear migration in the preblastoderm Drosophila embryo. *The Journal of Cell Biology* **122,** 113–121 (1993).
8. Foe, V., Odell, G. & Edgar, B. in *The Development of Drosophila melanogaster* 149–300 (Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY, 1993).
9. Von Dassow, G. & Schubiger, G. How an actin network might cause fountain streaming and nuclear migration in the syncytial Drosophila embryo. *The Journal of Cell Biology* **127,** 1637–1653 (1994).
10. Ji, J.-Y., Haghnia, M., Trusty, C., Goldstein, L. S. & Schubiger, G. A genetic screen for suppressors and enhancers of the Drosophila cdk1-cyclin B identifies maternal factors that regulate microtubule and microfilament stability. *Genetics* **162,** 1179–1195 (2002).
11. Furtney, J. *scikit-fmm* `https://pythonhosted.org/scikit-fmm/`. 2012–2019.

12. Sethian, J. A. *Level Set Methods and Fast Marching Methods* tech. rep. (Cambridge University Press, 1999).

13. Donoughe, S., Kim, C. & Extavour, C. G. High-throughput live-imaging of embryos in microwell arrays using a modular specimen mounting system. *Biology Open* **7,** bio031260 (2018).

What are a friend's books for if not to be borrowed?

Tom Stoppard, Arcadia

2 Geometry of Insect Wings

2.1 Background

The geometry of the world around us has fascinated mankind for centuries. Over time, the study of pattern formation has become increasingly quantified, with specific natural mechanisms giving rise to particular patterns. One excellent example is Turing patterning, based on (at least) two chemicals that interact with one another through time where the different chemicals can interconvert.[152] This has led to many beautiful collaborations between observation of the natural world, physics, and mathematics.[94,146] More recently, this has expanded to more broadly include computation as well. Mathematical tools have been deployed for understanding geometric structure that span from anthropology to zoology, from the nanoscale to the exascale.

In the growing era of automated tools for data science, datasets themselves have become an increasingly valuable resource. However, centuries of data collection often sits in a non-digitized form in older books and artwork. For this project, we made extensive use of data from many different previously published sources. Along with each of the two manuscripts described below, we included the segmented scans that we used in hope that future scientists can make use of the dataset.

While at first glance it may not appear so, Voronoi cells[157] are *not* a good approximation to many of the shapes in Dragonfly wings. In fact, the shapes on dragonfly wings have not been well studied, and often it seems that their geometric regularity lulls one into a false sense of understanding. Even D'Arcy Thomson oversimplifies the geometry of dragonfly wings in "On Growth and Form", commenting that angles appear to be either 90° or 120°.[146] To understand how the shapes emerge, we calculated the circularity and absolute area of over 100,000 wing domains from over 500 individual wings, characterizing wings based on the distribution of shapes along the proximal–distal (P-D) axis. From here, we asked whether we could create a testable model of wing development that captured the diverse set of wing domains observed.

With Professor L. Mahadevan and his graduate student, Mary Salcedo, we deployed many of the same quantitative characterizations over a larger dataset of insect wings, which included representatives from every winged insect family. Rather than attempt to unravel underlying developmental hypotheses, in this case we chose to deploy a hierarchical set of quantitative geometric analyses over these wings.

The work in Hoffmann *et al.* (2018)[75] proposes a developmental model for the complex patterns present in odonate wings. In the manuscript, we rely on "inhibitory" centers that equilibrate over time. Between these centers, domains form, and then the wing undergoes a transformation. We have begun work where we replace this aspect of the model with a reaction–diffusion (RD) equation approach.[106,152]

2.2 Contributions

2.2.1 PNAS 2018

I developed and wrote all code for this project, including the image segmentation tool, the optimization tool, the plotting code, and analyzed wing geometry. Seth and I designed the figures together, producing my favorite set of figures in any published manuscript I have seen. I wrote a large part of the mathematical supplement (included in this document as a supplement) and assisted in writing the main text of the manuscript. I took an active role in directing the progress of the research, and with Seth we conceived of the optimization code that led to the developmental hypothesis. All of the code I developed for the project is available on my GitHub page. Additionally, we uploaded all of the segmented odonate wings for future work. This represents a very large dataset of preprocessed data that other scientists can easily access.

2.2.2 bioRxiv 2019

A large part of this work relied on code I developed for the previous manuscript. The segmentation code needed to be expanded to handle more diverse sets of images. Additionally, to handle the more diverse set of wing domains, we needed to expand the polygonization code to handle wing domain shapes that where not present in the original dataset of *Odonata* wings. Along with Mary Salcedo, I helped collect the new dataset that we used for the manuscript. Again, we uploaded all of the segmented odonate wings for future work along with all of the code needed for the manuscript. I performed all the calculations and all clustering presented in the manuscript.

2.3 Publication

This project resulted in two publications. The first of which focuses primarily on odonate wings and the second of which extends the ideas to other species as well.

A simple developmental model recapitulates complex insect wing venation patterns

Jordan Hoffmann[a,1], Seth Donoughe[b,1,2], Kathy Li[c], Mary K. Salcedo[d], and Chris H. Rycroft[a,e,2]

[a]Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138; [b]Department of Molecular Genetics and Cell Biology, University of Chicago, Chicago, IL 60637; [c]Applied Physics and Applied Mathematics Department, Columbia University, New York, NY 10027; [d]Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, MA 02138; and [e]Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA 94720

Edited by Charles S. Peskin, New York University, New York, NY, and approved July 20, 2018 (received for review January 3, 2018)

Insect wings are typically supported by thickened struts called veins. These veins form diverse geometric patterns across insects. For many insect species, even the left and right wings from the same individual have veins with unique topological arrangements, and little is known about how these patterns form. We present a large-scale quantitative study of the fingerprint-like "secondary veins." We compile a dataset of wings from 232 species and 17 families from the order Odonata (dragonflies and damselflies), a group with particularly elaborate vein patterns. We characterize the geometric arrangements of veins and develop a simple model of secondary vein patterning. We show that our model is capable of recapitulating the vein geometries of species from other, distantly related winged insect clades.

insect wings | patterning | image segmentation | computational modeling | Odonata

Insect wings are a marvel of evolution and biological engineering. They are lightweight, strong, durable, and flexible—traits made possible by "wing veins," the thickened, strut-like structures embedded in the wing surface. The density and spatial arrangement of wing veins vary tremendously among insects (1–3), wherein they serve many functions: stiffening the wing (4), resisting crack propagation (5–7), forming the vertices of corrugation (8–10), conducting hemolymph (11, 12), supporting sensory structures (13, 14), and contributing to an architecture that undergoes useful passive deformation in response to aerodynamic forces (4, 9, 15, 16).

The study of wing veins has mostly focused on "primary veins"—those whose relative positions are shared between left and right wings of the same individual and among individuals of the same species. The morphology of primary veins has served as essential evidence in the effort to place long-extinct insects into a comprehensive insect phylogeny (17, 18). Likewise, subtle shifts in the positions of homologous primary veins, quantified with the tools of comparative morphometrics, have provided insight into evolutionary patterns (19, 20) and fluctuating asymmetry—deviations from perfect symmetry that indicate developmental noise (21, 22).

In addition to primary veins, many insect species also have "secondary veins." These veins, sometimes referred to as "crossveins," cannot be matched one-to-one on the left and right wings of the same individual (2) (see labels on Fig. 1*A*). In some taxa, secondary veins comprise a large majority of wing veins yet remain poorly described. For species that have them, secondary veins form a unique pattern on every wing, which suggests that a stochastic patterning mechanism is responsible for their formation. To our knowledge, the geometric arrangement of secondary veins has not been quantitatively characterized for any species. It is not known whether a universal developmental process generates the diverse secondary vein arrangements found among insects. In fact, because the best-studied model species (e.g., *Drosophila melanogaster*) do not have secondary veins, the developmental basis of their patterning remains a mystery.

We collected original high-resolution micrographs and combined them with published wing tracings, resulting in vein patterns of 468 wings from 232 insect species. This dataset is composed of wings that span a 36-fold range in area, and it includes representatives from three taxonomic orders. We developed computational tools to segment images of wings and used them to calculate geometric traits for each digitized wing image, including vein lengths, connectivities, angles, and densities.

With the resulting data, we describe clade-specific distributions of secondary vein arrangements; we also show that these distributions scale with wing size. Then, we synthesize our work with published developmental data to create a minimal geometric model of secondary vein development based on evenly spaced inhibitory signaling centers. This model is able to recapitulate the vast majority of secondary vein arrangements that are observed in our dataset. Furthermore, our model allows us to make specific, testable hypotheses about wing development for all insects with stochastically patterned secondary veins, a group that collectively spans ∼400 My of evolution (23).

Results

We initially focus on dragonflies and damselflies (order: Odonata), a group of aerial predators whose wings have especially complex venation patterns. An overlapping projection of the left and right wings of an example dragonfly, *Erythremis simplicicolis*, allows us to identify the primary and secondary veins, as defined above (Fig. 1*A*; see *SI Appendix* for details). This categorization of veins is similar to those used in previous studies (24, 25).

Significance

The wing veins of the fruit fly *Drosophila melanogaster* have long been studied as an example of how signaling gradients in a growing tissue can generate precise, reproducible patterns. However, fruit fly wings represent only a small slice of wing diversity. In many insect species, wings are like human fingerprints: even the left and right wings of the same individual have unique vein patterns. We analyze wing geometry in many species and then present a minimal developmental model for how vein patterns can be formed. This model will serve as a hypothesis for future empirical work.

Author contributions: J.H., S.D., and C.H.R. designed research; J.H., S.D., K.L., and M.K.S. performed research; J.H. contributed new reagents/analytic tools; J.H., S.D., and K.L. analyzed data; and J.H., S.D., and C.H.R. wrote the paper.

The authors declare no conflict of interest.

This article is a PNAS Direct Submission.

Data deposition: The data and code used in this study are available at https://github.com/hoffmannjordan/insect-wing-venation-patterns.

[1]J.H. and S.D. contributed equally to this work.

[2]To whom correspondence may be addressed. Email: seth.donoughe@gmail.com or chr@seas.harvard.edu.

This article contains supporting information online at www.pnas.org/lookup/suppl/doi:10.1073/pnas.1721248115/-/DCSupplemental.

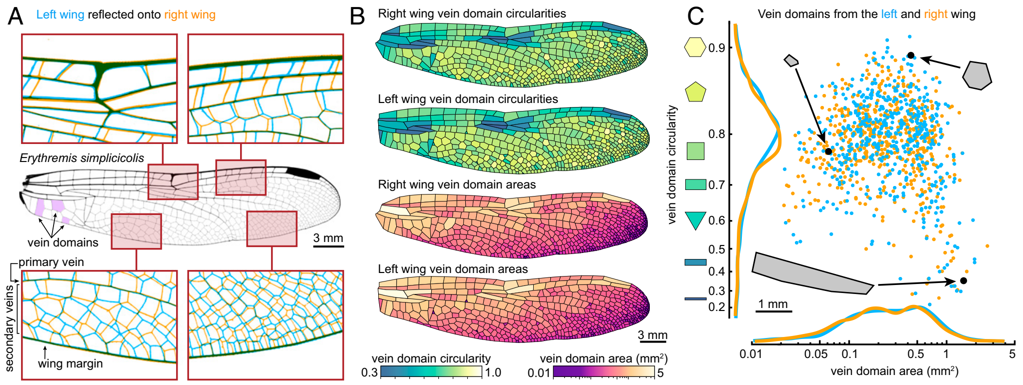


Fig. 1. Secondary veins form a unique pattern on every wing. (*A*) Overlay of the left (blue) and right (orange) forewing of the same individual of *Erythremis simplicicolis*. (*B*) Left and right wings of the same individual, with domains colored by circularity and area. Left wings have been reflected for display. (*C*) Area and circularity of each vein domain. Each point represents a single domain (blue points, left wing; orange points, right wing).

In many insect species, wing veins form tens to thousands of closed polygonal shapes called "vein domains" (Fig. 1*A*, examples highlighted in purple). Characterizing the areas and shapes of vein domains is a tractable way to study the geometric properties of veins. We present a custom method to segment wing images based on level sets (*SI Appendix*). This approach is well-suited to studying the morphologies of diverse wing vein patterns, robust to variations in image resolution, and it requires minimal parameter adjustment. This allows us to precisely calculate attributes—such as area and circularity—of every vein domain in a wing. Circularity is defined as the ratio of a domain's area to the area of a circle whose perimeter is equal to that of the domain. The left and right wings of *E. simplicicolis* are shown, with each domain colored according to its area and circularity (Fig. 1*B*). When vein domain area is plotted against circularity for left and right wings (Fig. 1*C*), it is clear that each wing's set of domain shapes is a unique fingerprint, yet the marginal distributions of each trait are strikingly similar.

Our dataset includes published wing tracings from 215 odonate species, including representatives from 17 families, whose wings range from 20 to 725 mm^2 in area (Fig. 2*A*). We took high-resolution micrographs of example species to verify that the wing tracings accurately capture the geometric arrangement of veins (*SI Appendix*, Figs. S1 and S4–S6). We segmented the vein domains on all wings in the dataset, finding that the number of vein domains scales allometrically with wing size: species with larger wings have larger and more numerous vein domains (Fig. 2*B*). The full segmented dataset contains 150,000+ vein domains, from <0.01 to >5 mm^2 in area (*SI Appendix*, Fig. S13*A*).

These data enable us to explore how vein domain area and circularity vary along the proximal to distal (P–D) axis (i.e., from a wing's base to its tip) for each forewing and hindwing. We divide wings into 21 equally spaced rectangular bins along the P–D axis. For a given bin, we determine the area and circularity of each vein domain within it, and then calculate an area-weighted mean for the entire bin. For each wing, we plot the P–D morphology trace of its vein domains in a 2D space determined by circularity and area (Fig. 2*C*). By plotting P–D traces of many species, we show that damselflies and dragonflies exhibit distinct, clade-specific patterns (Fig. 2 *D* and *E*), and within each group, P–D morphology traces are related to wing size (*SI Appendix*, Fig. S13 *B* and *C*).

In nature, there are many developing structures that can be approximated as a flat tissue that is stochastically partitioned. Examples include leaf vascularization (26, 27), reptile scale formation (28), and a variety of pigmentation patterns (29). Theoretical and empirical work has shown that such patterns can form in different ways—as a bifurcating process in which branches grow toward secreted signal sources (26), a mechanism wherein

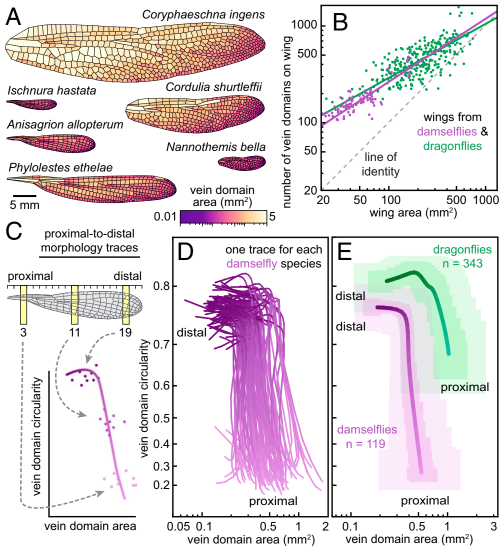
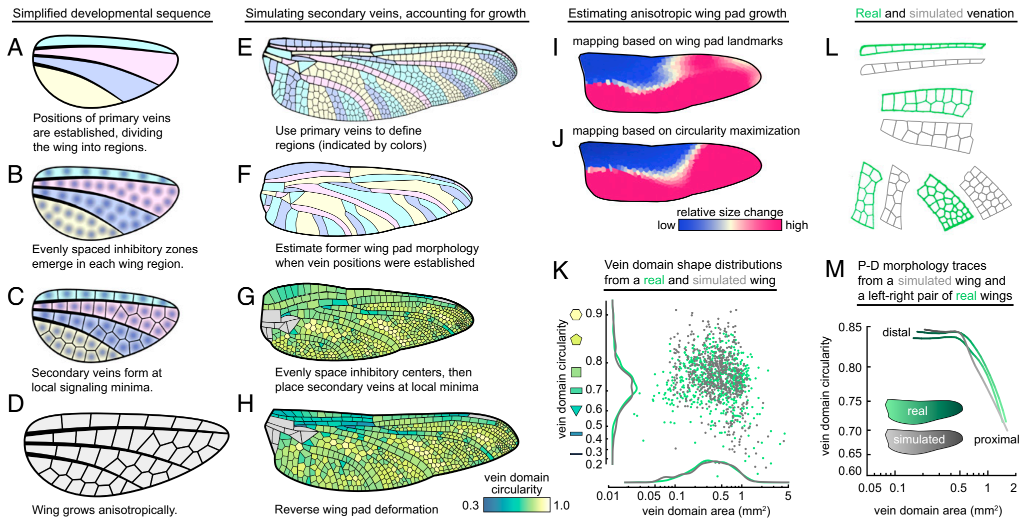


Fig. 2. Comparing vein domains across species. (*A*) Forewings of the smallest, median, and largest dragonfly (*Right*) and damselfly (*Left*) species included in our dataset. Vein domains are colored by area on the same scale. (*B*) Area of the entire wing (in square millimeters) versus the total number of wing domains on a log scale for each dragonfly (green; $n = 343$) and damselfly (purple; $n = 119$). Best fits are shown as solid lines, with an identity line in dashed gray. For both dragonflies and damselflies, the exponent on the fit is less than 1. (*C*) Schematic of the process used to create P–D morphology traces. The wing is divided into a series of rectangular bins. For a given bin, mean area and circularity are computed; the value for each domain is weighted by its overlap with the rectangular bin. P–D traces are smoothed with a Gaussian of width 3. (*D*) P–D traces of all damselflies in the dataset. (*E*) Distribution of P–D morphology traces for damselflies (purple) and dragonflies (green).



Simplified developmental sequence
A
Positions of primary veins are established, dividing the wing into regions.
B
Evenly spaced inhibitory zones emerge in each wing region.
C
Secondary veins form at local signaling minima.
D
Wing grows anisotropically.
Simulating secondary veins, accounting for growth
E
Use primary veins to define regions (indicated by colors)
F
Estimate former wing pad morphology when vein positions were established
G
Evenly space inhibitory centers, then place secondary veins at local minima
H
Reverse wing pad deformation
vein domain circularity
0.3
1.0
Estimating anisotropic wing pad growth
I
mapping based on wing pad landmarks
J
mapping based on circularity maximization
relative size change
low
high
K
Vein domain shape distributions from a real and simulated wing
vein domain circularity
vein domain area (mm²)
Real and simulated venation
L
M
P-D morphology traces from a simulated wing and a left-right pair of real wings
distal
proximal
real
simulated
vein domain circularity
vein domain area (mm²)

Fig. 3. A model for simulating secondary vein patterning. (*A–D*) A simplified schematic of secondary vein development. (*E–H*) Process for simulating secondary veins while taking tissue growth into account. Gray domains in *G* and *H* are bounded on all sides by primary wing veins; these are not simulated in the model. (*I* and *J*) Mapping an example dragonfly (*Anax junius*) wing pad to the adult wing shows that the tissue grows heterogenously; two methods of calculating local growth produce similar spatial distributions of relative size changes: (*I*) map generated using landmarks on a nymphal wing pad; (*J*) map based on a wing pad morphology that was estimated by maximizing vein domain circularities. (*K*) The area and circularity distributions of the simulated wing (in gray) compared with the real P–D morphology trace (in green). (*L*) Simulated vein patterns for a hindwing of *D. spinosus* next to real vein patterns. (*M*) The P–D traces of the true left and right wings (green) compared with the P–D trace of a simulated wing (gray).

stresses in the growing tissue trigger localized differentiation (27, 28), or diffusion-based systems with feedback loops that generate evenly spaced domains from a noisy precursor signal (29). Each class of processes produces characteristic geometric patterns.

Secondary veins in odonate wings have several features that are consistent with a simultaneous, diffusion-based patterning mechanism: (*i*) secondary veins that terminate in space are extraordinarily rare (*SI Appendix*, Table S1), (*ii*) 180° joints rarely occur among secondary veins (*SI Appendix*, Fig. S15), and (*iii*) domains made of secondary veins tend to be approximately the same size as their immediate neighbors (*SI Appendix*, Fig. S16). Last, rectangles tend to form between closely spaced parallel primary veins while pentagons and hexagons predominate in regions where primary veins are distantly spaced. Collectively, these features are conspicuously similar to those of a Voronoi tessellation of evenly spaced seeds in a 2D region (*SI Appendix*, Fig. S29) (30). A Voronoi tessellation is produced by taking a set of seed locations on a plane, and then partitioning every seed into its own region in space. The shape of each region is given by the set of all points that are closer to its seed than to any other. Voronoi tessellations are mathematically tractable, and they appear in nature in different contexts (31–33); we use them as the basis for a minimal model of secondary vein patterning.

We hypothesized that, to a first approximation, the development of secondary veins proceeds as follows. First, the positions of primary veins are established on the wing pad (Fig. 3*A*) (34–42). Second, an as-yet-undescribed stochastic patterning mechanism generates evenly spaced inhibitory centers within the regions bound by the primary veins (Fig. 3*B*) [see, for example, mammalian hair follicle patterning (43) and avian feather bud patterning (44)]. Third, secondary veins arise at the inhibitory signal's local minima, which can be well approximated by Voronoi cells (Fig. 3*C*). Finally, during subsequent nymphal development and wing eclosion, the wing undergoes anisotropic growth (41, 45) (Fig. 3*D*). This simplified sequence of steps serves as a tool to generate testable hypotheses about the mechanisms of wing vein patterning.

We use the developmental sequence described above to simulate the formation of secondary veins in an example wing, the hindwing of the dragonfly *Dromogomphus spinosus*. To start with the simplest possible model, we ignore wing growth altogether by simulating secondary veins as though they emerge on a fully formed adult wing (*SI Appendix*, Fig. S21 *A–C*). First, we manually divide the wing into regions that are bounded by primary veins and the wing margin. Next, we use the following procedure to generate a set of evenly spaced inhibitory centers for each wing region: we randomly place "inhibitory centers" equal in number to the number of vein domains in the matching region of the real wing, and then use Voronoi iteration as a method to evenly space the inhibitory centers (46). Finally, we position secondary veins at local minima of the inhibitory signal. When we compare the simulated wing to a left–right pair of real wings, we find that this model recovers some natural vein features, yet it systematically overestimates vein domain circularity (*SI Appendix*, Fig. S21*D*).

Next, we modified the model to include wing pad growth and shape change. As above, we use primary veins to define wing regions (Fig. 3*E*), and then estimate the former morphology of the wing pad (Fig. 3*F*; described below). We evenly space inhibitory centers on the wing pad and place secondary veins at local minima (Fig. 3*G*). Last, we simulate anisotropic growth by reforming the wing pad into the shape of the mature wing (Fig. 3*H*).

To estimate the shape of a wing pad, we make two assumptions about wing development based on earlier literature (34–41): the wing pad develops as a roughly convex shape, and the pattern of secondary veins that forms on the wing pad is composed of well-spaced polygons, which tend to maximize the circularity of vein domains. We use the mature wing to calculate a corresponding wing pad shape via a coordinate transformation that maximizes the circularity of all vein domains, while constraining the wing pad to be approximately convex (an example transformation for *D. spinosus* is shown in Fig. 3 *E* and *F*; see *SI Appendix* for further details).

To assess the effectiveness of our wing pad shape estimation, we use a published micrograph of the hindwing pad from the dragonfly *Anax junius*, which was dissected from the nymph before secondary veins had formed (47). We use primary veins as landmarks to map the adult wing onto the wing pad, and then color each vein domain according to its relative size change (Fig. 3*I* and *SI Appendix*, Fig. S23). This shows that the nymphal wing pad-based map produces a coordinate transformation that is strikingly similar to the map we independently calculate using the circularity maximization procedure described in the previous paragraph (Fig. 3*J*).

When we employ this computational model to simulate secondary veins for the hindwing of *D. spinosus*, it results in the secondary-venation pattern shown in Fig. 3*H*. Real and simulated veins from different parts of the wing are shown side-by-side in Fig. 3*L*. The simulated wing has a vein domain area distribution, circularity distribution, and P–D morphology trace that closely match those of the true wing (Fig. 3 *K* and *M*; left and right wings of the same individual shown for comparison). We simulated secondary veins for odonates from several different families, and in each case the same simple model recapitulates the observed geometric rearrangements of veins (e.g., *SI Appendix*, Fig. S30); conversely, the model does not generate any arrangements that are not seen in true wings.

Next, we apply the secondary vein simulation model to representatives from orders Orthoptera and Neuroptera. With respect to Odonata, these orders are drawn from distantly related parts of the insect phylogeny (Fig. 4*A*)—the last common ancestor of the three orders may have been the shared ancestor of all extant winged insects (23). As above, we treat primary veins as boundaries and simulate secondary veins within them. Likewise, the model recapitulates most of the secondary vein patterns in each example species (Fig. 4 *B* and *C*), producing distributions of vein domain size and circularity that are broadly similar to those of the true wings. However, there are a few subregions in the wing of each species where the model does not capture vein domain geometry as accurately. For instance, in lacewing and grasshopper, vein domains along the trailing wing margin in the real wings have a systematically lower circularity than the analogous vein domains in simulations. A possible explanation for this mismatch is considered in *Discussion*.

We assess model sensitivity to variation in the density of inhibitory centers by resimulating secondary venation at a range of densities. The resulting venation patterns have substantial changes to their vein domain area and circularity distributions (*SI Appendix*, Fig. S27*D*), demonstrating that it is essential for the model to include an accurate estimate of the number of inhibitory centers in each region. With the model described so far, the density of inhibitory centers in each region is drawn directly from real wings, effectively "baking it" into the model. Therefore, we ask whether it is possible to accurately model the secondary vein pattern using primary vein morphology as the only input. The thickness of primary veins varies substantially across a wing (10, 48). We hypothesized that if primary veins are the source of a morphogen that affects inhibitory centers in nearby tissue, primary vein thickness on the wing pad could indicate the strength or concentration of that signal. This, in turn, would determine the length scale of the pattern generator. We show in an example wing that the relative thicknesses of primary veins are correlated with the thicknesses of the corresponding veins on the adult wing (*SI Appendix*, Fig. S26). Therefore, we use primary vein thickness on the adult wing as a proxy for relative thickness on the wing pad. We inquire how the thickness of primary veins is related to the area of

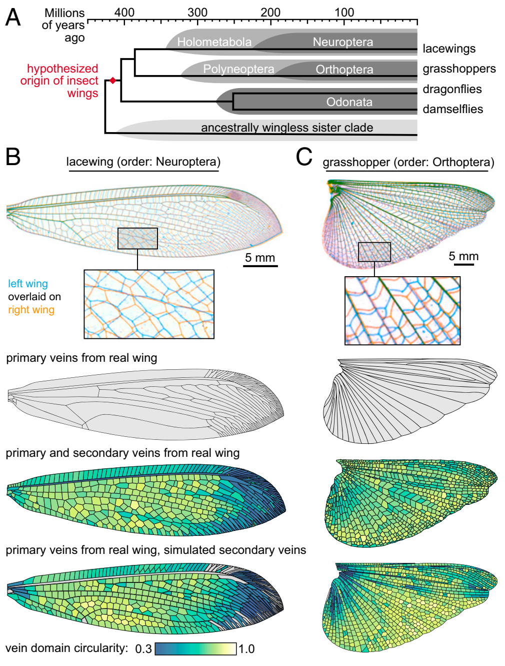

A
Millions of years ago
400
300
200
100
Holometabola
Neuroptera
lacewings
Polyneoptera
Orthoptera
grasshoppers
hypothesized origin of insect wings
dragonflies
Odonata
damselflies
ancestrally wingless sister clade
B
lacewing (order: Neuroptera)
C
grasshopper (order: Orthoptera)
5 mm
5 mm
left wing
overlaid on
right wing
primary veins from real wing
primary and secondary veins from real wing
primary veins from real wing, simulated secondary veins
vein domain circularity: 0.3
1.0

Fig. 4. Modeling the secondary veins of distantly related species. (*A*) The evolutionary relationships among the insects considered in this study. (*B* and *C*) The model applied to a (*B*) lacewing (order: Neuroptera) and (*C*) grasshopper (order: Orthoptera). Gray vein domains in *B* are bounded on all sides by primary wing veins; these are not simulated in the model.

nearby vein domains. We do so by measuring the thickness of each primary vein segment in an adult wing. Then, we calculate the shortest distance from each vein domain to every primary vein and compute a proximity-weighted primary vein thickness (Fig. 5*A*; see *SI Appendix* for details). Across multiple species, we find a positive relationship between primary vein thickness and vein domain area (Fig. 5*C* and *SI Appendix*, Fig. S28). We use this relationship to simulate secondary veins without predetermining the number of inhibitory centers in each wing region. Using high-resolution micrographs of wings from the dragonflies *Libellula cyanea* and *Sympetrum vicinum*, we measure primary vein thickness to produce a distribution of thicknesses and domain areas. We use this distribution to simulate a wing from *E. simplicicolis*—a species that was not used to generate the sample distribution. We stochastically simulate an *E. simplicicolis* wing repeatedly, computing the P–D morphology trace each time. The variation between simulated wings is comparable to the disparity we observe between left and right real wings of the same individual (Fig. 5*B*). Therefore, knowing only the thickness and arrangement of primary veins on a wing, we are able to simulate secondary veins whose pattern is comparable to that of a real wing.

Discussion

The molecular basis of primary wing vein patterning has been studied extensively in *Drosophila melanogaster* (49), but because fruit flies do not have secondary veins, the developmental basis of these "fingerprint" veins is still unknown. Some previous researchers have qualitatively described secondary veins and

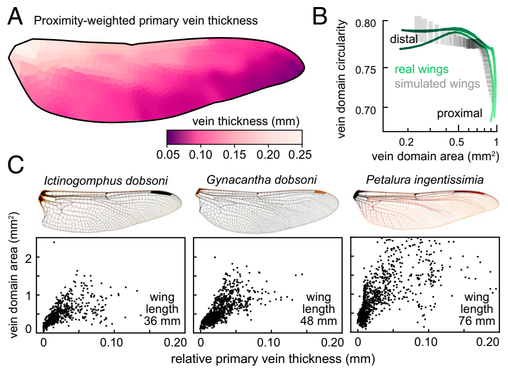


Fig. 5. Primary vein thickness correlates with the length scale of secondary vein spacing. (*A*) An example wing from the dragonfly *Petalura gigantean*, with vein domains colored by proximity-weighted primary vein thickness (see text for details). (*B*) P–D morphology traces of many simulated wings, generated by drawing from the distribution of primary vein thickness vs. vein domain density. Gray boxes show 25th to 75th percentiles for bins along the P–D axis of simulated wings. Green traces represent the real left and right wings of a single individual. (*C*) In dragonflies, there is a relationship between the thickness of nearby long veins and the size of domains. The *x* axis shows the thickness of the nearby long veins; the *y* axis shows the size of wing domains. Wing images courtesy of Wikimedia Commons/John Tann.

speculated about how their curious geometric patterns are formed. K. G. Andrew Hamilton (2) described the patterns from diverse species and attempted to place them in broad morphological categories. D'Arcy Thompson (50) likened them to the shapes formed by interfaces equilibrating under tension, as one can observe in clusters of soap bubbles. However, only with the advent of high-throughput digital image quantification tools has it become possible for us to chart the geometric attributes of secondary veins in detail and test a model to explain their patterning.

The model we present is not molecularly specific, but it allows us to make falsifiable hypotheses about the development of secondary veins: first, the position of a primary vein is established before the positions of neighboring secondary veins. Second, there exists an inhibitory signal that restricts secondary vein formation to certain locations in the developing wing. Third, there is a stochastic input to the process that evenly spaces inhibitory centers. These are probably generated by a reaction–diffusion process (29), a simple feedback system that is able to generate diverse patterns, including pigmentation patterns that are observed on insect wings (51). Similarly, a reaction–diffusion-based hypothesis for wing vein patterning has been proposed for the wings of *Orosanga japonicus* (52). Finally, we hypothesize that once secondary vein locations have been established, vein morphogenesis itself is deterministic. Our model is consistent with empirical observations, but we have not proven that the model captures a specific molecular mechanism. Further testing of the model would entail a mechanistic investigation into the abovementioned hypotheses that form its foundation. This will require detailed developmental description, as well as functional genetic and developmental experiments on developing wings of species that have reticulate secondary veins.

Our model is largely effective in recapitulating secondary vein arrangements in three orders of insect wings. However, there are two features of real wings for which the model is systematically inaccurate: (*i*) wing regions in which there is a pronounced gradient in the size of domains, and (*ii*) wing regions in which secondary vein segments are arranged in an atypically collinear manner (*SI Appendix*, Figs. S31 and S32). The existence of the latter case suggests that a strict dichotomy of primary veins and secondary veins cannot fully describe wing vein identities. We hypothesize that after primary vein positions are established, secondary veins can take on primary vein–like function and morphology in wing regions that are sufficiently distant from an inhibitory signal that emanates from primary veins [for further discussion of vein identity, see other authors (53, 54) who have reviewed the evolution and development of diverse wing veins in closer detail].

To discern the functional ramifications of a given arrangement of wing veins, it is necessary to consider additional aspects of wing morphology beyond the topology and thickness of veins (3). In odonates, for instance, wing veins are tubular struts composed of several different layers of cuticle (48) that are joined together in a variety of mechanically complex ways (9, 55–58). It will be instructive to integrate large-scale vein arrangement data with functional manipulations of wings. The biomechanical effects of venation patterns can be assessed from another perspective as well: recent work on miniature winged robots has used natural veined insect wings as models for biomimetic wings (59, 60). The present study, by illuminating principles of geometric wing design, could guide efforts to generate life-like, synthetic vein patterns, and in turn be used to examine how vein patterns affect the mechanical properties of a wing.

A variety of open questions in morphological evolution could be addressed using the approach we take in the present study. Phenotypic description is typically the most expensive aspect of a project and usually requires a great deal of expertise (61, 62). The centuries-long documented history of life science scholarship is rich with observations that were recorded as images, but they remain mostly untapped for large-scale investigations, partly because phenotypes have not been recorded in a precise, machine-parsable manner (63). We suggest that the method used here could be applied to many biological questions that are answerable with existing image-based data. To demonstrate the possibilities of this approach, we apply our quantification tools to diverse patterned tissues (*SI Appendix*, Fig. S3), finding that it enables us to effectively characterize each of them. This offers a fruitful avenue for future research.

Methods

Microscopy and Collecting Published Images. Wings were dissected from specimens and imaged with a flat-mount scanner, macroscopic photography, or dissection microscope. Each technique produces a 2D image of the 3D wing. Since some wings are corrugated (10, 64, 65), capturing them in 2D introduces a slight distortion to vein domain shapes. For typical vein domains, this distortion results in an underestimate of vein domain area and circularity by 1–5% (*SI Appendix*, Figs. S8–S10). When measuring the thicknesses of primary veins, we found that lighting conditions could affect the measured lengths by altering the apparent thickness of a vein. To compare the data from multiple species, we plotted relative vein thickness, calculated by subtracting the smallest vein thickness from every measured thickness on that wing. Images in electronic publications were extracted digitally; images in printed publications were digitally scanned (40, 45, 66–68).

Segmenting Wing Images and Calculating Vein Domain Attributes. Segmentation of wing images was accomplished with a code based on the fast marching method with a variable background velocity field (69, 70). The segmented images were used to make a polygonal representation of each vein domain, which provided two advantages over the segmented image: (*i*) domain perimeter was a more rigorously defined quantity, and (*ii*) geometric computations were less sensitive to segmentation-related noise. Segmented wing images are available for all wings examined in this paper. See https://github.com/hoffmannjordan/insect-wing-venation-patterns.

Full methods are available in *SI Appendix*. This includes mathematical details on calculating wing attributes, simulating secondary veins, measuring primary vein thickness, and validating the use of wing tracings from published sources.

ACKNOWLEDGMENTS. We are grateful to L. Mahadevan for fruitful discussions on the project and to Christina Baik, Ben Blonder, John Boyle, James Crall, Richard Childers, Albert Kao, and Shruti Mishra for their helpful feedback. This research was supported by a US Department of Energy (DOE) Computational Science Graduate Fellowship (to J.H.), National Science Foundation Graduate Training Fellowships (to S.D. and M.K.S.), and the Applied Mathematics Program of the US DOE Office of Advanced Scientific Computing Research under Contract DE-AC02-05CH11231 (to C.H.R.). We also acknowledge support from the NSF-Simons Center for Mathematical and Statistical Analysis of Biology at Harvard University, supported by NSF Grant DMS-1764269, and the Harvard Faculty of Arts and Sciences Quantitative Biology Initiative.

1. Hamilton KGA (1972) The insect wing, Part III. Venation of the orders. *J Kans Entomol Soc* 45:145–162.
2. Hamilton KGA (1972) The insect wing, Part IV. Venational trends and the phylogeny of the winged orders. *J Kans Entomol Soc* 45:295–308.
3. Combes SA, Daniel TL (2003) Flexural stiffness in insect wings. I. Scaling and the influence of wing venation. *J Exp Biol* 206:2979–2987.
4. Wootton RJ (1992) Functional morphology of insect wings. *Annu Rev Entomol* 37: 113–140.
5. Dirks J-H, Taylor D (2012) Veins improve fracture toughness of insect wings. *PLoS One* 7:e43411.
6. Li XJ, et al. (2014) Antifatigue properties of dragonfly *Pantala flavescens* wings. *Microsc Res Tech* 77:356–362.
7. Rajabi H, Darvizeh A, Shafiei A, Taylor D, Dirks JH (2015) Numerical investigation of insect wing fracture behaviour. *J Biomech* 48:89–94.
8. Rees CJC (1975) Form and function in corrugated insect wings. *Nature* 256:200–203.
9. Newman D, Wootton RJ (1986) An approach to the mechanics of pleating in dragonfly wings. *J Exp Biol* 125:361–372.
10. Jongerius SR, Lentink D (2010) Structural analysis of a dragonfly wing. *Exp Mech* 50: 1323–1334.
11. Arnold JW (1964) Blood circulation in insect wings. *Mem Entomol Soc Can* 96:5–60.
12. Chintapalli RTV, Hillyer JF (2016) Hemolymph circulation in insect flight appendages: Physiology of the wing heart and circulatory flow in the wings of the mosquito *Anopheles gambiae*. *J Exp Biol* 219:3945–3951.
13. Hartenstein V, Posakony JW (1989) Development of adult sensilla on the wing and notum of *Drosophila melanogaster*. *Development* 107:389–405.
14. Dickerson BH, Aldworth ZN, Daniel TL (2014) Control of moth flight posture is mediated by wing mechanosensory feedback. *J Exp Biol* 217:2301–2308.
15. Ennos AR (1988) The importance of torsion in the design of insect wings. *J Exp Biol* 140:137–160.
16. Mountcastle AM, Combes SA (2013) Wing flexibility enhances load-lifting capacity in bumblebees. *Proc Biol Sci* 280:20130531.
17. Kukalová-Peck J, Peters JG, Soldán T (2009) Homologisation of the anterior articular plate in the wing base of Ephemeroptera and Odonatoptera. *Aquat Insects* 31: 459–470.
18. Bybee SM, Ogden TH, Branham MA, Whiting MF (2008) Molecules, morphology and fossils: A comprehensive approach to odonate phylogeny and the evolution of the odonate wing. *Cladistics* 24:477–514.
19. Johansson F, Söderquist M, Bokma F (2009) Insect wing shape evolution: Independent effects of migratory and mate guarding flight on dragonfly wings. *Biol J Linn Soc Lond* 97:362–372.
20. Debat V, Bégin M, Legout H, David JR (2003) Allometric and nonallometric components of *Drosophila* wing shape respond differently to developmental temperature. *Evolution* 57:2773–2784.
21. Klingenberg CP, Badyaev AV, Sowry SM, Beckwith NJ (2001) Inferring developmental modularity from morphological integration: Analysis of individual variation and asymmetry in bumblebee wings. *Am Nat* 157:11–23.
22. Klingenberg CP, McIntyre GS, Zaklan SD (1998) Left-right asymmetry of fly wings and the evolution of body axes. *Proc Biol Sci* 265:1255–1259.
23. Misof B, et al. (2014) Phylogenomics resolves the timing and pattern of insect evolution. *Science* 346:763–767.
24. Rajabi H, et al. (2015) A comparative study of the effects of constructional elements on the mechanical behaviour of dragonfly wings. *Appl Phys A Mater Sci Process* 122: 1–13.
25. Suárez-Tovar CM, Sarmiento CE (2016) Beyond the wing planform: Morphological differentiation between migratory and nonmigratory dragonfly species. *J Evol Biol* 29:690–703.
26. Runions A, et al. (2005) Modeling and visualization of leaf venation patterns. *ACM Trans Graph* 24:702–711.
27. Laguna MF, Bohn S, Jagla EA (2008) The role of elastic stresses on leaf venation morphogenesis. *PLOS Comput Biol* 4:e1000055.
28. Milinkovitch MC, et al. (2013) Crocodile head scales are not developmental units but emerge from physical cracking. *Science* 339:78–81.
29. Kondo S, Miura T (2010) Reaction-diffusion model as a framework for understanding biological pattern formation. *Science* 329:1616–1620.
30. Voronoi G (1908) Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *J Reine Angew Math (Crelle's J)* 1908:97–102.
31. Barlow GW (1974) Hexagonal territories. *Anim Behav* 22:876–878.
32. Bock M, Tyagi AK, Kreft J-U, Alt W (2010) Generalized voronoi tessellation as a model of two-dimensional cell tissue dynamics. *Bull Math Biol* 72:1696–1731.
33. Sánchez-Gutiérrez D, et al. (2016) Fundamental physical cellular constraints drive self-organization of tissues. *EMBO J* 35:77–88.
34. Comstock JH, Needham JG (1898) The wings of insects. Chapter III. The specialization of wings by reduction. *Am Nat* 32:231–257.
35. Comstock JH, Needham JG (1898) The wings of insects. Chapter III (continued). The venation of the wings of Hymenoptera. *Am Nat* 32:413–424.
36. Comstock JH, Needham JG (1899) The wings of insects. Chapter IV (concluded). The specialization of wings by addition. *Am Nat* 33:573–582.
37. Comstock JH, Needham JG (1899) The wings of insects. Chapter V. The development of wings. *Am Nat* 33:845–860.
38. Tillyard RJ (1914) On some problems concerning the development of the wing venation of the Odonata. *Proc Linn Soc N S W* 39:163–216.
39. Tillyard RJ (1915) On the development of the wing-venation in zygopterous dragonflies, with special reference to the Calopterygidae. *Proc Linn Soc N S W* 40: 212–230.
40. Tillyard RJ (1921) On an anisozygopterous larva from the Himalayas (order Odonata). *Rec Indian Mus* 22:93–107.
41. Holdsworth RP (1942) The wing development of *Pteronarcys proteus* Newman (Pteronarcidae: Plecoptera). *J Morphol* 70:431–461.
42. Aoki T (1999) Larval development, emergence and seasonal regulation in *Asiagomphus pryeri* (Selys) (Odonata: Gomphidae). *Hydrobiologia* 394:179–192.
43. Sick S, Reinker S, Timmer J, Schlake T (2006) WNT and DKK determine hair follicle spacing through a reaction-diffusion mechanism. *Science* 314:1447–1450.
44. Harris MP, Williamson S, Fallon JF, Meinhardt H, Prum RO (2005) Molecular evidence for an activator-inhibitor mechanism in development of embryonic feather branching. *Proc Natl Acad Sci USA* 102:11734–11739.
45. Needham JG (1951) Prodrome for a manual of the dragonflies of North America, with extended comments on wing venation systems. *Trans Am Entomol Soc* 77:21–62.
46. Lloyd S (1982) Least squares quantization in PCM. *IEEE Trans Inf Theory* 28:129–137.
47. Needham JG, Westfall MJ, May ML (2014) *Dragonflies of North America: The Odonata (Anisoptera) Fauna of Canada, the Continental United States, Northern Mexico and the Greater Antilles* (Scientific Publishers, Gainesville, FL).
48. Appel E, Heepe L, Lin C-P, Gorb SN (2015) Ultrastructure of dragonfly wing veins: Composite structure of fibrous material supplemented by resilin. *J Anat* 227:561–582.
49. Blair SS (2007) Wing vein patterning in *Drosophila* and the analysis of intercellular signaling. *Annu Rev Cell Dev Biol* 23:293–319.
50. Thompson DW (1942) *On Growth and Form* (Cambridge Univ Press, Cambridge, UK).
51. Nijhout HF (2010) Molecular and physiological basis of colour pattern formation. *Advances in Insect Physiology* (Academic Press, London), Vol 38, Chap 6, pp 219–265.
52. Yoshimoto E, Kondo S (2012) Wing vein patterns of the Hemiptera insect *Orosanga japonicus* differ among individuals. *Interface Focus* 2:451–456.
53. Kukalová-Peck J (1978) Origin and evolution of insect wings and their relation to metamorphosis, as documented by the fossil record. *J Morphol* 156:53–125.
54. De Celis JF, Diaz-Benjumea FJ (2003) Developmental basis for vein pattern variations in insect wings. *Int J Dev Biol* 47:653–663.
55. Gorb SN (1999) Serial elastic elements in the damselfly wing: Mobile vein joints contain resilin. *Naturwissenschaften* 86:552–555.
56. Donoughe S, Crall JD, Merz RA, Combes SA (2011) Resilin in dragonfly and damselfly wings and its implications for wing flexibility. *J Morphol* 272:1409–1421.
57. Appel E, Gorb SN (2011) Resilin-bearing wing vein joints in the dragonfly *Epiophlebia superstes*. *Bioinspir Biomim* 6:046006.
58. Appel E, Gorb SN (2014) *Comparative Functional Morphology of Vein Joints in Odonata* (Schweizerbart Science Publishers, Stuttgart).
59. Shang JK, Combes SA, Finio BM, Wood RJ (2009) Artificial insect wings of diverse morphology for flapping-wing micro air vehicles. *Bioinspir Biomim* 4:036002.
60. Tanaka H, Wood RJ (2010) Fabrication of corrugated artificial insect wings using laser micromachined molds. *J Micromech Microeng* 20:075008.
61. Freimer N, Sabatti C (2003) The human phenome project. *Nat Genet* 34:15–21.
62. Houle D (2010) Colloquium papers: Numbering the hairs on our heads: The shared challenge and promise of phenomics. *Proc Natl Acad Sci USA* 107:1793–1799.
63. Deans AR, et al. (2015) Finding our way through phenotypes. *PLoS Biol* 13:e1002033.
64. Kesel AB (2000) Aerodynamic characteristics of dragonfly wing sections compared with technical aerofoils. *J Exp Biol* 203:3125–3135.
65. Wootton RJ, Evans KE, Herbert R, Smith CW (2000) The hind wing of the desert locust (*Schistocerca gregaria* Forskål). I. Functional morphology and mode of operation. *J Exp Biol* 203:2921–2931.
66. Westfall MJ, May ML (1996) *Damselflies of North America* (Scientific Publishers, Gainesville, FL), 649 p.
67. Garrison RW, von Ellenrieder N, Louton JA (2006) *Dragonfly Genera of the New World: An Illustrated and Annotated Key to the Anisoptera* (JHU Press, Baltimore).
68. Garrison RW, von Ellenrieder N, Louton JA (2010) *Damselfly Genera of the New World: An Illustrated and Annotated Key to the Zygoptera* (JHU Press, Baltimore).
69. Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J Comput Phys* 79:12–49.
70. Chopp DL (2001) Some improvements of the fast marching method. *SIAM J Sci Comput* 23:230–244.

Size, shape and structure of insect wings

Mary K. Salcedo,[1, a)] Jordan Hoffmann,[2, a)] Seth Donoughe,[3] and L. Mahadevan[1, 2, b)]

[1)] *Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, MA*

[2)] *School of Engineering and Applied Sciences, Harvard University, Cambridge, MA*

[3)] *Department of Molecular Genetics and Cell Biology, University of Chicago, Chicago, IL*

The size, shape and structure of insect wings are intimately linked to their ability to fly. However, there are few systematic studies of the variability of the natural patterns in wing morphology across insects. We assemble a comprehensive dataset of insect wings and analyze their morphology using topological and geometric notions in terms of i) wing size and contour shape, ii) vein geometry and topology, and iii) shape and distribution of wing membrane domains. These morphospaces are a first-step in defining the diversity of wing patterns across insect orders and set the stage for investigating their functional consequences.

[a)]Equal contribution

[b)]Electronic mail: Lmahadev@g.harvard.edu

INTRODUCTION

Of all the multicellular species on our planet, insects are the most speciose, with more than one million species[1]. Most are capable of flight owing to that remarkable evolutionary innovation, wings which themselves show a range of hierarchies of complexity, varying greatly in size and venation, stiffness and flexibility, pigmentation[6] and flight behaviors[2–5], while being subject to strong selective pressures by ecological niche specialization[7].

From a physical perspective, insect wings are slender quasi two-dimensional membranes criss-crossed by a network of tubular veins. The patterns formed by veins can partition some wings into just a few domains and others into many thousands. The venation network allows for fluid and nutrient transport across the structure while providing a mechanical skeleton that stiffens the wing[3,8–10]. While phylogenetic analysis and geometric morphometrics have been deployed to understand the variation of and selection pressures on wing morphology[3,6,11–14], their scope has been limited to a few species or orders at most. Here we complement these studies by using a set of simple quantitative measures to compare morphological variation in wing size, shape, and structure of insect wings across species, families and orders.

We start by assembling 555 wings drawn from 24 taxonomic orders, sampling representatives from nearly every extant insect order. We then deploy a range of geometrical and topological methods on these structures, and establish three complementary approaches to quantify wing size, shape, venation complexity and domain size and shape. Our dataset and analysis will, we hope, serve as a first step in functional and phylogenetic approaches to test hypotheses about wing evolution and physiology.

MATERIALS AND METHODS

Image collection and segmentation

Our dataset of wing images with representatives from most recognized insect orders is depicted in Fig. 1. We use a combination of original micrographs (hand-caught and donated specimens), a collection of scans from entomological literature (1840s - 1930s) sourced at the Ernst Mayr Library, Harvard University (Cambridge, MA) and online through the Biodiversity Heritage Library[15](see SI for details).

Entomological texts were chosen based on the quality and diversity of insects described and how well wing shape, venation, and morphological data were represented. This is not an unbiased sample of insects but we attempted to maximize diversity at the order level. By incorporating newly available data[16], we obtained taxonomic coverage enriched for Odonata, a group with particularly complex wings. Insects are referred to by their common order-level names.

To quantitatively characterize an insect wing, we first segment a wing image using a Level-Set approach[16] (see SI for details and dataset availability). Using this polygonal reconstruction, we are able to accurately and efficiently compute many geometric properties of an insect wing that is only possible with well-segmented data. For each wing, we also use the connectivity of neighboring vein domains and vertices to construct an adjacency matrix describing topological relationships between neighboring vertices.

RESULTS

Since absolute wing size in insects is roughly correlated with body size[1], we do not consider size directly. However, using these geometrical and topological datasets, we then calculate the basic geometric features of a wing: scaled venation length (Fig. 2a), scaled contour curvature (Fig. 2b), and scaled venation length (Fig. 2c). For each case, we normalize shape and wing contour to compare with a circle. We also studied the topology of venation using methods from network analysis (Fig. 2d). Finally, we studied the distributions of geometric domains, regions bound by veins, in terms of two simple statistical measures (Fig. 2e): 1) Circularity (K): the ratio of domain perimeter to the circumference of a circle with the same area and 2) Fractional area (W): the ratio of each individual domain area to the area of the entire wing. Together, these metrics serve as complementary features for quantifying the range of morphological characteristics of insect wings.

Wing shape, venation length and contour curvature

Our first morphometric feature characterizes all interior venation relative to wing contour, in terms of the scaled wing perimeter P (contour), and the scaled interior venation network, L (which excludes perimeter). Both of these geometric features are normalized by the square root of the wing area in order to isolate total vein length from overall wing size (Fig. 2a). Then, we define contour, the normalized perimeter,

$$P = \tilde{P} - 2\sqrt{\pi}, \tag{1}$$

so that a circle with unit area would have $P = 0$. Additionally, L, the summation of all lengths of interior vein connectivity within the wing, yields the density of wing venation. In Fig. 3a, we plot the scaled wing contour length L against the venation contour P, noting that a wing at the origin $(0, 0)$ corresponds to a circular wing without any internal venation. We see that wings with dense venation (e.g. locust, Orthoptera) occupy the upper middle/right sections of the graph and wings with sparse venation (e.g. fruit fly (Diptera)) occupy the lower/left regions of the morphospace.

Our second morphospace characterizes wing shape complexity treated in terms of its boundary curvature $\kappa(s)$ as a function of the arc-length distance from the wing hinge. In Fig. 3b, we show a plot of wing curvature varying from wing base ($s = 0$ and $s = 1$) to wing tip ($s = 0.5$) for representative wings from six orders (top to bottom): *Diptera, Plecoptera, Odonata, Neuroptera, Orthoptera, Phasmatodea*. There seems to be no particular pattern to the distribution of curvature that we can discern.

Wing vein topology

Wing venation forms a physical network, with the intersection of veins as nodes (Fig. 4). We use tools from network analysis[17,18] to clustering the network into communities quantifying a third major trait of a wing: a topological measure of the complexity of venation patterns. We start by building an unweighted symmetric adjacency matrix, A, where every node$_{ij}$ = node$_{ji}$ (see Fig. 2d). To partition a wing network into clusters or communities[19], we use the maximum modularity measure, which compares a given network to a randomly generated network and is maximized by a partition factor[20] (see SI for other network versions comparison with more methods). This allows us to determine the number and size of clusters, each of which indicates a higher density of internal connections within a group of nodes, relative to connections across clusters.

We deployed simple network analysis to understand venation hierarchies and patterns. deployed. Clustering a network into communities quantifies a third major trait of a wing: a topological measure of the complexity of venation patterns. In Fig. 4 we show range of venation patterns seen in wings, with sparse venation in Diptera at one end, and the dense venation in Odonata at the other end. For wings with sparse venation there are few clusters, e.g. Diptera, Hymenoptera, whereas those with dense venation shows many clusters, e.g. Orthoptera, Odonata. Our comparative network analysis is a first step in understanding how we use venation topology as a precursor to quantifying mechanical and hydrodynamical aspects of wings. It is an open question whether topology metrics will provide insight into wing function.

Wing domain size and shape distribution

In addition to whole-wing topological and geometric features, we also considered fine-grain features. We found that wings with sparse venation tend to have more rectangular domains (Fig.5a), while wings with dense venation (Fig.5a) tend to have higher numbers of more circular domains. In Figure 5b, we show the domain distributions from representatives of six insect orders with varying complexities of wing shape and venation. This morphospace quantifies domain shapes (circularity, K), their distribution in a wing and how much area they occupy within a wing (fractional area, W). Within this space, domains at $(0, 1)$ are small and circular while domains at (0,0) are small and rectangular.

Building on the recent work[16], we plotted domain shapes and how they vary in space across the span of a wing. In Fig. 5c, we consider the proximal to distal axis (P–D axis, wing base to tip) of the wing (similar to wing span). This axis is divided into $N = 25$, rectangular bins, where each bin encompasses all domains across that chord (distance from leading edge and trailing edge of the wing). Following the method of Hoffmann *et al*, we then compute the area-weighted mean area and circularity of all domains within each bin. Then we apply a set of normalized coordinates on this P–D axis, through the computed domain area-circularity space, which is smoothed with a Gaussian of width twice the number of bins (2/25 here) and rescaled by the wing's perimeter. Similarly, portions of the P–D curves located near $(0, 1)$ describe regions of the wing that are dominated by small, circular domains while portions of curves near (1,0) contain domains are characterized by larger, rectangular domains.

In Fig. 5a, we show that domain circularity and fractional area vary within a wing, providing a geometrically minimal description of the internal structure of a wing. Dipteran wings have larger, more rectangular domains, while Odonate wings are made up of smaller, rounder domains. While sparsely veined wings comprise a handful of rectangular domains, e.g. fruit flies (gray, Fig. 5b) and stoneflies (purple, Fig. 5b), other species have thousands of domains. For example, species within the order Megaloptera (Fig. 1) have large, elongate wings, but contain mostly rectangular domains. Furthermore, we see an asymmetry in the distribution of domains, where, as domain number increases, higher numbers of smaller, more circular domains are found in the wing's trailing edge.

Following the method described in Hoffman *et al.*[16], in Fig. 5c, we summarize the distribution of domains across the span of a wing, from the proximal to the distal region, showing six species, each of which belongs to a different order. The resultant J-shaped curves represent the entire distribution of domains across the span of a wing. For a dragonfly wing (Odonata, green) the domain morphospace includes more rectangular domains at its wing base (faded green) than at the wing tip, where domains are more circular and more numerous (light green). In contrast, a Plecopteran wing (stonefly, purple), has the opposite domain distribution: larger domains (increased fractional area) and domains are more rectangular at wing tip (dark purple) than at the wing tip. For larger, more elongate wings, rectangular domains tend to be found near the proximal end of the wing while the distal end tends to have smaller, more circular wing domains (Fig. 5a). For approximately 468 Odonate wings (Fig. 5b,c, green), domains near the wing base tend to be rectangular, taking up a larger fractional area of the entire wing. Near the distal end, domains are more circular, taking up less area. In contrast, Neuropteran wings have more elongate and rectangular domains towards the wing tip. Some domains makeup over 10% the total area of the wing (i.e. Diptera), while the smallest account for only 1/10,000 the entire area (i.e. Odonata) of an insect wing. With these domain distributions, our P-D curve morphospace (Fig. 5c) categorizes the spatial geometries of domains across the span of a wing.

DISCUSSION

Our study has assembled a large dataset of wings from across insect orders. Using this data set, we analyzed segmented wings to create morphospaces comparing simple topological and geometric features of the insect wing, compartmentalizing normalized shape, size and venation structure across the insect phylogeny. By providing these data of segmented images and adjacency matrices, our hope is that others can use different approaches from geometric morphometrics to understand the evolution of wing venation patterns, while also informing modeling efforts for wing flexibility[9,21–23].

The morphospace of Fig. 3, parses the complexity of wing shape and venation, characterizing contour and internal venation. Results suggest that fliers characterized as forewing dominated tend to have more sparse venation (i.e. Diptera, Hymenoptera) than those that are hindwing or both wing dominated fliers.

The range in shape and branching internal venation also relate to an insect's flight characteristics and its resistance to damage. Insects with higher numbers of cross veins (thus higher numbers of domains), especially towards the trailing edge, are more likely to reduce tearing and fracturing of the wing that might occur throughout its lifespan[24,25]. Within Odonata, species with larger wings have larger and more numerous domains[16]. While not always applicable across orders, an asymmetry regarding domain number and size across the wing span could be beneficial; from a structural integrity perspective, asymmetry provides fracture toughness[24]. Since cross-veins effectively transfer tensile stresses to neighboring wing domains[25], wings with higher numbers of small domains (increased cross veins) in the trailing edge could reduce damage propagation[24].

Our paper is but the first step in addressing the origins and functional consequences of insect wings, and we hope that others will take up the challenges posed by these questions using the datasets and the simple morphometric approaches that we have outlined.

Supporting Information (SI)

SI Datasets

Full methods available in the SI Appendix. This includes: detail on the geometric and topological analysis and lists of species shown in figures. Code: https://github.com/hoffmannjordan/Fast-Marching-Image-Segmentation and https://github.com/hoffmannjordan/size-and-shape-of-insect-wings.

FUNDING

MKS and SD were supported by the NSF-GRFP.

ACKNOWLEDGEMENTS

The authors would like to thank the librarians at the Ernst Mayr Library, Harvard University for their help in finding insect images, and fruitful discussions with Stacey Combes and Chris **H.** Rycroft.

REFERENCES

[1]D. Grimaldi and M. Engel, *Evolution of Insects*. New York, New York: Cambridge University Press, 2005.

[2]C. Betts and R. Wootton, “Wing shape and flight behavior in butterflies (lepidoptera: Papilionoidea and hesperioidea): A preliminary analysis,” *Journal of experimental biology*, vol. 138, no. 17, pp. 271–288, 1988.

[3]R. Wootton, “Functional morphology of insect wings,” *Annual Review of Entomology*, vol. 37, pp. 113–140, 1992.

[4]R. Wootton, “Support and deformability in insect wings,” *Journal of Zoology*, vol. 193, pp. 447–468, 1981.

[5]F. Johansson, M. Soderquist, and F. Bokma, “Insect wing shape evolution: independent effects of migratory and mate guarding flight on dragonfly wings,” *Biological Journal of the Linnean Society*, vol. 97, no. 2, pp. 362–372, 2009.

[6]R. Parchem, M. Perry, and N. Patel, “Patterns on the insect wing,” *Current Opinion in Genetics and Development*, vol. 17, no. 4, pp. 300–308, 2007.

[7]P. DeVries, C. Penz, and R. Hill, "Vertical distribution, flight behaviour and evolution of wing morphology in morpho butterflies," *Journal of Animal Ecology*, vol. 79, no. 5, pp. 89–94, 2010.

[8]A. Brodsky, *The Evolution of Insect Flight*. New York, New York: Oxford University Press Inc, 1994.

[9]S. Jongerius and D. Lentink, "Structural analysis of a dragonfly wing," *Experimental Mechanics*, vol. 50, no. 9, pp. 1323–1334, 2010.

[10]C. J. Rees, "Form and function in corrugated insect wings," *Nature*, vol. 256, p. 200, 1975.

[11]H. Nijhout, "Elements of butterfly wing patterns," *Journal of experimental zoology (Mol Dev Evol)*, vol. 291, pp. 213–225, 2001.

[12]V. Debat, A. Debelle, and I. Dworkin, "Plasticity, canalization and developmental stability of the *Drosophila* wing: Joint effects of mutations and developmental temperature," *Evolution*, vol. 63, no. 11, pp. 2864–2876, 2009.

[13]N. MacLeod, *Automated taxon identification in systematics: Theory, approaches and applications*. Natural History Museum, London, UK: CRC Press: Taylor and Francis Group, 2007.

[14]C. Pelabon, C. Firmat, G. H. Bolstad, K. L. Voje, D. Houle, J. Cassara,A. Le Rouzic and T. F. Hansen, "Evolution of morphological allometry," *Annals of the New York Academy of Sciences*, 1320:58-75, 2014.

[15]"Image from the biodiversity heritage library. biodiversity heritage library." http://www.biodiversitylibrary.org. Accessed Jan 2017.

[16]J. Hoffmann, S. Donoughe, K. Li, M. K. Salcedo, and C. H. Rycroft, "A simple developmental model recapitulates complex insect wing venation patterns," *Proceedings of the National Academy of Sciences*, 115 (40) 9905-9910, 2018.

[17]J. Lasser and E. Katifori, "Netfi: a new framework for the vectorization and examination of network data," *Source Code for Biology and Medicine*, vol. 12, no. 1, p. 4, 2017.

[18]M. Dirnberger *et al.*, "Net: Network extraction from images," *Scientific Reports*, vol. 5, no. 15669, 2015.

[19]G. Morrison and L. Mahadevan, "Discovering communities through friendship," *PlosOne*, 7 (7), e38704, 2012.

[20]M. Newman, "Equivalence between modularity optimization and maximum likelihood methods for community detection," *Physical Review E*, vol. 94, p. 052315, 2016.

[21]H. Rajabi, A. Shafiei, A. Darvizeh, J. Dirks, E. Appel, and S. Gorb, "Effect of microstructure on the mechanical and damping behaviour of dragonfly wing veins," *Royal Society Open Science*, vol. 3, no. 2, 2016.

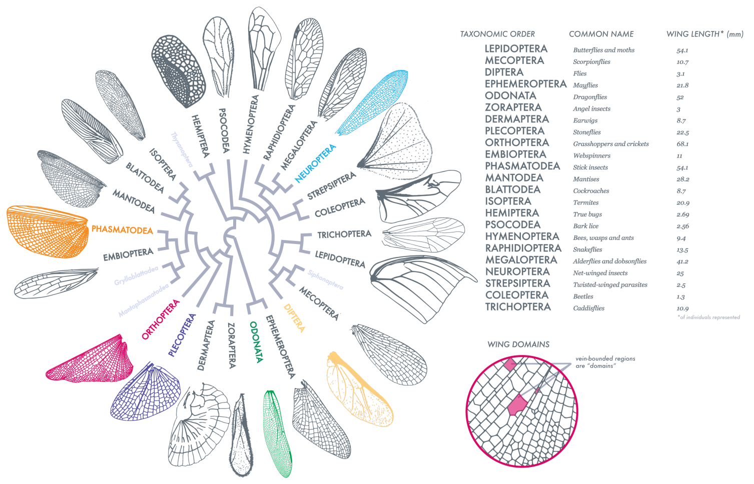
[22]J. Sun, M. Lin, C. Pan, D. Chen, J. Tong, and X. Li, "Biomimetic structure design of dragonfly wing venation using topology optimization method," *Journal of Mechanics in Medicine and Biology*, vol. 14, no. 4, p. 1450078, 2014.

[23]T. Mengesha, R. Vallance, M. Barraja, and R. Mittal, "Parametric structural modeling of insect wings," *Bioinspiration and Biomimetics*, vol. 4, no. 3, p. 036004, 2009.

[24]J. Dirks and D. Taylor, "Veins improve fracture toughness of insect wings," *PlosOne*, vol. 7, no. 8, pp. 1–9, 2012.

[25]H. Rajabi, A. Darvizeh, A. Shafiei, D. Taylor, and J. Dirks, "Numerical investigation of insect wing fracture behaviour," *Journal of Biomechanics*, vol. 48, no. 1, pp. 89–94, 2015.

[26]B. Misof *et al.*, "Phylogenomics resolves the timing and pattern of insect evolution," *Science*, vol. 346, no. 6210, pp. 763–767, 2014.



TAXONOMIC ORDER	COMMON NAME	WING LENGTH* (mm)
LEPIDOPTERA	*Butterflies and moths*	54.1
MECOPTERA	*Scorpionflies*	10.7
DIPTERA	*Flies*	3.1
EPHEMEROPTERA	*Mayflies*	21.8
ODONATA	*Dragonflies*	52
ZORAPTERA	*Angel insects*	3
DERMAPTERA	*Earwigs*	8.7
PLECOPTERA	*Stoneflies*	22.5
ORTHOPTERA	*Grasshoppers and crickets*	68.1
EMBIOPTERA	*Webspinners*	11
PHASMATODEA	*Stick insects*	54.1
MANTODEA	*Mantises*	28.2
BLATTODEA	*Cockroaches*	8.7
ISOPTERA	*Termites*	20.9
HEMIPTERA	*True bugs*	2.69
PSOCODEA	*Bark lice*	2.56
HYMENOPTERA	*Bees, wasps and ants*	9.4
RAPHIDIOPTERA	*Snakeflies*	13.5
MEGALOPTERA	*Alderflies and dobsonflies*	41.2
NEUROPTERA	*Net-winged insects*	25
STREPSIPTERA	*Twisted-winged parasites*	2.5
COLEOPTERA	*Beetles*	1.3
TRICHOPTERA	*Caddisflies*	10.9

*of individuals represented

Figure 1. **Wing taxonomy: size, shape and structure** Adapted from Misof et al.[26], this order-level representation of insect wing size, venation patterns, shapes and wing domains (as defined in the figure on the right) exhibits the range and diversity of our sampling. Insect orders labeled in light gray are not sampled or characterized as wingless (full species list in SI). Scale bars are for wings represented on the phylogeny.

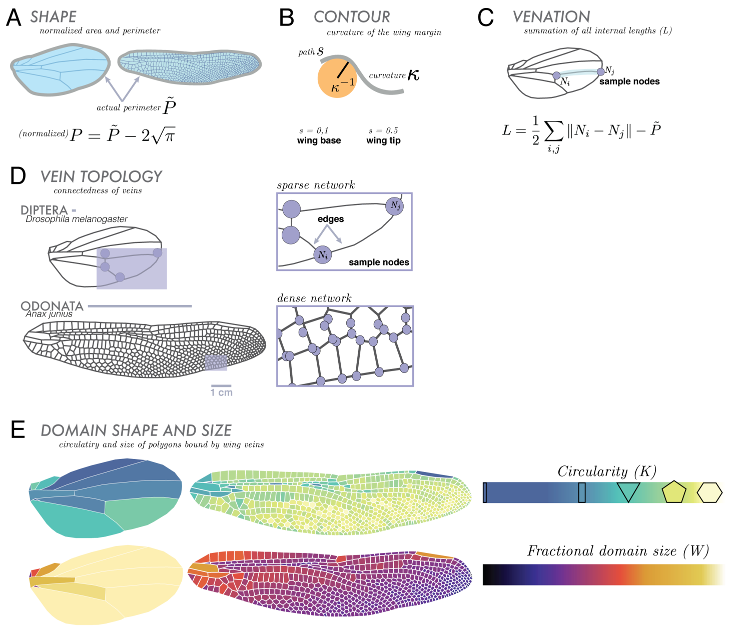

A SHAPE
normalized area and perimeter
actual perimeter $\tilde{P}$
(normalized) $P = \tilde{P} - 2\sqrt{\pi}$
B CONTOUR
curvature of the wing margin
path s
κ^{-1}
curvature κ
s = 0.1
wing base
s = 0.5
wing tip
C VENATION
summation of all internal lengths (L)
N_i
N_j
sample nodes
$L = \frac{1}{2}\sum_{i,j} \|N_i - N_j\| - \tilde{P}$
D VEIN TOPOLOGY
connectedness of veins
DIPTERA - Drosophila melanogaster
ODONATA Anax junius
1 cm
sparse network
edges
sample nodes
dense network
E DOMAIN SHAPE AND SIZE
circularity and size of polygons bound by wing veins
Circularity (K)
Fractional domain size (W)

Figure 2. **Wing morphometrics** We focus on broad comparative geometric and topological components, illustrated here using as examples Diptera (*Drosophila melanogaster*) and Odonata (*Anax junius*) wings. For geometric features, we analyze curvature, shape and area, and internal venation. **(A)** Contour, κ, is given by the radius of curvature or κ^{-1} (where s is arc length along the wing). **(B)** Shape: all wings are normalized to have an area equal to that of a circle with an area of unity (removing absolute size effects). Wing shape is characterized by its scaled perimeter, P, where $\tilde{P}$ is the actual perimeter of the wing. **(C)** Venation is treated as a network, and quantified in terms of the sum of its total internal length, L, where N_i and N_j are representative nodes. We continue analyzing venation using topological measures where **(D)** the wing is a network of vein junctions (nodes) and the lengths of vein between them (edges). Lastly, we observe the geometries and distributions of vein domains. **(E)** Domains are characterized by their circularity (shape relative to that of a circle) and fractional domain size (domain area relative to area of entire wing).

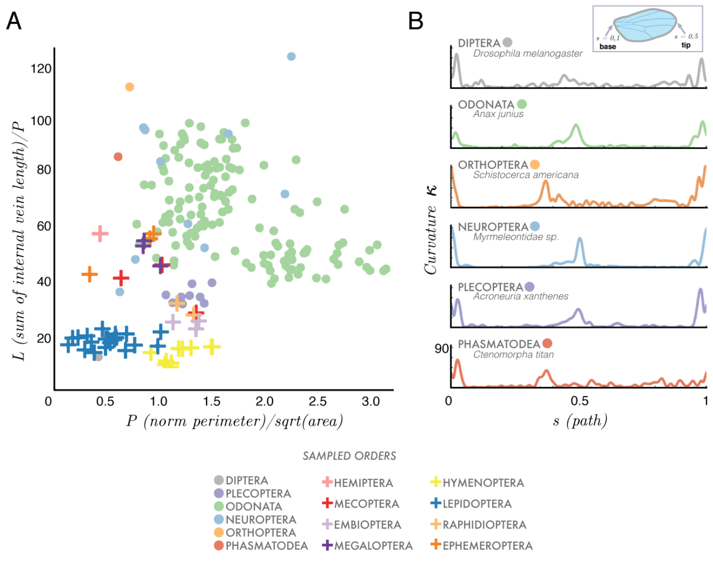


Figure 3. **Wing shape, wing contour and internal venation** Comparison of three geometric traits (of all sampled orders) where **(A)** contour is defined using a scaled curvature (κ) (scaled by total perimeter P) as a function of arc length, s, where the wing base is $s = 0$ and the wing tip is $s = 0.5$. **(B)**. The total sum of all internal vein lengths L (scaled by the perimeter P), as a function of the normalized perimeter P (scaled by the square root of the area of the wing, see text) characterizes venation density. Species (per insect order) are represented by either circles or crosses.

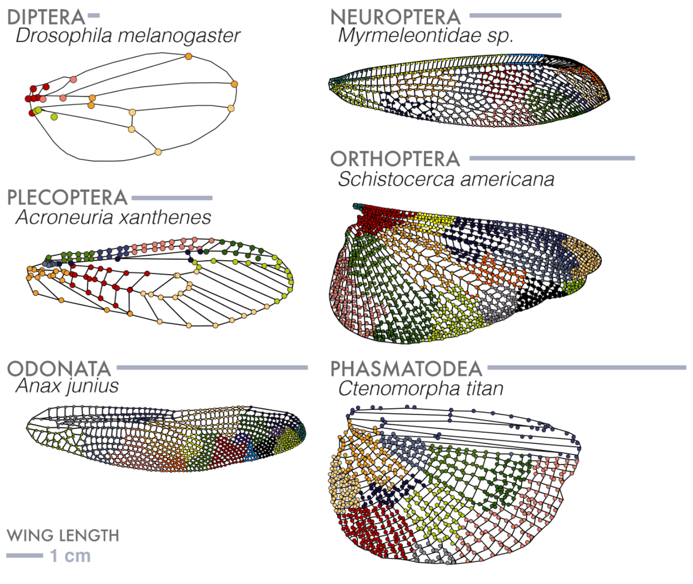


Figure 4. **Wing venation network analysis** A wing is a network made up of vein junctions (nodes) and the lengths between them (edges). Using segmented wing images, where each 2D component of the wing is mapped out, we characterize a network using a common community detection algorithm, maximum modularity (see text for details and definitions). Here we show a sampling of wing types and their resultant patterning of clusters.

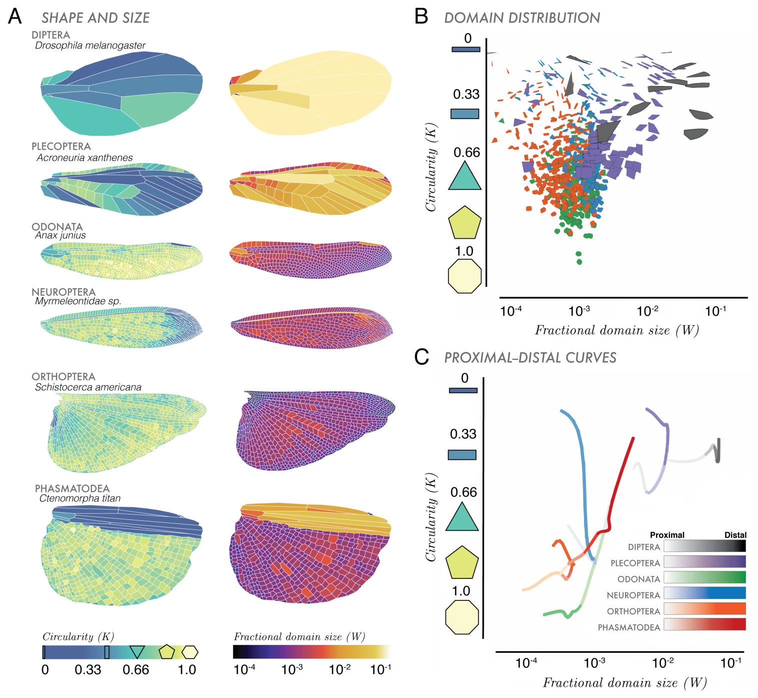


Figure 5. **Wing domain sizes and shapes (A)** Circularity (K) and fractional area (W) characterize the distribution of polygonal shapes that make up the vein-bounded domains within wing (see text for definitions). **(B)** Circularity as a function of Fractional Domain Size. **(C)**, Along the proximal to distal (P–D, wing span from base to tip) axis across a wing, we show circularity varies by fading color (lighter = wing base, darker = wing tip), for six insect orders.

2.3.1 Mathematical Supplement to Hoffmann, *et. al.* 2018. PNAS

A simple developmental model recapitulates complex wing venation patterns in insects

Supplemental Materials

Jordan Hoffmann[†,1], Seth Donoughe[†,2], Kathy Li[3], Mary K. Salcedo[2], Chris H. Rycroft[1,4]

Contents

† Jordan Hoffmann and Seth Donoughe contributed equally to this work. Author order is random.

1 Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, United States
2 Department of Organismic and Evolutionary Biology, Harvard University, Cambridge, MA 02138, United States
3 Applied Physics and Applied Mathematics Department, Columbia University, New York, NY 10027, United States
4 Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA 94720, United States

Overview of this document

We have divided this supplement into six main sections: (1) Data collection, image processing, and segmentation, (2) Quantitatively characterizing the arrangement of wing veins, (3) Procedure for simulating secondary vein patterns, (4) Simulation results, (5) Availability of data and code, and (6) Acknowledgements for images obtained from published books.

1 Data collection, image processing, and segmentation

The dataset of insect wings studied in this paper is a combination of original micrographs of wild-collected specimens and published wing tracings from the scientific literature. In this section, we describe our techniques for acquiring those images and a custom computational method that we used to segment the wing veins. We also show that this method is robust and effective for segmenting many different types of biological images. Finally, we assess segmented wing images to confirm that wings from different sources are comparable.

1.1 Original micrographs

Some specimens were collected by the authors in the vicinity of Swarthmore College in Swarthmore, PA or the Concord Field Station, in Bedford, MA. Other specimens were donated from the Harvard University entomology teaching collections. Insects were caught with nets, euthanized, pinned, spread, and identified. Wings were removed with forceps at the wing hinge and then imaged with either a scanner or microscope. *Imaging with a flatmount scanner:* Wings were placed under a glass slide on a film and document scanner (CanoScan 9000F Mark II), illuminated with white reflected light, and then scanned at a resolution of 2400 dots per inch (DPI). *Imaging with a microscope:* Wings were illuminated with transmitted white light and imaged with a Zeiss AxioZoom V16 microscope at 35× optical magnification. Using Zen Pro software (Zeiss), the stage was moved to tile the entire wing with overlapping imaging positions. A

Z-stack was captured at each position. Z-stacks were fused using the "Tiling" and "Extended Depth of Focus" functions in `Fiji/ImageJ` (NIH).

1.2 Published wing images

We used Google Scholar and Web of Science to compile a maximally inclusive list of candidate books, reviews, and field guides for Odonata. We then pared down the list to those sources that could be obtained as a physical copy, contained descriptions of multiple species, and included detailed wing drawings that were printed at a high enough resolution to resolve even the smallest vein domains [1, 2, 7, 9]. We scanned each wing image from each source at 1200 DPI. In cases where there was notation on the wing diagrams, we used Adobe Photoshop to manually remove the arrows and/or text if it was possible to do so without disturbing the drawn wings veins. Otherwise, graphically annotated wings were discarded. There is a highly conserved vein domain called the pterostigma that is often pigmented. We manually filled this domain in with white pixels to ensure that it properly segmented as a wing domain. For all published wing images, if the length of the wing was given in the text, we used it. When the length of the wing was not listed, we used the length given in *The Encyclopedia of Life* [20]. When multiple wing measurements were listed, we used the mean length.

1.3 Segmenting wing images

Naturalists have used images—drawings, photographs, and micrographs—to capture the shapes and patterns of biological tissues for hundreds of years. Historically, measurements of morphological traits have been made by hand, and it is especially laborious to extract phenotypes from complex biological images in a high dimensional, quantitative way. Software-based image analysis tools have made this process dramatically faster, but most morphological traits are still not readily quantifiable in an automated manner.

Many diverse tissues and patterns take the form of bounded shapes on a surface, such as the edges of cell junctions in epithelia, coat patterning in mammals, and insect wing veins. Thus, the process of taking a bitmap image of bounded shapes and segmenting into "edges" and "background" is a common problem in many disciplines. Several general purpose tools have been created for this purpose, such as the widely used image processing toolbox `Fiji/ImageJ` or the machine learning based tool `ilastik` [10]. Additionally, a few tissue specific segmentation tools have been created [11, 12, 13, 14].

We found that most standard segmentation tools were ineffective for our purpose of segmenting a wide range of images coming from different sources. In Fig. S1 we show a selection of the wing images used in the present study, including micrographs and scans from published sources.[1] Additionally, we wanted our segmentation tool to unambiguously resolve contacts between wing domains, a capability that is not offered by some existing segmentation methods. Therefore, to segment wing images, we developed a custom code based on the fast marching method with a variable background velocity field [17, 18]. We implement it using the open-source `scikit-fmm` python library [19] which solves the Eikonal equation

[1] *Note:* Most images used in this study are between 2000 and 4000 pixels along the long axis.

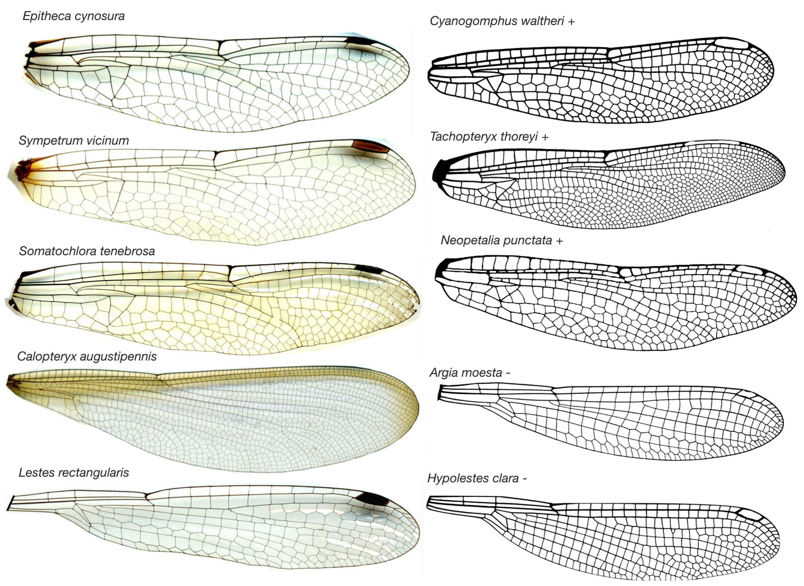


Figure S1: **Example wing images used in manuscript.** The left hand column shows wing micrographs reprinted with permission from [8]. Images on the right come from published sources where authors have traced wings. Images marked with '+' are from [2], p. 16, Figure 53; p. 97, Figure 587; p. 100, Figure 604; p. 134, Figure 704; p. 157, Figure 864. (c) 2006 Johns Hopkins University Press. Reprinted with permission of Johns Hopkins University Press. Images marked '−' are reprinted with permission from the artist [7]. Note that in the published wing images, we have filled in the pterostigma domain (typically pigmented) with white pixels to ensure it is properly segmented as a wing domain.

$F(\vec{x})|\nabla T(\vec{x})|= 1$ where $T(\vec{x})$ is the travel time from a set of pre-determined seed points, and $F(\vec{x})$ is a spatially varying speed.

We begin by generating candidate seed points using either a watershed-based segmentation or connected components after Otsu thresholding.[2] Figure S2 outlines the process of our segmentation. Panel **A** shows the original wing image. Panel **B** shows the background velocity field we construct from our image. We threshold the original image and find connected components, each of which is given a different random color in panel **C**. To generate seed points from our connected components, we only include components that have an area of at least 30 pixels, which excludes very small components generated by imaging noise. If a component's dimensions are greater than 500 pixels in both x and y, we treat it as background by randomly dropping a large number of additional seed points in it. We do this in order to chop up the background space and ensure that wing domains do not bleed outside the wing. Remaining connected components are treated as wing domains; the centroid of each domain is assigned a single

[2]Which segmentation method we use depends on the type of input image. For published drawings, we used watershed, whereas for micrographs we used Otsu thresholding.

seed at its centroid. Panel **D** shows our seed points as white crosses. From each seed point, we compute $T(\vec{x})$, shown in Panel **E**.

We use the wing image to create a background velocity field where we set the velocity in the darkest pixels to be 1000 times slower than in the lightest pixels. Thus, when the fronts from two different seed points collide, they collide in the vein. This was essential as it allowed us to accurately resolve neighbor information, something that we leverage to convert each domain shape into a polygon. This is helpful for analyzing vein geometry, because a polygon-based representation of vein domains minimizes the effect of pixel-scale noise on the quantitative features of domains.[3] Due to imperfections in the printing process of book images, there are occasionally very small domains that are not seeded accurately. However, when we calculate attributes for all the domains in a wing, we weight values by their constituent domain's area, meaning that these seeding errors have a negligible effect.

1.4 Segmenting other biological images

We show that our segmentation method is effective on a variety of biological images in Fig. S3.

1.5 Assessing the accuracy and consistency of wing images

Since many of the images used in the present study are published wing tracings, we establish that authors of those sources are able to accurately and consistently recapitulate the veins of wings. We do so by comparing our original micrographs to drawings of the wings of the same species in published sources. We also establish that the authors of our published sources do not substantially differ from one another in their drawings of wings.

1.5.1 Comparing between original wing micrographs and published sources

In Fig. S4, we show examples of real wings we captured and imaged directly, compared to wing images of the same species taken from published sources. We find that they are quite similar and that the drawings accurately capture the characteristics of the wings used in this manuscript. For two examples, we show area and circularity maps.

1.5.2 Comparing between wing images from different published sources

In Fig. S5 we compare wings of the same species from two sources. Overall, we find that the two sources agree with one another quite well. In Fig. S6 we compare the size and circularity of domains in a *Dromogomphus spinosus* from both sources.

[3]Segmentation imperfections along veins tend to be approximately the same thickness (a few pixels in size). Therefore, the fractional change in properties like area and perimeter is larger for smaller domains. Converting domains to polygons minimizes this effect.

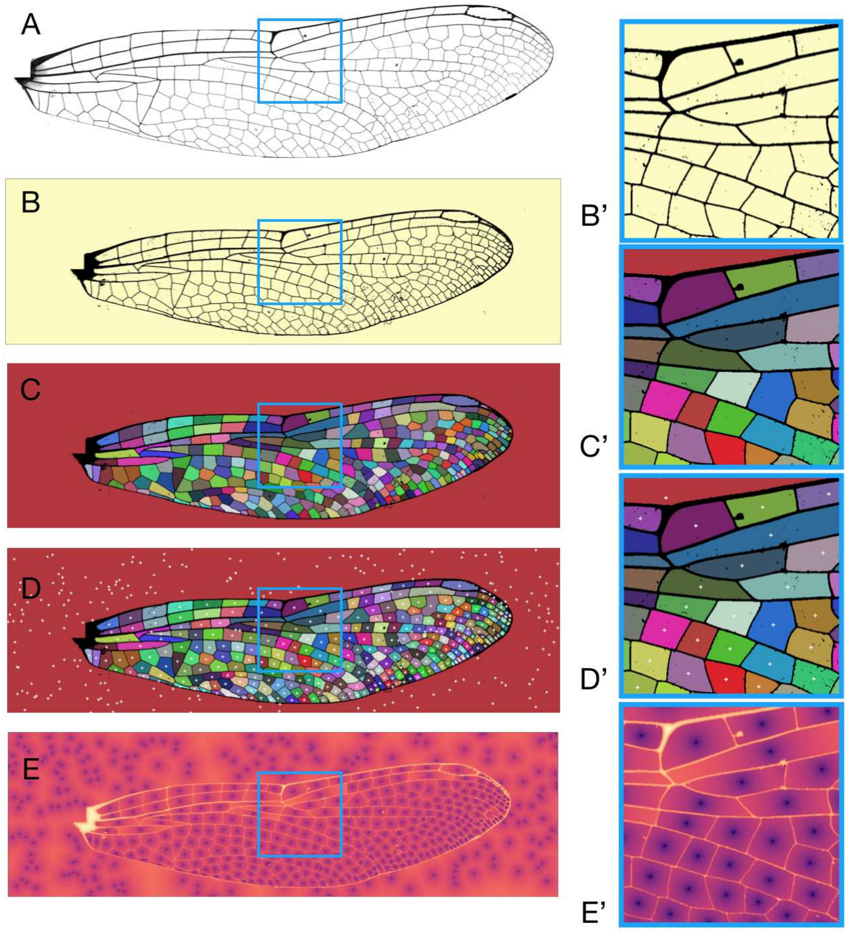


Figure S2: **Segmentation of wing images.** Panel **A** shows the raw image of an insect wing. Panel **B** shows the background velocity field created from the raw wing image. A black pixel has a value of 0.0 and a white pixel to have a value of 1.0. To set the velocity, the formula $v_{i,j} = p_{i,j} + 0.001$ is used, where the offset of 0.001 ensures that the speed through a black vein is 1000 times slower than through free space. Panel **C** shows connected components after thresholding. Points within these components are used as seeds for our fast marching method. Panel **D** shows the seed points from each connected component denoted by a white cross. Panel **E** shows the travel time matrix plotted on a logarithmic scale obtained after solving an equation of the form $F(\vec{x})|\nabla T(\vec{x})| = 1$. $F(\vec{x})$ represents the speed and $T(\vec{x})$ represents the travel time matrix. The figures on the right (**B'**, **C'**, **D'**, **E'**) are magnifications of the regions boxed in blue.

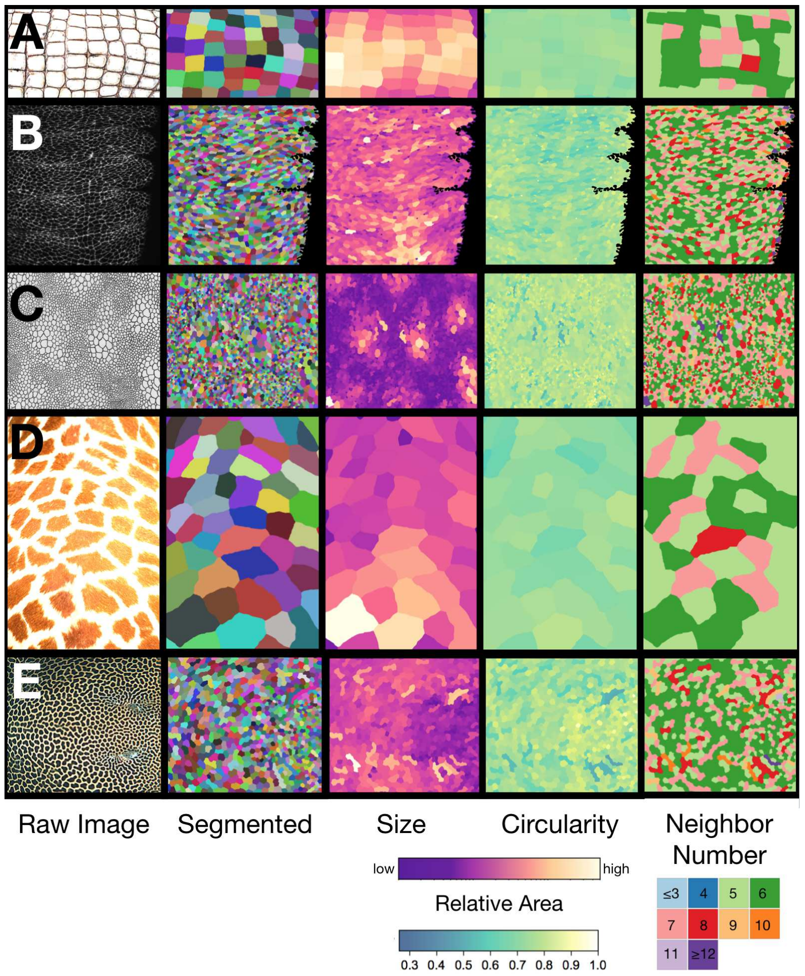


Figure S3: **Segmenting patterns in biology.** Row **A**: Alligator scales. Image courtesy of Matt Eich (photographer) [32]. Row **B**: Developing ventral epidermis on an embryo of *Drosophila melanogaster*. Fixed and stained to show adherens junctions, following immunohistochemistry methods described in Simone and Dinardo (2010) [33]. Row **C**: Abdominal scale imprint of a *Trachodon*. Image courtesy of Wikimedia Commons/Henry Fairfield Osborn [34]. Row **D**: Giraffe Image courtesy of Public Domain Pictures/Petr Kratochvil [35]. Row **E**: Reticulate whipray. Image courtesy of Krystof Tichy (photographer) [36]. First column shows the input image, the second column shows the segmented image with domains colored randomly, the third column shows the relative area of different regions, the fourth column shows the circularity, and the fifth column shows neighbor number.

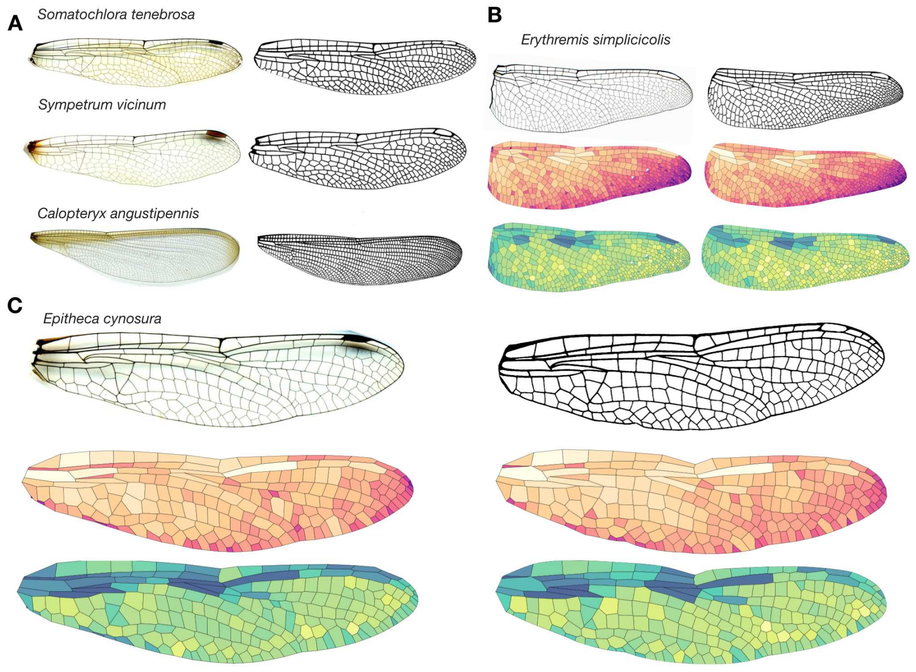


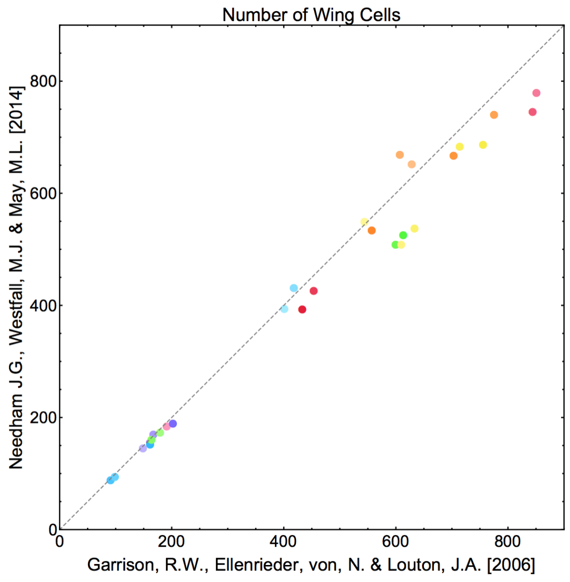
Figure S4: **Comparing original micrographs and published wing images.** Panel **A** shows a series of original dragonfly wing micrographs (on the left) and published images (on the right). Panels **B** and **C** show the segmentation of two different wings, colored by size and circularity, following the color scales in Fig. 1 of the main text. Reprinted with permission from Michael L. May (artist) [1]. From [2], p. 16, Figure 53; p. 97, Figure 587; p. 100, Figure 604; p. 134, Figure 704; p. 157, Figure 864. (c) 2006 Johns Hopkins University Press. Reprinted with permission of Johns Hopkins University Press.

2 Quantitatively characterizing the arrangement of wing veins

2.1 Calculating circularity

For each wing polygon $\mathcal{P}_i$, we compute its circularity by computing the length of its perimeter (∂_i) and its area ($\mathcal{A}(\mathcal{P}_i)$). Defining the radius of a circle with the same perimeter to be $R_c = \partial_i/(2\pi)$, we can then write that the area of the equivalent circle is $A_c = R_c^2\pi$. The circularity is then given by

$$\mathcal{C}(\mathcal{P}_i) = \frac{\mathcal{A}(\mathcal{P}_i)}{A_c}. \tag{1}$$



●	*Archilestes grandis* Hindwing	●	*Ischnura hastata* Hindwing	●	*Aeshna juncea* Hindwing
●	*Archilestes grandis* Forewing	●	*Ischnura hastata* Forewing	●	*Aeshna juncea* Forewing
●	*Tachopteryx thoreyi* Hindwing	●	*Lanthus vernalis* Hindwing	●	*Basiaeschna janata* Hindwing
●	*Tachopteryx thoreyi* Forewing	●	*Lanthus vernalis* Forewing	●	*Basiaeschna janata* Forewing
●	*Zoniagrion exclamationis* Hindwing	●	*Tanypteryx hageni* Hindwing	●	*Epiaeschna heros* Hindwing
●	*Zoniagrion exclamationis* Forewing	●	*Tanypteryx hageni* Forewing	●	*Epiaeschna heros* Forewing
●	*Hesperagrion heterodoxum* Hindwing	●	*Microneura caligata* Hindwing	●	*Nasiaeschna pentacantha* Hindwing
●	*Hesperagrion heterodoxum* Forewing	●	*Microneura caligata* Forewing	●	*Nasiaeschna pentacantha* Forewing
●	*Enacantha caribbea* Hindwing	●	*Oplonaeschna armata* Hindwing		
●	*Enacantha caribbea* Forewing	●	*Oplonaeschna armata* Forewing		

Figure S5: **Comparison of wing images from two sources.** Plot showing the number of wing domains that are present in equivalent wings of the same species from the two sources used [1, 2].

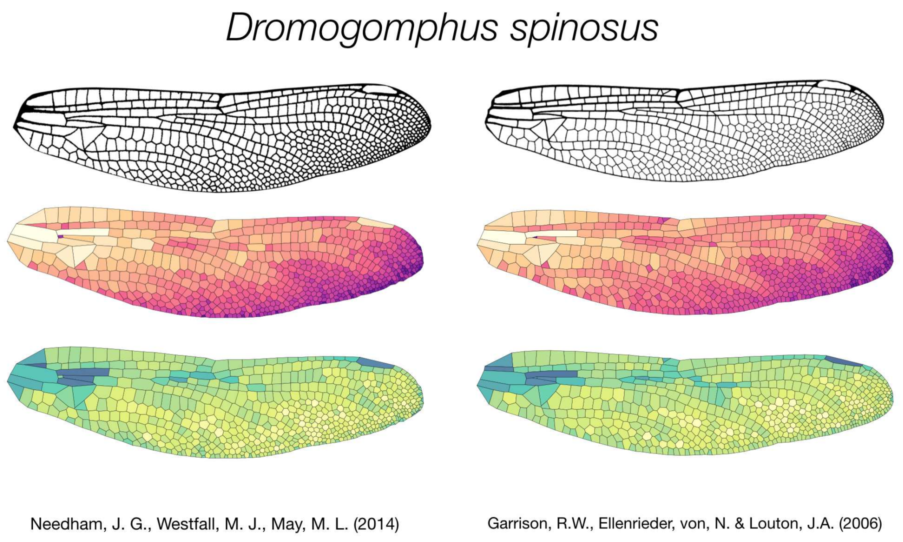

Dromogomphus spinosus
Needham, J. G., Westfall, M. J., May, M. L. (2014)
Garrison, R.W., Ellenrieder, von, N. & Louton, J.A. (2006)

Figure S6: **Comparison of a *Dromogomphus spinosus* wing from two sources.** The original wing (top), the wing with domains colored by size (middle), and the wing with domains colored by circularity (bottom). Reprinted with permission from Michael L. May (artist) [1]. From [2], p. 16, Figure 53; p. 97, Figure 587; p. 100, Figure 604; p. 134, Figure 704; p. 157, Figure 864. (c) 2006 Johns Hopkins University Press. Reprinted with permission of Johns Hopkins University Press.

2.2 Vein domain area and circularity for dragonfly and damselfly forewings

In Fig. S7 we show the area and circularity of domains in the smallest, median, and largest dragonfly and damselfly wings.

2.3 Correcting for corrugation

Our original micrographs and the published wing tracings are both two-dimensional projections of wings that have some three-dimensional corrugation [21]. This results in distortion to the observed shapes, illustrated in Fig. S8. Here we attempt to bound the effect of this corrugation on our quantification of domain sizes and shapes. We consider the effect of corrugation slope on fractional area and circularity. We consider "worst case" scenarios, but most of the wing does not have extreme corrugation [21, 22]. In fact, near the tip of the wing, corrugation is nearly absent [22].

2.3.1 The effect of corrugation on area measurements

Let us consider a rectangle with sides of lengths a and b. We can write that $b = \gamma a$ where γ represents the aspect ratio of the rectangle. We observe the rectangle as having $A_o = \gamma a^2$. Suppose that the wing

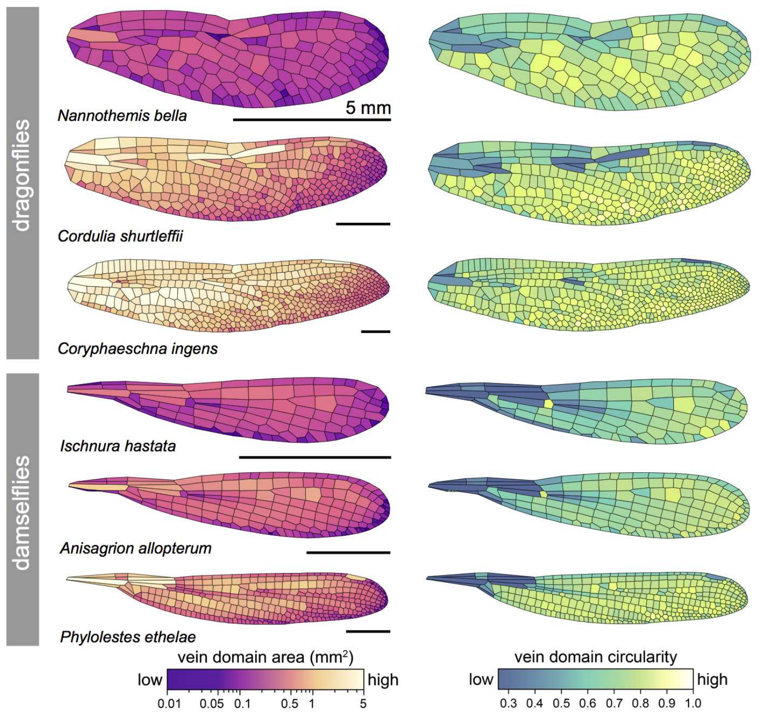


Figure S7: **Dragonfly and damselfly wings with domains colored by area and circularity.** The smallest, median, and largest dragonfly and damselfly wings in our dataset of segmented wings. Scale bar represents 5 mm for all images.

is angled in the direction along the side γa. The true length of this side is then $h = (\gamma a)/\cos(\theta)$. We can write

$$A_t = a\left(\frac{a\gamma}{\cos\theta}\right) \tag{2}$$

as the true area. The ratio of the observed area to the true area is therefore

$$\frac{A_o}{A_t} = \frac{\gamma a^2}{a\left(\frac{a\gamma}{\cos\theta}\right)} = \cos\theta. \tag{3}$$

In Fig. S9 we plot the distortion of area that results as a function of θ. Note that for most domains, the angle is between 0° and 5°. Very few domains lie on corrugations that are angled more than 20° (see the profiles in Fig. S9 Panel **B**).

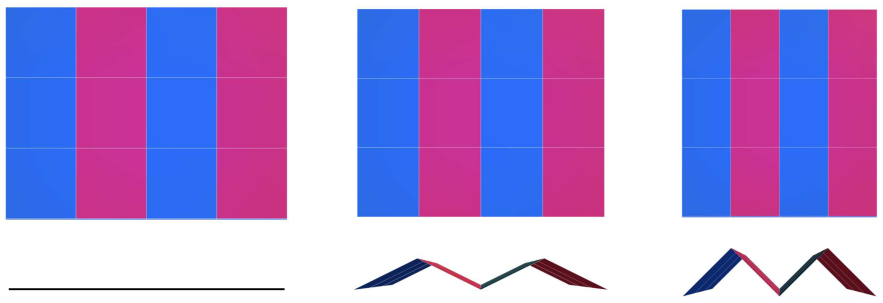


Figure S8: **Corrugation of a wing.** Odonate wings are corrugated. Here we show a top-down projection (**top row**) and side-on view (**bottom row**) of a corrugated sheet. From the top-down views, the squares appear distorted along the axis of corrugation. This distortion is investigated in Figure S9.

2.3.2 The effect of corrugation on circularity measurements

Here we consider how various angles of corrugation would affect the circularity calculation of wing domains that have the shape of a regular triangle, square, pentagon, and hexagon. We also consider the effect on a rectangular domain with aspect ratio 4. In Fig. S10 we show how the circularity of various shapes changes as a function of θ.

We find that for the range of angles that we consider, the effect on circularity is quite small. For the entire range of shapes we observe, deviations of 0–10° of the measured circularity differ from true circularity by less than 1%. Larger deviations in angles only occur in a very limited region of the wing and still result in only minor deviations. The most extreme effect that we see is for a relatively oblong rectangular shape where the change is 11%. For a pentagonal shape, the change is 1.4%.

2.4 Displaying vein domain shape and size features for a single wing

Vein domain shapes and sizes vary across each wing. In an effort to capture this diversity and compare among wings, we plot domain data from whole wings in several different ways. We use wing maps with each domain colored in according to its area or circularity (e.g. Fig. 1B, C in the main text). We also show single-wing scatter plots with each point representing a single domain from a wing, alongside marginal distributions of circularity and area (e.g. Fig. 1C in the main text). We show wings with data grouped by wing region (e.g. Fig. S33). We also plot vein domain features, averaged within bins that span a particularly important biological axis in wings: the proximal-to-distal axis.

2.4.1 Proximal-to-distal morphology traces

The proximal-to-distal (P–D) axis is a well-defined and widely studied axis of insect wings. The P–D axis begins at the hinge where the wing joins the body wall and extends to the farthest tip of the wing. It is

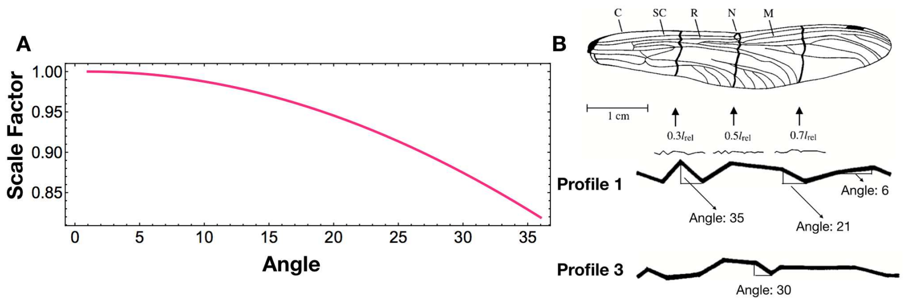


Figure S9: **Effect of corrugation on area.** Panel **A** shows the distortion of area that results from varying its corrugation slope θ for a rectangular object. Note that the curve plotted is merely $\cos(\theta)$. Panel **B** shows cross sections of the wing where the corrugation has been measured. Angles for the most extreme cases have been computed. The 'C', 'SC', 'R', 'N', and 'M' refer to the costa, subcosta, radius, nodus, and mediana primary veins, respectively. Republished with permission of Company of Biologists, from [21]; permission conveyed through Copyright Clearance Center, Inc.

often used as a way to orient morphological or biomechanical attributes of a wing [39, 40, 42, 43]. It has also been identified as a key axis of tissue specification in the wing [38, 41]. In this study we use "P–D wing traces" to summarize how vein domain characteristics vary along the long axis of the wing, which enables us to compare among distantly related insects. It also provides results that can be directly compared to much of the existing morphological, biomechanical, and developmental literature on insect wings. This approach necessarily obscures differences between domains that are in the same position along the P–D axis. Therefore, this is certainly not the only way to summarize vein domain characteristics within a wing. It is nonetheless effective at capturing similarities and differences among insects that are both closely and distantly related. For readers that are interested in comparing species without the dimensional reduction of P–D traces, we have included complete sets of domain attributes for a wing from each odonate species that we study in this paper at `https://github.com/hoffmannjordan/insect-wing-venation-patterns`.

To construct a P–D wing trace, we represent a wing as a curve through area–circularity space as one moves from the hinge at the wing base to the farthest tip of the wing. To do this, we begin by orienting the wing with the main principal component along the x-axis. We then partition the wing into 21 equal-width rectangular bins along the long axis. We then compute the circularity of each polygonal domain and then compute the area of overlap between the given polygon and the bin we are considering. That is, to calculate the mean circularity of slice i, which we denote $\mathcal{S}_i$, we compute

$$\frac{1}{\sum_{\mathcal{P}_j} \text{Area}(\mathcal{S}_i \cap \mathcal{P}_j)} \sum_{\mathcal{P}_j} \text{Area}\left(\mathcal{S}_i \cap \mathcal{P}_j\right) \mathcal{C}(\mathcal{P}_j) \tag{4}$$

where the sum is taken over all polygons $\mathcal{P}_j$. This produces a smoothly varying mean circularity as we move along the long axis of the wing. For polygon $\mathcal{P}_i$, we obtain the area fraction f_i that overlaps with

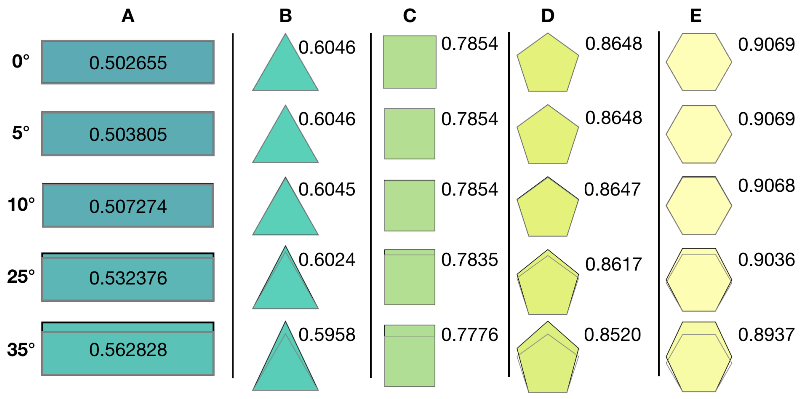


Figure S10: **Effect of corrugation on circularity.** For a series of shapes and angles, the circularity of each polygon is shown. Column **A** shows a rectangle with aspect ratio 4. Columns **B–E** show a triangle, square, pentagon, and hexagon. Even for the most extreme corrugation slopes, the error in calculated circularity is quite small.

the bin. We construct the vector of all area fractions in bin i, denoted $\vec{F_i}$. We also have the vector of all circularities $\vec{C}$ and all areas $\vec{A}$. For each bin, we compute

$$(\vec{F_i} \cdot \vec{A}, \vec{F_i} \cdot \vec{C}) \tag{5}$$

giving us the weighted mean area and the mean circularity of the wing domains in the region.

2.5 Comparing forewings and hindwings of the same species

As many workers have observed, forewings and hindwings are more similar to one another in damselflies as compared to dragonflies. In Figure S11, we show the forewing and hindwing of two example dragonflies and damselflies using P–D traces, with images of the wings themselves inset. As expected, the difference between P–D traces of damselfly forewings and hindwings is less than that of dragonflies.

2.6 Intraspecific and intra-individual wing comparisons

Within our dataset, we have 3 sets of *Sympetrum rubicundulum* hindwings. Each hind wing was imaged and segmented to allow us to look at the variation in P–D trace within an individual compared to the variation within a species. In Fig. S12, we show the left/right traces for each individual (Panel **A**) as well as the variation between the different individuals (Panels **B–D**). With only three individuals, we cannot conclude that we have effectively captured intraspecific variation, although it is a promising avenue for future work.

In Fig. S13, we show all vein domains in our dataset (over 150,000). Additionally, we show how P–D trajectories vary for the largest and smallest three damselflies and dragonflies in our dataset.

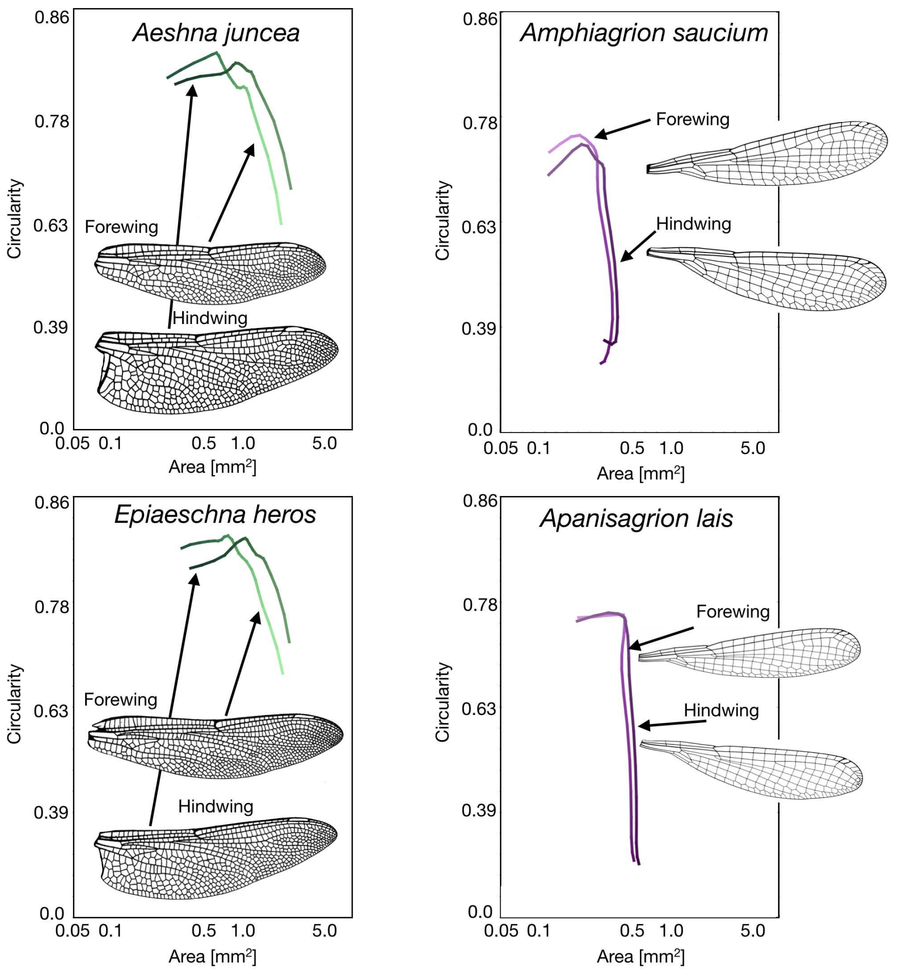


Figure S11: **Comparing forewings and hindwings of the same species.** A comparison of the forewing and hindwings of two dragonflies (on the left) and two damselflies (on the right). Reprinted with permission from Michael L. May (artist) [1]. From [2], p. 85, Figure 353; p. 119, Figure 506; p. 196, Figure 1138; p. 209, Figure 1211. (c) 2010 Johns Hopkins University Press. Reprinted with permission of Johns Hopkins University Press.

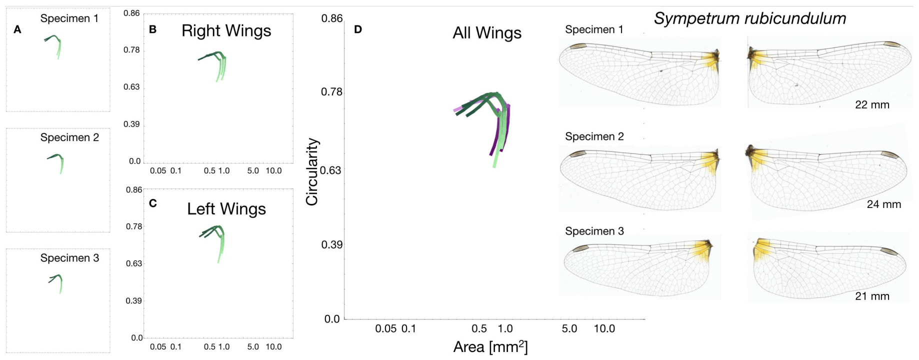


Figure S12: **Variation in P–D traces within individual/species.** Panel **A** shows the P–D traces for the left/right wings of three individuals. In Panels **B–C** just the left/right wings are shown. In Panel **D**, all wings are shown, the left wings in purple. The six wings are shown on the right.

2.7 Domain features of dragonfly in genus *Epiophlebia*

We computed domain features from a drawing of *Epiophlebia superstes*. This genus of dragonfly is of particular interest, as they have the oldest last common ancestor with the rest of the dragonflies. We plotted its P–D trace alongside distributions of traces from dragonflies and damselflies. We used an image from Tillyard [31]. Interestingly, the trace through area–circularity space of the *Epiophlebia* appears to be intermediate between the distributions of damselflies and other dragonflies, as shown in Fig. S14.

2.8 Other quantitative measures of wings

2.8.1 Interior angle distribution

Different regions of the wings are bound by a different number of primary veins. Between two veins that are relatively close together, we typically observe rectilinear domains that form a brick-like pattern. As these long, relatively straight veins diverge, domains have more freedom in forming complex angle distributions. We choose four different wings (two dragonflies and two damselflies) and randomly selected a set of domains bound by zero, one, and two primary veins. For each of these wing domains, we measured all internal angles and in Fig. S15 we show smoothed angular distributions.

2.8.2 Domain sizes in a region

In Fig. S16, we show four wings (two dragonflies and two damselflies) and how each domain's area compares to the mean area of all neighboring domains. We see that there is a very strong positive relationship.

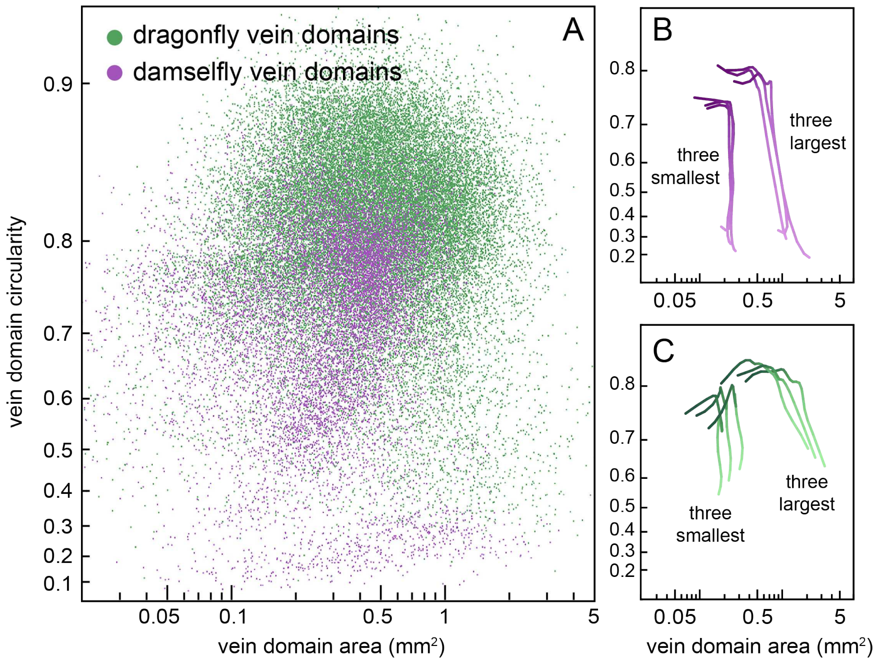


Figure S13: **Variation in domains and P–D traces.** Panel **A** shows area versus circularity for all vein domains in our dataset (n > 150,000); dragonflies in green and damselflies in purple. Panels **B, C** show P–D traces of the three largest and three smallest damselflies **B** and dragonflies **C**.

2.8.3 Unconnected vein ends

We observe very few domains that contain incomplete veins. We take a selection of wings imaged directly by the authors and count how many unconnected ends we see in the table below.

2.8.4 Comparison of the number of domains in corresponding regions in left and right wings

In Fig. S17, we count and highlight the number of wing domains in corresponding regions. Shown are the left and right forewing and hindwing from the same individual (*Erythremis simplicicolis*, shown as an example).

3 Procedure for simulating secondary vein patterns

Here we include further details about the vein simulation procedure that is described in the main text and schematized in Fig. 3. Below we have included a subsection for each step of the procedure:

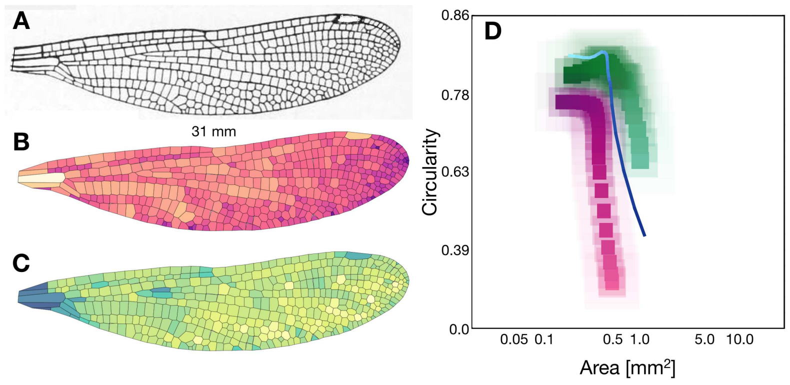


Figure S14: **Wing trace of an *Epiophlebia*.** Panel **A** shows the drawing of a *Epiophlebia superstes* from Tillyard [31]. Panels **B–C** show a segmented wing with domains colored by area and circularity. The wing is 31 mm long. Panel **D** shows the curve of the wing (in blue) juxtaposed next to the distributions of wing traces of dragonflies and damselflies.

Aeshna constricta	1
Aeshna verticalis	2
Epitheca cynosura	0
Erythemis simplicicollis	0
Somatochlora tenebrosa	1
Sympetrum rubicundulum	0
Sympetrum vicinum	0

Table 1: Number of unconnected vein ends

- Identifying primary veins for the species we will simulate
- Transforming the adult wing to the shape of its former wing pad
- Seeding inhibitory centers on the wing pad and evenly spacing them using Lloyd's algorithm
- Calculating secondary vein placements as the Voronoi tessellation of the inhibitory centers
- Transforming the wing pad back to the shape of the adult wing

3.1 Identifying primary veins for the species we will simulate

When we simulate secondary veins, primary veins are treated as fixed boundaries within which a stochastic process operates. Thus, the first step of our modeling procedure is to identify the primary veins. We manually identify primary veins, based on three sources of information: (1) overlap among left–right wing pairs, (2) morphological criteria based on published anatomical literature, and (3) conservation of relative positions for consensus vein identifications in the published taxonomic literature. The specific details of this

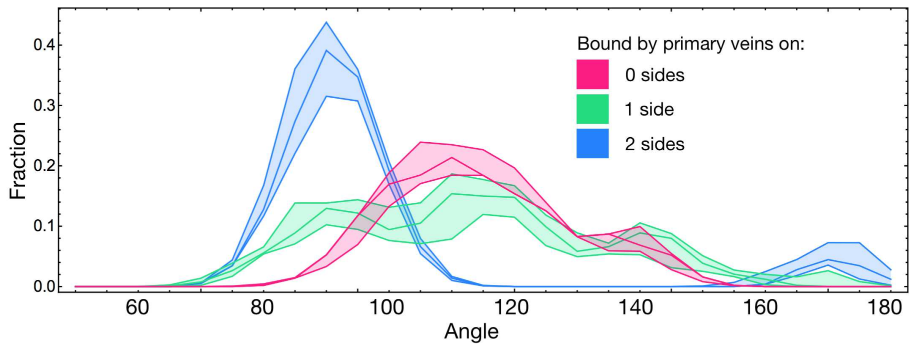


Figure S15: **Interior angle distribution of domains.** Smoothed histograms showing the interior angle distributions of domains bounded by primary veins on zero, one, or two sides (magenta, green, and blue, respectively). Shaded region represents the 1st and 3rd quantiles.

literature-based process for primary vein identification is described in the section below (Section 3.1.1). We then describe two computational alternatives that would allow one to identify primary veins without relying on published anatomical descriptions (Section 3.1.2). We show that these alternatives are not sufficiently effective for the present dataset, but we argue that they should be practical with datasets that include more example wings from a given species. Finally, we show that our technique for simulating secondary veins produces life-like vein patterns even when we vary the primary vein arrangements, demonstrating that our vein simulation results are *not* highly sensitive to primary vein identification (these results are discussed in Section 4.2.2 and shown in Fig. S34).

3.1.1 Literature-based approach for identifying primary veins

As described in the main text, we define a "primary vein" as any wing vein that has a matching counterpart on the opposite wing of the same individual and on the wings of other individuals of the same species. Put more simply: *primary veins are all the veins that are conserved at the species level.*

Thus, the set of primary veins includes long connected segments of veins that are largely collinear, as well as a few conserved cross-veins. Conversely, we define a "secondary vein" as any wing vein that does *not* have a consistently identifiable counterpart on the opposite wing of the same individual and on the wings of other individuals.[4]

[4]Naming conventions for insect wing veins has been a matter of considerable debate in the literature because assigning a label to a specific vein is sometimes an act of proposing (or supporting) a hypothesis about evolutionary homology. In this study, we use terms with no intended homology-related implications. For instance, we use "wing vein" to refer to any thickened, strut-like structure in the wing, irrespective of its developmental or evolutionary origin. For the extant taxa included in this paper, the wing veins that we examine also happen to be pigmented and opaque, but we do not consider these traits to be essential attributes of a wing vein. The terms "main vein," "longitudinal vein," "cross-vein," "transverse vein," "fixed vein," and "variable vein"—not to mention "branch," "sector," and numerous taxon-specific terms—have been employed in the literature to refer to subsets of wing veins that may differ from the subsets of veins that we refer to as "primary" and "secondary" veins. For a more comprehensive discussion of wing vein terminology, we refer readers to the many excellent treatments of the topic [3, 4, 5, 6].

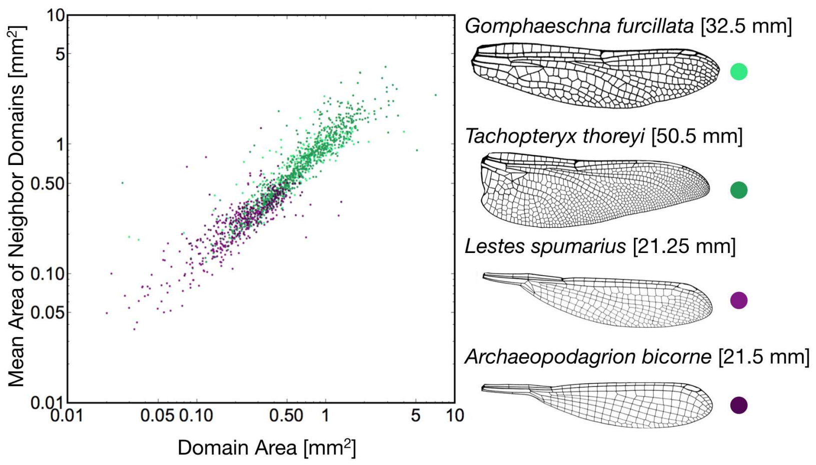


Figure S16: **Domain size compared to mean neighbor domain size.** For each domain in one of four wings, shown on the right, we compute the area of the wing domain. The *x*-axis shows the area of a given wing domain and the *y*-axis shows the mean area of all neighboring domains. Reprinted with permission from Michael L. May (artist) [1]. From [2], p. 85, Figure 353; p. 119, Figure 506; p. 196, Figure 1138; p. 209, Figure 1211. (c) 2010 Johns Hopkins University Press. Reprinted with permission of Johns Hopkins University Press.

The simplest way to identify a primary vein is to examine a number of wings from the same species and find the veins whose relative positions can be unambiguously matched across all the wings of that species. Fortunately, that is precisely what entomologists have been doing for over a century. The set of well-conserved veins has been described in many sources [1, 2, 3, 4, 5, 6, 9, 15, 16]. This work has shown that for the majority of primary veins, they are conserved across a wide diversity of species, and even at the level of taxonomic family and order. This previous research has also shown that conserved veins share many morphological features as well: they form the peaks and troughs of wing corrugation [21, 22], often take on a pattern of collinear segments [1, 3, 4, 5, 16], and bear microscopic cuticular structures that are either different from or absent in the rest of the wing [8, 15, 44, 45].

First, we overlap and align the left and right wings from the same individual; see examples in Fig. S25. Then, we manually identify vein placements that are unambiguously shared between the overlapping wings. In cases where veins are close, but not precisely overlapping, we rely on the expertise of previous entomologists to resolve ambiguities. We do so by only identifying a candidate vein as a primary vein if it has the position and/or neighbor collinearity of previously described, well-conserved veins. Finally, we connect all primary veins to form closed shapes by connecting each unconnected end to its closest neighbor.

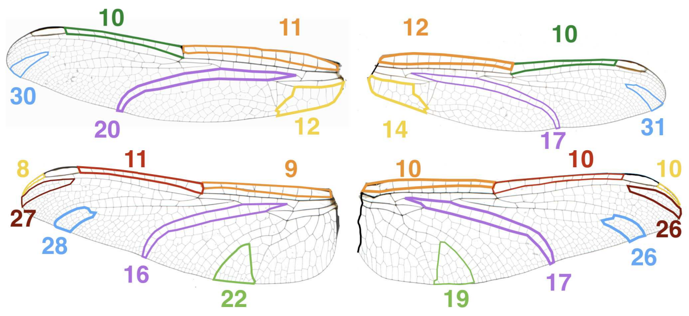


Figure S17: **Number of domains in regions from the left and right wings of a *Erythremis simplicicolis*.** For a selection of regions bounded by primary veins, we show the number vein domains in the corresponding regions of paired wings. Even in the same individual, corresponding wing regions on the left and right wings have a variable number of domains.

3.1.2 Automated approaches for identifying primary veins

Here we discuss two approaches that we have implemented to automatically detect of primary veins. The first method is premised on the fact that primary veins are generally connected to other collinear primary vein segments. Therefore, we score segments based on their collinearity with their neighbors. For this approach, we take one focal vein segment at a time, and for each of its two ends we consider each adjoining vein segment. For each of these, we compute the correlation between the focal vein segment and each of its connected neighbors. We take the maximum value of these correlations at each of the two ends, and discard the smaller values. This results in two correlations. Finally, we compute a "primary vein score" for the focal segment. This is the sum of 0.9 times the larger correlation and 0.1 times the smaller. This weighting scheme was chosen so that intersections that end in a $\perp$ do not get discarded. In Figure S18 we show the results of this procedure as applied to the left and right wings of a single individual, wherein vein segments with a "primary vein score" over 0.75 are displayed. In Figure S19 we mirror the left wing onto the right wing to assess whether the veins are shared. We find that when we use this approach on a single left–right pair of wings—but without using any additional additional information from the published literature—we identify long connected primary veins effectively. However, there are also many other scattered, short segments that meet the straightness criterion, but whose positions are rarely shared between the left and right wings. It seems likely that most of these are collinear with their neighbors due to chance alone. If one had access to a larger set of wings, by comparing across more individual wings, one could exclude non-shared segments.

Therefore, we developed an alternative automated approach for identifying primary veins based on correlations of vein segments between multiple wings. After aligning two wings of the same individual, we use a sliding window to calculate the correlation between both wings in a box centered at each pixel. The box side length is allowed to vary across the wing, so that it is always proportional to the local domain

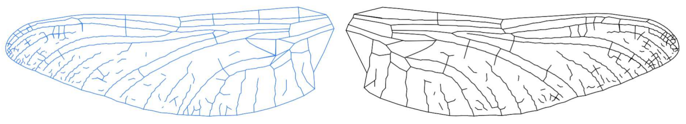


Figure S18: **Vein correlations** Each vein segment has been assigned a "primary vein score" based on its collinearity with its neighbors. Those that have a correlation over a critical value (0.75) are shown in both the left and the right wing (blue and black, respectively).

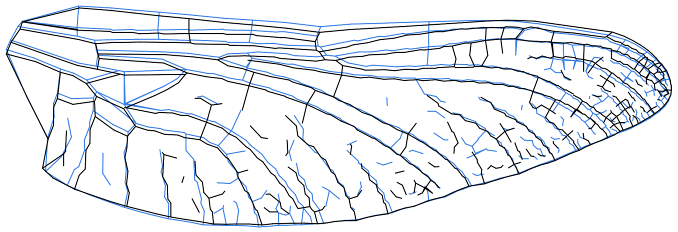


Figure S19: **Vein correlations, aligned.** We flip the left wing onto the right wing where the only veins shown are those that meet a straightness criterion.

size (square root of area). This accounts for the fact that domain size itself varies across the wing. Then, we take the maximum correlation at each pixel and multiply this resulting matrix by a binarized input vein matrix, which gives a correlation value to each vein pixel. These can also be treated as "primary vein scores," assigned at the pixel level. Preliminary results are shown in Figure S20. The result is that veins in an arrangement that is conserved over multiple wings have larger correlation values than those that are not conserved. In the present study we only have one or at most a few wings for each species, so this is not an approach we use for the simulations included in this paper.

Both of these automated approaches require multiple individuals of a given species. In the future, either of these techniques (or perhaps both in combination) could be a promising avenue for high-throughput analysis of even larger wing datasets.

3.2 Transforming the adult wing to the shape of its former wing pad

In Fig. S21, we show the result of our model without properly estimating the shape on the wing pad. Once primary veins have been identified, we estimate the shape of the wing earlier in development—that is, when it was a wing pad and the veins were only beginning to form. The geometry of the wing that we observe is not the same as the geometry of the developing wing. The vein arrangement that formed on the nascent wing pad grows and distorts as the wing itself grows [1, 3, 37].[5] We approximate the effects

[5]As described in the main text, we outline the development of wing in broad strokes by supposing that all the primary veins form first, followed by secondary veins. This is almost certainly an oversimplification, but it is suitable for our general

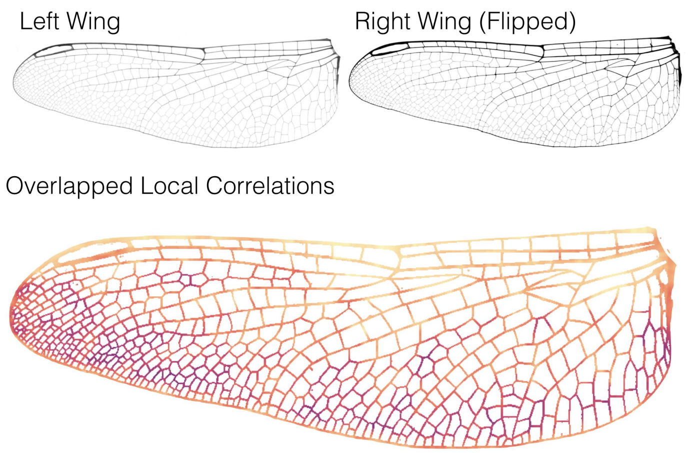


Figure S20: **Local correlation of two wings.** We flip the left wing onto the right wing and determine the local correlation of vein arrangements. Regions with higher correlations are brighter colors than those with smaller correlations.

of this process of anisotropic growth by a process of maximization of the circularity of vein domains.

3.2.1 Remapping a wing to maximize area-weighted circularity of polygonal domains

In our minimal model of vein patterning, we suppose that inhibitory centers emerge stochastically and equilibrate in space, producing a roughly even spacing. These secrete some diffusible signal. Veins form at local minima of this inhibitory signal. Based on this model, the most likely geometry of veins on the wing pad is an arrangement with maximally circular polygonal domain shapes. We consider a transformation that will map a coordinate $(x, y) \to (x', y')$ where the unprimed coordinates represent what we observe and the primed coordinates are transformed such as that the mean polygon is most circular.

Let us represent an (x, y) coordinate as $\vec{x}$. We write a transformation

$$\vec{x}' = \vec{x} + \sum_{i=0}^{N} \sum_{j=0}^{N} \vec{\alpha}_{i,j} T_i \left(\frac{x}{x_{\max}} \right) T_j \left(\frac{y}{y_{\max}} \right) \tag{6}$$

for some integer N (we use $N = 3$), where we want to find the two-vectors $\vec{\alpha}_{i,j}$ to maximize our objective function. T_i represents the i-th Chebyshev polynomial. When all $\vec{\alpha}_{i,j}$ are $\vec{0}$, we have the identity mapping. We divide by $x_{\max}$ and $y_{\max}$ in x and y to ensure that the x and y coordinates are bounded by -1 and 1.

model. An alternative is that secondary vein inhibitory centers develop after their adjacent primary veins, but before the primary veins have completed their development. We show in the main text that our model can recapitulate secondary vein patterns in Odonata, Orthoptera, and Neuroptera. This suggests that although these distantly related taxa have differing vein ontogenies, they share essential features that are consistent with the assumptions of our model. See [3, 37] for a more detailed discussion of the development of diverse insect wing veins.

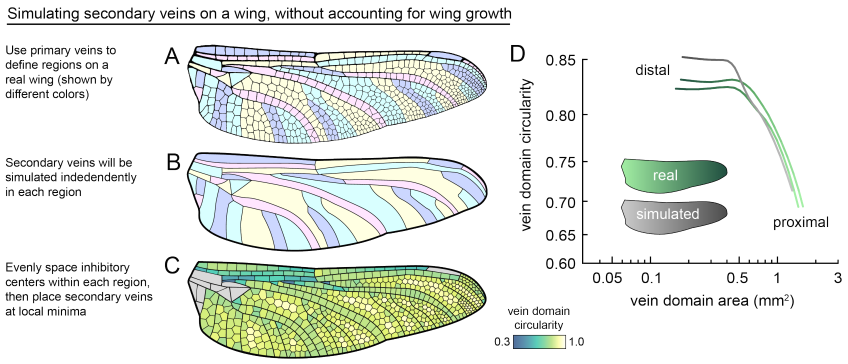


Figure S21: **A model for simulating secondary vein patterning.** Panels **A-C** show a process for simulating secondary veins without taking tissue growth into account (*D. spinosus* shown as an example). Gray domains in **C** are bounded on all sides by primary wing veins; these are not simulated in the model. Panel **D** shows P–D morphology traces for the simulated wing and a left-right pair of real wings.

To compute the $\vec{\alpha}_{i,j}$, we want to maximize the area-weighted circularity, where we weight the circularity of each domain by its original area. However, in order to keep the resulting wing pad shapes consistent with examples in the biological literature (see references 34, 36, 38, 39, 40, 41, and 42 in the main text for examples), we impose the constraint that the resulting shape be convex. Since the wings are generally long and thin, we found that maximizing area-weighted circularity alone would lead to a poorly constrained degree of freedom corresponding to a bending mode along the proximal-to-distal axis. Requiring that the shapes be convex is therefore a biologically reasonable way to fully constrain the maximization problem. To penalize shapes that are not convex, we multiply the circularity term by a factor γ that we define to be

$$\gamma = \frac{\sum_i \text{Area}_{\text{transformed}}(\mathcal{P}_i)}{\text{Area}(\text{Convex Hull})}. \tag{7}$$

In the above equation we define the convex hull to be that of the entire wing. In Figure S22, we investigate how different methods of including the γ term affect the resulting shape. When the term is omitted, the bending mode along the P–D axis is visible. We choose to use a linear term, but find that the specific exponent of γ has negligible effect on the wing pad shape that results. For our optimization routine, we want to find

$$\max \gamma \sum_i \left(\text{Area}_{\text{orig}}(\mathcal{P}_i)\text{Circ}_{\text{trans}}(\mathcal{P}_i)\right). \tag{8}$$

In the expression above, $\text{Area}_{\text{orig}}(\mathcal{P}_i)$ represents the original area of polygon $\mathcal{P}_i$ and $\text{Circ}_{\text{trans}}(\mathcal{P}_i)$ represents the circularity of the transformed polygon $\mathcal{P}_i$. While we can no longer easily differentiate this quantity, by artificially perturbing each component of the $\vec{\alpha}_{i,j}$ we are able to estimate the derivative. We use our numerically computed gradients for the Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimization method [29].

γ	Area divided by that of convex hull
$\text{Area}_{\text{orig}}(\mathcal{P}_i)$	Original Area of Polygon i
$\text{Circ}_{\text{trans}}(\mathcal{P}_i)$	Circularity of Transformed Polygon i

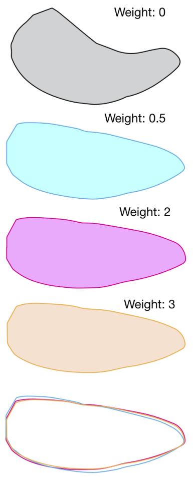


Figure S22: **The effect of gamma.** We define γ to be the ratio of the total area of the wing domains to the convex hull enclosing the wing domains. In our optimization routine we incorporate a multiplicative factor of γ^1 into our objective function. Without a γ (or equivalently, with γ^0), we get shapes that are not biologically realistic. In this figure we show γ^0 on top, $\gamma^{0.5}$ second from the top, γ^2 in the middle, and γ^3 second from the bottom. On the bottom, we overlay the final three shapes. We find they are all quite similar and the choice of exponent is not crucial for our analysis.

3.2.2 Comparing circularity maximization to biological data

In order to assess the accuracy of our circularity maximization routine for estimated wing pad shape, we use an example published wing pad for hindwing of a developing *Anax junius* [1]. A tracing of the wing pad is shown in Figure S23. Many primary veins are visible on the pad, with the identities of specific wing morphologies indicated by the author. We begin by tracing primary veins that are identifiable on both the wing pad and the adult wing. Next, we randomly select 15,000 pixels that belong to veins from both the observed wing and the image of the wing on the wing pad. To compute the mapping, we write a transformation of the form in Equation 6. We denote points within the veins of the wing pad as p_i and points in the (transformed) observed wing as q_j. We can then write the cost function as

$$\text{cost} = \sum_i \min_j(d(p_i, q_j)) + \sum_j \min_i(d(q_j, p_i)). \tag{9}$$

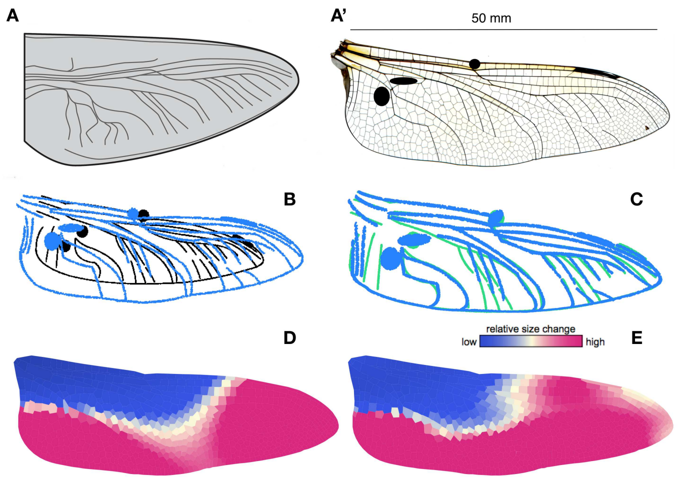


Figure S23: **Comparing circularity maximization with landmark-based growth of the wing pad.** Panel **A** shows the developing wing pad of (*Anax junius*), as adapted from [1]. Panel **A'** shows a wing image of an adult from the same species. The adult wing is 50 mm long. Corresponding vein regions are marked in black. Panel **B** shows the wing veins we marked in **A'** in blue along with landmarks identified from the wing pad in black. Panel **C** shows the result after using the landmarks to map the wing pad onto the adult wing. The transformed wing pad landmarks are shown in green. Panel **D** shows the relative local growth in the wing pad, mapped onto the adult wing, as estimated via the circularity maximization routine described in the text. The colormap represents the estimated relative local growth of the wing. Blue domains are estimated to grow the least, while magenta grows the most. Panel **E** shows the relative local growth of the wing, as estimated from landmarks on the wing pad and the adult wing. Scale bar in Panel **A'** applies to Panel **D** and **E**. Panels **B** and **C** are unscaled. The original source for Panel **A** [1] did not include scale information; given that we are estimating *relative* growth, the absolute scale is not needed.

We include two symmetric terms such that we minimize the distance between points from the transformed wing to the wing pad and from the wing pad to the transformed wing (this ensures that all points are not collapsed in space). The procedure and its results are outlined in Fig. S23, Panels A–C.

We assess the accuracy of our circularity maximization routine for estimating the wing pad by comparing the wing-to-pad transformation for the estimated wing to the transformation for the true wing. For each wing domain, we compute the fractional area before and after the deformation. The results are shown in Fig. S23 Panel D. On the whole, we found that our circularity maximization routine broadly captures patterns of anisotropic growth of the wing. Specifically, we find that base of the wing grows relatively less that the rest of the wing during development, while other parts of the wing get relatively larger; this pattern is recapitulated with the circularity maximization routine.

3.3 Seeding inhibitory centers on the wing pad and evenly spacing them using Lloyd's algorithm

To equilibrate seed points in space, we use an algorithm known as Lloyd's algorithm or Voronoi Iteration [30]. A Voronoi cell is defined to be the region around a seed point that is closer to the seed than to any other seed. Lloyd's algorithm proceeds by iteratively moving all seeds to the centroid of their Voronoi cell.

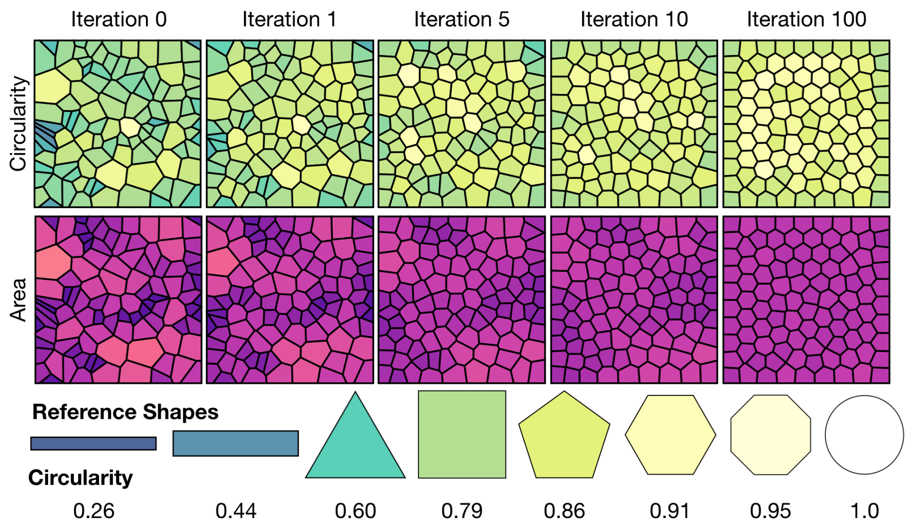


Figure S24: **Iterations of Lloyd's algorithm.** The random initial configuration (iteration 0), and iterations 1, 5, 10 and 100 of Lloyd's algorithm. At each iteration, domains are colored by circularity (top row) and size (middle row). The bottom row shows various reference shapes colored by their circularity.

Figure S24 shows several iterations of Lloyd's algorithm. A random collection of points equilibrate into an equally spaced lattice. Note that the geometry of domains around the boundary are different from those in the interior.

In the transformed wing, we drop a number of seed points into each primary vein bound region. The number of seeds is either determined empirically from real wings (main text, Figs. 3–4) or estimated from the local thickness of primary veins (main text, Fig. 5); we discuss this further in the next section. We find that the initial spatial distribution of seeds in each region does not matter at all. After iterating their position through Lloyd's algorithm, they equilibrate to indistinguishable distributions. The number of required iterations varied depending on wing size and number of domains. We terminated Lloyd's algorithm once the mean change in seed positions dropped below 0.1% of the length of the wing.[6] After updating the locations of all points, we remap the distorted wing back to the untransformed wing geometry.

[6]Typically, we ran between 100 and 1000 iterations to reach the endpoint.

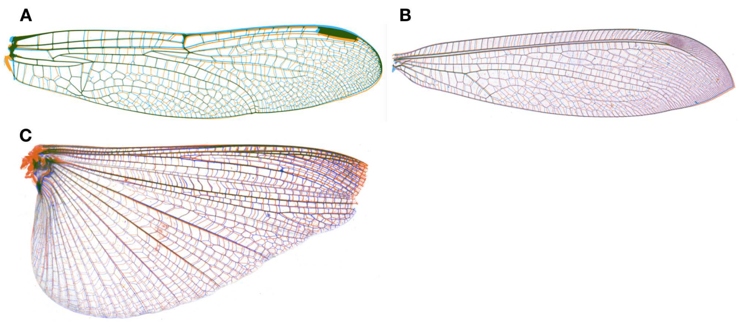


Figure S25: **Overlaying left/right wings of the same individual.** Panel **A**: Dragonfly, Panel **B**: Neuroptera, Panel **C**: Orthoptera.

3.3.1 Determining the number seed points in each region

As described in the main text, we use two approaches to determine the number of inhibitory centers that we seed into regions bound by primary veins: (1) we use the empirical count of domains in each region in the true wing (shown in main text Fig. 3); (2) we use information based on the thickness of surrounding primary veins (shown in main text Fig. 5). The latter approach is described in the main text and in further detail in this section.

Near larger domains, primary veins are thicker; near smaller domains, primary veins are thinner (see main text Fig. 5, and additional information below in Section 3.3.2). We hypothesized that the primary veins on the wing pad are the source of a morphogen whose concentration determines the local length scale of the pattern generator. If so, one might expect that the thickness of the primary veins would be positively related to the strength of the secreted signal. There is little data available on the morphology of primary veins on the wing pad, but in one example case, we find that the thicknesses of primary veins on the wing pad are correlated with their thicknesses on the adult wing (Fig. S26). Therefore, we use adult vein thickness as a proxy for wing pad vein thickness hereafter.

For three different species for which we had original wing micrographs, we used `Fiji` to manually measure the primary vein thicknesses in pixels (the measured primary veins from an example wing are shown in Fig. S27 Panel A) across the wing. Then, for each wing domain centroid we compute the distance to each primary vein. We take a weighted average of primary vein thickness, where we weight each thickness by

$$\text{weight} = \exp\left(-\frac{r}{0.515663\text{ mm}}\right) \tag{10}$$

where r is the distance from the centroid of the domain in consideration to the closest point on the primary vein. We also introduce a scale factor to convert the pixel value to a width in mm. This results in a locally weighted "primary vein thickness" (in mm) for each domain (Fig. S27 Panel B). We emphasize that the exponential decay function in Eq. (10) is a modeling assumption, and that other functional

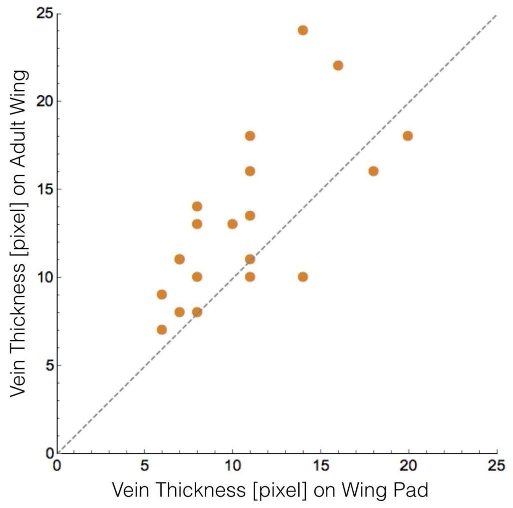


Figure S26: **Vein thickness correlates between wing pad and adult wing.** Various non ambiguous vein thicknesses were manually measured in `ImageJ/Fiji` on the wing pad and on the adult wing. Pixel values were rescaled so the wings were the same size. The correlation coefficient is 0.7246.

forms, such as a Gaussian decay, $\exp(-r^2/L^2)$ could be used. We suggest that an exponential decay is a reasonable choice, since it is the steady-state solution for a signal that is diffusing from a vein and decaying with a constant rate. We expect that our results are insensitive to the precise choice of this decay, but this could be investigated further in future work. The pooled data from three example species is shown in Fig. S27 Panel C.

We use the empirical relationship between primary vein thickness and domain size (Fig. S27 Panel C) to determine the number of inhibitory seeds in each wing region for a species that was *not* included in our initial dataset of primary vein thicknesses. We measure the primary vein thicknesses of the new wing, and then randomly draw domains from the thickness distribution shown in Fig. S27 Panel C until the cumulative domain area exceeds the true area of the region. This results in a simulated wing venation pattern whose circularity-area P–D trace is shown in gray in Fig. S27 Panel D, compared to the true left and right wing traces shown in green.

We also evaluated the sensitivity of our simulation results to the number of inhibitory seeds, finding that varying the number of seeds in a region alters the local domain geometries (Fig. S27 Panel D). With too many seeds, domains are both smaller and more circular than the true wing. With too few seeds, domains tend to be larger and more rectangular. To assess this sensitivity, and the relative success of our model, we perform simulations where we seed each region with 0.5, 0.75, 1.25, and 1.5 as many seed points as are found in the real wing (Fig. S27 Panel D).

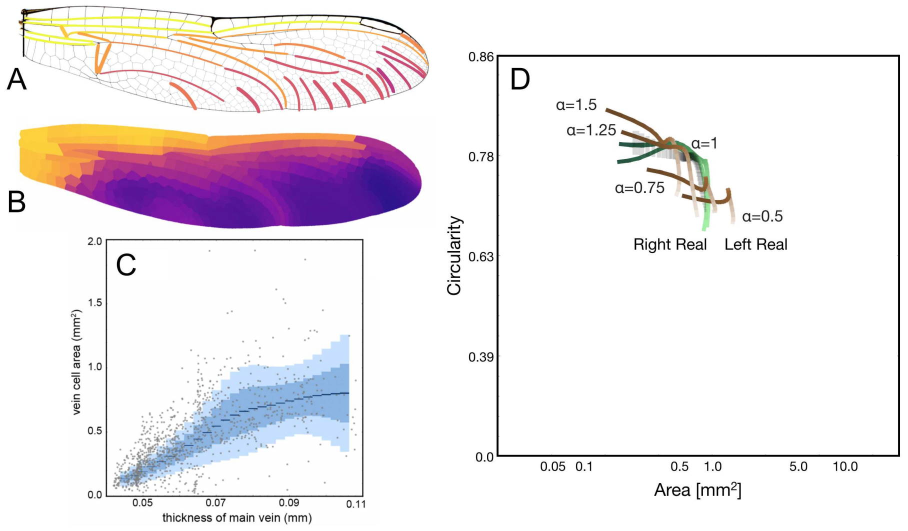


Figure S27: **Primary vein thickness correlates with nearby domain sizes.** Panel **A** shows the primary vein segments used colored by their thickness. Panel **B** shows weighted local primary vein thickness (in mm) for each domain. In Panel **C** we plot vein thickness against domain area for all domains. Panel **D** shows the P–D trace distribution of simulations of a new wing, with seed number determined solely by measuring primary vein thickness and drawing from the distribution of thicknesses and domain areas **C**. The results are shown in gray. The two green curves represent the true left and right wings of the simulated species. We also assessed the sensitivity of the simulation to inhibitory seed number by purposefully overseeding/underseeding regions by scale factors ranging from 0.5 to 1.5. Wing traces from those simulations are shown in brown.

3.3.2 Scaling of vein thickness and domain size across species

As described above, there is a positive relationship between the size of a domain and the thickness of nearby primary veins. To see the extent of this relationship across species, we use a selection of wing images and follow the procedure described in Section 3.3.1. Results are shown in Fig. S28. We find that subtle differences in illumination can affect the measured vein thickness in a wing-specific manner. Therefore we shift all profiles towards the origin by subtracting the smallest vein thickness among all measured veins on that wing. We then fit the relationship between domain size and vein thickness in all wings using an equation of the form domain size $= a(\text{vein thickness})^b$. We report a 90% confidence interval on a and b for each fit.

3.4 Calculating secondary vein placements as the Voronoi tessellation of the inhibitory centers

As described above, a Voronoi tessellation is defined to be the region around a seed point that is closer to the seed point than to any other point. Many features of odonate wing domains are qualitatively similar

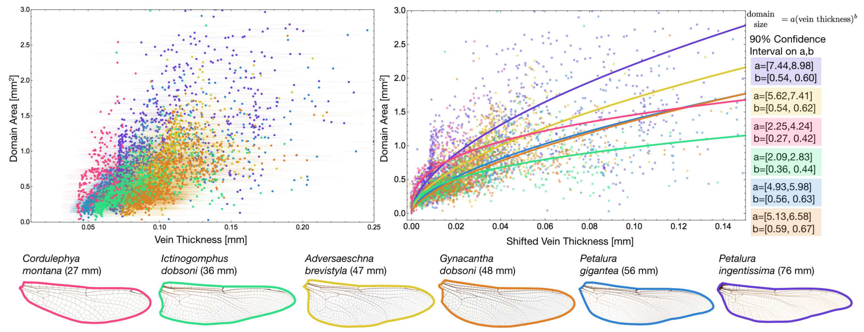


Figure S28: **Scaling of vein thickness with domain size.** Left panel shows the scaling of vein thickness and domain size for six wings (outlined in corresponding colors underneath). Error bars represent a measurement error of one pixel. To minimize the effect of illumination on measured thickness, we shift all points towards the origin before fitting (right panel). Fits are of the form $a(\text{vein thickness})^b$. 90% confidence intervals of the parameters are given. Wing images courtesy of Wikimedia Commons/John Tann [23, 24, 25, 26, 27, 28].

to shapes that can result from a Voronoi tessellation around equilibrated points. For example, domains in large regions are mostly hexagonal or pentagonal shapes while those in more confined regions tend to be more rectilinear shapes (see the angle distribution in Fig. S15), and cells tend to be similar sizes to their neighbors (see Fig. S16). In Fig. S29, we show various shapes that result from a Voronoi tessellation of evenly spaced seeds.

Our simplified model of secondary vein development is premised on the idea that an inhibitory signal emanates from seeds whose positions are equilibrated in space. In simulating secondary vein geometries on the wing pad, we simply take the equilibrated seed locations in each region and calculate the Voronoi tessellation to simulate the secondary veins.

3.5 Transforming the wing pad back to the shape of the adult wing

Once the secondary veins have been calculated on the estimated wing pad, we reverse the transformation described in Section 3.2 to recover the shape of the adult wing. As described above in Section 3.2.2, we find that our procedure for estimating the growth of the wing pad effectively recapitulates an example wing pad from the literature.

4 Simulation results

4.1 Modeling a damselfly

In the main text we demonstrate simulated venation for a dragonfly, grasshopper, and lacewing. Here, we also apply the same code to model a damselfly, *Hypolestes clara*. Results are shown in Fig. S30.

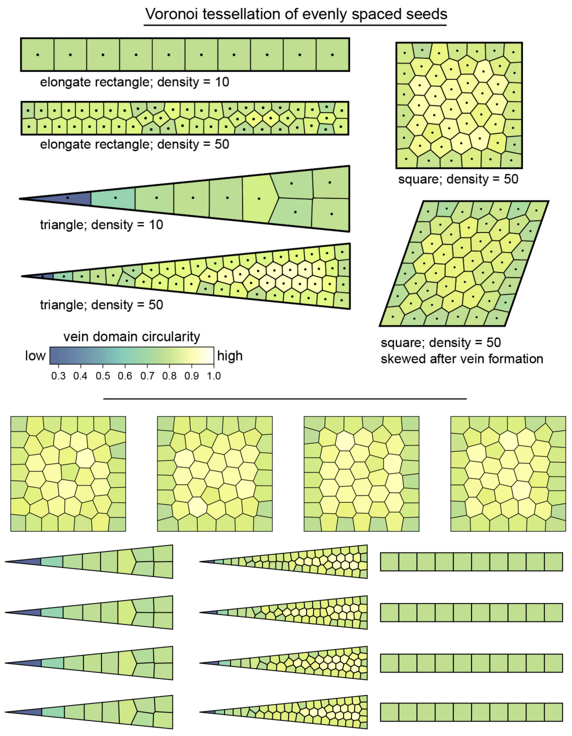


Figure S29: **Example Voronoi Shapes.** In various geometries and densities, the Voronoi cells of equilibrated points are shown, colored by circularity. For the case of a square, we show how circularities transform under a shear. The bottom panel shows near-equilibrated distributions for four runs of different biologically reasonable geometries and densities.

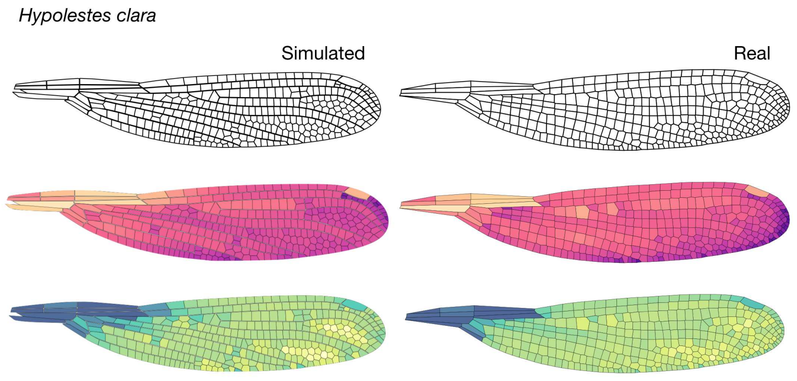


Figure S30: **Modeling *Hypolestes clara*.** The model applied to the wing of *Hypolestes clara*.

4.2 Accuracy of vein simulations

4.2.1 Absolute and relative error of vein simulations

In Fig. S32, we compute the relative error in circularity and area between the real vein pattern and our simulated vein pattern for an example dragonfly and damselfly. In Fig. S32, we show the relative error. To calculate these error values, we begin by finding the length of the longest axis, L. We also subdivide each polygonal domain into many small triangles. Each triangle vertex is assigned the area and circularity of the polygon it comes from. We then smooth an area map and a circularity map over the entire wing, where we use a weight proportional to $\exp(-d_{ij}/(0.025L))$ where d_{ij} represents the distance between vertices i and j. Consistent with the results from our P–D wing traces, we see that we underestimate the size of wing domains near the trailing edge of the wing in both dragonflies and damselflies. As a result, we slightly overestimate the area near the leading edge of the wing. It is worth noting that our largest error in the dragonfly comes from a mismatch for just the largest single wing domain in the dragonfly at the base of the wing.

We also assess the error of the simulation in each of the regions bound by primary veins. In Fig. S33 we show the relative error in each region between a simulated dragonfly and damselfly. Domains that are bound by primary veins on all sides are shown in gray.

4.2.2 Sensitivity of primary vein selection on simulated vein patterns

There is broad consensus on which veins are conserved (that is, which ones are primary veins), but there are probably a few of cases in which the published literature includes errors in identifying veins—e.g. secondary veins erroneously identified as primary veins—and vice versa. It is also possible that a simple dichotomy of primary and secondary veins does not fully describe all veins on all species. Therefore, we do not assume that our vein categorizations are perfect. In order to assess the sensitivity of our simulated

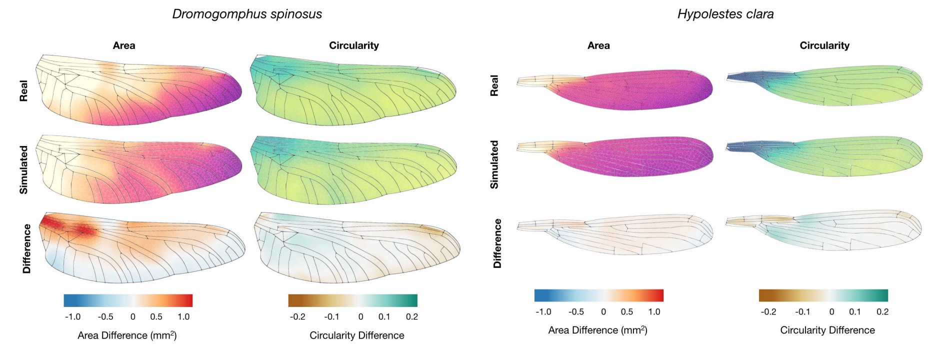


Figure S31: **Absolute error of the simulation.** Smoothed area and circularity maps for a real wing (top row) and model (middle row). The difference between these are shown in the bottom row.

results to our identification of primary veins, we take an example wing and simulated secondary veins under different sets of primary veins.

With each set of primary veins, we use them as region boundaries for our simulation approach as described in Section 3. Here, we assess several slightly different sets of primary vein identifications. In one case, we use just the primary veins named in [1] (Fig. S34, top row). The simulation captures most of wing effectively. In the middle part of the wing, however, simulated domains are slightly more circular than in the real wing.

In second case, we adapt the findings of Appel and Gorb (2014) [15]. Specifically, they show that in many species, long, straight veins also have microscopic cuticular structures that are shared in common with the conserved veins defined by Needham [1, 15]. Under the assumption that shared cuticular morphology indicates shared primary vein identity, we increase the set of primary veins using Appel and Gorb's findings as a guide. In doing so, we improve the performance of the vein simulation (Fig. S34, second row).

Lastly, we show that we can randomly perturb the identifications of primary veins without severely influencing the results of the model. As described above, our approach for simulating secondary veins involves equilibrating the positions of inhibitory seeds in closed regions. Thus, after identifying primary veins, we connect unconnected ends in order to close all regions. We do so by joining each unconnected end to the closest primary vein neighbor (Section 3.1.1). In the bottom third and fourth rows of Fig. S34, we remove some primary veins and alter how we connect unconnected vein ends. This shows that the specifics of these choices do not greatly influence our results.

5 Availability of data and code

Integer `csv` files of segmented images of all data along with code used in the manuscript are freely available. All code and data can be found at https://github.com/hoffmannjordan/insect-wing-venation-patterns.

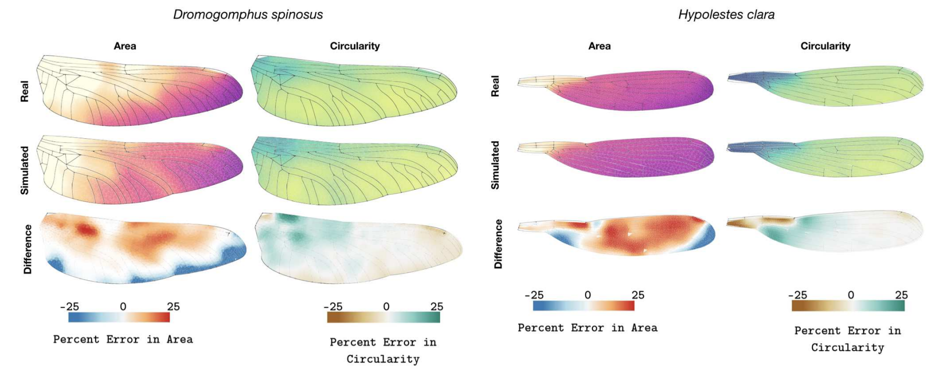


Figure S32: **Relative error of the simulation.** Smoothed area and circularity maps for a real wing (top row) and model (middle row). The difference between these are shown in the bottom row, where percent error is plotted.

5.1 Dragonfly wing data

In Fig. S35, we show segmented and polygonized plots of every dragonfly used in the manuscript colored by size and circularity.

A high quality version of this image is available at: `http://seas.harvard.edu/~chr/wing_data/Dragonfly_Wing_Aug_14.png` [157.8 MB]

5.2 Damselfly wing data

In Fig. S36, we show segmented and polygonized plots of every damselfly used in the manuscript colored by size and circularity.

A high quality version of this image is available at: `http://seas.harvard.edu/~chr/wing_data/Damselfly_Wing_Aug_14.png` [28.4 MB]

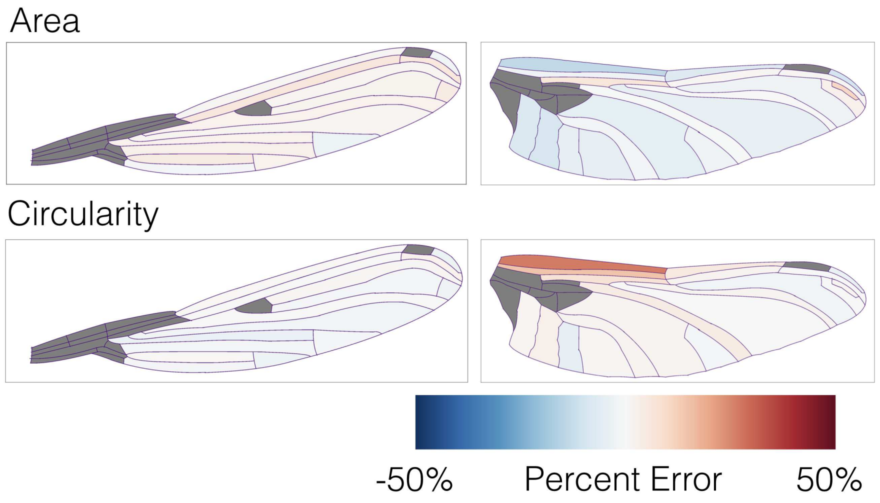


Figure S33: **Error of the model by region.** For each region bound by primary veins of a damselfly and dragonfly, we have plotted the relative error.

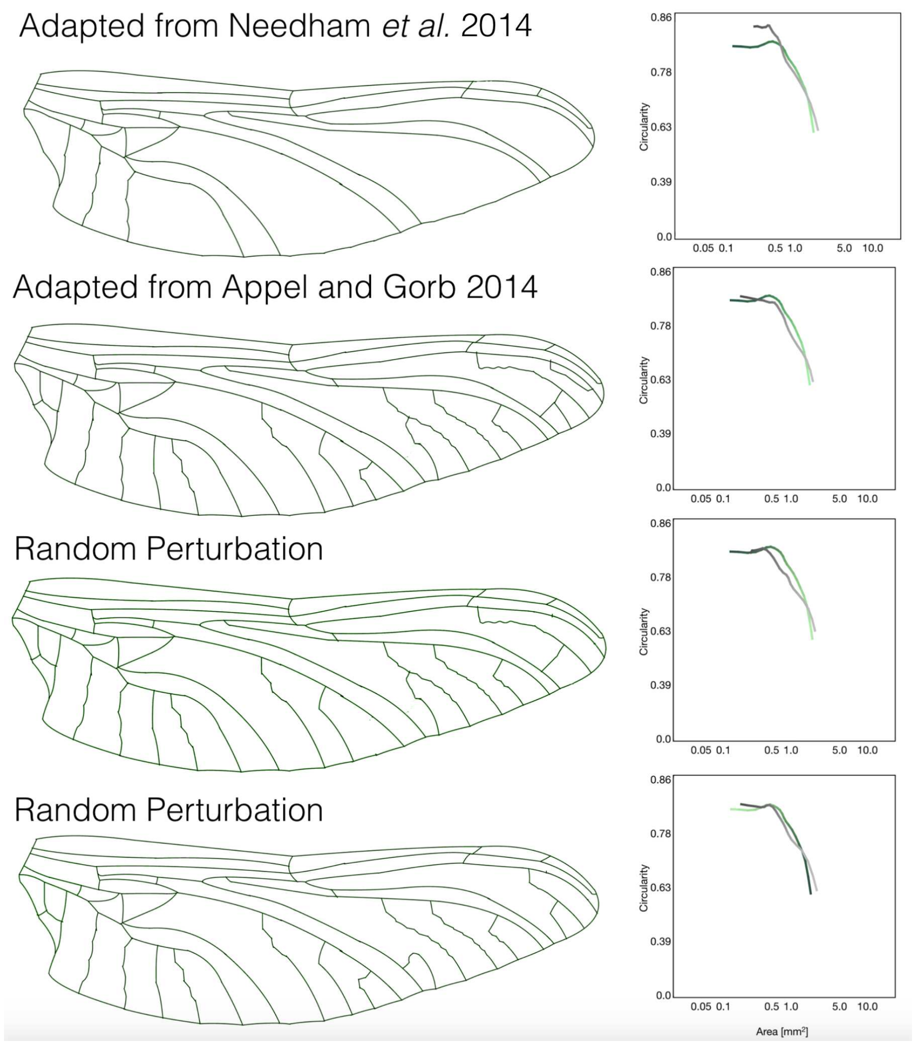


Figure S34: **Effect of region boundaries on trace.** Four different region boundaries and their corresponding P–D wing traces. The wing is an *Arigomphus villosipes*. The wing is 32.5 mm long. In the top row, we identify primary veins as those indicated as broadly conserved in Needham 2014 [1]. In the second row, we show veins adapted from Appel and Gorb 2014 [15] where they assessed veins for a variety of cuticular structures and found a subset of veins that have common features. In the bottom two rows, we begin with the set of primary veins shown in the second row, alter some primary veins, and then randomly alter the connections formed in linking the unconnected vein ends. The simulated P–D wing traces are shown in grey; real wing traces are shown in green.

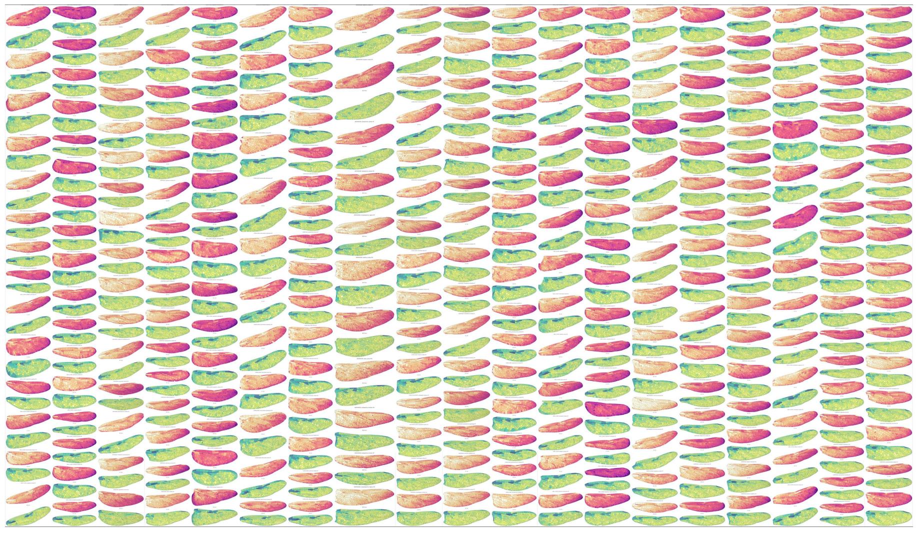


Figure S35: All dragonflies used in the manuscript, colored by size and circularity.

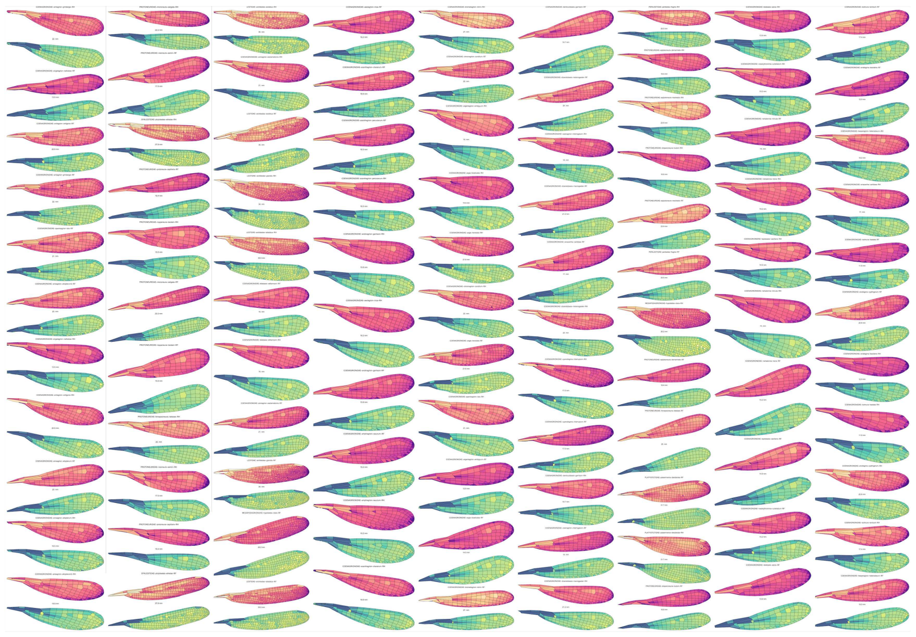


Figure S36: All damselflies used in the manuscript, colored by size and circularity.

References

[1] Needham, J. G., Westfall, M. J., and May, M. L. *Dragonflies of North America: the Odonata (Anisoptera) fauna of Canada, the continental United States, northern Mexico and the Greater Antilles*. Third edition, Gainesville, FL: Scientific Publishers. 2014.

[2] Garrison, R.W., von Ellenrieder, N., and Louton, J. A. *Dragonfly genera of the New World : an illustrated and annotated key to the Anisoptera.* Baltimore, MD: JHU Press. 2006.

[3] Kukalova-Peck, R. *Origin and Evolution of Insect Wings and Their Relation to Metamorphosis, as Documented by the Fossil Record*. J. Morph. 156, 53–126 (1978).

[4] Wootton, R. *Function, homology and terminology in insect wings*. Systematic Entomology 4, 81–93 (1979).

[5] Carpenter, F. M. *The Lower Permian insects of Kansas. Part 11*. Psyche, Cambridge, 73, 46–88 (1966).

[6] Béthoux, O. *Wing venation pattern of Plecoptera (Insecta: Neoptera)*. Illiesia 1(9), 52–81 (2005).

[7] Westfall, M. J. , May, M. L *Damselflies of North America*. Third edition, Gainesville, FL: Scientific Publishers. 1996.

[8] Donoughe, S., Crall, J. D., Merz, R. A., and Combes, S. A. *Resilin in Dragonfly and Damselfly Wings and Its Implications for Wing Flexibility.* Journal of Morphology 272, 1409–1421 (2011).

[9] Garrison, R.W., von Ellenrieder, N., and Louton, J. A. *Damselfly genera of the New World : an illustrated and annotated key to the Zygoptera*. Baltimore, MD: JHU Press. 2010.

[10] Sommer, C., Strähle, C., Köthe, U., and Hamprecht, F. A. *ilastik: Interactive Learning and Segmentation Toolkit*. Eighth IEEE International Symposium on Biomedical Imaging (ISBI). Proceedings, 230–233 (2011).

[11] Cilla, R., Mechery, V., Hernandez de Madrid, B., Del Signore, S., Dotu, I., and Hatini, V. *Segmentation and Tracking of Adherens Junctions in 3D for the Analysis of Epithelial Tissue Morphogenesis*. PLoS Comput. Biol. 11(4), e1004124 (2015).

[12] Heller, D., Hoppe, A., Restrepo, S., Gatti, L., Tournier, A. L., Tapon, N., Basler, K., and Mao, Y. *EpiTools: An Open-Source Image Analysis Toolkit for Quantifying Epithelial Growth Dynamics*. Development Cell 36, 103–116 (2016).

[13] Etournay, R., Merkel, M., Popović, M., Brandl, H., Dye, N. A., Aigouy, B., Salbreux, G., Eaton, S., and Jülicher, F. *TissueMiner: A multiscale analysis toolkit to quantify how cellular processes create tissue dynamics.* eLife 5, e14334 (2016).

[14] Lamprecht, M. R., Sabatini, D. .M., Carpenter, A. E. *CellProfiler: free, versatile software for automated biological image analysis*. BioTechniques 42, 71–75 (2007).

[15] Appel, E. and Gorb, S. N. *Comparative functional morphology of vein joints in Odonata.* Zoologica. 159, (2014).

[16] Hamilton, K. G. Andrew *The Insect Wing, Part III. Venation of the Orders.* Kansas Entomological Society. Vol. 45 No. 2, (1972).

[17] Osher, S. and Sethian, J. A. *Fronts Propagating with Curvature Dependent Speed: algorithms Based on Hamilton–Jacobi Formulations.* J. Comput. Phys. 79, 12–49 (1988).

[18] Chopp, D. *Some Improvements of the Fast Marching Method.* SIAM J. Sci. Comput. 23(1), 230–244 (2006).

[19] https://github.com/scikit-fmm/scikit-fmm

[20] Encyclopedia of Life. Available from http://www.eol.org. Accessed August 2017.

[21] Kesel, A. B. *Aerodynamic Characteristics of Dragonfly Wing Sections Compared With Technical Aerofoils.* The Journal of Experimental Biology 203, 3125–3135 (2000).

[22] Jongerius, S. R., and Lentink, D. *Structural Analysis of a Dragonfly Wing.* Experimental Mechanics 50, 1323–1334 (2010).

[23] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Petalura_gigantea#/media/File:Petalura_gigantea_female_wings_(34921024531).jpg Accessed Sept. 18, 2017.

[24] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Petalura_ingentissima#/media/File:Petalura_ingentissima_female_wings_(34665011080).jpg Accessed Sept. 18, 2017.

[25] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Ictinogomphus_dobsoni#/media/File:Ictinogomphus_dobsoni_male_wings_(34928349381).jpg Accessed Sept. 18, 2017.

[26] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Blue-spotted_hawker#/media/File:Adversaeschna_brevistyla_female_wings_(34888576182).jpg Accessed Sept. 18, 2017.

[27] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Gynacantha_dobsoni#/media/File:Gynacantha_dobsoni_male_wings_(35019600076).jpg Accessed Sept. 18, 2017.

[28] Wikimedia Commons, John Tann, CC BY 2.0 https://en.wikipedia.org/wiki/Cordulephya_montana#/media/File:Cordulephya_montana_female_wings_(34927994381).jpg Accessed Sept. 18, 2017.

[29] William H. Press, Saul Teukolsky, William T. Vetterling, and Brian P. Flannery *Numerical Recipes: The Art of Scientific Computing* Third Edition (2007), 1256 pp. Cambridge University Press.

[30] Lloyd, S. P. *Least squares quantization in PCM.* IEEE Transactions on Information Theory 28(2), 129–137 (1982).

[31] Tillyard, R. J. *Fauna of British India. Odonata. Volume 2: On an Anisozygopterous Larva from the Himalayas (Order Odonata).* Records of the Indian, 93–107 (1921).

[32] http://matteich.photoshelter.com/image/I00004Z1IRt8Yx.Q

[33] Simone, R. P. and DiNardo, S. *Actomyosin contractility and Discs large contribute to junctional conversion in guiding cell alignment within the Drosophila embryonic epithelium.* Development, 1385–94 (2010).

[34] https://commons.wikimedia.org/wiki/File:Osborn_1912_trachodon_scale_pattern.png

[35] http://www.publicdomainpictures.net/pictures/130000/velka/giraffe-skin-pattern-14425724840w4.jpg

[36] http://4.bp.blogspot.com/_RqNCCCrU4dw/TK2LmG2wchI/AAAAAAAAAMY/px1VwYGsThU/s1600/uarnak.jpg

[37] De Celis, José F. and Diaz-Benjumea, Fernando J. *Developmental basis for vein pattern variations in insect wings* The International Journal of Developmental Biology, 653–664 (2003).

[38] Blair, Seth S. *Wing vein patterning in Drosophila and the analysis of intercellular signaling* Annual Review of Cell and Developmental Biology, 293–319 (2007).

[39] Combes, S A and Daniel, T. L. *Flexural stiffness in insect wings I. Scaling and the influence of wing venation* The Journal of experimental biology, vol 206. No 17. (2003).

[40] Combes, S. and Daniel, T. *Flexural stiffness in insect wings II. Spatial distribution and dynamic wing bending* The Journal of experimental biology, (2003).

[41] Stark, J. and Bonacum, J. and Remsen, J. and DeSalle, R. *The evolution and development of dipteran wing veins: a systematic approach* Annual review of Entomology, vol 44. (1999).

[42] Lehmann, Fritz-Olaf and Gorb, Stanislav and Nasir, Nazri and Schützner, Peter *Elastic deformation and energy loss of flapping fly wings* The Journal of experimental biology, vol 214. (2011).

[43] Mountcastle, Andrew M. and Combes, Stacey A. *Biomechanical strategies for mitigating collision damage in insect wings: structural design versus embedded elastic materials* The Journal of Experimental Biology, vol 217. (2014).

[44] Gorb, Stanislav N. *Serial elastic elements in the damselfly wing: mobile vein joints contain resilin* Naturwissenschaften, vol 86. (1999).

[45] Appel, Esther and Gorb, Stanislav N. *Resilin-bearing wing vein joints in the dragonfly Epiophlebia superstes* Bioinspiration & Biomimetics, vol 6. (2011).

2.3.2 Mathematical Supplement to Salcedo, *et. al.* 2019. Biology Open

Size, shape and structure of insect wings
Supplemental Materials
Mary K. Salcedo, Jordan Hoffmann, Seth Donoughe, L. Mahadevan

Contents

Overview of this document

Here we provide an overview of the data used in the manuscript. We establish the reliability of the collected data, which comes from a wide variety of sources. We then briefly outline the mathematical and computational tools used in the manuscript, all of which are made freely available. Lastly, we provide the algorithms and calculations used in the manuscript.

1 Data collection, image processing, and segmentation

The data used in this manuscript comes from a large number of different sources. A few individuals are from original micrographs, however the majority of the data is sourced from previously published sources. While we sampled broadly, our data was limited to those insect wings with low pigmentation, or accurate drawings. Wing images from dense entomological texts can have have beautifully accurate wing drawings, but not always the morphological information (i.e. size). Since we normalized for wing size, we sampled hundreds of wings to gain insight on geometric complexity of the wings. A detailed list of sources is provided in section 2 below.

1.1 Original micrographs

Original micrographs of actual wings were taken using a CanonScan 9000F Mark II and then processed in `Fiji/ImageJ` using the stitching and extended depth of focus tools.

1.2 Published wing images

Published wing images where either taken (1) from online resources (Biodiversity Heritage Library)[1] or (2) from books at the Ernst Mayr Library, Cambridge, MA. For images taken from online resources, images were contrast-adjusted with Adobe Photoshop when needed. For images taken from textbooks, images were scanned again with the CanonScan. For insect wings represented within Odonata, segmented images were taken from Hoffmann *et al.*[2].

1.3 Use of images

The supplemental material of Hoffmann *et al.*[2] show an extensive analysis of comparing insect wing drawings to micrographs. The authors show that these drawings are true-to-form and accurately capture the geometries that we are interested in.

1.4 Segmenting wing images

The segmentation of wing images was done using the code used in Hoffmann *et al.*[2]. This code is available online at `https://github.com/hoffmannjordan/insect-wing-venation-patterns`. Due to the increased diversity of wing domain shapes, the procedure needed to be modified to handle an increased diversity of wing domains. Specifically, regions that are particularly non-convex need to be properly accounted for. This was done through a combination of code changes (available online) and by manually adding looped regions in certain parts of the wing. These looped regions serve to help in wing domains that are not convex. A loop will not polygonize out, but allows us to add nodes when we polygonize that would not be otherwise captured.

Wings segmented in our study range in the segmentation quality from poor to well-segmented.

1.5 Vein domain shape and size features for a single wing

After segmenting a wing, each domain is turned into a polygonal representation. This allows us to more accurately compute a variety of statistics about the shape of individual domains, and when grouped, about the shape of the entire wing.

2 Orders and species represented

While our data encompasses 789 wings, we represent several orders with key examples, understanding that within an order the variation of wing venation is highly variable.

Table 1. Insects represented in Figure 1

Species	Family	Order	Reference
Blattela germanica	Ectobiidae	Blattodea	[3], Pg. 126, Fig. 199
Isonychia bicolor	Isonychiidae	Ephemeroptera	[3], Pg. 265, Fig. 264
Ogcodes adaptatus	Acroceridae	Diptera	[4], Fig. 27
Merope tuber	Meropiedae	Mecoptera	[3], Pg. 305, Fig. 317
Anaea archidona (Hewitson)	Nymphalidae	Lepidoptera	[5], Plate 1, Fig. 1
Costora iena Mosley	Sericostomatidae	Trichoptera	[6], Pg. 47, Fig. 23
Paramigdolus tetropioides	Vesperidae	Coleoptera	[7], Pg. 140, Fig. 172
Stylops crawfordi Pierce	Stylopidae	Strepsiptera	[8], Plate 1, Fig. 5b
Steniolia duplicata	Crabronidae	Hymenoptera	[9], Plate 1, Fig. 8
Paracaecilius anareolatus	Caeciliusidae	Psocodea	[10], Pg. 499, Fig. 1
Forficula auricularia	Forficulidae	Dermaptera	[3], Pg. 295, Fig. 305
Zorotypus hubbardi Caudell	Zorotypidae	Zoraptera	[11], Pg. 96, Plate 6
Schistocerca gregaria Forskal	Locustidae	Orthoptera	[12], Fig. 4
Eusthenia spectabilis	Eustheniidae	Plecoptera	[3], Pg. 247, Fig. 246
Raphidia adnixa	Raphidiidae	Raphidioptera	[3], Pg. 173, Fig. 168
Zootermopsis angusticollis	Termopsidae	Isoptera	[3], Pg. 132, Fig. 126
Ctenomorpha titan	Phasmatidae	Phasmatodea	[13], Pg. 122, Fig. 46
Clothoda nobilis	Clothodidae	Embioptera	[3], Pg. 265, Fig. 265
Corydalus primitivus	Corydalidae	Megaloptera	[3], Pg. 155, Fig. 149
Cryptoleon nebulosum	Myrmeleonidae	Neuroptera	[3], Pg. 205, Fig. 202
Anax junius	Aeshnidae	Odonata	Salcedo
Mantis religiosa	Mantidae	Mantodea	[14], Fig. 4
Leptocysta novatis	Tingidae	Hemiptera	[15], Pg. 66, Fig. 14

2.1 List of species in Figure 1

Table 1 lists species in the Figure 1 phylogeny. Insects hand-caught by MK Salcedo were captured in Bedford, MA at the Concord Field Station in 2015.

2.2 List of species for six representative wings

Table 2 lists the species of our six representative insect wings shown in Figures 2 - 5. The listed Myrmeleontidae sp. was collected by Dino J. Martins in North Kajiado, Oloosirkon, Kenya in 2008.

Table 2. Six representative insect wings

Species	Family	Order	Reference
Drosophila melanogaster	Drosophilidae	Diptera	[16], Pg. 166
Acroneuria xanthenes Newm.	Acroneurinae	Plecoptera	[17], Plate 16, Fig. 10
Anax junius	Anax junius	Odonata	Salcedo
Myrmeleontidae sp	Myrmeleontidae	Neuroptera	Martins
Schistocerca americana	Acrididae	Orthoptera	Salcedo
Ctenomorpha titan	Phasmatidae	Phasmatodea	[13], Pg. 122, Fig. 46

3 Quantitative characterization

3.1 Initialization

For each polygonized wing, the coordinates are rescaled such that the entire wing has area 1. This effectively removes all size information for each wing, allowing comparison between species. In each of the analyses discussed below, the area is first rescaled before applying the technique [2].

3.2 Network analysis

We deployed a suite of network analysis tools on the diverse set of insect wing geometries. We tried both weighted [18] and unweighted analysis [19](techniques are described below).

3.2.1 Unweighted

After polygonizing an insect wing, we construct a list of vertices. We then construct an adjacency matrix, $M = N \times N$ matrix of 0's, where N represents the number of vertices. For each i, j pair of vertices that are connected, we set $M_{i,j} = 1$.

3.2.2 Weighted

Applying weighted connections between nodes (vein junctions) with length, L, allowed weighting $M_{i,j} = 1/L^n$ where L is the distance between nodes i and j and $n = 1, 2$. We also looked at an analysis using the resistance between nodes, L/r^4, where r was computed using the 2D width from segmented images of original micrographs (giving us an approximate thickness of veins). Rather than measure the radius of each vein segment, we took a handful of original micrographs and measured approximately 50 cross vein radii and their location. We then interpolated over the wing as a proxy.

3.3 Curvature

When computing curvature [20], κ, we do not use our polygonized wing. Instead, we use our original segmentation and extract the boundary of the wing region. Then we orient each wing such that the base of the wing is on the left and the perimeter of the wing runs clockwise. We choose a distance of $N = 0.02L_P$, and for each point on the perimeter, P, we take p_i, p_{i-N} and p_{i+N} where L is length of the entire perimeter. From here, we perform a linear-least square fit to a circle, where we define $\kappa = 1/R$.

3.4 Internal venation length

For all nodes N, we sum up the distances between all nodes. In doing this calculation, we then subtract off the perimeter, which we calculate separately. In the corresponding main text figure, we omit a selection of wings to improve visibility.

3.4.1 Proximal-to-Distal morphology traces

We characterize representatives of our large dataset using the P–D morphology traces introduced in [2]. We divide the wing into 25 equal spaced bins along the long axis of the insect wing. To calculate the mean circularity of each slice i, which we denote $\mathcal{S}_i$, we compute

$$\frac{1}{\sum_{\mathcal{P}_j} \text{Area}(\mathcal{S}_i \cap \mathcal{P}_j)} \sum_{\mathcal{P}_j} \text{Area}\,(\mathcal{S}_i \cap \mathcal{P}_j)\,\mathcal{C}(\mathcal{P}_j) \tag{1}$$

where the sum is applied over all polygons $\mathcal{P}_j$. This produces a smoothly varying mean circularity as we move along the long axis of the wing from base to wing tip. For polygon $\mathcal{P}_i$, we obtain the area fraction f_i that overlaps with the bin. We construct the vector of all area fractions in bin i, denoted $\vec{F}_i$. We also have the vector of all circularities $\vec{C}$ and all areas $\vec{A}$. For each bin, we compute

$$(\vec{F}_i \cdot \vec{A}, \vec{F}_i \cdot \vec{C}) \tag{2}$$

giving us the weighted mean area and the mean circularity of the wing domains in the region.

4 Availability of data and code

Integer `csv` files of segmented images of all data along with code used in the manuscript are freely available. All code and data can be found at:
https://github.com/hoffmannjordan/size-and-shape-of-insect-wings. A more general segmentation code can be found at
https://github.com/hoffmannjordan/Fast-Marching-Image-Segmentation.

References

[1] "Image from the biodiversity heritage library. biodiversity heritage library." http://www.biodiversitylibrary.org. Accessed Jan 2017.

[2] J. Hoffmann, S. Donoughe, K. Li, M. Salcedo, and C. Rycroft, "A simple developmental model recapitulates complex insect wing venation patterns," *PNAS*, 2018.

[3] J. Comstock, *The Wings of Insects*. Ithaca, NY: The Comstock Publishing Company, 1918.

[4] E. Schlinger, *Acroceridae. in: McAlpline, J.F. (Ed.), Manual of Nearctic Diptera*. Ottawa, Canada: Agriculture Canada, 1981.

[5] W. Comstock, *Butterflies of the American Tropics, The Genus Anaea, Lepidoptera Nymphalidae*. New York: The American Museum of Natural History, 1961.

[6] M. E. Mosely and D. Kimmins, *The Trichoptera (Caddis-flies) of Australia and New Zealand*. Dorking, Surrey: Printed by order of the trustees of the British Museum. Adlard and Son Ltd., 1953.

[7] R. Crowson, *The Natural Classification of the Families of Coleoptera*. Middlesex, England: EW Classey Ltd., 1967.

[8] W. D. Pierce, *Genera Insectorum, Strepsiptera*. Brussels, Belgium: Directed by P. Wytsman, 121 ed., 1911.

[9] J. Parker, *Proceedings of the United States National Museum. A generic revision of the fossorial wasps of the tribes Stizini and Bembicini with notes and descriptions of new species*. Washington, DC: United States Government Printing Office, vol. 75, article 5 ed., 1929.

[10] C. Lienhard, *Revue suisse de zoologie: Two unusual new psocids from Vietnam*, vol. 115. GenÃĺve,Kundig [etc.], 2008.

[11] A. Caudell, "Proceedings of the entomological society of washington.," vol. 20, no. 5, pp. 84–97, 1920.

[12] R. Herbert, P. Young, C. Smith, R. Wootton, and K. Evans, "The hind wing of the desert locust (*Schistocerca gregaria forskal*). iii. a finite element analysis of a deployable structure," *Journal of Experimental Biology*, vol. 203, pp. 2945–2955, 2000.

[13] A. Sharov, *Phylogeny of the Orthopteroidea*, vol. 118. Jerusalem, Israel: Israel Program for Scientific Translations, 1971.

[14] S. K. Brannoch, F. Wieland, J. Rivera, K.-D. Klass, O. BÃľthoux, and G. J. Svenson, "Manual of praying mantis morphology, nomenclature, and practices (insecta, mantodea)," *ZooKeys*, vol. 696, pp. 1–100, 2017.

[15] S. Montemayor, "Article review of the genus leptocysta stÃël with descriptions of two new species (hemiptera: Heteroptera: Tingidae) from argentina," *zootaxa*, vol. 2641, pp. 62–68, 10 2010.

[16] W. Brook, F. Diaz-Benjumea, and S. Cohen, "Organizing spatial pattern in limb development," *Ann Rev Cell Dev Biol*, vol. 12, pp. 161–180, 1996.

[17] J. N. ham and P. Claassen, *A monograph of the Plecoptera or Stoneflies of America North of Mexico, Vol II.* LaFayette, Indiana: The Thomas Say Foundation of the Entomological Society of America, 1925.

[18] G. Morrison and L. Mahadevan, "Discovering communities through friendship," *PloS one*, vol. 7, no. 7, pp. 1–9, 2012.

[19] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[20] W. Gander, G. H. Golub, and R. Strebel, "Least-squares fitting of circles and ellipses," *BIT Numerical Mathematics*, vol. 34, no. 4, pp. 558–578, 1994.

The unpredictable and the predetermined unfold together to make everything the way it is.

Tom Stoppard, Arcadia

3 Machine Learning & Crumpled Sheets

3.1 Background

The "big-data" requirements of traditional machine learning[40,87,23] are not typically available in a bench-top environment. Experiments are often laborious, and collecting data can be a slow, tedious process. Trying to remedy this disparity has been a hot topic of research, and has been approached from many different angles.

Some scientists have worked on designing specific neural networks and regularization methods that attempt to reduce the affects of over-fitting.[139,63,101] Others have worked on ways to best leverage a small amount of training data to extract the most information.[107,39,15,87,134,166]

In rigid flat-folding, there is a very clear relationship between the ridges and valleys in the crease network.

1. Creases cannot begin or terminate in the interior of a sheet—they must either reach the boundary or create closed loops.
2. The number of ridge and valley creases that meet at each vertex differs by two (Maekawa's theorem).[153]
3. Alternating sector angles must sum to π (Kawasaki's theorem).[153]

Together, these rules necessitate that provided the ridge network, one can infer the locations of the valleys with near certainty.

In crumpled sheets, these constraints are relaxed and therefore make the predictions far more complicated. As a result, the degree to which the same question can be answered is unclear.

3.2 Contributions

For this project, I wrote all code for the machine learning aspect of the project. With Yohai Bar-Sinai and Shruti Mishra, I designed the figures and discussed what code should be written. I participated in all discussions about how to use machine learning as a tool to try to get at the underlying physics of the system. Chris H. Rycroft wrote the majority of the $\mathrm{C}{+}{+}$ flat-folding code, which I modified to help streamline the process for generating *in silico* training data. Yohai and I wrote the main text and supplement of the manuscript which was then edited with the help of all of the other coauthors. As these things go, a lot of exploratory work went into this project that didn't make it into the paper. With Yohai and Shruti, I did preliminary work looking at recurring motifs and how these evolve.[90] We also tried "differentiating" the prediction to look for any underlying patterns.

3.3 Publication

APPLIED SCIENCES AND ENGINEERING

Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets

Jordan Hoffmann[1]*, Yohai Bar-Sinai[1]*†, Lisa M. Lee[1], Jovana Andrejevic[1], Shruti Mishra[1], Shmuel M. Rubinstein[1]†, Chris H. Rycroft[1,2]

Machine learning has gained widespread attention as a powerful tool to identify structure in complex, high-dimensional data. However, these techniques are ostensibly inapplicable for experimental systems where data are scarce or expensive to obtain. Here, we introduce a strategy to resolve this impasse by augmenting the experimental dataset with synthetically generated data of a much simpler sister system. Specifically, we study spontaneously emerging local order in crease networks of crumpled thin sheets, a paradigmatic example of spatial complexity, and show that machine learning techniques can be effective even in a data-limited regime. This is achieved by augmenting the scarce experimental dataset with inexhaustible amounts of simulated data of rigid flat-folded sheets, which are simple to simulate and share common statistical properties. This considerably improves the predictive power in a test problem of pattern completion and demonstrates the usefulness of machine learning in bench-top experiments where data are good but scarce.

INTRODUCTION

Machine learning is a versatile tool for data analysis that has permeated applications in a wide range of domains (*1*). It has been particularly well suited to the task of mining large datasets to uncover underlying trends and structure, enabling breakthroughs in areas as diverse as speech and character recognition (*2*–*5*), medicine (*6*), games (*7*, *8*), finance (*9*), and even romantic attraction (*10*). The prospect of applying machine learning to research in the physical sciences has likewise gained attention and excitement. Data-driven approaches have been successfully applied to data-rich systems such as classifying particle collisions in the Large Hadron Collider (LHC) (*11*, *12*), classifying galaxies (*13*), segmenting large microscopy datasets (*14*, *15*), or identifying states of matter (*16*, *17*). Machine learning has also enhanced our understanding of soft matter systems: In a recent series of works, Cubuk, Liu, and collaborators (*18*–*20*) have used data-driven techniques to define and analyze a novel "softness" parameter governing the mechanical response of disordered, jammed systems.

All examples cited above address experimentally, computationally, or analytically well-developed scientific fields supplied by effectively unlimited data. By contrast, many systems of interest are characterized by scarce or poor-quality data, a lack of established tools, and a limited data acquisition rate that falls short of the demands of effective machine learning. As a result, the applicability of machine learning to these systems is problematic and would require additional tools. This would potentially be of high value to the experimental physics community and would require novel ways of circumventing the data limitations, either experimentally or computationally. Here, we study crumpling and the evolution of damage networks in thin sheets as a test case for machine learning–aided science in complex, data-limited systems that lack a well-established theoretical, or even a phenomenological, model.

Crumpling is a complicated and poorly understood process: As a thin sheet is confined to a small region of space, stresses spontaneously localize into one-dimensional regions of high curvature (*21*–*23*), forming a damage network of sharp creases (Fig. 1B) that can be classified according to the sign of the mean curvature: Creases with positive and negative curvature are commonly referred to as valleys and ridges, respectively. Previous works on crumpled sheets have established clear and robust statistical properties of these damage networks. For example, it has been shown that the number of creases at a given length follows a predictable distribution (*24*), and the cumulative amount of damage over repeated crumpling is described by an equation of state (*25*). However, these works do not account for spatial correlations, which is the structure we are trying to unravel. The goal of this work was to learn the statistical properties of these networks by solving a problem of network completion: Separating the ridges from valleys, can a neural net be trained to accurately recover the location of the ridges, presented only with the valleys? For later use, we call this problem partial network reconstruction. The predominant challenge we are addressing here is a severe data limitation. As detailed below, we were unable to perform this task using experimental data alone. However, by augmenting experimental data with computer-generated examples of a simple sister system that is well understood, namely, rigid flat folding, we trained an appropriate neural network with significant predictive power.

The primary dataset used in this work was collected in a previous crumpling study (*25*), where the experimental procedures are detailed and are only reviewed here for completeness. Mylar sheets (10 cm by 10 cm) are crumpled by rolling them into a 3-cm-diameter cylinder and compressing them uniaxially to a specified depth within the cylindrical container, creating a permanent damage network of creasing scars embedded into the sheet. To extract the crease network, the sheet is carefully opened up and scanned using a custom-made laser profilometer, resulting in a topographic height map from which the mean curvature is calculated. The sheet is then successively recrumpled and scanned between 4 and 24 times, following the same procedure. The curvature map is preprocessed with a custom algorithm based on the Radon transform (for details, see section S1) to separate creases from

[1]John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA. [2]Computational Research Division, Lawrence Berkeley Laboratory, Berkeley, CA 94720, USA.
*These authors contributed equally to this work.
†Corresponding author. Email: ybarsinai@gmail.com (Y.B.-S.); shmuel@seas.harvard.edu (S.M.R.)

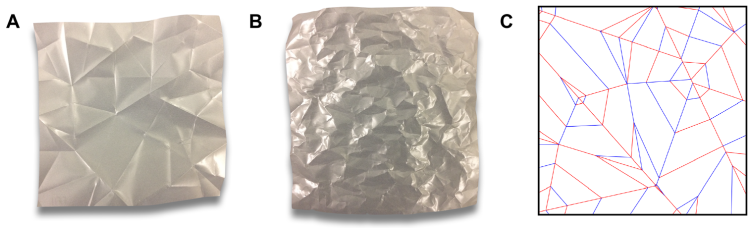


Fig. 1. Examples of crease networks. (**A**) A 10 cm by 10 cm sheet of Mylar that has undergone a succession of rigid flat folds. (**B**) A sheet of Mylar that has been crumpled. (**C**) A simulated rigid flat-folded sheet. The sheet has been folded 13 times. Ridges are colored red, and valleys are blue.

the flat facets and fine texture in the data (Fig. 2A). The complete dataset consists of a total of 506 scans corresponding to 31 different sheets.

RESULTS

Failures with only experimental data

As stated above, the task we tried to achieve is partial network reconstruction: inferring the location of the ridges given only the valleys (Fig. 2A). Our first attempts were largely unsatisfactory and demonstrated little to no predictive power. Strategies for improving our results included subdividing the input data into small patches of different length scales, varying the network architecture, data representation, and loss function, and denoising the data in different ways. We approached variants of the original problem statement, trying to predict specific crease locations, distance from a crease, and changes in the crease network between successive frames. In all these cases, our network invariably learned specific features of the training set rather than general principles that hold for unseen test data, a common problem known as overfitting. The main culprit for this failure is insufficient data: The dataset of a few hundred scans available for this study is small compared with standard practices in machine learning tasks [for example, the problem of handwritten digit classification using the MNIST database, which is commonly given as an introductory exercise in machine learning, consists of 70,000 images (*5*)]. Moreover, as creases produce irreversible scars, images of successive crumples of the same sheet are highly correlated, rendering the effective size of our dataset considerably smaller.

Overfitting can be addressed by constraining the model complexity through insights from physical laws, geometric rules, symmetries, or other relevant constraints. Alternatively, it can be mediated by acquiring more data. Sadly, neither of these avenues is viable: Current theory of crumpling cannot offer significant constraints about the structure or evolution of crease networks. Furthermore, adding a significant amount of experimental data is prohibitively costly: Achieving a dataset of the size typically used in deep learning problems, say 10^4 scans, would require thousands of lab hours, given that a single scan takes about 10 min. Last, data cannot be efficiently simulated since, while preliminary work on simulating crumpling is promising (*26*, *27*), generating a simulated crumpled sheet still takes longer than an actual experiment. A different approach is needed.

Turning to a sister system: Rigid flat folding

An alternative strategy is to consider a reference system free from data limitations alongside the target system, with the idea that similarities between the target and reference systems allow a machine learning model of one to inform that of the other. This is similar to transfer learning (*28*), but in this case, rather than repurpose a network, we supplement the training data with that of a reference system. In our case, a natural choice of such a system is a rigid flat-folded thin sheet, effectively a more constrained version of crumpling that is well understood. Rigid flat folding is the process of repeatedly folding a thin sheet along straight lines to create permanent creases, keeping all polygonal faces of the sheet flat during folding. For brevity, we will henceforth omit the word "rigid" and refer simply to flat folding.

Known rules constrain the structure of the flat-folded crease network: Creases cannot begin or terminate in the interior of a sheet—they must either reach the boundary or create closed loops; the number of ridge and valley creases that meet at each vertex differs by two (Maekawa's theorem); last, alternating sector angles must sum to π (Kawasaki's theorem) (*29*). Given these rigid geometric rules, we expect partial network reconstruction of rigid flat-folded sheets to be a much more constrained problem than that of crumpled ones.

However, while experimentally collecting flat-folding data is only marginally less costly than collecting crumpling data, simulating it on a computer is a straightforward task, which provides a dataset of a practically unlimited size. We wrote a custom code to do this using the Voro++ library (*30*) for rapid manipulation of the polygonal folded facets, as described in section S2. Typical examples are shown in Figs. 1C and 2B and fig. S1.

Having flat folding as a reference system provides foremost a convenient setting for comparing the performance of different network architectures. The vast parameter space of neural networks requires testing different hyperparameters, loss functions, optimizers, and data representations with no standard method for finding the optimal combination. This problem is exacerbated when it is not at all clear where the failure lies: Is the task at all feasible? If so, is the network architecture appropriate? If so, is the dataset sufficiently large? Answering these questions with our limited amount of experimental data is very difficult. In contrast, for flat-folded sheets, we are certain that the task is feasible and our dataset is comprehensive, so experimentation with different networks is easier. After testing many architectures, we identified a network capable of learning the desired properties of our data, reproducing linear features and maintaining even nonlocal angle relationships between features.

Network structure

The chosen network is a modified version of the fully connected SegNet (*31*) deep convolutional neural net. As outlined in Fig. 2A, each crease network is separated into its valleys and ridges. The neural net, $\mathcal{N}$, is given as an input binary image of the valleys, denoted $\mathbf{X}$ ("input" in Fig. 2). The output of the network, $\mathcal{N}(\mathbf{X})$, is the predicted distance transform of the ridges, $\mathbf{Y}$. That is, for each pixel, $\mathbf{Y}$ is the distance to the nearest ridge pixel (“target” in Fig. 2). Training is performed by minimizing the L_2 distance (the “loss”) between the predicted distance transform, $\hat{\mathbf{Y}} = \mathcal{N}(\mathbf{X})$, and the real one

$$L = \sum_i (\hat{Y}_i - Y_i)^2 \tag{1}$$

where the summation index i represents image pixels. The motivation for this choice of representation is that creases are sharp and narrow features, and therefore, if we require $\mathcal{N}$ to predict the precise location of a crease, even slight inaccuracies would lead to vanishing gradients of L, making training harder. See Materials and Methods below for full details of the implementation.

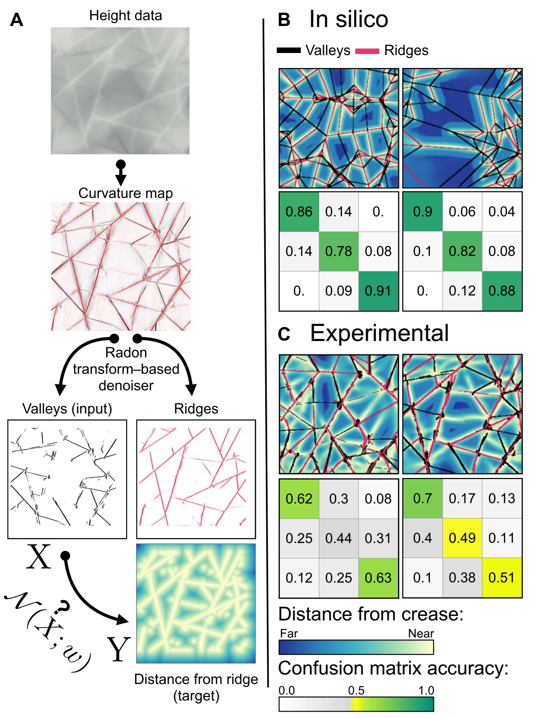


Fig. 2. A schematic of the processing pipeline. (**A**) From the height map, a mean curvature map is calculated and denoised with a Radon transform–based method. Valleys (black) and ridges (red) are separated. The binary image of the valleys (**X**) is the input to the neural network ($\mathcal{N}$). The distance transform of the binary image of the ridges is the target (**Y**). Brighter colors represent regions closer to ridges. These color conventions are consistent through all the figures in this paper. (**B**) Two samples of predictions on generated data. The true fold network is superimposed on the predicted distance map. It is seen that the true ridges (red) coincide perfectly with the bright colors, demonstrating strong predictive power. Below the predictions, we show confusion matrices, with the nearest third of pixels, the middle third, and the furthest third. (**C**) Two predictions, as well as their corresponding confusion matrices, using the network trained on generated data (without noise) and applied to experimental scans.

In silico flat folding

For exclusively in silico–generated flat-folding data, the trained network performs partial network reconstruction with nearly perfect accuracy, as demonstrated in Fig. 2B: The agreement between the true location of the valleys (red lines) and their predicted location (bright colors) is visibly flawless. As a means of quantifying accuracy, we present the confusion matrices of the predicted and true output (Fig. 2B).

Confusion matrices are a common way to quantify classification errors, and since we are predicting the distance from a crease, the problem can be thought of as a classification problem: Choosing some thresholds according to typical values of the distances, we can ask for each point in space whether it is close to a crease, far from it, or at an intermediate distance. The confusion matrix measures what percentage of each class is correctly classified and, if not, what class it is wrongly classified as. We define three equal bins, based on the relative distance from the predicted ridges. The upper row in the matrix corresponds to pixels that are closest to the ridges, and the lower row corresponds to the farthest pixels. Similarly, the first and last columns correspond to the closest and farthest predicted distances. Thus, the top left entry in the matrix contains the probability of correctly predicting regions closest to a ridge, which is approximately 90%.

Partial network reconstruction of in silico flat-folded sheets is itself a nontrivial task requiring the knowledge of a complicated set of geometrical rules. Tasked to a human, inferring these rules from the data would require non-negligible effort in writing an explicit algorithm. The neural network, however, solves this problem with relative ease.

Experimental flat folding

As an intermediate step between in silico flat-folding and experimental crumpling data, we next examine the performance of the neural network on experimental flat-folding scans. Figure 2C reveals that the resulting prediction weakens by comparison, a consequence of noise present in experimental data that is absent from the in silico samples. Noise occurs in the form of varying crease widths, fine texture, and missing creases that are undetected in image processing. In some cases, even the true creases that are missed during processing are correctly predicted, which also introduces error to our accuracy metric (see, for example, the center of the second panel of Fig. 2C). While sufficient data of experimental flat folding would likely allow the network to distinguish signal from noise, in our data-limited regime, noise must be added to the generated in silico data to help the network learn to accurately predict experimental scans and avoid overfitting.

We examine the effect of adding several types of noise on the prediction accuracy on experimental input (Fig. 3, A to E). We observe considerable improvement and find that adding experimentally realistic noise (Fig. 3E) is more effective than toggling individual pixels randomly (Fig. 3, B and D). We found that the noise type that leads to optimal training is to randomly add and remove patches of input that are approximately the same length scale as the noise in the experimental scans. We also find that it is important to provide input data with lines of variable width to prevent the network from expecting only creases of a particular width. For complete details of the different noise properties, see Materials and Methods.

While the values in the confusion matrices in Fig. 3E might seem low, it is noteworthy that the metric used here is not trivial to interpret: It compares the L_2 distance from a distance map, which is particularly sensitive to noise since a localized noise speckle in a region remote from valleys perturbs a large region of space (essentially, of the size of its Voronoi cell). To gauge the effect of noise on the accuracy metric, we randomly toggle a fraction P of pixels in an otherwise perfect flat-folding example and recompute the entries of its confusion matrix, as presented in Fig. 3F. With realistic noise levels, i.e., $P \sim 10^{-3}$, we can expect accuracy values between 0.75 and 0.80 in the upper left and lower right entries of the confusion matrix, comparable to the values reported in Fig. 3E. That is, for experimental flat folding, we achieve accuracy levels that are comparable to what is expected for a perfect prediction with noisy preprocessing.

Experimental crumpling

For crumpling, we train the neural network using a combination of 30% experimental crumpling and 70% in silico flat-folding data, which was noised as described above. We also tried pretraining on in silico data prior to training on crumpling data but observed no improvement. Training on this combined dataset, the resulting predictions accurately reconstruct key features of the crease networks in crumpled sheets, which were not achieved in prior attempts. In Fig. 4, we present predictions on entire sheets (Fig. 4A) and a few close-ups on selected regions (Fig. 4B). The confusion matrices suggest that the network is often relatively accurate in predicting regions that are directly near a crease (upper left entry) and large open spaces (lower right entry), classifying these regions with 50 to 60% accuracy. In addition, fig. S3 shows the prediction on each of the 16 successive crumples of the same sheet held out from training.

The ratio of 70% in silico data was chosen since it provides optimal predictions, as shown in Fig. 5A. We present three different metrics to quantify the predictive power: the L_2 loss of Eq. 1, the Pearson correlation between the prediction and the target, and the average of the upper left and lower right of the confusion matrix (classification accuracy). We find that all accuracy metrics are optimized for training on 50 to 70% in silico data. It is also interesting to see in what way this affects the prediction: In Fig. 5D, we show that when trained solely on experimental data, the neural network produces a blurred and indecisive prediction, while for 100% flat-folding data, the network predicts only unrealistic straight and long creases.

In addition to these metrics, one can compare the network's output to "random" network completion, i.e., to a network that construes a pattern having the statistical properties of a crease network but is only

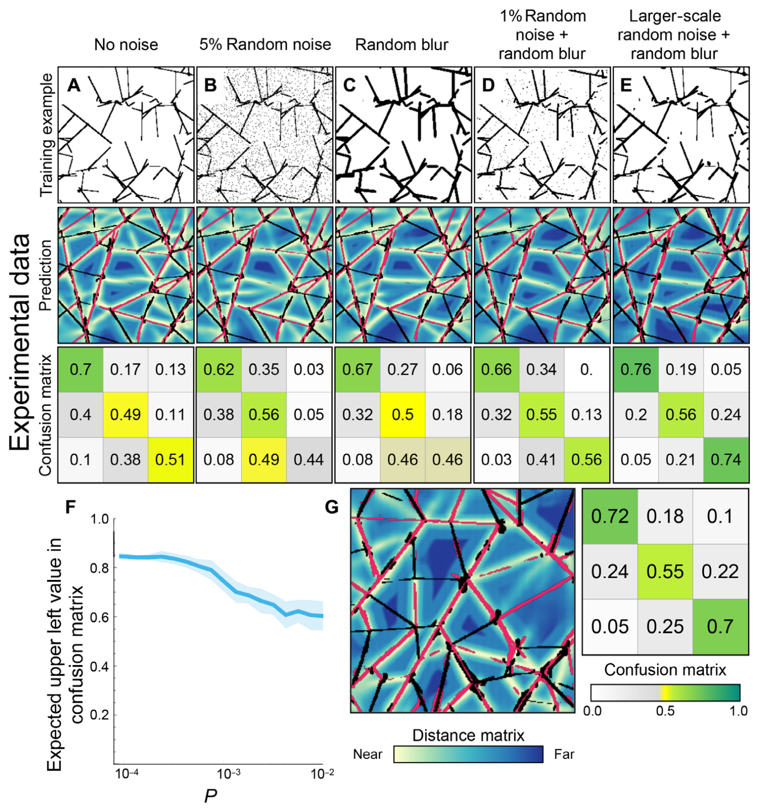


Fig. 3. Effect of noise type on prediction. (**A** to **E**) An example noised image (top), an example prediction (middle), and the corresponding confusion matrix (bottom) for different types of artificial noise. Noise types are described concisely in the title of each panel, and complete specifications are given in Materials and Methods. (**F**) The upper left value of the confusion matrix when each pixel of the near-perfect prediction from Fig. 2B was randomly toggled with probability *P*. (**G**) The network from (E) applied on an additional experimental scan (from left panel of Fig. 2C). The average confusion matrix on all experimental scans is shown.

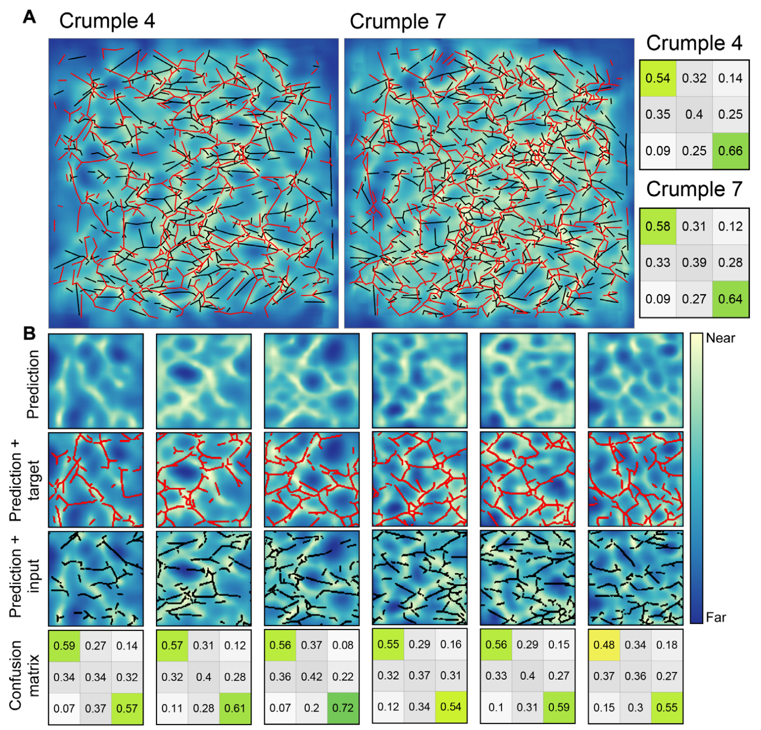


Fig. 4. Predictions on crumpling. (**A**) One sheet that was successively crumpled, shown after four and seven crumpling iterations. Color code follows Fig. 2. (**B**) Closeups on selected smaller patches from the same image, broken down to prediction, prediction and target, and prediction and input.

weakly correlated with the input image. Although a generative model for crease networks is not available, we can sample crease patterns from the experimental data and compare the predicted distance maps to those measured from these randomly selected samples. This is discussed in section S5, where it is seen that our prediction for a given crease pattern is overwhelmingly closer to the truth than any sampled patch from other experiments (fig. S5).

The similarity of flat folding and crumpling

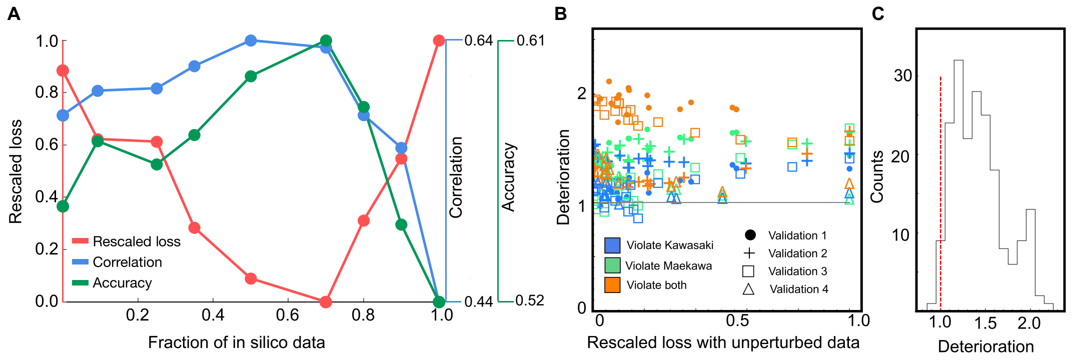
These results demonstrate that augmenting the dataset with in silico–generated flat-folding data allows the network to discern some underlying geometric order in crease networks of experimental crumpling data. This suggests that the two systems share some common statistical properties, and it is interesting to ask how robust this similarity is. One may suspect that the main contribution of the in silico data is merely having a multitude of intersecting straight lines, which are the main geometric feature that is analyzed, but that the specific statistics of these lines is not crucial.

As explained above, flat-folding networks are characterized by two theorems: Maekawa's theorem, which constrains the curvatures (ridge/valley) of creases joined in each vertex, and Kawasaki's theorem, which constrains the relative angles at vertices. We tested the sensitivity of our prediction to replacing the in silico data used in training with crease networks that violate these rules: We obtained crease networks that violate Maekawa's theorem by taking flat-folding networks and randomly reassigning curvatures to each crease, and crease networks that violate Kawasaki's theorem by perturbing all vertex positions. Last, we obtained crease networks that violate both rules by performing both perturbations simultaneously. Examples of perturbed networks are shown in fig. S6, with additional details about the perturbation process.

The effect is quantified in Fig. 5 (B and C). We define, for a given sheet, the "deterioration" as the ratio between the loss of a network trained on 70% experimental data and 30% perturbed flat-folding data to that of a network trained on the same ratio of experimental and unperturbed data. It is seen that breaking the flat-folding rules leads to consistently worse performance for all types of perturbations.

We cross-validated with four different experiments covering a total of 198 sheets. Although for some small fraction (<5%) of the sheets training on perturbed data has led to marginally better performance, this happened mostly in sheets with low loss, and the improvement is negligible. On average, the network trained on perturbed data has a loss approximately 35% higher than that of the network trained on unperturbed data.

These results, namely, that training on perturbed flat-folding networks led to inferior performance, again suggest a similarity between crumpled crease networks and flat-folded networks. We did not quantitatively study the detailed effect of the different kinds of perturbations—i.e., whether violating Kawasaki's rule, Maekawa's rule,



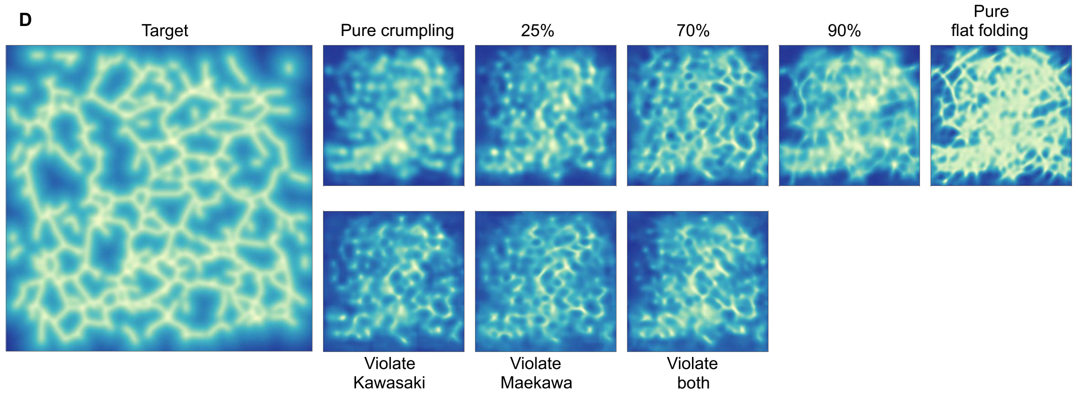


Fig. 5. Effect of fraction generated data. (**A**) Three quantifications of the predictive power of the model when trained on varying amounts of generated data and a constant amount of crumpling data. Strong predictive power corresponds to low loss (red) and large Pearson correlation and classification accuracy (blue and green, respectively). (**B**) Deterioration (see main text) for each sheet in the validation set, as a function of the rescaled loss. Colors correspond to different perturbations and marker styles to cross-validation sample. It is seen that all tested perturbations lead to worse predictive power (above the gray reference line). The few points below the reference line occur at high crumple number and low absolute loss. (**C**) Histogram of all points in (B). Values to the right of the red line correspond to deterioration when using unphysical data. (**D**) Example target and predictions for the various models considered in previous panels.

or both results in more or less accurate predictions. Instead, equipped with this physical insight, we propose to directly probe the statistical similarity with traditional methods by measuring vertex properties in crease networks, a study that will be reported elsewhere.

DISCUSSION

Experimental data are paramount to our understanding of the physical world. However, prohibitive data acquisition rates in many experimental settings require augmenting experimental data to draw meaningful conclusions. In particular, computer simulations now play a significant role in exploratory science; many experimental conditions can be accurately simulated to corroborate our understanding of empirical results.

Despite these advances, the simulation of certain phenomena is inhibited by insufficient theoretical knowledge of the system or by demanding computational resources and development time. For crumpling, without a deeper understanding that would allow the use of simplified/reduced models, simulations require prohibitively small time steps, small domain discretization, or both (*26*). Here, we show that even with a small experimental training set, augmenting the dataset by computer-generated, artificially noised data of flat folding, salient features of the ridge network can be predicted from the surrounding valleys: The network successfully predicts the presence of certain creases, as well as their pronounced absence in certain locations (see Fig. 4B). Moreover, our results demonstrate a statistical similarity between flat folding and crumpling, evidenced by the fact that when flat-folding data are replaced with data of similar geometry but different statistics, the algorithm does not succeed in learning the underlying distribution to the same extent (Fig. 5B).

Our results demonstrate the capacity of a neural network to learn, at least partially, the structural relationship of ridges and valleys in a crease pattern of crumpled sheets. The next step is to understand the network's decision process, with the aim of uncovering the physical principles responsible for the observed structure. However, while interpretation of trained weights is currently a heavily researched topic [see (*32*–*34*), among many others], there is not yet a standard method to do so. Our ongoing work seeks to probe the network's inner workings by perturbing the input data. For example, we can individually alter input pixels and quantify the effect of perturbation on the prediction relative to the original target. Alternatively, we can examine the effect of adding or removing creases or test the prediction on inputs that do not occur naturally in crumpled sheets. Some preliminary results are discussed in section S4.

Improving the experimental dataset by performing dedicated experiments or replacing the simulated flat folding with simulated crumpling data is also a promising future direction. While we have only demonstrated the advantages of data augmentation for one problem, it is tempting to imagine how it may apply to other systems in experimental physics. In addition to providing insights into the structure of crease patterns, a quantitative predictive model (i.e., an oracle) could serve as an important experimental tool that allows for targeted experiments, especially when experiments are costly or difficult. As shown above, a trained neural network is able to shed light on where order exists, even if the source of the order is not apparent.

Replacing the scientific discovery process with an automated procedure is risky. Frequently, hypotheses that were initially proposed are not the focal points of the final works they germinated, as observations and insights along the way sculpt the research toward its final state. This serendipitous aspect of discovery has been of immense importance to the sciences and is difficult to include in automated data exploration methods, which is an area of ongoing research (*35*–*37*). By showing that data-driven techniques are able to make nontrivial predictions on complicated systems, even in a severely data-limited regime, we hope to demonstrate that these tools should become a valuable tool for experimentalists in many different fields.

MATERIALS AND METHODS

Experiments

Experimental flat-folding and crumpling data were performed on 10 cm by 10 cm sheets of 0.05-mm-thick Mylar. Flat folds were performed successively at random, without allowing the paper to unfold between successive creases. Crumpled sheets were obtained by first rolling the sheet into a 3-cm-diameter cylinder and then applying axial compression to a specified depth between 7.5 and 55 mm. Sheets were successively crumpled between 4 and 24 times.

To image the experimental crease network, crumpled/flat-folded sheets were opened up, and their height profile was scanned using a home-built laser profilometer. The mean curvature map was calculated by differentiating the height profile and then denoised using a custom Radon-based denoiser (the implementation details of which are given in section S1). A total of 506 scans were collected from 31 different experiments.

Network architecture and training

Data were fed into a fully convolutional network, based on the SegNet architecture (*31*) with the final soft-max layer removed, as we did not perform a classification problem. The depth of the network allows for long-range interactions to be incorporated without fully connected layers. The network was implemented in Mathematica, and optimization was performed using the Adam optimizer (*38*) on a Tesla 40c graphics processing unit (GPU) with 256 GB of random-access memory (RAM) and a computer with a Titan V GPU and 128 GB of RAM. Code is freely available. See "materials availability" below.

For training, the in silico–generated input data were augmented with standard data augmentation methods: Symmetric copies of each original were generated by reflection and rotation. All images were down sampled to have dimensions of 224 by 224 pixels. For crumpling data, creases were also linearized to look more similar to the experimental input. An example of the effect of linearizing is shown in fig. S2.

Noise

Noise was added to the input in a few different ways (presented in Fig. 3B). The noise of each panel was generated as follows:

A. No noise.

B. "White" noise: Each pixel was randomly toggled with 5% probability.

C. Random blur: Input was convolved with a Gaussian with a width drawn uniformly between 0 and 3. The array was then thresholded at 0.1. Here and below, "thresholded at z" means a pointwise threshold was imposed on the array, such that values smaller than z were set to 0 and otherwise set to 1.

D. Each pixel was randomly toggled with 1% probability and then passed through random blur (C).

E. Input was random blurred [as in (C)] but thresholded at 0.55. We denote the blurred-and-thresholded input as $\tilde{X}$. Then, $\tilde{X}$ was noised using both additive and multiplicative noise, as follows: Y and Z are two random fields drawn from a pointwise uniform distribution between 0 and 1 and convolved with a Gaussian of width seven (pixels) and thresholded at 0.55. Last, the "noised" input is

$$\min(\tilde{X} + (1 - Y),\ 1)(1 - Z)$$

SUPPLEMENTARY MATERIALS

Supplementary material for this article is available at http://advances.sciencemag.org/cgi/content/full/5/4/eaau6792/DC1

Section S1. Radon transform–based detection method
Section S2. In silico generation of flat-folding data
Section S3. Prediction on 16 sheets
Section S4. Probing the network: Ongoing work
Section S5. Another approach to error quantification
Section S6. Perturbing the in silico data
Fig. S1. In silico–generated flat-folded crease networks.
Fig. S2. Comparison between the preprocessed curvature map and the linearized version.
Fig. S3. Prediction on a sheet that was crumpled 16 times.
Fig. S4. Additional test results.
Fig. S5. Prediction accuracy.
Fig. S6. Examples of perturbed in silico data.
Reference (*39*)

REFERENCES AND NOTES

1. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. *Nature* **521**, 436–444 (2015).
2. A.-R. Mohamed, G. Dahl, G. Hinton, Deep belief networks for phone recognition, in *Proceedings of the NIPS Workshop on Deep Learning for Speech Recognition and Related Applications* (Neural Information Processing Systems Foundation, 2009), p. 39.
3. G. Dahl, D. Yu, L. Deng, A. Acero, Large vocabulary continuous speech recognition with context-dependent DBN-HMMS, in *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2011), pp. 4688–4691.
4. L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, A. Acero, Recent advances in deep learning for speech research at Microsoft, in *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (IEEE, 2013), pp. 8604–8608.
5. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).

6. K. Shameer, K. W. Johnson, B. S. Glicksberg, J. T. Dudley, P. P. Sengupta, Machine learning in cardiovascular medicine: Are we there yet? *Heart* **104**, 1156–1164 (2018).
7. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning. arXiv:1312.5602 [cs.LG] (19 December 2013).
8. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge. *Nature* **550**, 354–359 (2017).
9. J. B. Heaton, N. G. Polson, J. H. Witte, Deep learning for finance: Deep portfolios. *Appl. Stoch. Models Bus. Ind.* **33**, 3–12 (2017).
10. S. Joel, P. W. Eastwick, E. J. Finkel, Is romantic desire predictable? Machine learning applied to initial romantic attraction. *Psychol. Sci.* **28**, 1478–1489 (2017).
11. W. Bhimji, S. A. Farrell, T. Kurth, M. Paganini, Prabhat, E. Racah, Deep neural networks for physics analysis on low-level whole-detector data at the LHC. arXiv:1711.03573 [hep-ex] (9 November 2017).
12. P. Baldi, P. Sadowski, D. Whiteson, Searching for exotic particles in high-energy physics with deep learning. *Nat. Commun.* **5**, 4308 (2014).
13. M. Banerji, O. Lahav, C. J. Lintott, F. B. Abdalla, K. Schawinski, S. P. Bamford, D. Andreescu, P. Murray, M. J. Raddick, A. Slosar, A. Szalay, D. Thomas, J. Vandenberg, Galaxy zoo: Reproducing galaxy morphologies via machine learning. *Mon. Not. R. Astron. Soc.* **406**, 342–353 (2010).
14. C. Sommer, C. Straehle, U. Koethe, F. A. Hamprecht, Ilastik: Interactive learning and segmentation toolkit, in *Proceedings of the 2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro* (IEEE, 2011), pp. 230–233.
15. V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, J. Cuadros, R. Kim, R. Raman, P. C. Nelson, J. L. Mega, D. R. Webster, Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *JAMA* **316**, 2402–2410 (2016).
16. J. Carrasquilla, R. G. Melko, Machine learning phases of matter. *Nat. Phys.* **13**, 431–434 (2017).
17. M. Spellings, S. C. Glotzer, Machine learning for crystal identification and discovery. *AIChE J.* **64**, 2198–2206 (2018).
18. E. D. Cubuk, S. S. Schoenholz, J. M. Rieser, B. D. Malone, J. Rottler, D. J. Durian, E. Kaxiras, A. J. Liu, Identifying structural flow defects in disordered solids using machine-learning methods. *Phys. Rev. Lett.* **114**, 108001 (2015).
19. D. M. Sussman, S. S. Schoenholz, E. D. Cubuk, A. J. Liu, Disconnecting structure and dynamics in glassy thin films. *Proc. Natl. Acad. Sci. U.S.A.* **114**, 10601–10605 (2017).
20. S. S. Schoenholz, E. D. Cubuk, E. Kaxiras, A. J. Liu, Relationship between local structure and relaxation in out-of-equilibrium glassy systems. *Proc. Natl. Acad. Sci. U.S.A.* **114**, 263–267 (2017).
21. M. B. Amar, Y. Pomeau, Crumpled paper. *Proc. R. Soc. Lond. A* **453**, 729 (1997).
22. T. A. Witten, Stress focusing in elastic sheets. *Rev. Mod. Phys.* **79**, 643–675 (2007).
23. H. Aharoni, E. Sharon, Direct observation of the temporal and spatial dynamics during crumpling. *Nat. Mater.* **9**, 993–997 (2010).
24. C. A. Andresen, A. Hansen, J. Schmittbuhl, Ridge network in crumpled paper. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* **76**, 026108 (2007).
25. O. Gottesman, J. Andrejevic, C. H. Rycroft, S. M. Rubinstein, A state variable for crumpled thin sheets. *Commun. Phys.* **1**, 70 (2018).
26. R. Narain, T. Pfaff, J. F. O'Brien, Folding and crumpling adaptive sheets. *ACM Trans. Graph.* **32**, 1–8 (2013).
27. Q. Guo, X. Han, C. Fu, T. Gast, R. Tamstorf, J. Teran, A material point method for thin shells with frictional contact. *ACM Trans. Graph.* **37**, 147 (2018).
28. S. J. Pan, Q. Yang, A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **22**, 1345–1359 (2010).
29. N. Turner, B. Goodwine, M. Sen, A review of origami applications in mechanical engineering. *Proc. Inst. Mech. Eng. C J. Mech. Eng. Sci.* **230**, 2345–2362 (2016).
30. C. H. Rycroft, Voro++: A three-dimensional Voronoi cell library in C++. *Chaos* **19**, 041111 (2009).
31. V. Badrinarayanan, A. Kendall, R. Cipolla, Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **39**, 2481–2495 (2017).
32. D. Smilkov, N. Thorat, C. Nicholson, E. Reif, F. B. Vi'egas, M. Wattenberg, Embedding projector: Interactive visualization and interpretation of embeddings. arXiv:1611.05469 [stat.ML] (16 November 2016).
33. N. Frosst, G. Hinton, Distilling a neural network into a soft decision tree. arXiv:1711.09784 [cs.LG] (27 November 2017).
34. M. Sundararajan, A. Taly, Q. Yan, Axiomatic attribution for deep networks. arXiv:1703.01365 [cs.LG] (4 March 2017).
35. P. Raccuglia, K. C. Elbert, P. D. F. Adler, C. Falk, M. B. Wenny, A. Mollo, M. Zeller, S. A. Friedler, J. Schrier, A. J. Norquist, Machine-learning-assisted materials discovery using failed experiments. *Nature* **533**, 73–76 (2016).
36. E. A. Baltz, E. Trask, M. Binderbauer, M. Dikovsky, H. Gota, R. Mendoza, J. C. Platt, P. F. Riley, Achievement of sustained net plasma heating in a fusion experiment with the optometrist algorithm. *Sci. Rep.* **7**, 6425 (2017).
37. F. Ren, L. Ward, T. Williams, K. J. Laws, C. Wolverton, J. Hattrick-Simpers, A. Mehta, Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments. *Sci. Adv.* **4**, eaaq1566 (2018).
38. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization. arXiv:1412.6980 [cs.LG] (22 December 2014).
39. J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg, J. Beaulieu, P. J. Bentley, S. Bernard, G. Beslon, D. M. Bryson, P. Chrabaszcz, N. Cheney, A. Cully, S. Doncieux, F. C. Dyer, K. O. Ellefsen, R. Feldt, S. Fischer, S. Forrest, A. Frénoy, C. Gagné, L. L. Goff, L. M. Grabowski, B. Hodjat, F. Hutter, L. Keller, C. Knibbe, P. Krcah, R. E. Lenski, H. Lipson, R. MacCurdy, C. Maestre, R. Miikkulainen, S. Mitri, D. E. Moriarty, J.-B. Mouret, A. Nguyen, C. Ofria, M. Parizeau, D. Parsons, R. T. Pennock, W. F. Punch, T. S. Ray, M. Schoenauer, E. Shulte, K. Sims, K. O. Stanley, F. Taddei, D. Tarapore, S. Thibault, W. Weimer, R. Watson, J. Yosinski, The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. arXiv:1803.03453 [cs.NE] (9 March 2018).

Acknowledgments: We thank an anonymous referee for the suggestion to perturb the in silico input data. **Funding:** This work was supported by the National Science Foundation through the Harvard Materials Research Science and Engineering Center (DMR-1420570). S.M.R. acknowledges support from the Alfred P. Sloan Research Foundation. The GPU computing unit was obtained through the NVIDIA GPU grant program. J.H. was supported by a Computational Science Graduate Fellowship (DOE CSGF). Y.B.-S. was supported by the JSMF postdoctoral fellowship for the study of complex systems. C.H.R. was partially supported by the Applied Mathematics Program of the U.S. DOE Office of Advanced Scientific Computing Research under contract no. DE-AC02-05CH11231. **Author contributions:** S.M.R. and C.H.R. conceived the concepts of the research. Y.B.-S., J.H., S.M.R., and C.H.R. designed the research. L.M.L. performed new experiments. J.A. and J.H. performed the image preprocessing. J.A. wrote the Radon transform methods code. Y.B.-S., J.H., and S.M. contributed to the machine learning. C.H.R. and J.H. wrote the flat-folding code. All authors analyzed the data and wrote the paper. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** Source code is available at https://github.com/hoffmannjordan/Crumpling. All data needed to evaluate the conclusions in the paper are present in the paper and/or the Supplementary Materials. Additional data related to this paper may be requested from the authors.

Submitted 5 July 2018
Accepted 6 March 2019
Published 26 April 2019
10.1126/sciadv.aau6792

Citation: J. Hoffmann, Y. Bar-Sinai, L. M. Lee, J. Andrejevic, S. Mishra, S. M. Rubinstein, C. H. Rycroft, Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets. *Sci. Adv.* **5**, eaau6792 (2019).

Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets

Jordan Hoffmann, Yohai Bar-Sinai, Lisa M. Lee, Jovana Andrejevic, Shruti Mishra, Shmuel M. Rubinstein and Chris H. Rycroft

Sci Adv **5** (4), eaau6792.
DOI: 10.1126/sciadv.aau6792

3.4 Supplemental Information

Science Advances
AAAS

advances.sciencemag.org/cgi/content/full/5/4/eaau6792/DC1

Supplementary Materials for

Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets

Jordan Hoffmann, Yohai Bar-Sinai*, Lisa M. Lee, Jovana Andrejevic, Shruti Mishra,
Shmuel M. Rubinstein*, Chris H. Rycroft

*Corresponding author. Email: ybarsinai@gmail.com (Y.B.-S.); shmuel@seas.harvard.edu (S.M.R.)

Published 26 April 2019, *Sci. Adv.* **5**, eaau6792 (2019)
DOI: 10.1126/sciadv.aau6792

This PDF file includes:

Section S1. Radon transform–based detection method

Here we detail the detection method used to identify crease networks from maps of mean curvature prior to machine learning. We refer to our technique as a Radon-based detection method, as it repurposes the key principle behind a Radon transform—recovering a signal through integration along directed paths—for crease detection. By integrating a quantity of interest, in our case the mean curvature, along paths of regularly spaced orientations within local regions of the curvature map, we construct a signal array that enhances the signature of creases and reduces noise. A strong signal is recovered if an integration path coincides with the direction of an extended structure such as a crease; a weak signal is produced by features that are point-like or isotropic, representative of noise and fine texture in the data. The raw curvature maps of each 10 cm $\times$ 10 cm sheet are 3000 $\times$ 3000 pixels. Prior to processing, curvature maps are downsampled for computational efficiency. A downsampling factor of 4 was found to preserve the integrity of the crease pattern while providing a useful speedup in computation for a final resolution of 75 pixels per cm. Next, a linear integration path is centered about a given pixel of the curvature map, traversing the diameter of a fixed circular local window. The average curvature along a particular direction is computed by exact numerical integration of the bicubic interpolant on the grid defined by pixel centers. The integration direction is systematically rotated about the central pixel, and the maximum average curvature over all path orientations is selected as the signal. This process is repeated for all pixels in the curvature map, resulting in a signal array of only the average curvatures that are a maximum along local, linear paths. Integrals along 24 equally spaced path orientations on the interval of 0 to 180 degrees were considered at each pixel and the maximum selected as the signal. We examined a range of integration path lengths up to 8 mm, as the integration window defines a length scale that must accommodate features of varied sizes. While smaller integration paths can detect finer details particularly at low crease densities, they sacrifice some of the advantage afforded by longer paths in accruing a strong signal that is well separated from noise. An integration path length of 3.2 mm suitably mediated such effects and provided a clear crease network. Finally, global and local thresholds are applied to the signal array to separate the real creases from the background noise. A combination of the two was observed to work well in retaining the desired crease network: The global threshold is more permissive of noise but acts uniformly across the signal array, while the local threshold accommodates variations in signal intensity, and thus provides sensitivity to softer (less sharp) creases. We use a global threshold of 0.12 as the minimum signal intensity retained as a crease (0.12 is approximately 10% the magnitude of the largest creases), and set the local threshold to label as noise any pixel whose intensity falls below 20% of the maximum signal in a 3.2 mm $\times$ 3.2 mm neighborhood centered about the pixel. In training with crumpled sheets, the crease networks were also linearized as shown in fig. S2. This was done with a custom script that skeletonized the input and used the *Mathematica* function `MorphologicalGraph`.

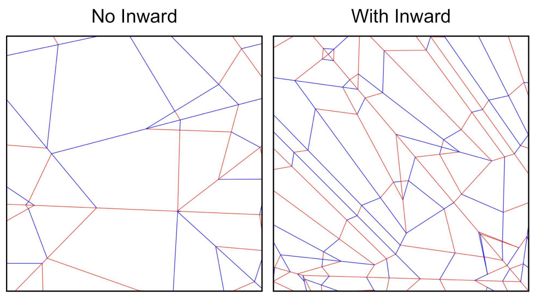


Fig. S1. **In silico–generated flat-folded crease networks.** Two random flat-folding patterns, with (left) and without (right) inward folding. Ridge folds are colored red and valley folds are colored in blue.

Section S2. In silico generation of flat-folding data

A custom code was written in C++ to simulate flat folding. The code makes use of the Voro++ software library [30], which provides routines for fast manipulation of polygons. To begin, the sheet is represented as a single square. To simulate a simple flat fold on a given chord, the square is cut into two polygons, and one polygon is reflected about the chord. Subsequent flat folds are simulated by taking the collection of polygons representing the folded sheet, cutting them by a given chord, and reflecting those on one side about the chord. Through-

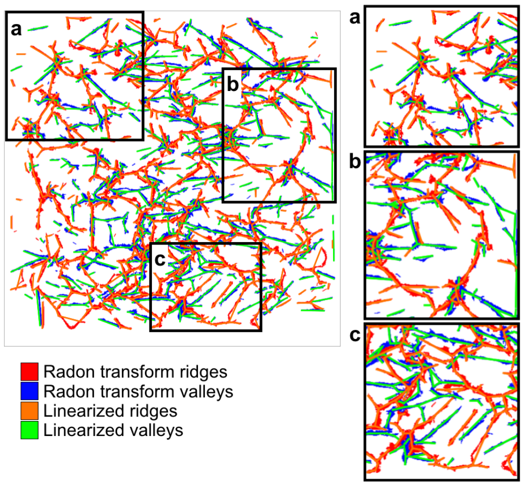


Fig. S2. **Comparison between the preprocessed curvature map and the linearized version.** The denoised curvature map of an entire crumpled sheet with three enlarged insets (a-c) for better visibility. Red and blue are creases retained after the Radon-based denoising, green and orange are the linearized representation.

out the process, each polygon keeps track of an affine transformation from its current configuration back to its position in the original square sheet. By transforming all polygons back to the original sheet, the flat folding map of valleys and ridges can be constructed. The code can also simulate inward folds where a ray is selected and the sheet is pinched in along this ray. For computational efficiency, the code computes a bounding circle during the folding process, whereby the collection of polygons representing the folded sheet is wholly within the circle.

While folding along a given chord is strictly well-defined, there is no natural way to draw a random chord from a distribution (e.g. Bertrand's paradox in probability theory) and a choice must be made regarding the way a chord is drawn. Our choice is the following: A fold is determined by a straight line in $\mathbb{R}^2$ and therefore can be parameterized by its angle and offset. At each iteration the angle is drawn uniformly in the range $[0, 2\pi)$ radians and the offset uniformly over the bounding circle. If the chosen fold line does not actually create a fold (because the line misses all polygons) then a new angle and displacement are chosen and so forth. For inward folds, we first choose a point uniformly inside the bounding circle, then determine if the point is inside any polygon; if not, we keep choosing new points until we find one that is. We then choose a random orientation for the ray from this point and two random angles α and β uniformly from $(0, \pi)$ for the first two folded segments that are counter-clockwise from the ray, after which the remaining two angles at the point are given by $\gamma = \pi - \alpha$ and $\delta = \pi - \beta$.

Our data set was generated by folding the sheet n times, where n is chosen uniformly in the range from 7 to 15. Each fold has a random sign (ridge or valley) with equal probability. For each sheet, the probability of inward folds was chosen uniformly over the range $[0\%, 50\%]$. Figure S1 shows a selection of generated crease patterns.

Section S3. Prediction on 16 sheets

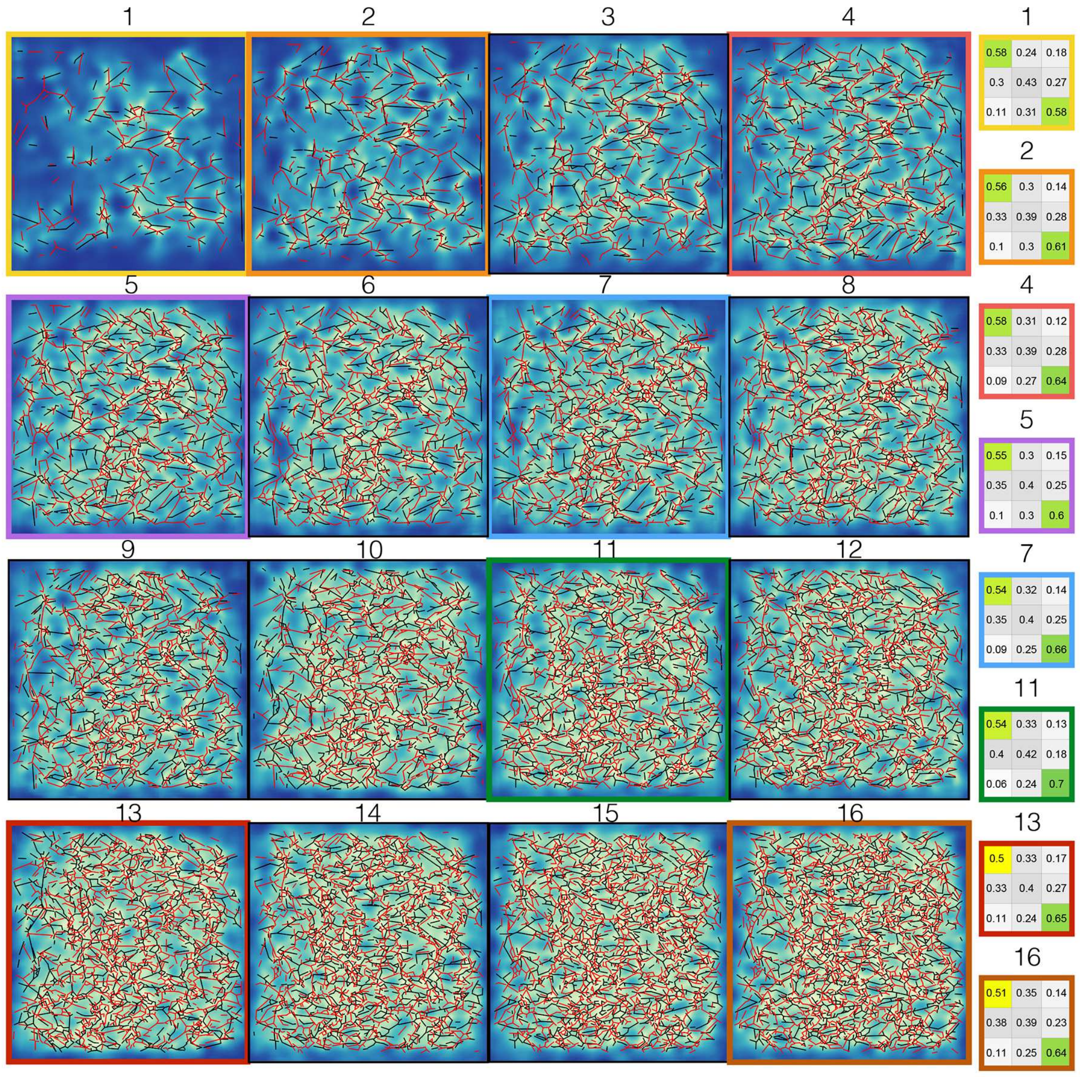
The validation set (an experiment held out from training) consists of 17 successive crumples of the same sheet of paper. In fig. S3, we show the prediction on the first sixteen of these sheets. For each prediction of an entire sheet, the image was computed in overlapping patches of size 224×224. Each pixel was considered to be the average value based on a sequence of predictions. Preliminary work was done on automatically detecting regions that were the best and the worst predicted. This along with aspects discussed below, are the topic of ongoing work.

Section S4. Probing the network: Ongoing work

In their paper, Lehman *et al.* discuss some computational oddities in the field of computational evolution [39]. They present a series of important ideas through short tales where the computer produced unexpected behavior that, when understood, were a key step in learning how to successfully use the computational tools. We think examples similar to these are important to share as the use of machine learning in the experimental sciences is still in its infancy. In this spirit, we discuss some of our attempts to tie the predictions of the network back to the underlying physics of crumpling.

A potential pitfall of using neural networks is that they will provide an output for any input, no matter how absurd either the input or output is. No warning appears. This is powerful, but requires caution, as neural networks allow for predictions on inputs that are physically impossible to create. Thus, one should take all the following probing attempts with a grain of salt.

It is tempting to "differentiate" the input signal to see if perturbations at any particular location cause large changes in the neural network's prediction. In fig. S4 A and A' we do this, perturbing the input (empty space and white lines) by making each pixel slightly more crease-like if it is not a crease or less crease-like if it is a crease. The background color shown is the magnitude of the change relative to the original prediction. Our hope was that this map may correlate with some known aspect of the physics. However, we do not think that this is the case. We tried aligning sequential images and estimating whether new creases tend to form with higher probability in regions that correlate with this sensitivity map—we do not find this to be the case. We are currently exploring more sophisticated ways of differentiating the trained network.



1

0.58	0.24	0.18
0.3	0.43	0.27
0.11	0.31	0.58

2

0.56	0.3	0.14
0.33	0.39	0.28
0.1	0.3	0.61

4

0.58	0.31	0.12
0.33	0.39	0.28
0.09	0.27	0.64

5

0.55	0.3	0.15
0.35	0.4	0.25
0.1	0.3	0.6

7

0.54	0.32	0.14
0.35	0.4	0.25
0.09	0.25	0.66

11

0.54	0.33	0.13
0.4	0.42	0.18
0.06	0.24	0.7

13

0.5	0.33	0.17
0.33	0.4	0.27
0.11	0.24	0.65

16

0.51	0.35	0.14
0.38	0.39	0.23
0.11	0.25	0.64

Fig. S3. **Prediction on a sheet that was crumpled 16 times**. The prediction is shown in blue for a given set of valleys (black). The true creases are overlaid in red .Confusion matrices for 8 of the 16 matrices are shown in the right. The color corresponds to the outline of the matrix.

Similarly, we can ask questions such as: What would happen if we translate a particular crease 5 mm to the left? What if we artificially remove parts of folds in flat-folding? What if we remove entire creases? In fig. S4B, we present the results of removing entire creases form a crumpled sheet. The ridges are colored by their total effect on the prediction, that is, if the original input is X and the perturbed input is $\tilde{X}$ (after removing a crease), we define the total magnitude of the change as

$$\sum_i \left(\mathcal{N}(X)_i - \mathcal{N}(\tilde{X})_i\right)^2 \qquad (1)$$

where i runs over all pixels. The hope is that these unphysical perturbations to the input can provide insight into the working of the network. However, as stated above,

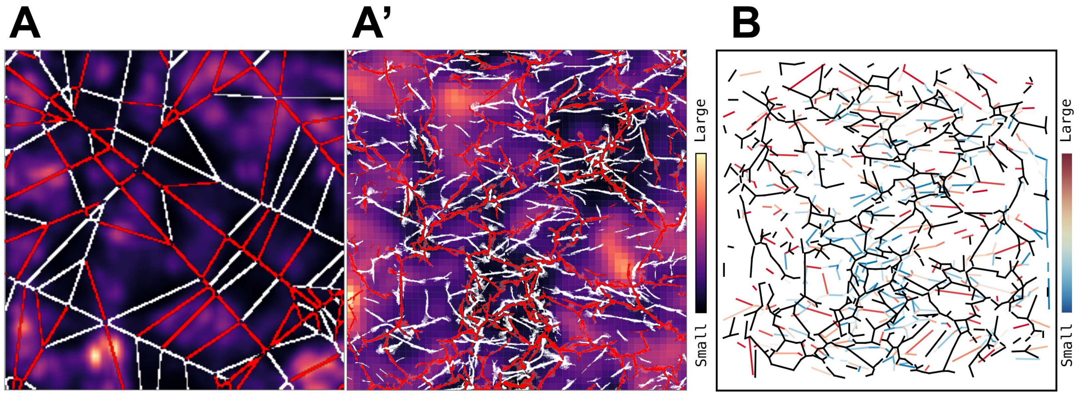


Fig. S4. **Additional test results.** **A** The result of approximate differentiation (see text in Sect. IV) on flat-fold (**A**) and crumpling (**A'**) inputs. Unfortunately, experimentally testing these results or correlating them with other physical quantities proved difficult. **B** Creases colored by the magnitude of the change caused by their removal (Eq. 1). Cooler colors correspond to weaker change and warmer colors to stronger change. While some trends are clearly discernible (e.g. there is a strong correlation between the change magnitude and the crease length), we are still trying to interpret these results in terms of the underlying physics.

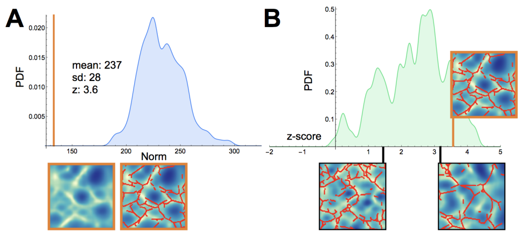


Fig. S5. **Prediction accuracy.** **A** The loss (orange line) of a given reconstruction (bottom) compared to the of losses distribution from all other patches from similarly crumpled sheets. From this data we calculate the z-score of this patch to be 3.6. **B** Repeating this procedure for all patches, we calculate the distribution of z-scores, giving an average z-score of nearly 3. Three representative patches are shown at their z-score location.

interpreting them should be done with care, since in these cases the input to the neural net might be too dissimilar to anything in the training data, making predictions less reliable.

Section S5. Another approach to error quantification

It is common to benchmark Machine Learning prediction accuracies with respect to a suitably-defined random guess. For example, in the `MNIST` digit recognition task, making random choices will achieve a 10% accuracy, because there are only ten classes to choose from. In our case, however, there exists no generative model for crease networks, so there is no random guess that we can compare the output of our network to. As a surrogate, we can draw a random crease network from our data. That is, we compare our predictions on a given patch to many patches from other, similar experiments. This is presented in fig. S5: For a given patch, we compute the loss of our prediction (Eq. 1 in the main text) compared to the true value. In fig. S5A we compare this loss with the distribution of losses obtained by comparing other patches to the true value. Examining hundreds of different predictions, in fig. S5B, we find that our predictions have an average z-score of nearly 3. The z-score for a patch is defined as $z = (\mu - L)/\sigma$ where L is the loss for this patch and μ, σ are, respectively, the mean and standard deviation of all losses calculated from other patches on the same true value. We find that the prediction returned by the net is substantially better than patches taken from other experiments.

Additionally, we can compute the Pearson correlation between the distance transform of the input and target, as well as between our prediction and the target. In the following table we show, for four representative crumple iterations, the Pearson correlation between the target distance map and either the distance map of the input or the network prediction.

Iteration	Input distance map	Prediction
1	0.44	0.68
3	0.35	0.66
6	0.31	0.54
11	0.52	0.74

It is seen that our prediction is significantly better than simply returning the distance transform of the input.

Section S6. Perturbing the in silico data

As discussed in the main text, we assessed the sensitivity of the prediction accuracy to perturbing the *in silico* data. In fig. S6 we present examples of perturbed crease networks (panels A-D) and the resulting validation loss as a function of the number of times the sheet was crumpled. It is seen that all perturbations lead to inferior predictions.

Perturbations were performed in the following manner:

1. Maekawa's theorem was violated by taking flat-folding networks and randomly reassigning curvatures (ridge/valley) to each crease. On average, Maekawa's rule is violated in 50% of the vertices.

2. Kawasaki's theorem was violated by perturbing the position of the vertices, while keeping the topology of the network fixed and ensuring that creases do not cross each other. This results in alternate angles that no longer sum to π. The average absolute deviation of from π is 0.4, amounting to $\sim 13\%$ change. The code is available on GitHub.

3. Finally, both rules were violated by combining both procedures 1-2.

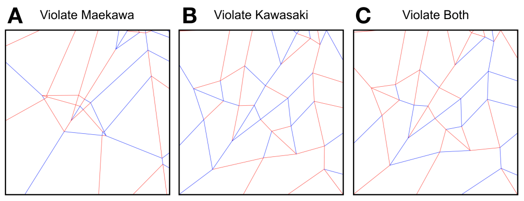


Fig. S6. **Examples of perturbed in silico data. A-C** One realization of each perturbed in silico data set, corresponding to the perturbations described in Sec. VI. Code to generate all types of perturbation available online.

It's the best possible time to be alive, when almost everything you thought you knew is wrong.

Tom Stoppard, Arcadia

4 Generative Model for 3-D Crystal Structures

4.1 Background

Generative models have proven to me remarkably effective in a wide variety of domains including image generation, text generation, and graph generation. Tools from this area brought many new words to the public vernacular, largely as generative adversarial networks (GANs) showed that photo-realistic images could be created of a wide variety of subjects.[64]

Generative models have also been particularly transformative in the sciences– particularly in the discovery of molecules. Drug-like molecules are of tremendous importance and their discovery has been very expensive, time consuming, and often a matter of luck. The hypothesized space of drug-like molecules is enormous with estimates beginning at 10^{40-60} possible candidates.[112] The idea, at a high level, is to use machine learning to learn a compressed representation of known molecules that follows a known distribution. Then, molecules can be reverse engineered by "decoding" samples of the known distribution. By attempting to encode additional auxiliary information in this compressed representation of molecules the hope is that one can perform an optimization procedure in this compressed space and then generate molecules with specific properties. A different approach is to condition the encoded latent space on properties of interest. In both cases, the key point is coming up with a useful representation of the molecules. Early work encoded and decoded the SMILES representation of molecules[160]– a string representation that works well for organic molecules.[61] However, this representation has many problems stemming from the fact that a specific grammar must be learned, and as a result, many random samples do not actually decode to physically realizable molecules. More recently, graph representations have been favored[82] as they do not suffer from generated molecules being unphysical.

For small drug-like molecules, these two representations allowed for tremendous progress. Drug discovery has been greatly impacted by AI with remarkable results in terms of the discovery, testing, and synthesis of new molecules.[172,129,31] Another important class of structures are crystal structures. Unlike many drug like molecules, crystal unit cells are inherently 3-D structures. Additionally, crystal structures contain elements found throughout the periodic table as opposed to drug-like molecules which tend to occupy a relatively small chemical space.[22,122] Combined, these two differences make the representations used for smaller molecules unhelpful here. An additional difficulty is that in addition to encoding and decoding the location and identity of atoms in a crystal unit cell, six degrees of freedom are also required. Specifically, the crystal unit cell side lengths (⍰, ⍰, ⍰) and angles (⍰, ⍰, ⍰) need to be encoded as well. In this work, we propose a representation that we show overcomes many of the difficulties that need to be addressed. We are able to accurately reconstruct the location of atoms in space as well as their approximate identity.

4.2 Contributions

Of the projects in my PhD, this perhaps represents the single project I feel the most ownership of, in fraction of the work done. I wrote all of the code used in the manuscript, generated the figures, and wrote the paper. This project also represented a substantial departure from my typical set of computational tools. Through this project, I learned a great deal. With Louis, we read and discussed the literature to understand what had already been done. Papers from many different fields were relevant, so reading and discussion was very important. Jian Tang and Yoshua Bengio advised me on the project and Jean Michel Sellier supervised Louis. Yoshihide Sawada provided the dataset and the code to generate the dataset from his collaborators.

4.3 Preprint

DATA-DRIVEN APPROACH TO ENCODING AND DECODING 3-D CRYSTAL STRUCTURES

Jordan Hoffmann^{1,2} **& Louis Maestrati**^{1} **& Yoshihide Sawada**^{1,3} **&**
Jian Tang^{1,4,5} **& Jean Michel Sellier**^{1} **& Yoshua Bengio**^{1,6,7}
^{1} Mila, Montréal, Canada ^{2} Harvard University, USA ^{3} Panasonic, Japan ^{4} HEC Montréal, Canada
^{5} CIFAR AI Research Chair ^{6} Université de Montréal ^{7} CIFAR Senior Fellow
jhoffmann@g.harvard.edu, maestratilouis@gmail.com
sawada.yoshihide@jp.panasonic.com, jian.tang@hec.ca
jeanmichel.sellier@mila.quebec, Yoshua.Bengio@mila.quebec

ABSTRACT

Generative models have achieved impressive results in many domains including image and text generation. In the natural sciences, generative models have led to rapid progress in automated drug discovery. Many of the current methods focus on either 1-D or 2-D representations of typically small, drug-like molecules. However, many molecules require 3-D descriptors and exceed the chemical complexity of commonly used dataset. We present a method to encode and decode the position of atoms in 3-D molecules from a dataset of nearly 50,000 stable crystal unit cells that vary from containing 1 to over 100 atoms. We construct a smooth and continuous 3-D density representation of each crystal based on the positions of different atoms. Two different neural networks were trained on a dataset of over 120,000 three-dimensional samples of single and repeating crystal structures, made by rotating the single unit cells. The first, an Encoder-Decoder pair, constructs a compressed latent space representation of each molecule and then decodes this description into an accurate reconstruction of the input. The second network segments the resulting output into atoms and assigns each atom an atomic number. By generating compressed, continuous latent spaces representations of molecules we are able to decode random samples, interpolate between two molecules, and alter known molecules.

1 INTRODUCTION

Generative models have recently seen tremendous success in generating 2-D images of every day objects (Kingma and Welling, 2013; Goodfellow et al., 2014; Brock et al., 2018; Razavi et al., 2019). The size and accuracy of generated results has greatly improved to the point where samples from the latent space decode to photo-realistic samples (Brock et al., 2018; Razavi et al., 2019). A very exciting and important future avenue for generative models is the generation of 3-D structures, like in the world around us. Adversarial networks and autoencoders have been extended into 3-D and have shown they are able to encode useful representations of everyday objects (Wu et al., 2016; Zhu et al., 2018; Brock et al., 2016; Achlioptas et al., 2017; Valsesia et al., 2018). Representations using point clouds have gained popularity as a way to capture the underlying distribution of different types of objects (Achlioptas et al., 2017; Yang et al., 2019).

As machine learning approaches are able to understand and recreate the underlying distributions for many different types of objects in our world, the application of these tools in the physical sciences is a very exciting direction. A clear field where generative models can have a tremendous impact is in material discovery, which matters for example to design new batteries or carbon capture devices for fighting climate change.

One very successful area of applying generative models to the sciences has been the field of drug discovery (Jin et al., 2018; Gómez-Bombarelli et al., 2018; Assouel et al., 2018) (see Schwalbe-Koda and Gómez-Bombarelli (2019) for an excellent summary of many methods). The success of machine learning in this domain has been enormously helpful as a process that was often painstakingly slow has been rapidly accelerated in searching an unimaginably large space of possible drug compounds, estimate to be up to 10^{60} (Polishchuk et al., 2013). Additionally, excellent datasets such as QM9 (Ramakrishnan et al., 2014) and ZINC (Irwin et al., 2012) which contain molecules of interest along with their precomputed properties have allowed for comparing different methods and developing state-of-the-art tools.

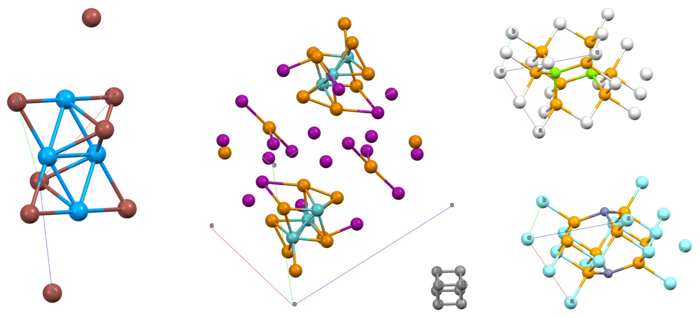


Figure 1: **Examples of crystal unit cells**. Each example shows the unit cell of a crystal. Different colors represent different atomic species. A red, green, and blue line represent the 3 axes of the crystal. Note that they vary in length and angle. Additionally, some unit cells have just one or two atoms (excluding equivalent positions due to translation) while others have nearly 100. Visualizations were made with Mercury (Macrae et al., 2008).

By learning a compressed, useful, latent space representation, this enormous space of molecules can be embedded in a simpler latent space that is easier to search. Using generative models, compounds hypothesized to have specific properties can be rapidly generated and then a targeted subset of these can be experimentally tested. For example, in Gómez-Bombarelli et al. (2018), an auxiliary property prediction task is introduced for a network separate from the encoder/decoder. This allows for optimization of properties in the latent space and then the resulting latent space vector is decoded into a candidate molecule that can undergo more rigorous computational testing before it is experimentally synthesized.

Most of the work combining generative models with chemical discovery has focused on molecules that can be represented in either 1 dimension (such as a `SMILES` string (Weininger, 1988)) or by leveraging a 2 dimensional representation of a molecule, such as a graph. While 1-D and 2-D representations have been very successful for many drug compounds, these representations are not sufficient to describe all molecules. For example, it is possible for different molecules, with very different properties, to have identical graphs (Gebauer et al., 2019). Additionally, there are more complex classes of compounds where the 3-D structure is integral to the molecules properties. Therefore, in this work we focus on generating 3-D representations of molecules. Generating 3-D structures is still a relatively nascent field, when compared to image and text generation. The data requirements for modeling 3-D structures are larger than their 2-D counterparts and there are fewer standard datasets (Nguyen-Phuoc et al., 2019). There is an extra dimension in 3-D problems, providing additional symmetries that often need to be learned (Weiler et al., 2018).

In addition to drug discovery, a promising avenue for molecular design is for minimizing environmental impact through the design of more efficient or more environmentally compounds for a variety of applications. Designing more efficient materials for photo-voltaic panels and more environmentally friendly materials for batteries are both very exciting research directions for using machine learning as a tool towards mitigating climate change (Niu et al., 2015; Tabor et al., 2018; Gebauer et al., 2019; Rolnick et al., 2019). Many of the compounds of interest are crystal structures made of a single small sub-unit that repeats in all directions. This sub-unit is referred to as a “unit cell” and can vary in size and shape as well as internal arrangement and chemical composition.

Preprint. Work in progress.

In this work, we present an alternative to standard methods for encoding 3-D chemical structures. We directly encode and decode 3-D volumes of density from two different data representations. In the first, we use single unit cells, as shown in Fig. 1. Each unit cell is centered in the cube, but randomly rotated. In the second representation, we repeat the unit cell along each axis such that the resulting sampled cube contains repeated unit cells of crystal structures, like those shown in Fig. 2 (and Fig. 5C). Many very interesting and important molecules, such as material for solar panels or batteries, are composed of crystalline units. We train a variational autoencoder (VAE) and a network to segment the decoded output based on the true locations of atoms in tandem. By coupling these two tasks, we are able to accurately encode and decode 3-D atomic positions and species. As far as we are aware, this is not an explored direction as a means to represent molecules for encoding and decoding their 3-D arrangement. We summarize our contribution as follows:

- We propose a method for encoding and decoding 3-D structures. By jointly training a VAE and a segmentation network on the output of the decoder, we train the entire network in an end-to-end fashion.
- We consider two different problems: (1) encoding and decoding single unit cells and (2) encoding and decoding repeated unit cells. In both cases we accurately reconstruct the locations of atoms. We also achieve good results in atom identification and future work will improve this front. By sampling from $\mathbf{z} \sim \mathcal{N}(0, 1)$, we generate complex structures that have physically realistic spacing between atoms.

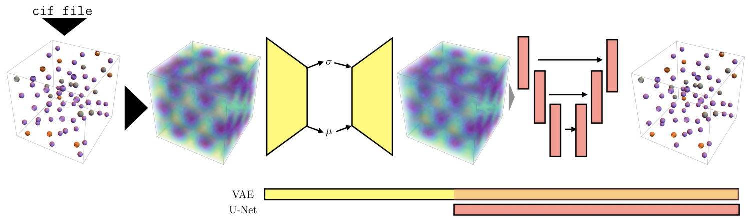


Figure 2: **Network Architecture**. We encode and decode a $30 \times 30 \times 30$ voxel grid representing 10 Å on each side. Each voxel contains the value of the density. The output of the decoder is passed into a 3-D U-Net. We train the two models in parallel. In the schematic, we show the crystal represented as a repeated unit cell rather than a single unit cell. The black arrows indicate deterministic transformations. From the `cif` file, the species matrix is constructed. From this, the density matrix is computed.

2 Preliminaries and Related Work

There are many classes of compounds that are of interest for data-driven discovery. There has been significant work exploring the generation of molecules with potential medicinal properties. These molecules tend to be organic molecules and can be represented efficiently using a string or a graph. A more complex class of molecules are inorganic crystal structures which vary in complexity across many different axes. Crystals are materials that are made up of a repeating pattern of a simpler "unit cell." Crystal structures are of key interest for many environmental problems, such as materials for solar panels and batteries (Niu et al., 2015).

In contrast to problems in drug discovery, many crystals are not composed solely of organic molecules. Crystals commonly contain many heavy (non-Hydrogen) atoms which make various quantum mechanical calculations of their energetic properties very difficult and time consuming. For example, the common benchmark dataset QM9 for predicting quantum mechanical properties includes over 130,000 molecules contains fewer than 9 "heavy" atoms (Ramakrishnan et al., 2014).

There have been two parallel lines of work applying machine learning to material sciences. The first deals with the prediction of physical properties from a compound without performing a computationally expensive density functional theory (DFT) calculation.

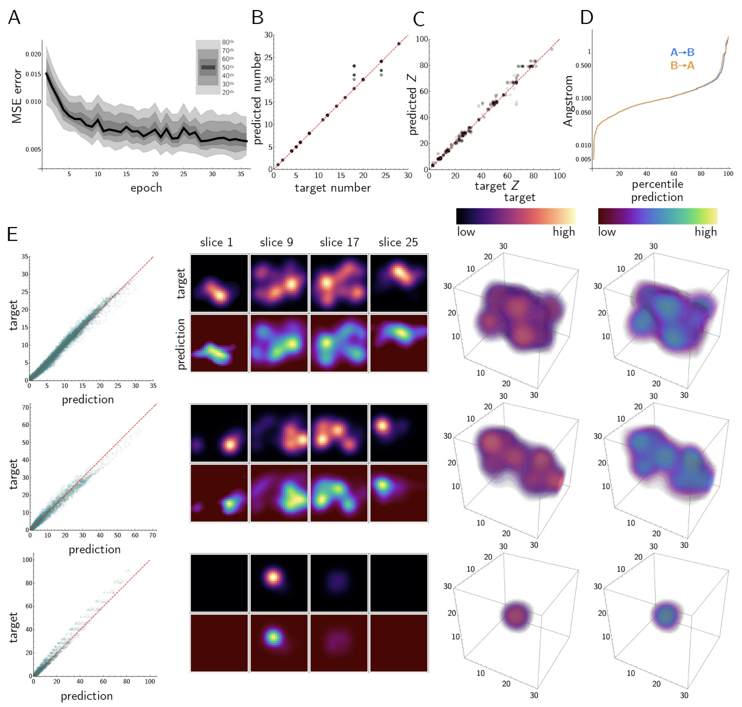


Figure 3: **Single unit cell accuracy**. **(A)** We show the voxel wise reconstruction error (plotted with mean square error) during training. **(B)** For random molecules in the test set, we plot the number of true atoms and the number of recovered atoms after segmentation. **(C)** For the reconstructed atoms, we plot the predicted atomic number versus the atomic number of the nearest true atom. **(D)** We compute the distance from each true atom to the nearest predicted atom and vice-versa (orange and blue, respectively). **(E)** For three different crystals we plot the predicted versus reconstructed density at each voxel. We also show 2D slices through the target and prediction, along with the 3-D reconstructions. For plotting, the density on each figure is normalized between 0 and 1 though is not decoded as such.

Recently, Xie and Grossman introduced crystal graph convolutional networks (CGCNN) for accurate prediction of 8 different DFT calculated properties on crystal structures (Xie and Grossman, 2018). MatErials Graph Network (MEGNet) uses graph neural networks (Chen et al., 2019) to predict a variety of energetic properties on both the QM9 dataset (Ramakrishnan et al., 2014) as well 69,000 crystal structures from the Materials Project (Jain et al., 2013). Another network, SchNet, uses 3-D spatial information to directly leverage the interactions between separate atoms. They then use a continuous-filter convolution for accurate property prediction (Schütt et al., 2018). These methods, and many more, have all achieved impressive results on many standard benchmark datasets. Cubuk, *et al.* use transfer learning to search an enormous space of molecules for promising Lithium ion conductors (Cubuk et al., 2019).

Another line of work is concerned with generating molecules, often molecules that have specific properties. In the past decade, generative models for drug discovery have achieved impressive results. Typically a representation of a compound, such as a `SMILES` string (Gómez-Bombarelli et al., 2018; Segler et al., 2017) or a graph (Jin et al., 2018; Assouel et al., 2018; Mansimov et al., 2019) is encoded and decoded. Then, by sampling the resulting latent space, novel molecules can be generated. Either by performing an auxiliary task with the latent space (as done in Gómez-Bombarelli et al. (2018)) and then performing optimization or by conditioning the latent space on desirable properties, novel chemical structures are obtained.

One possible representation of drug-like molecules is a `SMILES` string. However, a difficulty faced using the `SMILES` representation is ensuring that the decoder decoded a valid `SMILES` string (Gómez-Bombarelli et al., 2018; Kusner et al., 2017). Additionally, there is not a strong notion of distance between molecules and their `SMILES` representation (Jin et al., 2018). One approach, ChemTS, uses `SMILES` strings along with recurrent neural networks and Monte Carlo Tree Search to better ensure generated strings decode to real molecules (Yang et al., 2017). By explicitly penalizing invalid `SMILES` strings in their reward function they are able to bypass the difficulty in their decoded molecules being non-physical.

More recently, graph based methods have proven very successful in generating synthetically attainable molecules. By building a database of molecule fragments (like LEGO bricks), a graph is formed based on the connectivity of different structures (Jin et al., 2018). These methods have far fewer difficulties ensuring that the decoder produces physically valid molecules. For generation of physical molecules, this is a very important consideration, as there are many hard constrains that most be obeyed. Other graph based approaches include flow based models (Madhawa et al., 2019). Also using graph neural networks, E. Manismov and collaborators developed a method to, given a graph, recreate conformations of a 3-D molecule and predict its energetic properties (Mansimov et al., 2019). In Mansimov et al. (2019), the authors propose a graph-based generative method that is able to generate molecules with desired target properties. Impressively, they recover 3-D positional information for the atoms and show they achieve the required accuracy for relaxation.

An additional challenge for generative models in the physical sciences is ensuring that samples from the latent space, $\mathbf{z}$, decode into physically plausible objects. The decoded objects need to obey physical constraints and an object that may appear physical is not certain to actually be experimentally realizable. In many domains this is not of specific concern, for example a generated animal is scored on some likelihood based on its visual appearance, not whether or not the generated creature is genetically possible. Recently, there have been generative models leveraging the 3-D nature of the problem. G-SchNet, a generative model for 3-D molecules (Gebauer et al., 2019) leverages the SchNet architecture- a start-of-the-art property prediction network (Schütt et al., 2018). They show they are able to generate molecules, placing atoms in 3-D space in a rotationally invariant manner. By appropriately conditioning the placement of new molecules based on the location and identity of previous molecules, they ensure the symmetries that are required for the molecules to relax in a DFT calculation. They propose expanding G-SchNet to generating crystal structures as a future direction.

While there is a rich literature in the generation of organic compounds, recently there has been work in the generation of more complex crystal structures, such as CrystalGAN (Nouira et al., 2018). The authors of CrystalGAN introduce geometric constraints and show that their method is able to produce stable structures compared to methods that do not include such domain knowledge such as DiscoGAN (Kim et al., 2017). CrystalGAN uses a dataset with a much larger selection of elements than much previous work, but they limit themselves to molecules of particular chemical structure.

3 Methods

We use a dataset containing 46,744 `cif` files which contains the information describing the unit cells of properly relaxed crystal structures from the Materials Project (Jain et al., 2013; Xie and Grossman, 2018). We use 80% of the data for testing and the other 20% for training. Using the Python library `pymatgen` we preprocess all of the data into the density 3-D matrices (Ong et al., 2013) described below. The boundary box of a crystal structure is controlled by six different degrees of freedom (see Fig. 1). Each side can have a different length and the angles between the three sides is also variable. Additionally, the internal complexity of each crystal can also vary widely– some unit cells in our dataset contain a single atom while other unit cells can have over one hundred different atoms. This tremendous variability in structure makes a universal representation difficult.

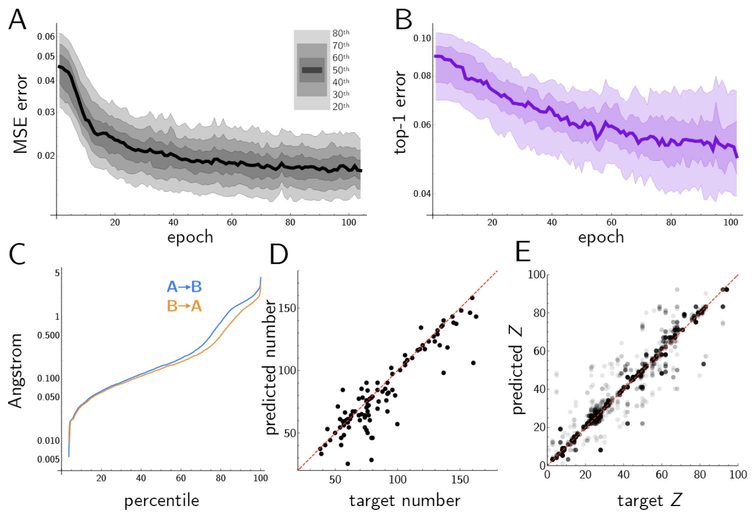


Figure 4: **Repeating unit cell accuracy**. **(A)** For each position in 3-D space, we compute the difference between the truth and the reconstruction. We show different percentile bands of the reconstruction error between the target and predicted density, plotted using the mean square error (MSE). **(B)** For the species matrix from U-Net we ask what the error of top-1 predictions is between our predictions and the ground truth. **(C)** For our segmented matrices, we ask the distance from the nearest segmented atom to the ground truth. We show the distance errors by percentile for both the nearest true atom to each predicted atom and vice-versa (orange and blue, respectively). **(D)** After segmenting the reconstructed density maps we show the predicted and true number of atoms. **(E)** We plot the predicted nearest atom species versus the true species of the closest corresponding atom, as long as the distance is less than 0.5 Å. We find 65.4% are correctly predicted.

3.1 Data Representation

As a first step for the problem of generating novel, physical crystal structures we begin with the simpler problem of encoding and decoding physical locations of atoms in space. We begin by considering a cube with side length 10 Å, which we represent by M. We divide this cube into 30 equally spaced bins, resulting in a $30 \times 30 \times 30$ cube with each voxel representing 0.33 Å on each side (see Fig. 2 and Fig. 3E). This data representation is similar to many computer vision tasks, so we use similar convolution based network architectures.

We consider the crystal structures in our dataset where the maximum side length is less than 10 Å. We consider **two different data representations** of different complexity. In the first, we shift each single unit cell to the center of our grid, then we randomly rotate it. In this representation, we encode and decode a single unit cell. We randomly sample 3 different rotations for each crystal. In this case, the encoded structures typically have between 1 and 30 atoms, with a few reaching between 50 and 100. In the second representation, we repeat these unit cells in each direction. This means that each cube contains at least one unit cell, but some will contain more than one cell. We choose one representation where a unit cell begins at $(0, 0, 0)$ and randomly sample 2 other cubes in this space resulting in a dataset of over 100,000 examples. In this representation, we typically encode the locations of between 20 to 100 atoms, with some having over 200. An advantage of our representation is that we are agnostic to the number of atoms that we are encoding and decoding.

Preprint. Work in progress.

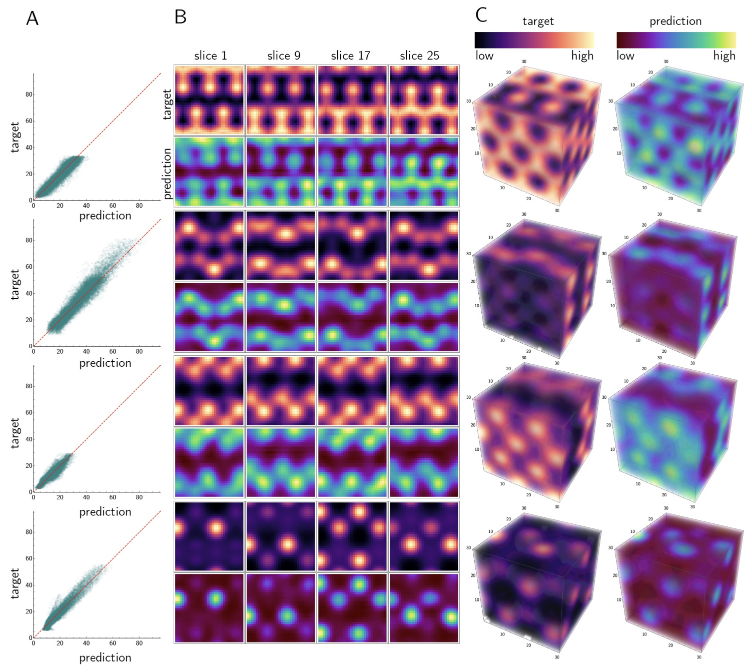


Figure 5: **Accuracy of Model**. **(A)** For each position in 3-D space, we plot the predicted and target density for 4 different random crystals from the test set. The red dashed line is an identity line. **(B)** For each of the panels in **(A)**, we show 4 different z-slices through the true and predicted density fields. **(C)** We show the full 3-D reconstruction of the prediction and the true density field.

Using both these representations, we compute a density field, where each pixel (i, j, k) is defined to have value

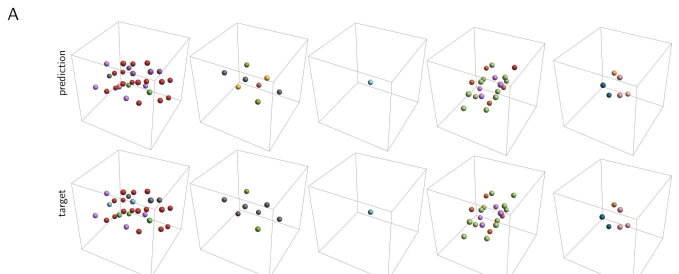
$$\boldsymbol{M}_{i,j,k} = \frac{1}{\sigma^3(2\pi)^{3/2}} \sum_m Z_m \exp\left(-\frac{d(\vec{Z}_m, (i,j,k))^2}{2\sigma^2}\right) \tag{1}$$

Where $d(\cdot,\cdot)$ represents the Euclidean distance between the two arguments. $\vec{Z}_m$ represents the 3-D coordinates of atom Z_m. We set σ to 1.0 Å. When plotting, we multiply the output by $\sigma^3(2\pi)^{3/2}$. We do this because when $\sigma = 1$ this results in the value of Z (the atomic number) being present at its location in space. Because of the structure of this representation, when the density of molecules increases, there are more interactions making it harder to reconstruct the true identity of an atom. Ultimately, our aim is to be able to reconstruct the location and species of the different atoms in this grid. To that end, we also construct a species matrix, $\boldsymbol{S}$, where each voxel is either a 0 or equal to the atomic number of an atom that is within 0.5 Å. We construct two neural networks that attempt to learn $\boldsymbol{M}$ and $\boldsymbol{S}$ in parallel. These two networks are trained together and are detailed in the following section.

3.2 Network Architecture

We use a variational autoencoder (Kingma and Welling, 2013) to encode and decode our 3-D density maps, $\boldsymbol{M}$. We use a β multiplier on the Kullback-Leibler (KL) loss term to encourage no correla-

A



B

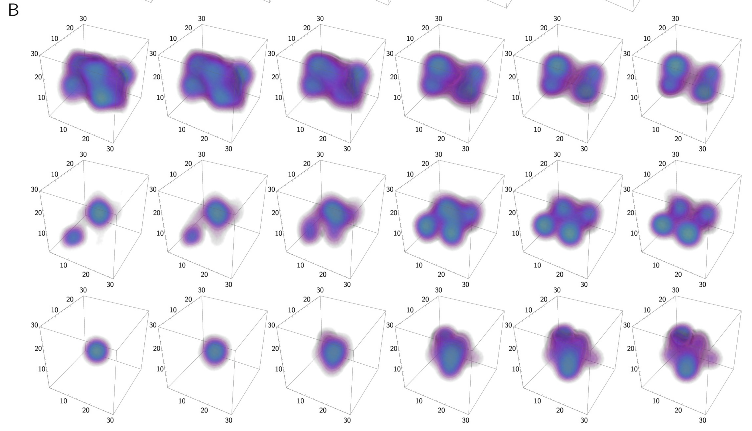


Figure 6: **Species reconstruction and latent space interpolation. (A)** For five different randomly selected crystal unit cells we show the target and our reconstruction. Atoms are colored using default atomic colors, atoms with similar atomic number do not necessarily have similar colors. **(B)** We show the reconstruction of latent space interpolation between two molecules for 3 random sets of targets. For a video, see the supplemental materials.

tions between different elements of the latent space (Higgins et al., 2017). Our encoder, $E(\cdot)$, is a convolutional neural network. The decoder, $D(\cdot)$, uses upsampling and convolutions in favor of transposed convolutions to avoid checkerboard artifacts (Odena et al., 2016). We use batch normalization (Ioffe and Szegedy, 2015), LeakyReLU activations, and the code is written in Pytorch (Paszke et al., 2017). The optimization is done using the Adam optimizer with a learning rate of 10^{-5} (Kingma and Ba, 2014). We use a latent space size of 300 for all experiments. This means that when considering a repeating unit cell, more information needs to be stored.

Simultaneously with training the VAE, we train a 3-D U-Net segmentation model, $\mathrm{U}_{\mathrm{net}}(\cdot)$, with an attention mechanism to segment the output of the decoder (Ronneberger et al., 2015; Oktay et al., 2018). For accurate segmentation, especially in the case of a repeating lattice, it is important to be able to capture the dependencies between different atoms. We found that a U-Net model worked well, though future work (discussed later) will explore sequentially classifying atoms. We include a weighted (by γ) loss from the segmentation in the loss of the encoder/decoder. In our experiments, we set γ to 0.1. We experimented with $\gamma = 0$ and $\gamma = 0.33$ and found that 0.1 proved an acceptable intermediate. With $\gamma = 0.33$ we found slightly improved segmentation results (approximately a 6%) improvement, but surprisingly slightly worse density reconstruction results. When we completely removed the effect of the species loss from the density reconstruction, species reconstruction results degraded by around 4% and the changes in density reconstruction were negligible. This allows us to train the entire network in an end-to-end fashion. The entire model is shown in Fig. 2.

We use three terms in the loss function for the VAE with the most weight given to the reconstruction of the density matrices. The segmentation network is only concerned with the final segmentation of the reconstruction from the decoder. Currently, we treat each atom type as a different class, with an additional (most common) class for "no atom." The loss function used for the VAE is

$$\mathcal{L}_{\text{VAE}} = L_{\text{RE}}(\hat{\boldsymbol{M}}, \boldsymbol{M}) + \beta(D_{\text{KL}}(q(\mathbf{z}|\boldsymbol{M})||p(\mathbf{z}))) + \gamma L_{\text{BCE}}(\hat{\boldsymbol{S}}, \boldsymbol{S}). \tag{2}$$

The first term is the reconstruction error and the third term is the binary cross entropy loss from the segmentation. The second term is the Kullback-Leibler divergence between the prior (set to $p(\mathbf{z}) = \mathcal{N}(0,1)$) and $q(\boldsymbol{z}|\boldsymbol{M})$ the posterior of the latent vector given input $\boldsymbol{M}$. We use a loss given by

$$\mathcal{L}_{\text{U-Net}} = L_{\text{BCE}}(\hat{\boldsymbol{S}}, \boldsymbol{S}) \tag{3}$$

for the segmentation. In the above expression, $\boldsymbol{M}$ is the input density field. $\mathbf{z} = E(\boldsymbol{M})$ and

$$\hat{\boldsymbol{M}} = D(\mathbf{z}) \tag{4}$$

is the reconstructed density field. The variable $\boldsymbol{S}$ represents a 1-hot species matrix. Lastly,

$$\hat{\boldsymbol{S}} = \text{U}_{\text{net}}\left(\hat{\boldsymbol{M}}\right) \tag{5}$$

represents the probability matrix from the segmentation routine, which will be of shape: [batch $\times$ classes $\times$ x-dim $\times$ y-dim $\times$ z-dim].

4 RESULTS

We find that we are able to accurately encode and decode 3-D representations of density fields (see Figs. 3 and 4). Using the segmentation network, we segment the output of the decoder into distinct atoms. Using a trained network, we show that random samples from the latent space decode to samples that obey many of the same statistics as the training distribution (see Figs. 7 and 9). Additionally, in the case of a repeating unit cell, we alter our training routine to condition the generation of molecules on the largest atomic number present.

4.1 ACCURACY ON A UNIT CELL

In Fig. 3 we plot the results of applying our model to single unit cells of crystals. We find out model is able to very accurately segment the locations of molecules with nearly 99% of atoms placed within 0.5 Å of their true location. Nearly 90% of unit cells are reconstructed with the correct number of atoms and 97% of unit cells are reconstructed with less than 2 atoms extra or missing. 66% of species are correctly classified in test set, with nearly 100 possible classes. When an atom is incorrectly classified, it is usually by one or two atomic numbers (98% are within 2 atomic numbers). Next, we will discuss the same network applied to repeating unit cells. We go into more detail on the reconstruction errors for repeated unit cells, as these are more complex than single unit cells. In all cases, the results for single unit cells are improvements over the results of multiple unit cells. For most figures, there are corresponding panels between the single unit cell and repeated cell results.

4.2 ACCURACY ON A REPEATING LATTICE

In Fig. 4 we assess the accuracy on a variety of different metrics for both the encoder-decoder network as well as the segmentation of the output.

4.2.1 RESULTS FROM THE ENCODER-DECODER

Our encoder-decoder architecture is able to accurately reconstruct the voxel-wise density (Fig. 4A and Fig. 5). We find a strong correlation between the predicted and target voxel value (Fig. 5A). This is essential for achieving an accurate segmentation of the correct atoms and their corresponding positions. Due to the strong penalty on the KL term we find that there is an increased spread in the

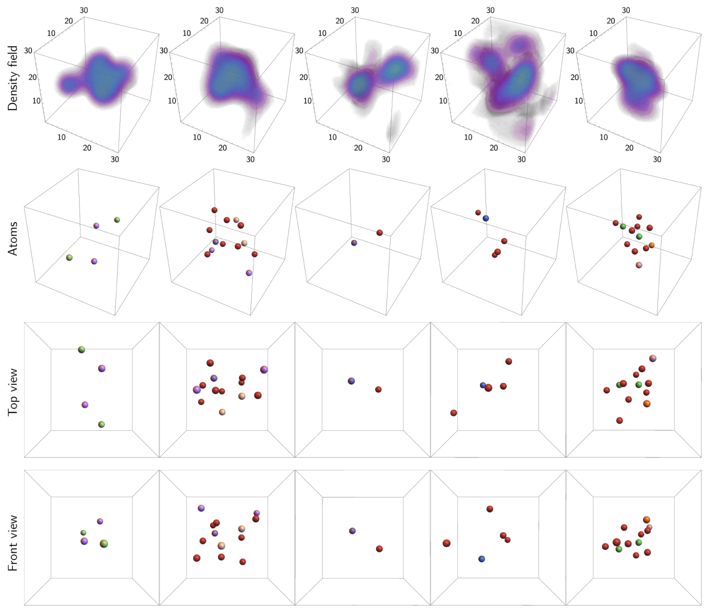


Figure 7: **Decode random latent space vectors.** For 5 different random latent space vectors we show the reconstructed density field and the resulting segmentation. Notice that in some cases small amounts of density are predicted but are not segmented into an atom (for example, in columns 3 and 4).

resulting predictions when attempting to encode and decode repeating unit cells. This is less of an issue when encoding and decoding single unit cells. In Fig. 12, we show reconstructions with $\beta/10$. As expected, reducing the effect of the KL penalty results in less blurry predicted density maps. However, by reducing β we find that there are correlations in our latent space that result in less physical samples when drawing from $\mathcal{N}(0, 1)$ in the case of repeated unit cells.

4.2.2 Results from Segmentation

Using the output of the segmentation network, we take the $\tilde{\mathbf{S}} = \text{argmax}(\mathbf{S})$. From this representation, we find the connected components and use majority voting to assign each cluster an atom identity. In Fig. 4B we show the accuracy of all top-1 predictions. We compare the results of our segmented matrix, $\tilde{\mathbf{S}}$ with the true values, $\hat{\mathbf{S}}$. From the center of mass of each predicted atom i, we compute the distance to the nearest true atom as follows

$$\min_j \, d(\tilde{\mathbf{S}}_i, \hat{\mathbf{S}}_j) \; \forall \; j \text{ and } \min_i \, d(\tilde{\mathbf{S}}_i, \hat{\mathbf{S}}_j) \; \forall \; i. \tag{6}$$

Similarly, we can compute the distance from each true atom to the nearest predicted atom. By computing this metric in both directions, we are able to verify that our model is placing atoms in the correct locations but only in those locations. In Fig. 4C we show the percentiles of all pairwise minimum distances in our reconstructed samples. We find that 50% of all reconstructed atoms are in 0.2 Å. For both directions, the 75th percentile error was under 1 Å and the 90th percentile of reconstructed atoms were within 2 Å. For predicted atoms that are within 0.5 Å of a true atom's location, we find that 65.4% of the time our prediction matches the atomic number exactly Fig. 4E.

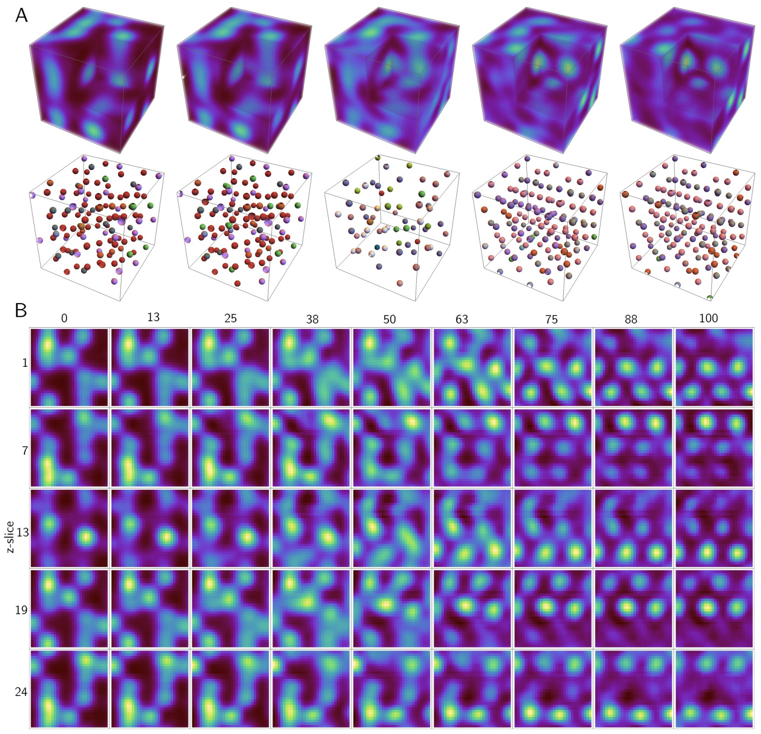


Figure 8: **Interpolation between two molecules**. In **(A)** we show the interpolation between two crystal density maps. We show three equally spaced intermediates the corresponding segmentation. In **(B)**, we show the same interpolation but highlight different two dimensional slices (y-axis). Along the x-axis we label the fraction of the way between the two latent space vectors.

For 120 randomly selected reconstructed crystals from the test set, we compare the number of atoms after segmentation with the true number (Fig. 4D and Fig. 3B). In Fig. 4E we compare the predicted species compared with the species of the nearest true atom, independent of the distance to the nearest atom. We find that based on the results of Fig. 4, our current model is able to more accurately reconstruct the locations of atoms than obtain their true atomic number. This makes sense, given the enormous number of possible classes. However for repeating unit cells, when an atom is correctly predicted to be within 0.33 Å of a true atom location nearly 70% of the time the atom is assigned exactly the correct Z value. The correlation between the true and predicted value is 0.98 suggesting that when the network makes an incorrect assessment, it chooses a nearby atomic number.

Further improvements in this placement and identification of atoms is necessary in achieving molecules that will appropriately relax in a DFT simulation for the calculation of quantum mechanical properties. To be able to generate potentially interesting crystal structures, it is important to ensure that different regions of the latent space decode to physically realistic molecules.

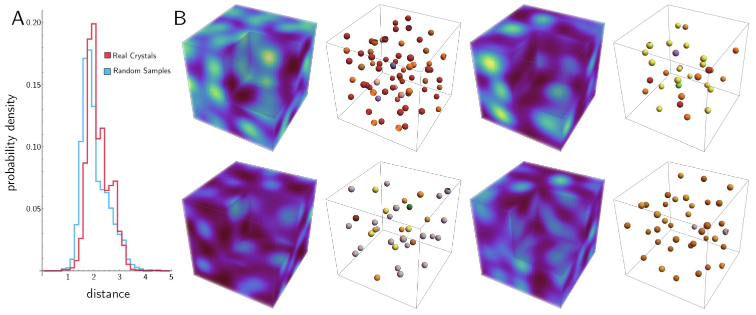


Figure 9: **Random latent space samples**. (**A**) We look at the spacing between nearest atoms from random draws from the latent space compared to those from real crystal structures. In blue we show the distribution of random reconstructions. In red we show the distribution of true inter-atomic spacing. (**B**) We pass $\mathbf{z} \sim \mathcal{N}(0, 1)$ into our decoder and then segment the output.

4.3 Latent Space Interpolation

We encode two different true density maps into vectors $\mathbf{z}_1$ and $\mathbf{z}_2$, respectively. We then linearly interpolate between these two values in latent space, constructing intermediate vectors $\tilde{\mathbf{z}}_i$. We decode the resulting latent space vectors which result in predicted density maps, $D(\tilde{\mathbf{z}})$. Passing these into the trained segmentation routine, we find that intermediate results segment into atoms (see results on a single unit cell in Fig. 3 and Fig. 6). See results on a repeating lattice in 8 and Fig. 11)[1]. More videos of latent space interpolation for both single unit cells and repeating unit cells are available in the Supplement.

4.4 Random Draws

We can randomly sample a latent space vector $\mathbf{z} \sim \mathcal{N}(0, 1)$ and we then decode the output. We do this for both single unit cells as well as for repeating lattices.

4.4.1 Unit cell

Using the network trained to encode and decode single unit cells, we decode a $\mathbf{z} \sim \mathcal{N}(0, 1)$. In Fig. 7 we show the decoded density fields along with the resulting segmented output. There are a few desirable features: single cells are centered, atoms tend to be of similar atomic number, even in different parts of the reconstructed output, and atoms are not ever too close together.

4.4.2 Repeating lattice

In Fig. 9 we plot a variety of random density fields from the latent space along with their segmented counterparts. While the resulting molecules are certainly not accurate enough to relax under a DFT calculation, we plot a histogram of of inter-atom distances. We find that the reconstructions obey a similar intra-molecule distance distribution as found in true samples.

4.5 Conditional VAE

Moving forward, an important step will be creating structures that have specific chemical properties. Moving towards this direction, we attempt to condition the generation of molecules on the largest

[1]See `https://youtu.be/ZpFN5tSo5Pg`
`https://youtu.be/pxYb8cnLxio`
`https://youtu.be/q2d8LZq8RW4`
`https://youtu.be/H3ca2NETRD0`

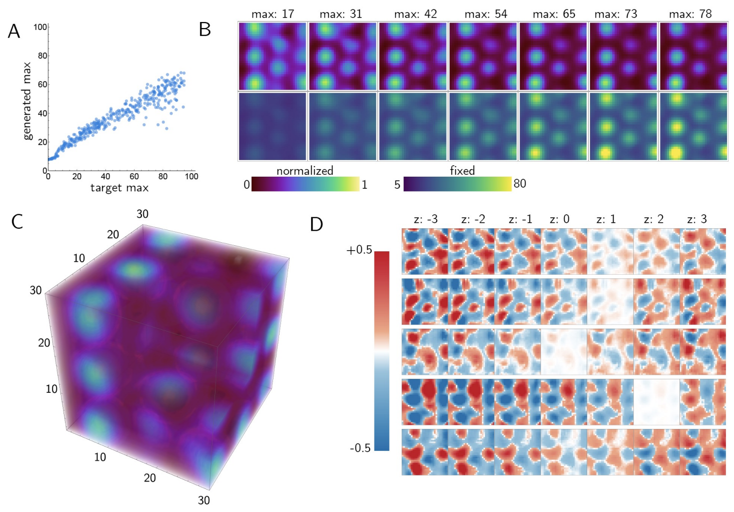


Figure 10: **Conditional generation of molecules**. We multiply the input and output of the bottleneck by the maximum density during training. Then, randomly decoding samples of $\alpha\mathbf{z}$ where $\mathbf{z} \sim \mathcal{N}(0, 1)$ and α is a random target, we find we are able to generate density maps that decode appropriately. If we take a true decoded value and re-scale the output we find we are able to control scale without affecting geometry. **(A)** We show the target and generated max value. **(B)** Using an encoded $\mathbf{z}$, we multiply it by different target values and decode them. We find that we are able to change scale without affecting the geometry. **(C)** We show a decoded sample. We cut away the right most corner facing the viewer. **(D)** For the molecule shown in **(C)**, we show the results of varying one value in the latent space from -3 to 3. We plot the difference in the slice when compared the the unaltered, decoded, $\mathbf{z}$ for a slice that is in-plane with many atoms.

species present (Sohn et al., 2015). To do this, we multiply the input and output of the bottleneck by the largest present density. By doing this, we are still able to get the network to train but find that we are able to control the magnitude of the resulting density field without perturbing the geometry (see Fig. 10B). We attempted to condition the encoder and decoder by concatenation though we found this was not sufficient to generate density fields with the desired property.

The ability to condition on specific properties has obvious applications in the targeted generation of molecules. In this case, we choose to condition on the largest present density as a way to generate molecules that do not have an atom with more than a specified maximum atomic number present. We find that by varying different parameters in the latent space, some vary the field in such a way that appears to correlate with the location of atoms (see Fig. 10C,D). However, unlike as in work on 2-D problems, the changes in the 3-D scalar field are much harder to interpret (Chen et al., 2016; Higgins et al., 2017). Future work will attempt to use improved factorizing techniques (Kim and Mnih, 2018; Chen et al.). Additionally, one could seek to condition the generation of single unit cells on quantum mechanical properties or by adding an auxiliary loss using the latent space to predict quantum mechanical properties.

5 Conclusions and Future Work

The ability to encode and decode 3-D structures is a very interesting direction of recent work and increasingly important in helping create models that can understand the world that we live in. In current material design, a lot of focus has been on molecules where the 3-D structure can be safely ignored. However, in many cases this is not the case. We do not know of an effective data representation for these 3-D molecules and find that by directly utilizing a proxy for density we are able to encode and decode the geometry of these molecules.

An important future direction of research is to come up with successful representations for encoding crystal structures such that they can be easily encoded and decoded. Coming up with a representation of crystal unit cells (and 3-D structures in general) so that decoded molecules are physically plausible is an important aspect that needs to be carefully considered.

Our approach is currently unable to generate molecules that are physically stable, but there are promising directions in this direction. Using ideas similar to G-SchNet, using the decoded density field, atoms can be placed sequentially, conditioned on the placement of previous atoms (Gebauer et al., 2019). Working towards decoding molecules that are able to relax is a very exciting direction.

Another improvement would be to alter the structure of our encoder/decoder. Currently, we use standard 3-D convolutions. However, using SE(3) equivariant kernels from M. Weiler *et al.* is a promising direction for improved performance (Weiler et al., 2018). We feel that our current approach could be extended beyond the domain of material science, as we are able to encode a 3-D distance map, this approach could be broadly applied to many objects. In essence, we are able to learn a distance transform from an object. This is more general than for encoding and decoding atomic structures.

To try to facilitate further research in the creation of 3-D representation of more complex atomic structures, we will release the code shortly.

6 Acknowledgements

We would like to thank Anirudh Goyal, Jian Tang's group, Rim Assouel, and Maksym Korablyov for useful input and discussion. J.H. was supported by the Harvard Faculty of Arts and Sciences Quantitative Biology Initiative.

References

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

Rim Assouel, Mohamed Ahmed, Marwin H Segler, Amir Saffari, and Yoshua Bengio. Defactor: Differentiable edge factorization-based probabilistic graph generation. *arXiv preprint arXiv:1811.09766*, 2018.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

Ricky TQ Chen, Xuechen Li, Roger Grosse, and David Duvenaud. Isolating sources of disentanglement in vaes.

Xi Chen, Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In D. D. Lee, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances In Neural Information Processing Systems 29*, pages 2172–2180. Curran Associates, Inc., 2016.

Ekin D Cubuk, Austin D Sendek, and Evan J Reed. Screening billions of candidates for solid lithium-ion conductors: A transfer learning approach for small data. *The Journal of chemical physics*, 150(21):214701, 2019.

Niklas WA Gebauer, Michael Gastegger, and Kristof T Schütt. Symmetry-adapted generation of 3d point sets for the targeted discovery of molecules. *arXiv preprint arXiv:1906.00957*, 2019.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 2(5):6, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52 (7):1757–1768, 2012.

Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, et al. Commentary: The materials project: A materials genome approach to accelerating materials innovation. *Apl Materials*, 1(1):011002, 2013.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1945–1954. JMLR. org, 2017.

Clare F Macrae, Ian J Bruno, James A Chisholm, Paul R Edgington, Patrick McCabe, Elna Pidcock, Lucia Rodriguez-Monge, Robin Taylor, J van de Streek, and Peter A Wood. Mercury csd 2.0–new features for the visualization and investigation of crystal structures. *Journal of Applied Crystallography*, 41(2):466–470, 2008.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Elman Mansimov, Omar Mahmood, Seokho Kang, and Kyunghyun Cho. Molecular geometry prediction using a deep generative graph neural network. *arXiv preprint arXiv:1904.00314*, 2019.

Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. *arXiv preprint arXiv:1904.01326*, 2019.

Guangda Niu, Xudong Guo, and Liduo Wang. Review of recent progress in chemical stability of perovskite solar cells. *Journal of Materials Chemistry A*, 3(17):8970–8980, 2015.

Asma Nouira, Jean-Claude Crivello, and Nataliya Sokolovska. Crystalgan: Learning to discover crystallographic structures with generative adversarial networks. *arXiv preprint arXiv:1810.11203*, 2018.

Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention u-net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson, and Gerbrand Ceder. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27(8): 675–679, 2013.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022, 2014.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *arXiv preprint arXiv:1906.00446*, 2019.

David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. URL `http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a`. (available on arXiv:1505.04597 [cs.CV]).

Kristof T Schütt, Huziel E Sauceda, P-J Kindermans, Alexandre Tkatchenko, and K-R Müller. Schnet–a deep learning architecture for molecules and materials. *The Journal of Chemical Physics*, 148(24):241722, 2018.

Daniel Schwalbe-Koda and Rafael Gómez-Bombarelli. Generative models for automatic chemical design. *arXiv preprint arXiv:1907.01632*, 2019.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1): 120–131, 2017.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pages 3483–3491, 2015.

Daniel P Tabor, Loïc M Roch, Semion K Saikin, Christoph Kreisbeck, Dennis Sheberla, Joseph H Montoya, Shyam Dwaraknath, Muratahan Aykol, Carlos Ortiz, Hermann Tribukait, et al. Accelerating the discovery of materials for clean energy in the era of smart automation. *Nature Reviews Materials*, 3(5):5, 2018.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Learning localized generative models for 3d point clouds via graph convolution. 2018.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, pages 10381–10392, 2018.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016.

Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Physical review letters*, 120(14):145301, 2018.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. *arXiv preprint arXiv:1906.12320*, 2019.

Xiufeng Yang, Jinzhe Zhang, Kazuki Yoshizoe, Kei Terayama, and Koji Tsuda. Chemts: an efficient python library for de novo molecular generation. *Science and technology of advanced materials*, 18(1):972–976, 2017.

Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: image generation with disentangled 3d representations. In *Advances in Neural Information Processing Systems*, pages 118–129, 2018.

7 Supplemental Information

7.1 Videos

7.1.1 Single Unit Cell Videos

```
https://youtu.be/ZpFN5tSo5Pg
https://youtu.be/3yPVdgd2mQ0
https://youtu.be/pxYb8cnLxio
https://youtu.be/ZZbHmZy7Idw
```

7.1.2 Repeating Lattice Videos

```
https://youtu.be/q2d8LZq8RW4
https://youtu.be/U5-x3jL2zcc
https://youtu.be/H3ca2NETRD0
```

8 Training Details

We used a Tesla V100 for training. We used a learning rate of 1e-5 and a batch size of 24.

8.1 Network Details

8.1.1 Encoder

3D convolution, kernel size of 5 and stride of 2. 16 channels.
Batch Norm and LeakyReLU activation.
3D convolution, kernel size of 3 and stride of 1. 32 channels.
Batch Norm and LeakyReLU activation.
3D convolution, kernel size of 3 and stride of 1. 64 channels.
Batch Norm and LeakyReLU activation.
3D convolution, kernel size of 3 and stride of 2. 128 channels.
Batch Norm and LeakyReLU activation.
Fully connected layer with a bottleneck size 300.

8.1.2 Decoder

Fully connected layer.
Reshape to 128 5,5,5
Trlinear upsample by a factor of two.
Conv3D with 64 channels, kernel size of 5, leakyReLU.
Trlinear upsample by a factor of two.
Conv3D with 32 channels, kernel size of 5, leakyReLU.
Trlinear upsample by a factor of two.
Conv3D with 16 channels, kernel size of 4, leakyReLU.
Conv3D with 1 channels, kernel size of 4, ReLU.

9 Accuracy of random draws in a repeating lattice

To test whether random samples from the latent space, $\tilde{\mathbf{z}}$, decode to physically realistic molecules, we trained a discriminator. We trained an auto-encoder that was able to very accurately reconstruct molecules by greatly reducing the penalty on β (see Fig. 12). Then, we compute real latent space representations of our different molecules, $\mathbf{z}$. We then draw a $\lambda \sim U(0,1)$ and construct a new latent space vector

$$\hat{\mathbf{z}} = \lambda \tilde{\mathbf{z}}_{\mathcal{N}(0,1)} + (1-\lambda)\mathbf{z}_{\text{real}} \tag{7}$$

Preprint. Work in progress.

where $\tilde{\mathbf{z}}$ is a random sample from a unit normal and $\mathbf{z}_{\text{real}}$ is a random "true" latent space. We pass this through the decoder and provide this a label $(1-\lambda, \lambda)$. We then randomly draw $\mathbf{z} \sim \mathcal{N}(0,1)$ from our trained network and pass this reconstruction $\tilde{\boldsymbol{D}}$ into the discriminator network, which outputs a prediction of the distance from a true crystal reconstruction versus a random draw from a latent space of a previously trained network. Applying this network to random samples from $\mathbf{z} \sim \mathcal{N}(0,1)$ we get a mean value of 0.84 with a standard deviation of 0.05. For true decoded samples, we find a mean value of 0.90 with a standard deviation of 0.01. In Fig. 13, we show results of the trained network and predictions on three different datasets. We find that the target latent space decodes to high-scoring samples (see Fig. 13B) and that out of distribution samples typically decode to much lower scores.

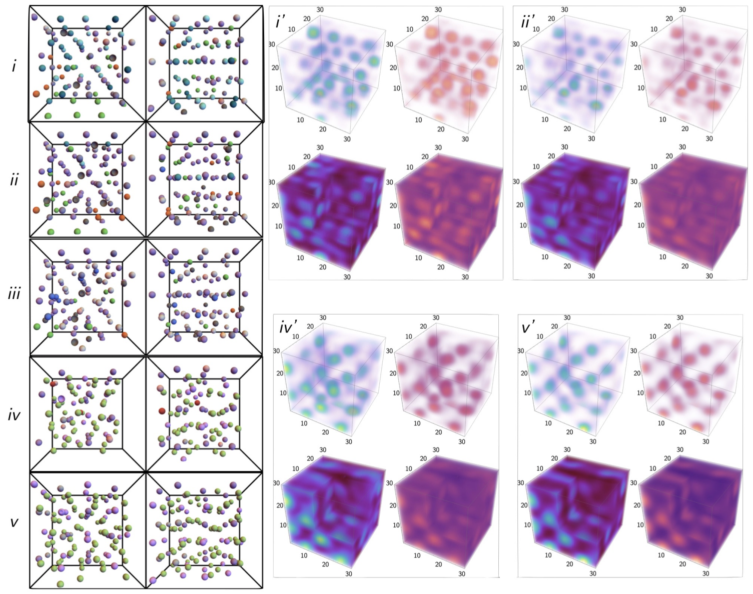


Figure 11: **Interpolation between two molecules**. From $i - v$ we show two views of the output from the segmentation routine decoding the latent space between two different molecules. In the primed corresponding figures we show the density fields that are output by the decoder. We show the output of each grid normalized between 0-1 in blue-green and on a fixed color map in magma. We also vary the opacity to show the locations of predicted atoms.

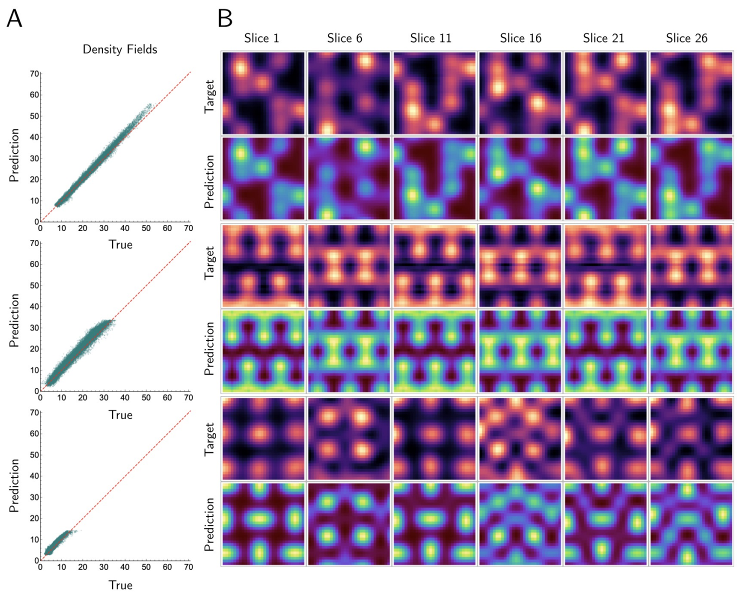


Figure 12: **Results with smaller** β. We decrease β by a factor of 10. By reducing the penalty on the Kullback-Leibler (KL) term we are able to generate more accurate reconstructions. However, sampling $\mathbf{z} \sim \mathcal{N}(0, 1)$ does not fully sample the encoded space of molecules.

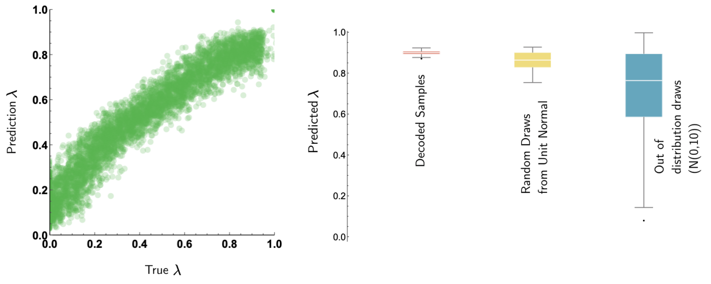


Figure 13: λ **prediction for different samples**. Using the network in Fig. 12 we train a discriminator network to predict the “distance” from a real crystal. We apply this discriminator to real decoded samples, random latent space draws, and out of distribution draws (right).

Preprint. Work in progress.

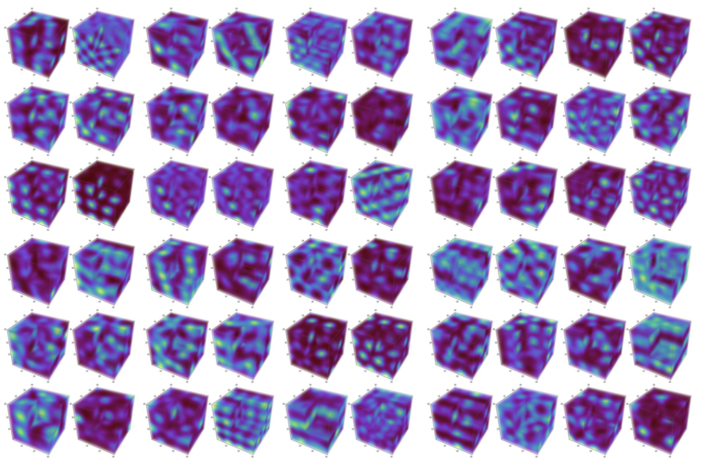


Figure 14: **Decoded samples of repeating unit cells**. Results from the encoder-decoder.

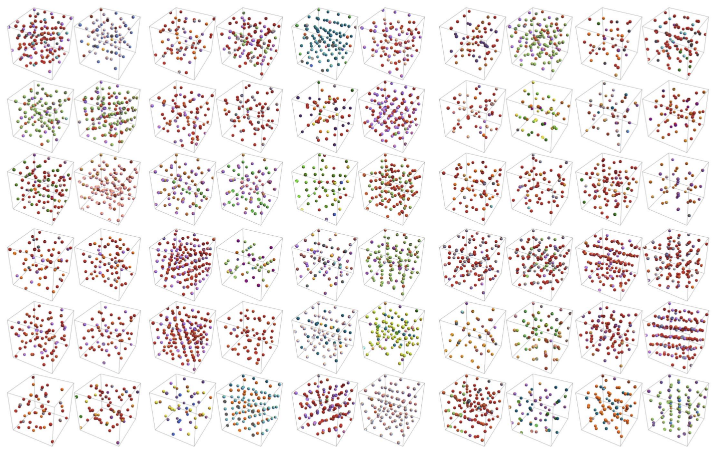


Figure 15: **Segmented Output from the decoded samples**. The segmented output from Fig. 14.

Now open even wider, Mr. Stevens... Just out of curiosity, we're going to see if we can also cram in this tennis ball.

Gary Larson

5 Other Problems

This chapter discusses a series of projects I worked on during my PhD that I was not first author on. Many of these projects represent quite a bit of work and also fantastic learning experiences.

5.1 RIMs model

This project is particularly exciting to me. I think that it is representative of the direction of the machine learning wind, and I am excited to see follow up work.

5.1.1 Background

Machine learning is a very powerful tool but a major limitation is poor generalization. Slight changes in datasets can have large effects on predictions and out of distribution inputs can have surprising outputs. We know that physical processes in the world often have a modular structure, with complexity emerging through combinations of simpler subsystems. Modular structures are aggregates of mechanisms that can perform functions without affecting the remainder of the system, and interact as needed. Despite this, most machine learning models employ the opposite inductive bias– specifically, that all processes interact and that the entire input is compressed into a single latent space. This can lead to poor generalization to out of distribution data and a lack of robustness to changing task distributions. Recent work has been interested in learning disentangled latent space representations,[25,71,30] but the aim of this work is quite different. The central question motivating this work is how a machine learning approach can learn independent, but sparsely interacting recurrent mechanisms in order to benefit from such modularity. In this paper, we introduce Recurrent Independent Mechanisms (RIMs), that partition the overall model into *k* small modules, each of which is recurrent in order to capture dynamics. Each of the RIMs have their distinct functions, including state dependent activation, that activated modules talk with one another, and that non active modules follow the default function. This has also the advantage of enabling a localization of the computation in the whole system. These learned mechanisms should not be too complex, otherwise it is easy for an individual mechanism to dominate (and hence the system may not learn anything meaningful).

5.1.2 Contributions

This project birthed from follow up work where Anirudh Goyal, Shagun Sofhani, and I were extending the original RIMs model (from Anirudh Goyal, Alex Lamb, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf) to the bouncing balls dataset. However, due to the conference cycle, we ended up merging most of the work into the first paper. I had a large part in interfacing and then running the code on the bouncing balls dataset. I analyzed results on the bouncing balls dataset, generated figures, etc. I also played a large role in writing and editing the manuscript. There are some exciting improvements that will hopefully be in a follow-up paper soon. I also generated several new datasets that we did not add to the current manuscript.

5.1.3 Publication: Recurrent Independent Mechanisms: A New Architecture for Improving Generalization

RECURRENT INDEPENDENT MECHANISMS

Anirudh Goyal[1], **Alex Lamb**[1], **Jordan Hoffmann**[1, 2, *], **Shagun Sodhani**[1, *], **Sergey Levine**[4]
Yoshua Bengio[1, **], **Bernhard Schölkopf** [3, **]

ABSTRACT

Learning modular structures which reflect the dynamics of the environment can lead to better generalization and robustness to changes which only affect a few of the underlying causes. We propose Recurrent Independent Mechanisms (RIMs), a new recurrent architecture in which multiple groups of recurrent cells operate with nearly independent transition dynamics, communicate only sparingly through the bottleneck of attention, and are only updated at time steps where they are most relevant. We show that this leads to specialization amongst the RIMs, which in turn allows for dramatically improved generalization on tasks where some factors of variation differ systematically between training and evaluation.

1 INTRODUCTION

Physical processes in the world often have a modular structure, with complexity emerging through combinations of simpler subsystems. Machine learning seeks to uncover and use regularities in the physical world. Although these regularities manifest themselves as statistical dependencies, they are ultimately due to dynamic processes governed by physics. These processes are often independent and only interact sparsely. For instance, we can model the motion of two balls as separate independent mechanisms even though they are both gravitationally coupled to Earth as well as (weakly) to each other. They may, however, occasionally strongly interact via collisions.

The notion of independent or autonomous mechanisms has been influential in the field of causal inference, where it is applied not only to dynamic processes but also to time independent datasets. For instance, it has been argued that the conditional distribution of the average annual temperature given the altitude of a place is an abstraction of a causal mechanism (subsuming complex physical processes involving air pressure, etc.) that is independent of the distribution of the altitudes of settlements (Peters et al., 2017), and will thus apply invariantly for, say, different countries in the same climate zone with different altitude distributions.

A complex generative model, temporal or not, can be thought of as the composition of independent mechanisms or "causal" modules. In the causality community, this is often considered a prerequisite of being able to perform localized interventions upon variables determined by such models (Pearl, 2009). It has been argued that the individual modules tend to remain robust or invariant even as other modules change, e.g., in the case of distribution shift (Schölkopf et al., 2012; Peters et al., 2017). One may hypothesize that if a brain is able to solve multiple problems beyond a single i.i.d. (independent and identically distributed) task, it would be economical to learn structures aligned with this, by learning independent mechanisms that can flexibly be reused, composed and re-purposed.

In the dynamic setting, we think of an overall system being assayed as composed of a number of fairly independent subsystems that evolve over time, responding to forces and interventions. A learning agent then need not devote equal attention to all subsystems at all times: only those aspects that significantly interact need to be considered jointly when taking a decision or forming a plan (Bengio, 2017). Such sparse interactions can reduce the difficulty of learning since few interactions need to be considered at a time, reducing unnecessary interference when a subsystem is adapted. Models learned this way may be more likely to capture the compositional generative (or causal) structure of the world, and thus better generalize across tasks where a (small) subset of mechanisms change while most of them remain invariant (Simon, 1991; Peters et al., 2017; Parascandolo et al., 2018). The central question motivating our work is how a machine learning approach can learn independent but sparsely interacting recurrent mechanisms in order to benefit from such modularity.

[1] Mila, University of Montreal,[2] Harvard University, [3] MPI for Intelligent Systems, Tübingen, [4] University of California, Berkeley, [**] Equal advising, [*] Equal Contribution. :anirudhgoyal9119@gmail.com

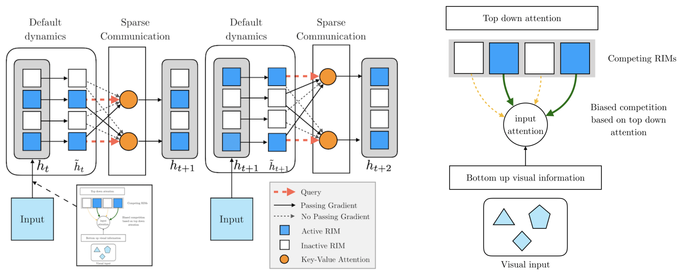


Figure 1: **Illustration of Recurrent Independent Mechanisms (RIMs)**. A single step under the proposed model occurs in four stages (left figure shows two steps). In the first stage, individual RIMs produce a query which is used to read from the current input. In the second stage, an attention based competition mechanism is used to select which RIMs to activate (right figure) based on encoded visual input (blue RIMs are active, based on attention score, white RIMs remain inactive). In the third stage, individual activated RIMs follow their own default transition dynamics while non-activated RIMs remain unchanged. In the fourth stage, the RIMs sparsely communicate information between themselves, also using attention.

2 Recurrent Independent Mechanisms with Sparse Interactions

Our approach to modelling a dynamical system of interest divides the overall model into k small subsystems (or modules), each of which is recurrent in order to be able to capture dynamics. We refer to these subsystems as *Recurrent Independent Mechanisms (RIMs)*, where each RIM has distinct functions that are learned automatically from data. We refer to RIM k at time step t as having state $h_{t,k}$, where $t = 1, \ldots, T$. Each RIM has parameters θ_k, which are shared across all time steps.

At a high level (see. Fig. 1), we want each RIM to have its own independent dynamics operating by default, and occasionally to interact with other relevant RIMs and with selected elements of the encoded input. The total number of parameters can be kept small since RIMs can specialize on simple sub-problems, similar to Parascandolo et al. (2018). This specialization and modularization not only has computational and statistical advantages (Baum & Haussler, 1989; Bengio et al., 2019), but also prevents individual RIMs from dominating and modelling complex, composite mechanisms. We expect this to lead to more robust systems than training one big homogeneous neural network (Schmidhuber, 2018). Moreover, modularity also has the desirable implication that a RIM should maintain its own independent functionality even as other RIMs are changed. A more detailed account of the desiderata for the model is given in Appendix A.

2.1 Independent RIM Dynamics

Now, consider the default transition dynamics which we apply for each RIM independently and during which no information passes between RIMs. We use $\tilde{h}$ for the hidden state after the independent dynamics are applied (and before attention is applied). First, for the RIMs which are not activated (we refer to the activated set as $\mathcal{S}_t$), the hidden state remains unchanged:

$$\tilde{h}_{t+1,k} = h_{t,k} \qquad \forall k \notin \mathcal{S}_t. \tag{1}$$

Note that the gradient still flows through a RIM on a step where it is not activated. For the RIMs that are activated, we run a per-RIM independent transition dynamics. The form of this is somewhat flexible, but in this work we opted to use either a GRU (Chung et al., 2015) or an LSTM (Hochreiter & Schmidhuber, 1997). We generically refer to these independent transition dynamics as D_k, and we emphasize that each RIM has its own separate parameters. Aside from being RIM-specific, the internal operation of the LSTM and GRU remain unchanged, and the active RIMs are updated by

$$\tilde{h}_{t+1,k} = D_k(h_{t,k}) = LSTM(h_{t,k}, A_k^{(in)}; \theta_k^{(D)}) \qquad \forall k \in \mathcal{S}_t \tag{2}$$

as a function of the attention mechanism $A_k^{(in)}$ applied on the current input, described in the next two subsections below, after explaining the key-value mechanism used to select arguments for this update.

Note that we are using the term *mechanism* both for the mechanisms that make up the world's dynamics as well as for the computational modules that we learn to model those mechanisms.

2.2 Key-Value Attention to Process Sets of Named Interchangeable Variables

Each RIM should be activated and updated when the input is relevant to it. We thus utilize competition to allocate representational and computational resources. As argued by Parascandolo et al. (2018), this tends to produce independence among learned mechanisms, provided the training data has been generated by a set of independent physical mechanisms. In contrast to Parascandolo et al. (2018), we use an *attention mechanism* for this purpose. In doing so, we are inspired by findings from experimental psychology in the study of the interplay of top-down attention and bottom-up information flow, conceptualized in the *biased competition theory* of selective attention (Desimone & Duncan, 1995): A brain's capacity for parallel processing of complex entities is limited, and many brain systems representing visual information use competition (operating in parallel across the visual field) to allocate resources, often biased by feedback from higher brain areas.

The introduction of content-based soft-attention mechanisms (Bahdanau et al., 2014) has opened the door to neural networks which operate on *sets of typed interchangeable objects*. This idea has been remarkably successful and widely applied to most recent Transformer-style multi-head dot product self attention models (Vaswani et al., 2017; Santoro et al., 2018), achieving new state-of-the-art results in many tasks. Soft-attention uses the product of a *query* (or *read key*) Q of dimensionality $N_r \times d$ matrix Q, and d dimension of each key) to a set of N_o objects each associated with a *key* (or *write-key*) matrix K^T ($N_o \times d$), and after normalization with a softmax yields outputs in the convex hull of the *values* (or *write-values*) V_i (row i of matrix V). Its result is computed as

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V,$$

where the softmax is applied to each row of its argument matrix, yielding a set of convex weights. As a result, one obtains a convex combination of the values V. If the attention is focused on one element for a particular row (i.e., the softmax is saturated), this simply selects one of the objects and copies its value to row j of the result. Note that the d dimensions in the key can be split into *heads* which then have their attention matrix and write values computed separately.

When the inputs and outputs of each RIM are a set of objects or entities (each associated with a key and value vector), the RIM processing becomes a generic object-processing machine which can operate on "variables" in a sense analogous to variables in a programming language: as interchangeable arguments of functions. Because each object has a key embedding (which one can understand both as a name and as a type), the same RIM processing can be applied to any variable which fits an expected "distributed type" (specified by a query vector). Each attention head then corresponds to a typed argument of the function computed by the RIM. When the key of an object matches the query, it can be used as input for the RIM. Whereas in regular neural networks (without attention) neurons operate on fixed variables (the neurons which are feeding them from the previous layer), the key-value attention mechanisms make it possible to select on the fly which variable instance (i.e. which entity or object) is going to be used as input for each of the arguments of the RIM dynamics, with a different set of query embeddings for each RIM. These inputs can come from the external input or from the output of other RIMs. So, if the individual RIMs can represent these "functions with typed arguments," then they can "bind" to whatever input is currently available and best suited according to its attention score: the "input attention" mechanism would look at the candidate input object's key and evaluate if its "type" matches with what this RIM expects (specified in the query).

2.3 Selective Activation of RIMs as a form of Top-Down Modulation

The proposed model learns to dynamically select those RIMs for which the current input is relevant. We give each RIM the choice between attending to the actual input instances or a special null input. The null input consists entirely of zeros and thus contains no information. At each step, we select the top-k_A (out of k_T) RIMs in terms of their value of the softmax for the real input. Intuitively, the RIMs must compete on each step to read from the input, and only the RIMs that win this competition will be able to read from the input and have their state updated.

In our use of key-value attention, the queries come from the RIMs, while the keys and values come from the current input. The mechanics of this attention mechanism follow (Vaswani et al., 2017; Santoro et al., 2018), with the modification that the parameters of the attention mechanism itself are separate for each RIM. The input attention for a particular RIM is described as follows. The input

x_t at time t is seen as a set of elements, structured as rows of a matrix (for image data, it can be the output of the CNN). We first concatenate a row full of zeros, to obtain

$$X = \emptyset \oplus x_t. \tag{3}$$

$\oplus$ refers to the row-level concatenation operator. Then, linear transformations are used to construct keys ($K = XW^k$, one per input element and for the null element), values ($V = XW^v$, again one per element), and queries ($Q = RW^q_k$, one per RIM attention head) where R is a matrix with each row r_i corresponding to the hidden state of an individual RIM (i.e $h_{t,k}$). W^v is a simple matrix mapping from an input element to the corresponding value vector for the weighted attention and W^k is similarly a weight matrix which maps the input to the keys. W^q_k is a per-RIM weight matrix which maps from the RIM's hidden state to its queries. The attention thus is

$$A^{(in)}_k = \text{softmax}\left(\frac{RW^q_k(XW^k)^T}{\sqrt{d_e}}\right) XW^v, \text{ where } \theta^{(in)}_k = (W^q_k, W^e, W^v). \tag{4}$$

Based on the softmax values in (4), we select the top k_A RIMs (out of the total K RIMs) to be activated for each step, which have the least attention on the null input (and thus put the highest attention on the input), and we call this set $\mathcal{S}_t$. Since the queries depend on the state of the RIMs, this enables individual RIMs to attend only to the part of the input that is relevant for that particular RIM, thus enabling selective attention based on a *top-down attention* process (see. Fig 1). In practice, we use multiheaded attention, and multi-headed attention doesn't change the essential computation, but when we do use it for input-attention we compute RIM activation by averaging the attention scores over the heads.

2.4 Communication between RIMs

Although the RIMs operate independently by default, the attention mechanism allows sharing of information among the RIMs. Specifically, we allow the activated RIMs to read from all other RIMs (activated or not). The intuition behind this is that non-activated RIMs are not related to the current input, so their value should not change. However they may still store contextual information that is relevant for activated RIMs. For this communication between RIMs, we use a residual connection as in (Santoro et al., 2018) to prevent vanishing or exploding gradients over long sequences.

$$Q_{t,k} = \tilde{W}^q_k \tilde{h}_{t,k}, \quad \forall k \in \mathcal{S}_t \tag{5}$$

$$K_{t,k} = \tilde{W}^e_k \tilde{h}_{t,k}, \quad \forall k \tag{6}$$

$$V_{t,k} = \tilde{W}^v_k \tilde{h}_{t,k}, \quad \forall k \tag{7}$$

$$h_{t+1,k} = \text{softmax}\left(\frac{Q_{t,k}(K_{t,:})^T}{\sqrt{d_e}}\right) V_{t,:} + \tilde{h}_{t,k} \quad \forall k \in \mathcal{S}_t, \text{ where } \theta^{(c)}_k = (\tilde{W}^q_k, \tilde{W}^e_k, \tilde{W}^v_k). \tag{8}$$

2.5 Variations on the RIMs Architecture

The RIMs architecture that we study is highly homogeneous and generally the only hyperparameters are the number of RIMs K and how many RIMs are activated on each time step K_A. All of the datasets that we consider are temporal, yet there is a distinction between datasets where the input on each time step is highly structured (such as a video, where each time step is an image) and where this is not the case (such as language modeling, where each step is a word or character). In the former case, we can get further improvements by making the activation of RIMs not just sparse across time but also sparse across the (spatial) structure.

Multiple Heads: As in Vaswani et al. (2017); Santoro et al. (2018), we use multiple heads (both for communication between RIMs as well as input attention (as in Sec 2.3) by producing different sets of queries, keys, and values to compute a linear transformation for each head (different heads have different parameters), and then applying the attention operator for each head separately in order to select conditioning inputs for the RIMs.

3 RELATED WORK

Neural Turing Machine (NTM) and Relational Memory Core (RMC): the NTM (Graves et al., 2014a) consists of a sequence of independent memory cells, and uses an attention mechanism while performing targeted read and write operations. This shares a key idea with RIMs: that input information should only impact a sparse subset of the memory by default, while keeping most of the memory unaltered. RMC (Santoro et al., 2018) uses a multi-head attention mechanism to share information between multiple memory elements. We encourage the RIMs to remain separate as much as possible, whereas Santoro et al. (2018) allow information between elements to flow on each step in an unconstrained way. Instead, each RIM has its own default dynamics, while in RMC, all the processes interact with each other.

Separate Recurrent Models: EnTNet (Henaff et al., 2016) and IndRNN (Li et al., 2018) can be viewed as a set of separate recurrent models. In IndRNN, each recurrent unit has completely independent dynamics, whereas EntNet uses an independent gate for writing to each memory slot. RIMs use different recurrent models (with separate parameters), but we allow the RIMs to communicate with each other sparingly using an attention mechanism.

Modularity and Neural Networks: A neural network is composed of several neural modules, where each module is meant to perform a distinct function, and hence can be seen as a combination of experts (Jacobs et al., 1991; Bottou & Gallinari, 1991; Ronco et al., 1997; Reed & De Freitas, 2015; Andreas et al., 2016; Parascandolo et al., 2018; Rosenbaum et al., 2017; Fernando et al., 2017; Shazeer et al., 2017; Kirsch et al., 2018; Rosenbaum et al., 2019) routing information through a gated activation of layers. These works generally assume that only a single expert is active at a particular time step. In the proposed method, multiple RIMs can be active, interact and share information.

Computation on demand: There are various architectures (El Hihi & Bengio, 1996; Koutnik et al., 2014; Chung et al., 2016; Neil et al., 2016; Jernite et al., 2016; Krueger et al., 2016) where parts of the LSTM's hidden state are kept dormant at times. The major differences as compared to the proposed architecture are that (a) we modularize the dynamics of recurrent cells (using RIMs), and (b) we also control the inputs of each module (using transformer style attention), while many previous gating methods did not control the inputs of each module, but only whether they should be executed or not.

4 EXPERIMENTS

The main goal of our experiments is to show that the use of RIMs improves generalization across changing environments and/or in modular tasks, and to explore how it does so. Our goal is not to outperform highly optimized baselines; rather, we want to show the versatility of our approach by applying it to a range of diverse tasks, focusing on tasks that involve a changing environment. We organize our results by the capabilities they illustrate: we address generalization based on temporal patterns, based on objects, and finally consider settings where both of these occur together.

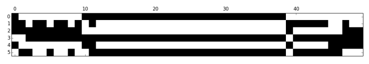


Figure 2: **Copying Task RIM Activation Pattern** for a model with $K = 6$ RIMs and $K_A = 3$ active RIMs per step (the activated RIMs are in black, non-activated in white). We can see that the RIM activation pattern is distinct during the dormant part of the sequence.

4.1 RIMS IMPROVE GENERALIZATION BY SPECIALIZING OVER TEMPORAL PATTERNS

We first show that when RIMs are presented with sequences containing distinct temporal patterns, they are able to specialize so that different RIMs are activated on different patterns. As a result, RIMs are able to generalize well when we modify a subset of the patterns (especially those unrelated to the class label) while most recurrent models fail to generalize well to these variations.

4.1.1 COPYING TASK

First we turn our attention to the task of receiving a short sequence of characters, then receiving blank inputs for a large number of steps, and then being asked to reproduce the original sequence. We can

Copying	k_T	k_A	h_{size}	Train(50) CE	Test(200) CE
RIMs	6	5	600	0.01	3.5
	6	4	600	**0.00**	**0.00**
	6	3	600	**0.00**	**0.00**
	6	2	600	**0.00**	**0.00**
	5	3	500	**0.00**	**0.00**
LSTM	-	-	300	0.00	2.28
	-	-	600	0.00	3.56
NTM	-	-	-	0.00	2.54
RMC	-	-	-	0.00	0.13
Transformers	-		-	0.00	0.54

Sequential MNIST	k_T	k_A	h_{size}	16 x 16 Accuracy	19 x 19 Accuracy	24 x 24 Accuracy
RIMs	6	6	600	85.5	56.2	30.9
	6	5	600	88.3	43.1	22.1
	6	4	600	**90.0**	**73.4**	**38.1**
LSTM	-	-	300	86.8	42.3	25.2
	-	-	600	84.5	52.2	21.9
EntNet	-	-	-	89.2	52.4	23.5
RMC	-	-	-	89.58	54.23	27.75
DNC	-	-	-	87.2	44.1	19.8
Transformers	-		-	**91.2**	51.6	22.9

Table 1: Performance on the copying task (left) and sequential MNIST resolution task right). **Error (CE on the last 10 time steps) on the copying task**. Note that while all of the methods are able to learn to copy for the length seen during training, the RIMs model generalizes to sequences longer than those seen during training whereas the LSTM, RMC, and NTM degrade. **Sequential MNIST resolution:** Test Accuracy % on the Sequential MNIST resolution generalization task (see text) after 100 epochs. Both the proposed and the Baseline model (LSTM) were trained on 14x14 resolution but evaluated at different resolutions; results averaged over 3 different trials.

think of this as consisting of two temporal patterns which are independent: one where the sequence is received and another "dormant" pattern where no input is provided.

As an example of out-of-distribution generalization, we find that using RIMs, we can extend the length of this dormant phase from 50 during training to 200 during testing and retain perfect performance (Table 1), whereas baseline methods including LSTM, NTM, and RMC substantially degrade. In addition, we find that this result is robust to the number of RIMs used as well as to the number of RIMs activated per-step. Our ablation results (Appendix C.5) show that all major components of the RIMs model are necessary to achieve this generalization. We consider this preliminary evidence that RIMs can specialize over distinct patterns in the data and improve generalization to settings where these patterns change.

4.1.2 Sequential MNIST Resolution Task

RIMs are motivated by the hypothesis that generalization performance can be improved by having modules which only activate on relevant parts of the sequence. For further evidence that RIMs can achieve this out-of-distribuution, we consider the task of classifying MNIST digits as sequences of pixels (Krueger et al., 2016) and assay generalization to images of resolutions different from those seen during training. Our intuition is that the RIMs model should have distinct subsets of the RIMs activated for pixels with the digit and empty pixels. As a result, RIMs should generalize better to greater resolutions by keeping the RIMs which store pixel information dormant over the empty regions of the image.

Results: Table 1 shows the result of the proposed model on the Sequential MNIST Resolution Task. If the train and test sequence lengths agree, both models achieve comparable test set performance. However, the RIMs model was relatively robust to changing the sequence length (by changing the image resolution), whereas the LSTM performance degraded more severely. This can be seen as a more involved analogue of the copying task, as MNIST digits contain large empty regions. It is essential that the model be able to store information and pass gradients through these regions. The RIMs outperform strong baselines such as Transformers, EntNet, RMC, as well as the Differentiable Neural Computer (DNC) (Graves et al., 2016).

4.2 RIMs learn to specialize over objects and generalize between them

We have presented evidence that RIMs can specialize over temporal patterns. We now turn our attention to showing that RIMs can specialize to objects, and show improved generalization to settings where we add or remove objects at test time.

4.2.1 Bouncing Ball Environment

We consider a synthetic "bouncing balls" task in which multiple balls (of different masses and sizes) move using basic Newtonian physics (Van Steenkiste et al., 2018). What makes this task particularly suited to RIMs is that the balls move independently most of the time, except when they collide. During training, we predict the next frame at each time step using teacher forcing (Williams & Zipser, 1989). We can then use this model to generate multi-step rollouts.

As a preliminary experiment, we train on sequences of length 51 (the previous standard), using a binary cross entropy loss when predicting the next frame. We consider LSTMs as baselines. We then produce rollouts, finding that RIMs are better able to predict future motion (examples in Figure 3, Figure 10 in Appendix and quantitative comparisons in Figure 4).

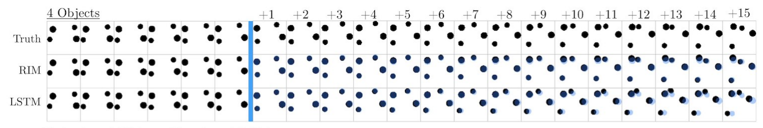


Figure 3: **Predicting Movement of Bouncing Balls**. The first 15 frames of ground truth are given (last 6 of those shown) and then the system is rolled out for the next 15 time steps. We find that RIMs perform better than the LSTMs (predictions are in black, ground truth in blue). Notice the blurring of LSTM predictions.

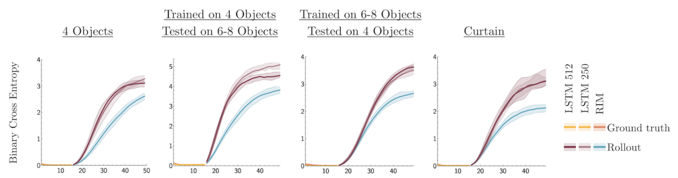


Figure 4: **Handling Novel Out-of-Distribution Variations**. Here, we study the performance of our proposed model compared to an LSTM baseline. The first 15 frames of ground truth are fed in and then the system is rolled out for the next 10 time steps. During the rollout phase, RIMs perform better than the LSTMs in accurately predicting the dynamics of the balls as reflected by the lower Cross Entropy (CE) [see blue for RIMs, purple for LSTM]. Notice the substantially better out-of-distribution generalization of RIMs when testing on a different number of objects than during training.

We take this further by evaluating RIMs on environments where the setup is different from the training setup. First we consider training with 4 balls and evaluating on an environment with 6-8 balls. Second, we consider training with 6-8 balls and evaluating with just 4 balls. Robustness in these settings requires a degree of invariance w.r.t. the number of balls.

In addition, we consider a task where we train on 4 balls and then evaluate on sequences where part of the visual space is occluded by a "curtain." This allows us to assess the ability of balls to be tracked (or remembered) through the occluding region. Our experimental results on these generalization tasks (Figure 4) show that RIMs substantially improve over an LSTM baseline. We found that increasing the capacity of the LSTM from 256 to 512 units did not substantially change the performance gap, suggesting that the improvement from RIMs is not primarily a result of increased capacity.

4.2.2 Environment with Novel Distractors

We next consider an object-picking reinforcement learning task from BabyAI (Chevalier-Boisvert et al., 2018) in which an agent must retrieve a specific object in the presence of distractors. We use a partially observed formulation of the task, where the agent only sees a small number of squares ahead of it. These tasks are difficult to solve (Chevalier-Boisvert et al., 2018) with standard RL algorithms, due to (1) the partial observability of the environment and (2) the sparsity of the reward, given that the agent receives a reward only after reaching the goal. During evaluation, we introduce new distractors to the environment which were not observed during training.

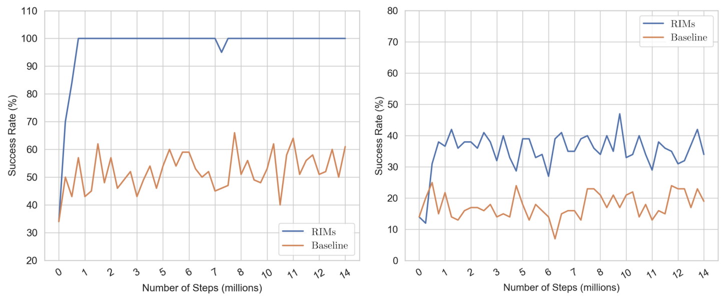


Figure 5: **Robustness to Novel Distractors:**. Left: performance of the proposed method compared to an LSTM baseline in solving the object picking task in the presence of distractors. Right: performance of proposed method and the baseline when novel distractors are added.

Figure 5 shows that RIMs outperform LSTMs on this task (details in appendix). When evaluating with known distractors, the RIM model achieves perfect performance while the LSTM struggles. When evaluating in an environment with novel unseen distractors the RIM doesn't achieve perfect performance but still outperforms the LSTM. An LSTM with a single memory flow may struggle to keep the distracting elements separate from elements which are necessary for the task, while the RIMs model uses attention to control which RIMs receive information at each step as well as what information they receive (as a function of their hidden state). This "top-down" bias results in a diminished representation of the distractor, not only enhancing the target visual information, but also suppressing irrelevant information. The notion that enhancement of the relevant information necessarily results in suppression of irrelevant information is fundamental to biased competition theory (Desimone & Duncan, 1995).

4.3 RIMs improve generalization in complex environments

We have investigated how RIMs use specialization to improve generalization to changing important factors of variation in the data. While these improvements have often been striking, it raises a question: what factors of variation should be changed between training and evaluation? One setting where factors of variation change naturally is in reinforcement learning, as the data received from an environment changes as the agent learns and improves. We conjecture that when applied to reinforcement learning, an agent using RIMs may be able to learn faster as its specialization leads to improved generalization to previously unseen aspects of the environment.

To investigate this we use an RL agent trained using Proximal Policy Optimization (PPO) (Schulman et al., 2017) with a recurrent network producing the policy. We employ an LSTM as a baseline, and compare results to the RIMs architecture. This was a simple drop-in replacement and did not require changing any of the hyperparameters for PPO. We experiment on the whole suite of Atari games and find that simply replacing the LSTM with RIMs greatly improves performance (Figure 6).

There is also an intriguing connection between the selective activation in RIMs and the concept of affordances from cognitive psychology (Gibson, 1977; Cisek & Kalaska, 2010). To perform well in environments with a dynamic combination of risks and opportunities, an agent should be ready to adapt immediately, releasing into execution actions which are at least partially prepared. This suggests agents should process sensory information in a contextual manner, building representations of potential actions that the environment currently affords. For instance, in Demon Attack, one of the games where RIMs exhibit strong performance gains, the agent must quickly choose between targeting distant aliens to maximize points and avoiding fire from close-by aliens to avoid destruction (indeed both types of aliens are always present, but which is relevant depends on the player's position). We hypothesize that in cases like this, selective activation of RIMs allows the agent to rapidly adapt its information processing to the types of actions relevant to the current context.

4.4 Ablations

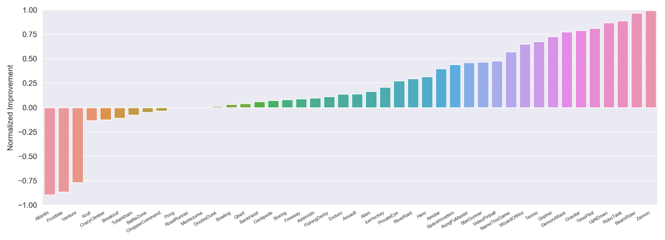


Figure 6: RIMs-PPO relative score improvement over LSTM-PPO baseline (Schulman et al., 2017) across all Atari games averaged over 3 trials per game. In both cases, PPO was used with the exact same settings, and the only change is the choice of recurrent architecture. More detailed experiments with learning curves as well as comparisons with external baselines are in Appendix C.

Role of Top-Down Modulation: Removing Input Attention We study the scenario where we remove the input attention process (Section 2.3) but still allow communication between RIMs (Section 2.4). We train this agent on 30 ATARI games for 30M time-steps each and compare the performance of this agent with the normal RIMs-PPO agent. We find that the RIMs agent still outperform this agent on 11 out of 30 games, while on 1 game (Frostbite) we see the proposed baseline agent substantially improves the performance. For more details regarding the training curves, refer to Fig. 25 (in Appendix).

Importance of communication between RIMs: For copying, we performed an ablation where we remove the communication between RIMs. We also varied the number of RIMs as well as the number of activated RIMs (Table 5). We found that the communication between RIMs is essential for good performance. We found similar results for the sequential MNIST resolution task.

Importance of sparsity of activation of the RIMs For the copying task, as well as for the sequential MNIST changed resolution task, we performed an ablation where we kept all RIMs active for all time steps (Table 5). We found that we were not able to achieve strong generalization as compared to the best performing RIMs model. On Atari we found that using $k_A = 5$ slightly improved over results compared with $k_A = 4$, but both had similar performance across the vast majority of games, suggesting that the k_A hyperparameter is reasonably flexible in practice.

Effect of complexity of individual RIM: Empirically, we found that if the learned mechanisms are too complex, it is easy for an individual mechanism to dominate (and hence not learn anything meaningful). If the learned mechanisms do not have enough capacity (i.e., if each RIM consists of about 100 units), then different RIMs have to work together.

Varying the number of attention heads for communication: Here, we study what happens if the output of RIMs only has one 'object' rather than multiple ones (Section 2.2). The intuition is that RIM processing can be applied to any "head" which matches the query by an individual RIM. So, having more heads should help, as different heads could be used by different RIMs, rather than every RIM competing for the same head. We study this in the context of bouncing balls. We found that using multiple heads improves the performance, thus validating our hypothesis (Sec. 2.2). See Appendix C.11 for details.

Randomly Dropping Out RIMs: Modular structures are aggregates of mechanisms that can perform functions without affecting the remainder of the system, and interact as needed. To what extent are trained RIMs able to model meaningful phenomena when other RIMs are removed? We performed an experiment on moving MNIST digits where we train normally and "dropout" a random RIM at test time. We found that in the absence of selective activation (i.e. when $k_A = k_T$, Section C.13) the performance degraded very badly, but the performance degrades much less with selective activation. See Appendix C.13 for details.

5 Conclusion

Many systems of interest comprise multiple dynamical processes that operate relatively independently and only occasionally have meaningful interactions. Despite this, most machine learning models employ the opposite inductive bias, i.e., that all processes interact. This can lead to poor generalization (if data is limited) and lack of robustness to changing task distributions. We have proposed a new architecture, Recurrent Independent Mechanisms (RIMs), in which we learn multiple recurrent modules that are independent by default, but interact sparingly. Our positive experimental results lend support to the consciousness prior (Bengio, 2017), i.e., the importance of computational elements which focus on few mechanisms at a time in order to determine how a high-level state evolves over time, with many aspects of the state not being affected by this attentive dynamics (i.e., following default dynamics). For the purposes of this paper, we note that the notion of RIMs is not limited to the particular architecture employed here. The latter is used as a vehicle to assay and validate our overall hypothesis (cf. Appendix A), but better architectures for the RIMs model can likely be found.

Acknowledgements

The authors acknowledge the important role played by their colleagues at Mila throughout the duration of this work. AG would like to thank Matthew Botvinick, Charles Blundell, Greg Wayne for their useful feedback, which has improved the work tremendously. The authors would like to thank Mike Mozer for brainstorming sessions. The authors would like to thank Vikas Verma for running some of the baselines used in the earlier version of the paper. The authors would also like to thank Rosemary Nan Ke, Nasim Rahaman, Stefan Bauer, Jonathan Binas, Min Lin, Disha Srivastava, Ali Farshchian, Sarthak Mittal, Owen Xu, for useful discussions. The authors would also like to thank Shahab Bakhtiari, Kris Sankaran, Felix E. Leeb, Georgios Arvanitidis, and Fabian Both for proof reading the paper. The authors are grateful to NSERC, CIFAR, Google, Samsung, Nuance, IBM, Canada Research Chairs, Canada Graduate Scholarship Program, Nvidia for funding, and Compute Canada for computing resources. We are very grateful to Google for giving Google Cloud credits used in this project. This project was also known by the name "Blocks" internally at Mila. Finally, we would like to thank various different Starbucks outlets in Downtown Montreal for providing free internet which acted as a stimulant for this project.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 39–48, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in neural information processing systems*, pp. 81–90, 1989.

Yoshua Bengio. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. *arXiv:1901.10912*, 2019.

Léon Bottou and Patrick Gallinari. A framework for the cooperation of learning algorithms. In *Advances in neural information processing systems*, pp. 781–788, 1991.

Matthew Botvinick and Todd Braver. Motivation and cognitive control: from behavior to neural mechanism. *Annual review of psychology*, 66, 2015.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: First steps towards grounded language learning with a human in the loop. *arXiv preprint arXiv:1810.08272*, 2018.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pp. 2980–2988, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Paul Cisek and John F Kalaska. Neural mechanisms for interacting with a world full of action choices. *Annual review of neuroscience*, 33:269–298, 2010.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

Robert Desimone and Jody Duncan. Neural mechanisms of selective visual attention. *Annual Review of Neuroscience*, 18:193–222, 1995.

A. Dickinson. Actions and habits: the development of behavioural autonomy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 308(1135):67–78, 1985. ISSN 0080-4622. doi: 10.1098/rstb.1985.0010. URL `http://rstb.royalsocietypublishing.org/content/308/1135/67`.

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pp. 493–499, 1996.

Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

James J Gibson. The theory of affordances. *Hilldale, USA*, 1(2), 1977.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Sergey Levine, and Yoshua Bengio. Infobot: Transfer and exploration via the information bottleneck. *arXiv preprint arXiv:1901.10902*, 2019a.

Anirudh Goyal, Shagun Sodhani, Jonathan Binas, Xue Bin Peng, Sergey Levine, and Yoshua Bengio. Reinforcement learning with competitive ensembles of information-constrained primitives. *arXiv preprint arXiv:1906.10667*, 2019b.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014a.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014b. URL `http://arxiv.org/abs/1410.5401`.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Yacine Jernite, Edouard Grave, Armand Joulin, and Tomas Mikolov. Variable computation in recurrent neural networks. *arXiv preprint arXiv:1611.06188*, 2016.

Nan Rosemary Ke, Anirudh Goyal, Olexa Bilaniuk, Jonathan Binas, Michael C Mozer, Chris Pal, and Yoshua Bengio. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems*, pp. 7640–7651, 2018.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687*, 2018.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pp. 2408–2418, 2018.

Wouter Kool and Matthew Botvinick. Mental labour. *Nature human behaviour*, 2(12):899–908, 2018.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in neural information processing systems*, pp. 3882–3890, 2016.

Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla, and Bernhard Schölkopf. Learning independent causal mechanisms. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 4033–4041, 2018. URL `http://proceedings.mlr.press/v80/parascandolo18a.html`.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, New York, NY, 2nd edition, 2009.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference - Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, USA, 2017. ISBN 978-0-262-03731-0.

David Raposo, Adam Santoro, David Barrett, Razvan Pascanu, Timothy Lillicrap, and Peter Battaglia. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Eric Ronco, Henrik Gollee, and Peter J Gawthrop. Modular neural networks and self-decomposition. *Technical Report CSC-96012*, 1997.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

Clemens Rosenbaum, Ignacio Cases, Matthew Riemer, and Tim Klinger. Routing networks and the challenges of modular and compositional computation. *arXiv preprint arXiv:1904.12774*, 2019.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pp. 4967–4976, 2017.

Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. *CoRR*, abs/1806.01822, 2018. URL `http://arxiv.org/abs/1806.01822`.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Jürgen Schmidhuber. One big net for everything. *arXiv preprint arXiv:1802.08864*, 2018.

Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. In J. Langford and J. Pineau (eds.), *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 1255–1262, New York, NY, USA, 2012. Omnipress.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Herbert A Simon. The architecture of complexity. In *Facets of systems science*, pp. 457–476. Springer, 1991.

Shagun Sodhani, Anirudh Goyal, Tristan Deleu, Yoshua Bengio, Sergey Levine, and Jian Tang. Learning powerful policies by using consistent dynamics model. *arXiv preprint arXiv:1906.04355*, 2019.

Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 4496–4506, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Sjoerd Van Steenkiste, Michael Chang, Klaus Greff, and Jürgen Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv preprint arXiv:1802.10353*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

H. L. F. von Helmholtz. *Handbuch der physiologischen Optik*, volume III. Voss, 1867.

Erich von Holst and Horst Mittelstaedt. Das reafferenzprinzip. *Naturwissenschaften*, 37(20):464–476, Jan 1950. doi: 10.1007/BF00622503.

Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pp. 4539–4547, 2017.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

A Desiderata for Recurrent Independent Mechanisms

We have laid out a case for building models composed of modules which by default operate independently and can interact in a limited manner. Accordingly, our approach to modelling the dynamics of the world starts by dividing the overall model into small subsystems (or modules), referred to as *Recurrent Independent Mechanisms (RIMs)*, with distinct functions learned automatically from data.Our model encourages sparse interaction, i.e., we want most RIMs to operate independently and follow their default dynamics most of the time, only rarely sharing information. Below, we lay out desiderata for modules to capture modular dynamics with sparse interactions.

Competitive Mechanisms: Inspired by the observations in the main paper, we propose that RIMs utilize competition to allocate representational and computational resources. As argued by (Parascandolo et al., 2018), this tends to produce independence among learned mechanisms if the training data has been generated by independent physical mechanisms.

Top Down Attention: The points mentioned in Section 2 in principle pertain to synthetic and natural intelligent systems alike. Hence, it is not surprising that they also appear in neuroscience. For instance, suppose we are looking for a particular object in a large scene, using limited processing capacity. The *biased competition theory* of selective attention conceptualizes basic findings of experimental psychology and neuroscience (Desimone & Duncan, 1995): our capacity of parallel processing of and reasoning with high-level concepts is limited, and many brain systems representing visual information use competition to allocate resources. Competitive interactions among multiple objects occur automatically and operate in parallel across the visual field. Second, the principle of selectivity amounts to the idea that a perceiver has the ability to filter out unwanted information and selectively *process* the rest of the information. Third, *top-down bias* originating from higher brain areas enables us to selectively devote resources to input information that may be of particular interest or relevance. This may be accomplished by units matching the internal model of an object or process of interest being pre-activated and thus gaining an advantage during the competition of brain mechanisms.

Sparse Information Flow: Each RIMs' dynamics should only be affected by RIMs which are deemed relevant. The fundamental challenge is centered around establishing sensible communication between RIMs. In the presence of noisy or distracting information, a large subset of RIMs should stay dormant, and not be affected by the noise. This way, training an ensemble of these RIMs can be more robust to out-of-distribution or distractor observations than training one big homogeneous neural network (Schmidhuber, 2018).

Modular Computation Flow and Modular Parameterization: Each RIM should have its own dynamics operating by *default*, in the absence of interaction with other RIMs. The total number of parameters (i.e. weights) can be reduced since the RIMs can specialize on simple sub-problems, similar to (Parascandolo et al., 2018). This can speed up computation and improve the generalisation ability of the system (Baum & Haussler, 1989). The individuals RIMs in the ensemble should be simple also to prevent individual RIMs from dominating and modelling complex, composite mechanisms. We refer to a parameterization as modular if most parameters are associated to individuals RIMs only. This has the desirable property that a RIM should maintain its own independent functionality even as other RIMs are changed (due to its behavior being determined by its own self-contained parameters).

B Extended Related Work

Table 2: A concise comparison of recurrent models with modular memory.

Method / Property	Modular Memory	Sparse Information Flow	Modular Computation Flow	Modular Parameterization
LSTM / RNN	✗	✗	✗	✗
Relational RNN (Santoro et al., 2018)	✓	✗	✓	✗
NTM (Graves et al., 2014b)	✓	✓	✗	✗
SAB(Ke et al., 2018)	✗	✓	✗	✗
IndRNN(Li et al., 2018)	✓	✗	✗	✓
RIMs	✓	✓	✓	✓

The present section provides further details on related work, thus extending Section 3.

Neural Turing Machine (NTM). The NTM (Graves et al., 2014a) has a Turing machine inspired memory with a sequence of independent memory cells, and uses an attention mechanism to move heads over the cells while performing targeted read and write operations. This shares a key idea with RIMs: that input information should only impact a sparse subset of the memory by default, while keeping most of the memory unaltered. The RIM model introduces the idea that each RIM has its own independent dynamics, whereas the mechanism for updating memory cells update is shared.

Relational RNN. The Relational Models paper (Santoro et al., 2018) is based on the idea of using a multi-head attention mechanism to share information between multiple parts of memory. It is related to our idea but a key difference is that we encourage the RIMs to remain separate as much as possible, whereas (Santoro et al., 2018) allows information between the parts to flow on each step (in effect making the part distribution only relevant to a particular step). Additionally, RIMs has the notion of each RIM having its own independent transition dynamics which operate by default, whereas the Relational RNN only does computation and updating of the memory using attention.

Sparse Attentive Backtracking (SAB). The SAB architecture (Ke et al., 2018) explores RNNs with self-attention across time steps as well as variants where the attention is sparse in the forward pass and where the gradient is sparse in the backward pass. It shares the motivation of using sparse attention to keep different pieces of information separated, but differs from the RIMs model in that it considers separation between time steps rather than separation between RIMs.

Independently Recurrent Neural Network (IndRNN). The IndRNN (Li et al., 2018) replaces the full transition matrix in a vanilla RNN (between time steps) to a diagonal transition weight matrix. In other words, each recurrent unit has completely independent dynamics. Intriguingly they show that this gives much finer control over the gating of information, and allows for such an RNN to learn long-term dependencies without vanishing or exploding gradients. Analysis of the gradients shows that having smaller recurrent transition matrices mitigates the vanishing and exploding gradient issue. This may provide further explanation for why RIMs perform well on long sequences.

Consciousness Prior (Bengio, 2017): This is based on the assumption of a sparse graphical model describing the interactions between high-level variables, using gating mechanisms to select only a subset of high-level variables to interact at any particular time. This is closely related to our work in the sense high level abstract representation is based on the representations of the RIMs, which are activated sparsely and interact sparsely. Our paper thus helps to validate the consciousness prior idea.

Recurrent Entity Networks: EnTNet (Henaff et al., 2016) can be viewed as a set of separate recurrent models whose hidden states store the memory slots. These hidden states are either fixed by the gates, or modified through a simple RNN-style update. Moreover, EntNet uses an independent gate for writing to each memory slot. Our work is related in the sense that we also have different recurrent models (i.e.,RIMs, though each RIM has different parameters), but we allow the RIMs to communicate with each other sparingly using an attention mechanism.

Capsules and Dynamic Routing: EM Capsules (Hinton et al., 2018) and the preceding Dynamic Capsules (Sabour et al., 2017) use the poses of parts and learned part $\rightarrow$ object relationships to vote for the poses of objects. When multiple parts cast very similar votes, the object is assumed to be present, which is facilitated by an interactive inference (routing) algorithm.

Relational Graph Based Methods: Recent graph-based architectures have studied combinatorial generalization in the context of modeling dynamical systems like physics simulation, multi-object scenes, and motion-capture data, and multiagent systems (Scarselli et al., 2008; Bronstein et al., 2017; Watters et al., 2017; Raposo et al., 2017; Santoro et al., 2017; Gilmer et al., 2017; Van Steenkiste et al., 2018; Kipf et al., 2018; Battaglia et al., 2018; Tacchetti et al., 2018). One can also view our proposed model as a relational graph neural network, where nodes are parameterized as individual RIMs and edges are parameterized by the attention mechanism. Though, its important to emphasize that the topology of the graph induced in the proposed model is dynamic, while in most graph neural networks the topology is fixed.

Default Behaviour: Our work is also related to work in behavioural research that deals with two modes of decision making (Dickinson, 1985; Botvinick & Braver, 2015; Kool & Botvinick, 2018): an automatic systems that relies on habits and a controlled system that uses some privileged information for making decision making. The proposed model also has two modes of input processing, RIMs which activate uses some external sensory information, and hence analogous to controlled system. RIMs which don't activate, they are synonymous to habit based system. There is some work done trying in Reinforcement learning, trying to learn *default policies*, which have shown to improve transfer and generalization in multi-task RL (Teh et al., 2017; Goyal et al., 2019a). The proposed method is different in the sense, we are not trying to learn *default policies* which effect the environment, instead we want to learn mechanisms, which try to understand the environment. State dependent activation of different primitive policies was also studied in (Goyal et al., 2019b), and the authors showed that they can learn different primitives, but they also consider that only a single primitive can be active at a particular time step. Also, note that primitive policies try to *effect* the environment, whereas mechanism try to *understand* the enviornment.

C Experimental Details and Hyperparameters

C.1 RIMs Implementation

The RIMs model consists of three main components: the input attention, the process for selecting activated RIMs, and the communication between RIMs. The input attention closely follows the attention mechanism of (Santoro et al., 2018) but with a significant modification: that all of the weights within the attention mechanism are separate per-block. Thus we remove the normal linear layers and replace them with a batch matrix multiplication over the RIMs (as each block has its own weight matrix). Note that the read-key (or query) is a function of the hidden state of each RIM.

For selecting activated RIMs, we compute the top-k attention weight on the null input over the RIMs. We then select the activated RIMs by using a mask.

We compute the independent dynamics over all RIMs by using a separate LSTM for each RIM. Following this, we compute the communication between RIMs as a multihead attention (Santoro et al., 2018), with the earlier-discussed modification of having separate weight parameters for each block, and also that we added a skip-connection around the attention mechanism. This attention mechanism used 4 heads and in general used a key size and value size of 32. We computed the updates for all RIMs but used the activated-block mask to selectively update only the activated subset of the RIMs.

The use of RIMs introduces two additional hyperparameters over an LSTM/GRU: the number of RIMs and the number of activated RIMs per step. We also observed that having too few activated RIMs tends to hurt optimization and having too many activated RIMs attenuates the improvements to generalization. For the future it would be interesting to explore dynamic ways of controlling how many RIMs to activate.

C.2 Detailed Model Hyperparameters

Table 3 lists the different hyperparameters.

Table 3: Hyperparameters

Parameter	Value
Optimizer	Adam(Kingma & Ba, 2014)
learning rate	$7 \cdot 10^{-4}$
batch size	64
Inp keys	64
Inp Values	Size of individual RIM * 4
Inp Heads	4
Inp Dropout	0.1
Comm keys	32
Comm Values	32
Comm heads	4
Comm Dropout	0.1

C.3 Future Architectural Changes

We have not conducted systematic optimizations of the proposed architecture. We believe that even principled hyperparameter tuning may significantly improve performance for many of the tasks we have considered in the paper. We briefly mention a few architectural changes which we have studied:

- On the output side, we concatenate the representations of the different RIMs, and use the concatenated representation for learning a policy (in RL experiments) or for predicting the input at the next time step (for bouncing balls as well as all other experiments). We empirically found that adding another layer of (multi-headed) key value attention on the output seems to improve the results. We have not included this change
- In our experiments, we shared the same decoder for all the RIMs, i.e., we concatenate the representations of different RIMS, and feed the concatenated representations to the decoder. In the future it would be interesting to think of ways to allow a more “structured” decoder. The reason for this is that even if the RIMs generalize to new environments, the shared decoder can fail to do so. So changing the structure of decoder could be helpful.

- For the RL experiments, we also tried providing the previous actions, rewards, language instruction as input to decide the activation of RIMs. This is consistent with the idea of *efference copies* as proposed by von Helmholtz (1867); von Holst & Mittelstaedt (1950), i.e., using copies of motor signals as inputs. Preliminary experiments shows that this improves the performance in Atari games.

C.4 Language Modeling

Table 4: Wikitext-2 results

Approach	Num. Parameters	Train PPL	Valid PPL	Test PPL
LSTM (2-layer)	21.2M	39.78	109.25	102.53
Relational Memory (Santoro et al., 2018)	11M	n/a	112.77	107.21
RIMs (2-layer, k_T = 6, k_A = 6)	23.7M	41.27	103.60	**98.66**

We investigate the task of word-based language modeling. We ran experiments on the wikitext-2 dataset (Merity et al., 2016). We ran each experiment for a fixed 100 epochs. These results are in Table 4. Our goal in this experiment is to demonstrate the breadth of the approach by showing that RIMs performs well even on datasets which are noisy and drawn from the real-world.

C.5 Copying Task

We used a learning rate of 0.001 with the Adam Optimizer and trained each model for 150 epochs (unless the model was stuck, we found that this was enough to bring the training error close to zero). For the RIMs model we used 600 units split across 6 RIMs (100 units per block). For the LSTM we used a total of 600 units. We did not explore this extensively but we qualitatively found that the results on copying were not very sensitive to the exact number of units.

The sequences to be copied first have 10 random digits (from 0-8), then a span of zeros of some length, followed by a special indicator “9” in the input which instructs the model to begin outputting the copied sequence.

In our experiments, we trained the models with “zero spans” of length 50 and evaluated on the model with “zero spans” of length 200. We note that all the ablations were run with the default parameters (i.e number of keys, values as for RIMs model) for 100 epochs. Tab. 5 shows the effect of two baselines as compared to the RIMs model (a) When we allow the input attention for activation of different RIMs but we dont allow different RIMs to communicate. (b) No Input attention, but we allow different RIMs to communicate with each other. Tab. 5 shows that the proposed method is better than both of these baselines. For copy task, we used 1 head in input attention, and 4 heads for RIMs communication. We note that even with 1 RIM, its not exactly same as a LSTM, because each RIM can still reference itself.

C.6 Sequential MNIST Task

In this task we considered classifying binary MNIST digits by feeding the pixels to an RNN (in a fixed order scanning over the image). As the focus of this work is on generalization, we introduced a variant on this task where the training digits are at a resolution of 14 x 14 (sequence length of 196). We then evaluated on MNIST digits of different higher resolutions (16 x 16, 19 x 19, and 24 x 24). When re-scaling the images, we used the nearest-neighbor based down-scaling and performed binarization after re-scaling. We trained with a learning rate of 0.0001 and the Adam optimizer. For RIMs we used a total of 600 hidden units split across 6 RIMs (100 units per block). For the LSTM we used a total of 600 units.

C.7 Imitation Learning: Robustness to Noise in State Distribution

Here, we consider imitation learning where we have training trajectories generated from an expert (Table 6). We evaluate our model on continuous control tasks in Mujoco (in our case, Half-Cheetah) (Todorov et al., 2012). We take the rendered images as input and compared the proposed model with recurrent policy (i.e., LSTM). Since, using rendered image of the input does not tell anything about the velocity of the Half-Cheetah, it makes the task partially observable. In order to test how well the proposed model generalizes during test, we add some noise (in the joints of the half-cheetah body). As one can see, after adding noise LSTM baselines performs poorly. On the other hand, for the proposed model, there’s also a drop in performance but not as bad as for the LSTM baseline.

We use the convolutional network from (Ha & Schmidhuber, 2018) as our encoder, a GRU (Chung et al., 2015) with 600 units as deterministic path in the dynamics model, and implement all other functions as two fully connected layers of size 256 with ReLU activations. Since, here we are using images as input, which makes

Table 5: **Error (CE for last 10 time steps) on the copying task**. Note that while all of the methods are able to learn to copy on the length seen during training, the RIMs model generalizes to sequences longer than those seen during training whereas the LSTM fails catastrophically.

Approach	Train Length 50	Test Length 200
RIMs	**0.00**	**0.00**
With input Attention and No Communication		
RIMs (k_T = 4, k_A = 2)	2.3	1.6
RIMs (k_T = 4, k_A = 3)	1.7	4.3
RIMs (k_T = 5, k_A = 2)	2.5	4,7
RIMs (k_T = 5, k_A = 3)	0.4	4.0
RIMs (k_T = 5, k_A = 4)	0.2	0.7
RIMs (k_T = 6, k_A = 2)	3.3	2.4
RIMs (k_T = 6, k_A = 3)	1.2	1.0
RIMs (k_T = 6, k_A = 4)	0.7	5.0
RIMs (k_T = 6, k_A = 5)	0.22	0.56
With No input Attention and Full Communication		
RIMs (k_T = 6, k_A = 6, h_{dim} = 600)	0.0	0.7
RIMs (k_T = 5, k_A = 5, h_{dim} = 500)	0.0	1.7
RIMs (k_T = 2, k_A = 2, h_{dim} = 256)	0.0	2.9
RIMs (k_T = 2, k_A = 2, h_{dim} = 512)	0.0	1.8
RIMs (k_T = 1, k_A = 1, h_{dim} = 512)	0.0	0.2

Table 6: **Imitation Learning:** Results on the half-cheetah imitation learning task. RIMs outperforms a baseline LSTM when we evaluate with perturbations not observed during training (left). An example of an input image fed to the model (right).

Method / Setting	Training Observed Reward	Perturbed States Observed Reward
LSTM (Recurrent Policy)	5400 ± 100	2500 ± 300
RIMs (k_T = 6, k_A = 3)	5300 ± 200	3800 ± 200
RIMs (k_T = 6, k_A = 6)	5500 ± 100	2700 ± 400

the task, partially observable. Hence, we concatenate the past 4 observations, and then feed the concatenated observations input to GRU (or our model). For our model, we use 6 RIMs, each of size 100, and we set $k_a = 3$. We follow the same setting as in (Hafner et al., 2018; Sodhani et al., 2019)

C.8 Generalization to Distractors: Algorithm Implementation Details

We evaluate the proposed framework using Adavantage Actor-Critic (A2C) to learn a policy $\pi_\theta(a|s, g)$ conditioned on the goal. To evaluate the performance of proposed method, we use a range of maze multi-room tasks from the gym-minigrid framework (Chevalier-Boisvert & Willems, 2018) and the A2C implementation from (Chevalier-Boisvert & Willems, 2018). For the maze tasks, we used agent's relative distance to the absolute goal position as "goal".

For the maze environments, we use A2C with 48 parallel workers. Our actor network and critic networks consist of two and three fully connected layers respectively, each of which have 128 hidden units. The encoder network is also parameterized as a neural network, which consists of 1 fully connected layer. We use RMSProp with an initial learning rate of 0.0007 to train the models. Due to the partially observable nature of the environment, we further use a LSTM to encode the state and summarize the past observations.

C.9 MiniGrid Environments for OpenAI Gym

The MultiRoom environments used for this research are part of MiniGrid, which is an open source gridworld package. This package includes a family of reinforcement learning environments compatible with the OpenAI

https://github.com/maximecb/gym-minigrid

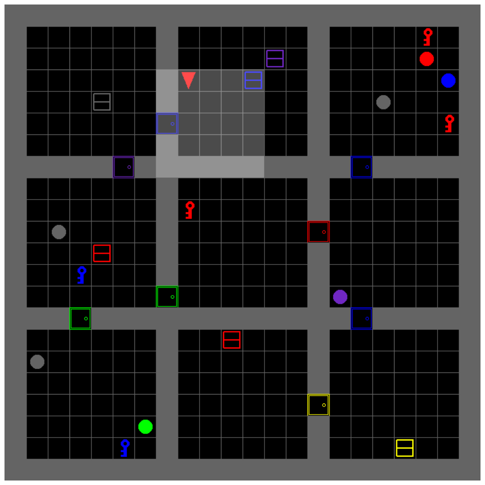


Figure 7: An example of the minigrid task.

Gym framework. Many of these environments are parameterizable so that the difficulty of tasks can be adjusted (e.g., the size of rooms is often adjustable).

C.9.1 The World

In MiniGrid, the world is a grid of size NxN. Each tile in the grid contains exactly zero or one object. The possible object types are wall, door, key, ball, box and goal. Each object has an associated discrete color, which can be one of red, green, blue, purple, yellow and grey. By default, walls are always grey and goal squares are always green.

C.9.2 Reward Function

Rewards are sparse for all MiniGrid environments. In the MultiRoom environment, episodes are terminated with a positive reward when the agent reaches the green goal square. Otherwise, episodes are terminated with zero reward when a time step limit is reached. In the FindObj environment, the agent receives a positive reward if it reaches the object to be found, otherwise zero reward if the time step limit is reached.

The formula for calculating positive sparse rewards is $1 - 0.9 * (step\_count/max\_steps)$. That is, rewards are always between zero and one, and the quicker the agent can successfully complete an episode, the closer to 1 the reward will be. The max_steps parameter is different for each environment, and varies depending on the size of each environment, with larger environments having a higher time step limit.

C.9.3 Action Space

There are seven actions in MiniGrid: turn left, turn right, move forward, pick up an object, drop an object, toggle and done. For the purpose of this paper, the pick up, drop and done actions are irrelevant. The agent can use the turn left and turn right action to rotate and face one of 4 possible directions (north, south, east, west). The move forward action makes the agent move from its current tile onto the tile in the direction it is currently facing, provided there is nothing on that tile, or that the tile contains an open door. The agent can open doors if they are right in front of it by using the toggle action.

C.9.4 Observation Space

Observations in MiniGrid are partial and egocentric. By default, the agent sees a square of 7x7 tiles in the direction it is facing. These include the tile the agent is standing on. The agent cannot see through walls or closed doors. The observations are provided as a tensor of shape 7x7x3. However, note that these are not RGB images. Each tile is encoded using 3 integer values: one describing the type of object contained in the cell, one describing its color, and a flag indicating whether doors are open or closed. This compact encoding was chosen for space efficiency and to enable faster training. The fully observable RGB image view of the environments shown in this paper is provided for human viewing.

C.9.5 Level Generation

The level generation in this task works as follows: (1) Generate the layout of the map (X number of rooms with different sizes (at most size Y) and green goal) (2) Add the agent to the map at a random location in the first room. (3) Add the goal at a random location in the last room. A neural network parameterized as CNN is used to process the visual observation.

We follow the same architecture as (Chevalier-Boisvert & Willems, 2018) but we replace the LSTM layer with BlockLSTM.

C.10 Bouncing balls

We use the bouncing-ball dataset from (Van Steenkiste et al., 2018). The dataset consists of 50,000 training examples and 10,000 test examples showing ∼50 frames of either 4 solid balls bouncing in a confined square geometry, 6-8 balls bouncing in a confined geometry, or 3 balls bouncing in a confined geometry with a random occluded region. In all cases, the balls bounce off the wall as well as off one another. We train baselines as well as proposed model for about 100 epochs using 0.0007 as learning rate and using Adam as optimizer (Kingma & Ba, 2014). We use the same architecture for encoder as well as decoder as in (Van Steenkiste et al., 2018). We train the proposed model as well as the baselines for 100 epochs. Below, we highlight a few different results.

C.10.1 Different RIMs attend to different balls

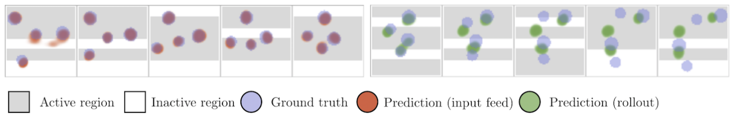


Figure 8: **Different RIMs attending to Different Balls**. For understanding what each RIM is actually doing, we associate each with a separate encoder, which are spatially masked. Only 4 encoders can be active at any particular instant and there are four different balls. We did this to check if there would be the expected geometric activation of RIMs. 1.) Early in training, RIM activations correlated more strongly with the locations of the four different balls. Later in training, this correlation decreased and the active strips did not correlate as strongly with the location of balls. As the model got better at predicting the location, it needed to attend less to the actual objects. The top row shows every 5th frame when the truth is fed in and the bottom shows the results during rollout. The gray region shows the active block. In the top row, the orange corresponds to the prediction and in the bottom, green corresponds to the prediction.

In order to visualize what each RIM is doing, we associate each RIM with a different encoder. By performing spatial masking on the input, we can control the possible spatial input to each RIM. We use six non-overlapping horizontal strips and allow only 4 RIMs to be active at a time (shown in Fig. 8). The mask is fixed mask of zeros with a band of ones that is multiplied by the input to each encoder. Therefore, each of the 6 encoders gets 1/6th of the input. The goal was to see how the RIM activation patterns changed/correlated with the locations of the balls. We find that early in training, the RIMs' activations are strongly correlated with the location of the 4 balls. However, after training has proceeded for some time this correlation deteriorates. This is likely because the predictable dynamics of the system do not necessitate constant attention.

C.10.2 Comparison with LSTM Baselines

In Figures 9, 10, 11, and 12 we highlight different baselines and how these compare to the proposed RIMs model.

C.10.3 Occlusion

In Fig. 13, we show the performance of RIMs on the curtain dataset. We find RIMs are able to track balls through the occlusion without difficulty. Note that the LSTM baseline, is also able to track the ball through the "invisible" curtain.

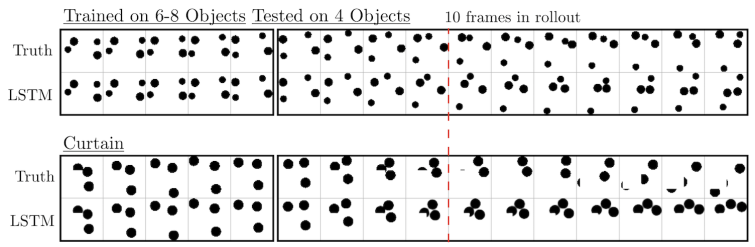


Figure 9: **Example of the other LSTM baselines**. For the 2 other experiments that we consider, here we show example outputs of our LSTM baselines. In each row, the top panel represents the ground truth and the bottom represents the prediction. All shown examples use an LSTM with 250 hidden units, as shown in Fig. 4. Frames are plotted every 3rd time step. The red line marks 10 rollout frames. This is marked because after this we do not find BCE to be a reliable measure of dissimilarity.

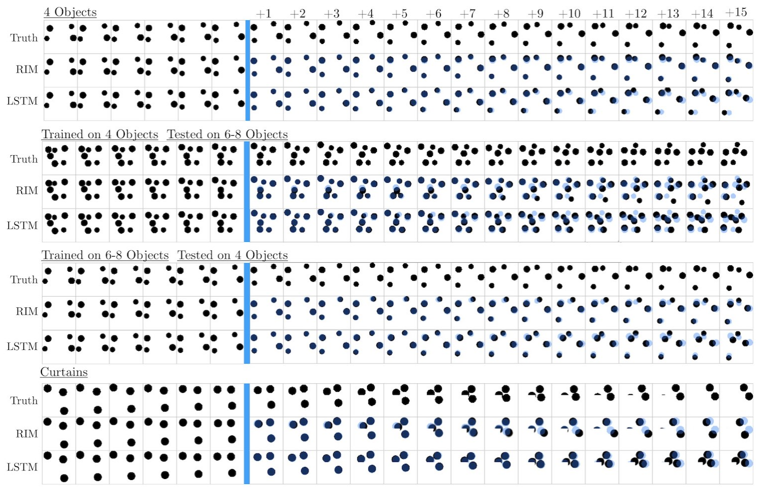


Figure 10: **Comparison of RIMs to LSTM baseline**. For 4 different experiments in the text, we compare RIMs to two different LSTM baselines. In all cases we find that during rollout, RIMs perform better than the LSTMs at accurately capturing the trajectories of the balls through time. Due to the number of hard collisions, accurate modeling is very difficult. In all cases, the first 15 frames of ground truth are fed in (last 6 shown) and then the system is rolled out for the next 15 time steps, computing the binary cross entropy between the prediction and the true balls at each instant, as in Van Steenkiste et al. (2018). See the Appendix for losses over the entire 35 frame rollout trajectory. In the predictions, the transparent blue shows the ground truth, overlaid to help guide the eye.

C.10.4 Study of Transfer

It is interesting to ask how models trained on a dataset with 6-8 balls perform on a dataset with 4 balls. In Fig. 14 we show predictions during feed-in and rollout phases.

C.11 Ablations

We present one ablation in addition to the ones in Section 4.4. In this experiment, we study the effect on input attention (i.e top down attention) as well as the use of multi-headed head key-value attention. We compare the proposed model (with input attention as well as multi-headed key value attention) with 2 baselines: (a) In which we remove the input attention (and force all the RIMs to communicate with each other (b) We use 1 head for key

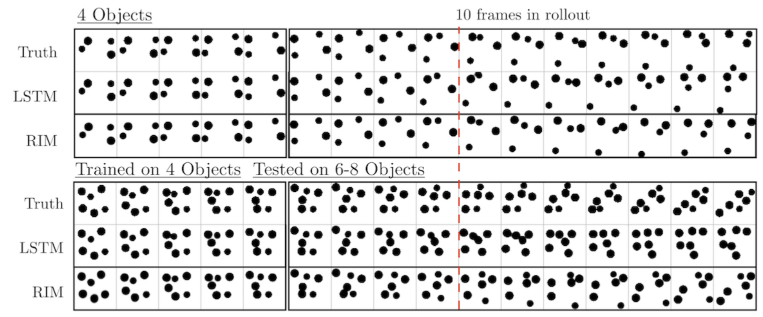


Figure 11: **Comparison between RIMs and LSTM baseline**. For the 4 ball task and the 6-8 ball extrapolation task, here we show an example output of from our LSTM baseline and from RIMs. All shown examples use an LSTM with 250 hidden units, as shown in Fig. 4. Frames are plotted every 3rd time step. The red line marks 10 rollout frames. This is marked because after this we do not find BCE to be a reliable measure of dissimilarity.

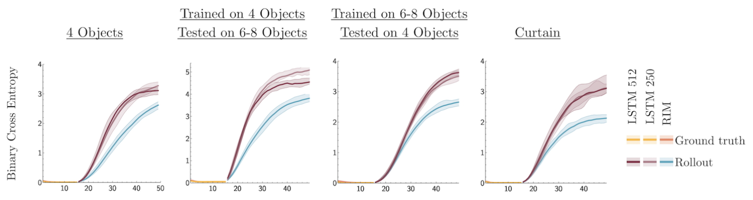


Figure 12: **Comparison of RIMs to LSTM baseline**. For 4 different experiments in the text, we compare RIMs to two different LSTM baselines. In all cases we find that during rollout, RIMs perform better than the LSTMs at accurately capturing the trajectories of the balls through time. Due to the number of hard collisions, accurate modeling is very difficult.

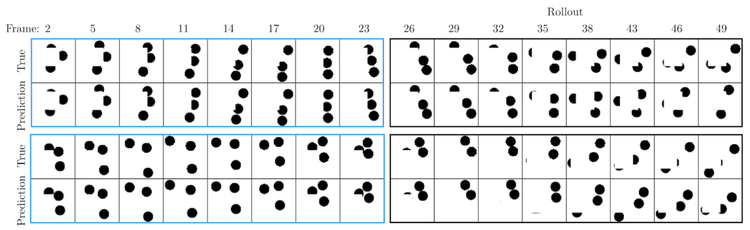


Figure 13: **RIMs on dataset with an occlusion**. We show two trajectories (top and bottom) of three balls. For the left frames, at each step the true frame is used as input. On the right, outlined in black, the previous output is used as input.

value attention as compared to multi-headed key-value attention. Results comparing the proposed model, with these two baselines is shown in Fig. 15. In Fig. 16, we show the predictions that result from the model with only one active head.

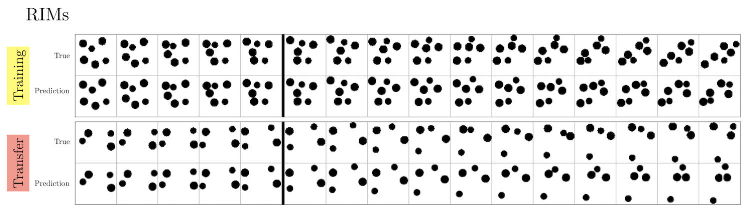


Figure 14: **RIMs transferred on new data**. We train the RIMs model on the 6-8 ball dataset (as shown in the top row). Then, we apply the model to the 4 ball dataset, as shown in the bottom.

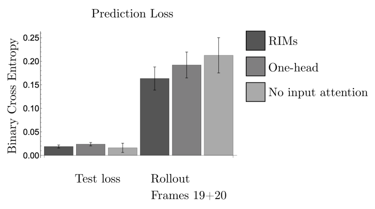


Figure 15: **Ablation loss** For the normal, a one-head model, and without input attention, we show the loss during training and the loss for the 4th and 5th frame of rollout. We find that the one-head and without input attention models perform worse than the normal RIMs model during the rollout phase.

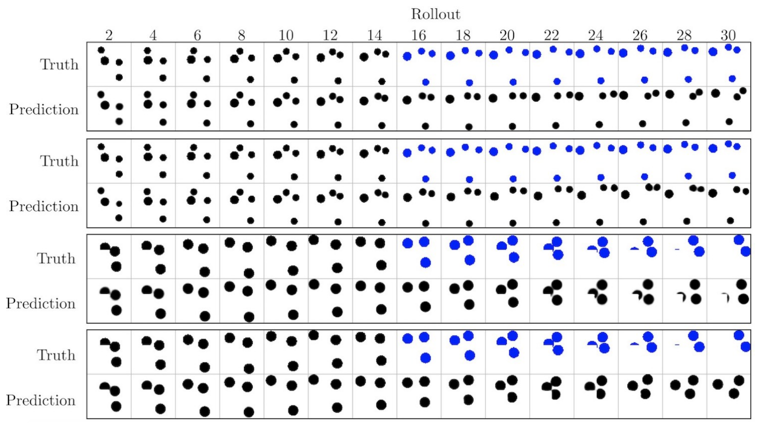


Figure 16: **One head and no attention** Using one head and no attention models, we show the rollout predictions in blue. On top we show results on the 4 ball dataset and on the bottom we show results on the curtains dataset.

C.12 Atari

We used open-source implementation of PPO from (Kostrikov, 2018) with default parameters. We ran the proposed algorihtm with 6 RIMs, and kept the number of activated RIMs to 4/5. We have not done any hyper-parameter search for Atari experiments.

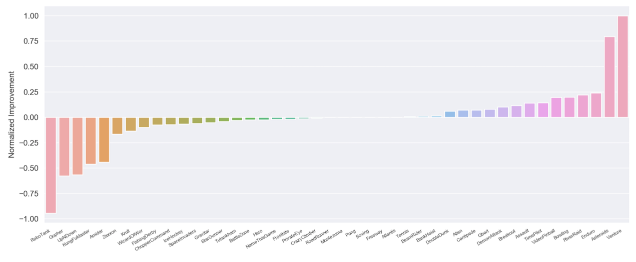


Figure 17: A comparison showing relative improvement of RIMs with $k_A = 5$ over a $k_A = 4$ baseline. Using $k_A = 5$ performs slightly worse than $k_A = 4$ but still outperforms PPO, and has similar results across the majority of games.

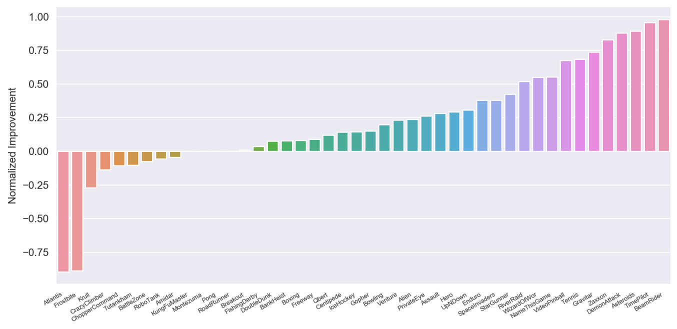


Figure 18: RIMs-PPO relative score improvement over LSTM-PPO baseline (Schulman et al., 2017) across all Atari games averaged over 3 trials per game. In both cases PPO was used with the exact same settings with the only change being the choice of the recurrent architecture (RIMs with $k_A = 5$).

C.12.1 Transfer on Atari

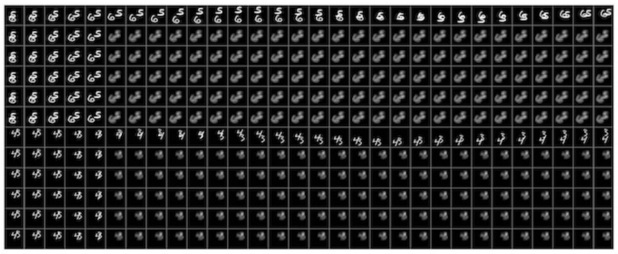
As a very preliminary result, we investigate feature transfer between randomly selected Atari games. In order to study this question, we follow the experimental protocol of Rusu et al. (2016).

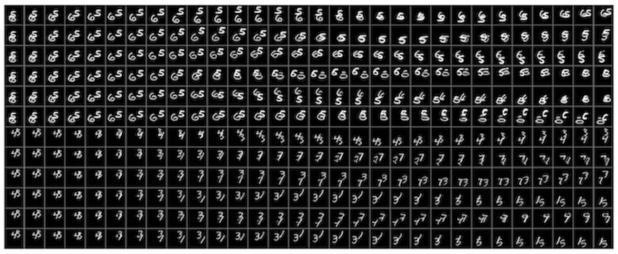
We start by training RIMs on three source games (Pong, River Raid, and Seaquest) and test if the learned features transfer to a different subset of randomly selected target games (Alien, Asterix, Boxing, Centipede, Gopher, Hero, James Bond, Krull, Robotank, Road Runner, Star Gunner, and Wizard of Wor). We observe, that RIMs result in positive transfer in 9 out of 12 target games, with three cases of negative transfer. On the other hand progressive networks (Rusu et al., 2016) result in positive transfer in 8 out of 12 target games, and two cases of negative transfer. We also compare to LSTM baseline, which yields positive transfer in 3 of 12 games.

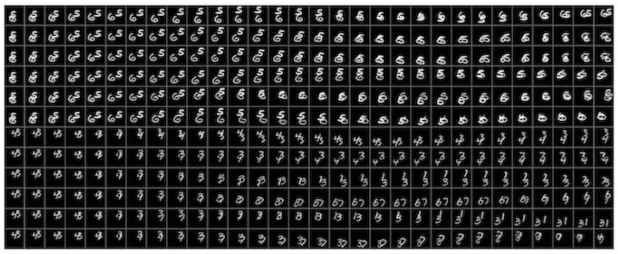
C.13 Bouncing MNIST: Dropping off RIMs

We use the Stochastic Moving MNIST (SM-MNIST) (Denton & Fergus, 2018) dataset which consists of sequences of frames of size 64×64, containing one or two MNIST digits moving and bouncing off the walls. Training sequences are generated on the fly by sampling two different MNIST digits from the training set (60k total digits) and two distinct trajectories.

Here, we show the effect of masking out a particular RIM and study the effect of the masking on the ensemble of RIMs. Ideally, we would want different RIMs not to co-adapt with each other. So, masking out a particular RIM should not really effect the dynamics of the entire model. We show qualitative comparisons in Fig. 19, 20, 21, 22, 23. In each of these figures, the model gets the ground truth image as input for first 5 time steps, and then asked to simulate the dynamics for next 25 time-steps. We find that sparsity is needed otherwise different RIMs co-adapt with each other (for ex. see Fig. 20, 22, 23). We tried similar masking experiments for different models like RMC, Transformers, EntNet (which learns a mixture of experts), LSTMs, but all of them failed to do anything meaningful (after masking). We suspect this is partly due to learning a homogeneous network.







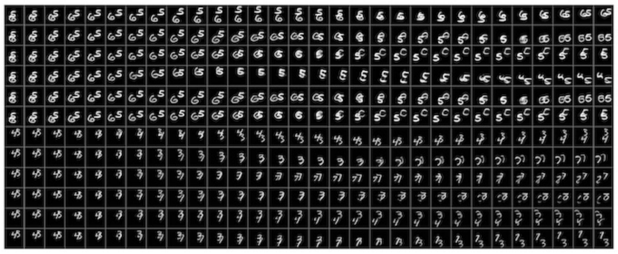
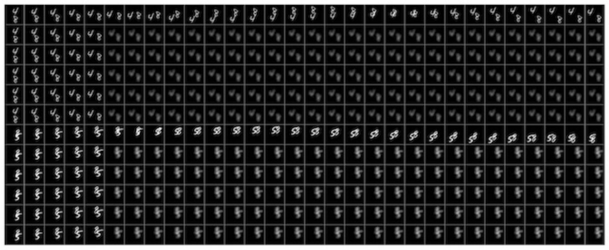
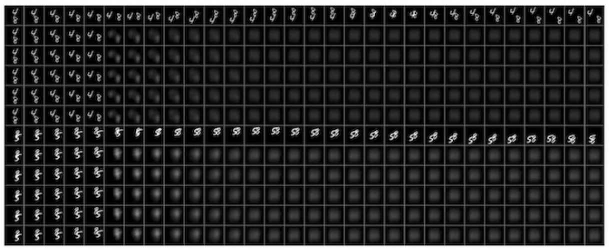
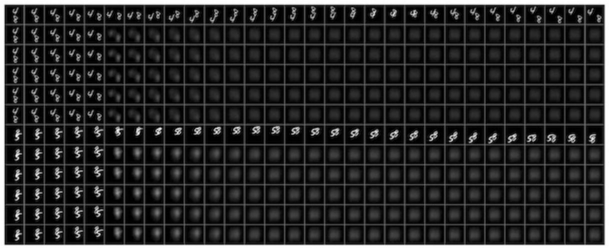


Figure 19: 4 RIMs, (top k = 2). Each sub-figure shows the effect of masking a particular RIM and studying the effect of masking on the other RIMs. For example, the top figure shows the effect of masking the first RIM, the second figure shows the effect of masking the second RIM etc.







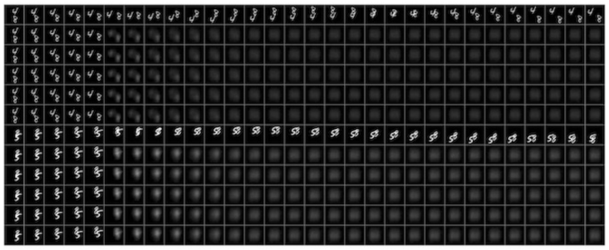


Figure 20: 4 RIMs, (top k = 3). Each sub-figure shows the effect of masking a particular RIM and studying the effect of masking on the other RIMs. For example, the top figure shows the effect of masking the first RIM, the second figure shows the effect of masking the second RIM etc.

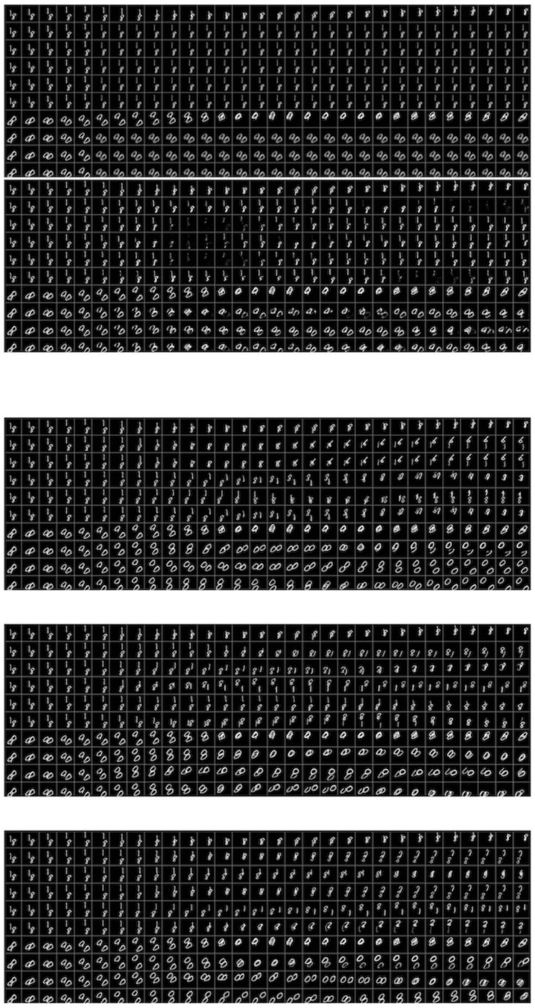


Figure 21: 400dim, 5 RIMs, (top k = 2). Each sub-figure shows the effect of masking a particular RIM and studying the effect of masking on the other RIMs. For example, the top figure shows the effect of masking the first RIM, the second figure shows the effect of masking the second RIM etc.

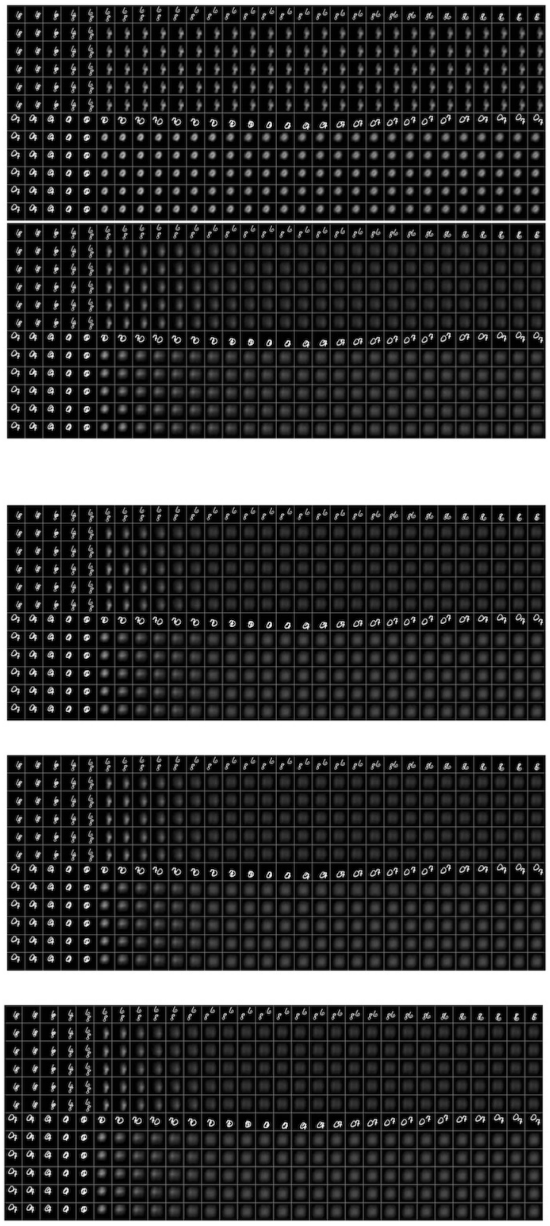


Figure 22: 400dim, 5 blocks, (top k = 3). Each sub-figure shows the effect of masking a particular RIM and studying the effect of masking on the other RIMs. For examples, the top figure shows the effect of masking the

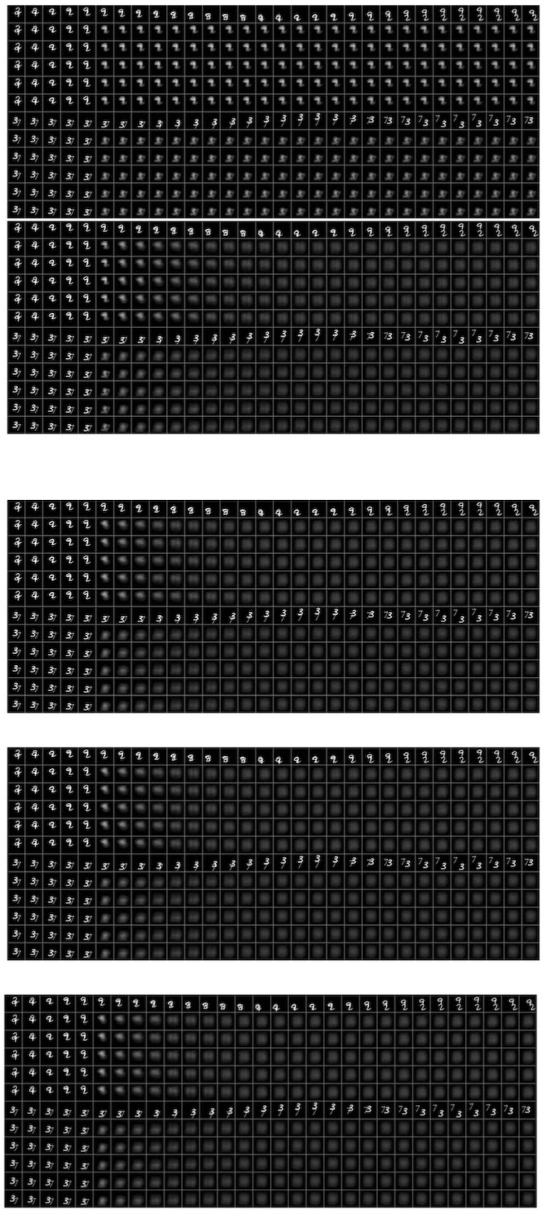


Figure 23: 400dim, 5 blocks, (top k = 4). Each sub-figure shows the effect of masking a particular RIM and studying the effect of masking on the other RIMs. For example, the top figure shows the effect of masking the

C.13.1 Atari Results: Comparison with LSTM-PPO

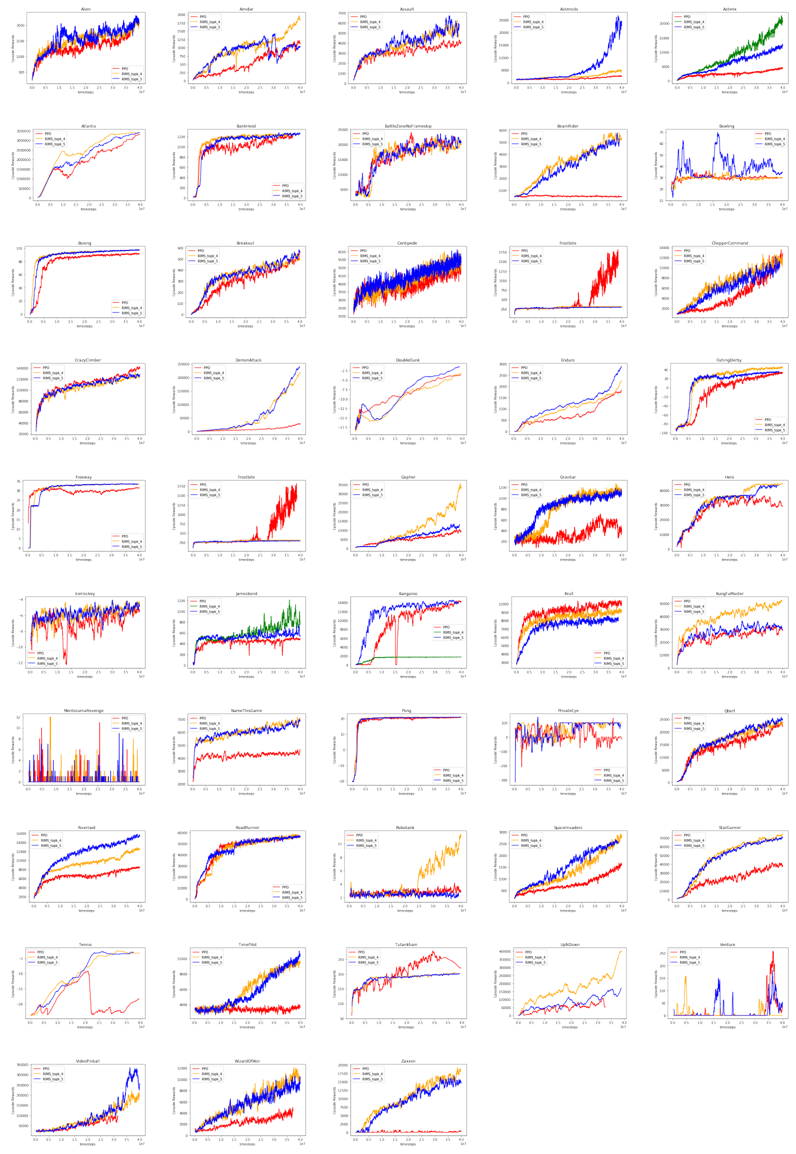


Figure 24: **Comparing RIMs-PPO with LSTM-PPO:** Learning curves for $k_A = 4$, $k_A = 5$ RIMs-PPO models and the LSTM-PPO baseline across all Atari games.

C.13.2 Atari Results: No Input attention

Here we compare the proposed method to the baseline, where we dont use input attention, and we force different RIMs to communicate with each at all the time steps.

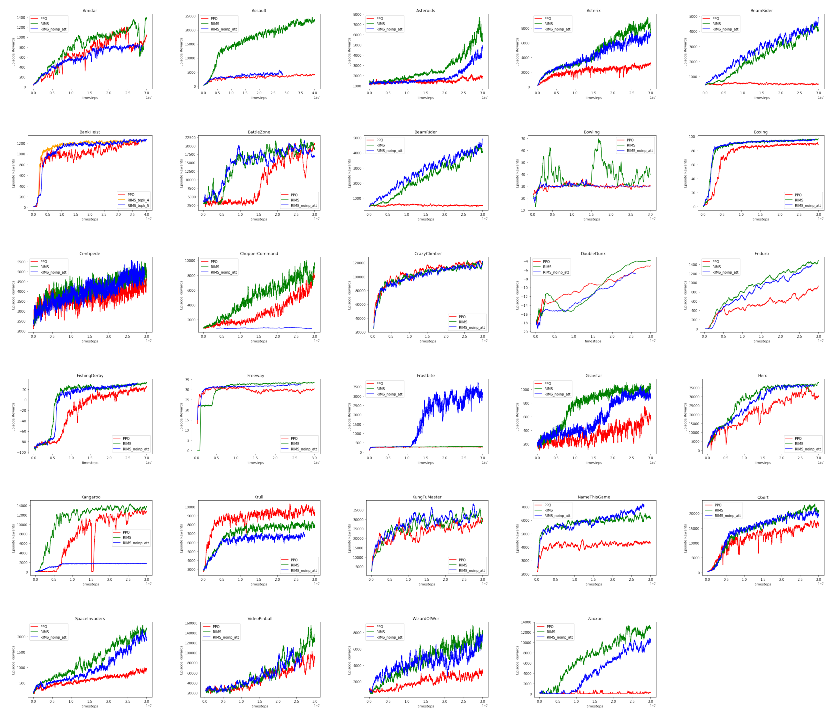


Figure 25: **Baseline agent with no input attention mechanism:** Here we compare the RIMs to the baseline, where their is no input attention (i.e., top down attention) as well as all the RIMs communicate with each other at all the time steps. Learning curves for RIMs-PPO models, Baseline Agent, the LSTM-PPO baseline across 30 Atari games.

5.2 Graph neural networks: Learning graph representations

5.2.1 Background

Graph structured data is ubiquitous in the sciences, social sciences, and many other fields. Common examples include, but are not limited to:

1. predicting properties of social networks,
2. clustering many different forms of data,
3. predicting properties of physical systems such as drug molecules or crystal structures,
4. predicting the properties of partially or unobserved nodes.

Convolutional neural networks have had profound effects in image analysis and processing.[87,144,136] For images, the spatial relationship between input pixels is apparent. However, for graph structured data the relationship is less clear. Graph (convolutional) Neural Networks (GNNs) are a very powerful tool that leverages much success from the image domain.[173,165] Developing new tools that leverage specific aspects of graph structured data and also reshaping tools developed for other domains to graph structured datasets are active ares of research.

In the two manuscripts discussed in this chapter, we develop new computational methods for graph structured data. In the first, we introduce vGraph, a method to learn node representations and community membership in a collaborative fashion. We show that this method achieves state-of-the-art performance in both tasks across a wide range of datasets.[143]

In the second manuscript, we introduce InfoGraph, a computational method that maximizes the mutual information between the graph-level representation and the representations of substructures of different scales.[142]

5.2.2 Contributions

I discussed ideas with Fan-Yun Sun and wrote code to implement these ideas. We both worked on extending InfoGraph to new datasets. We both wrote and edited both manuscripts.

5.2.3 Publication: vGraph: A Generative Model for Joint Community Detection and Node Representation Learning

vGraph: A Generative Model for Joint Community Detection and Node Representation Learning

Fan-Yun Sun[1,2], **Meng Qu**[2], **Jordan Hoffmann**[2,3], **Chin-Wei Huang**[2,4], **Jian Tang**[2,5,6]
[1]National Taiwan University,
[2]Mila-Quebec Institute for Learning Algorithms, Canada
[3]Harvard University, USA
[4]Element AI, Canada
[5]HEC Montreal, Canada
[6]CIFAR AI Research Chair
b04902045@ntu.edu.tw

Abstract

This paper focuses on two fundamental tasks of graph analysis: community detection and node representation learning, which capture the global and local structures of graphs, respectively. In the current literature, these two tasks are usually independently studied while they are actually highly correlated. We propose a probabilistic generative model called vGraph to learn community membership and node representation collaboratively. Specifically, we assume that each node can be represented as a mixture of communities, and each community is defined as a multinomial distribution over nodes. Both the mixing coefficients and the community distribution are parameterized by the low-dimensional representations of the nodes and communities. We designed an effective variational inference algorithm which regularizes the community membership of neighboring nodes to be similar in the latent space. Experimental results on multiple real-world graphs show that vGraph is very effective in both community detection and node representation learning, outperforming many competitive baselines in both tasks. We show that the framework of vGraph is quite flexible and can be easily extended to detect hierarchical communities.

1 Introduction

Graphs, or networks, are a general and flexible data structure to encode complex relationships among objects. Examples of real-world graphs include social networks, airline networks, protein-protein interaction networks, and traffic networks. Recently, there has been increasing interest from both academic and industrial communities in analyzing graphical data. Examples span a variety of domains and applications such as node classification [3, 26] and link prediction [8, 32] in social networks, role prediction in protein-protein interaction networks [16], and prediction of information diffusion in social and citation networks [22].

One fundamental task of graph analysis is community detection, which aims to cluster nodes into multiple groups called communities. Each community is a set of nodes that are more closely connected to each other than to nodes in different communities. A community level description is able to capture important information about a graph's global structure. Such a description is useful in many real-world applications, such as identifying users with similar interests in social networks [22] or proteins with similar functionality in biochemical networks [16]. Community detection has been extensively studied in the literature, and a number of methods have been proposed, including algorithmic approaches [1, 5] and probabilistic models [10, 20, 36, 37]. A classical approach to detect communities is spectral clustering [34], which assumes that neighboring nodes tend to belong to the same communities and detects communities by finding the eigenvectors of the graph Laplacian.

Another important task of graph analysis is node representation learning, where nodes are described using low-dimensional features. Node representations effectively capture local graph structure and are often used as features for many prediction tasks. Modern methods for learning node embeddings [11, 24, 26] have proved effective on a variety of tasks such as node classification [3, 26], link prediction [8, 32] and graph visualization [27, 31].

Clustering, which captures the global structure of graphs, and learning node embeddings, which captures local structure, are typically studied separately. Clustering is often used for exploratory analysis, while generating node embeddings is often done for predictive analysis. However, these two tasks are very correlated and it may be beneficial to perform both tasks simultaneously. The intuition is that (1) node representations can be used as good features for community detection (e.g., through K-means) [4, 25, 29], and (2) the node community membership can provide good contexts for learning node representations [33]. However, how to leverage the relatedness of node clustering and node embedding in a unified framework for joint community detection and node representation learning is under-explored.

In this paper, we propose a novel probabilistic generative model called vGraph for joint community detection and node representation learning. vGraph assumes that each node v can be represented as a mixture of multiple communities and is described by a multinomial distribution over communities z, i.e., $p(z|v)$. Meanwhile, each community z is modeled as a distribution over the nodes v, i.e., $p(v|z)$. vGraph models the process of generating the neighbors for each node. Given a node u, we first draw a community assignment z from $p(z|u)$. This indicates which community the node is going to interact with. Given the community assignment z, we generate an edge (u, v) by drawing another node v according to the community distribution $p(v|z)$. Both the distributions $p(z|v)$ and $p(v|z)$ are parameterized by the low-dimensional representations of the nodes and communities. As a result, this approach allows the node representations and the communities to interact in a mutually beneficial way. We also design a very effective algorithm for inference with backpropagation. We use variational inference for maximizing the lower-bound of the data likelihood. The Gumbel-Softmax [13] trick is leveraged since the community membership variables are discrete. Inspired by existing spectral clustering methods [6], we added a smoothness regularization term to the objective function of the variational inference routine to ensure that community membership of neighboring nodes is similar. The whole framework of vGraph is very flexible and general. We also show that it can be easily extended to detect hierarchical communities.

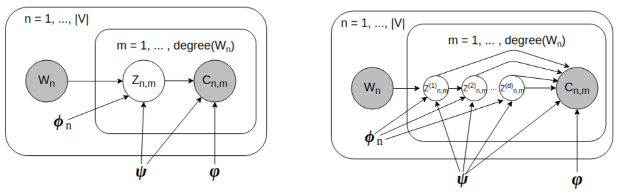
In the experiment section, we show results on three tasks: overlapping community detection, non-overlapping community detection, and node classification– all using various real-world datasets. Our results show that vGraph is very competitive with existing state-of-the-art approaches for these tasks. We also present results on hierarchical community detection. Relevant source codes have been made public [1].

2 Related Work

Community Detection. Many community detection methods are based on matrix factorization techniques. Typically, these methods try to recover the node-community affiliation matrix by performing a low-rank decomposition of the graph adjacency matrix or other related matrices [17, 18, 32, 36]. These methods are not scalable due to the complexity of matrix factorization, and their performance is restricted by the capacity of the bi-linear models. Many other studies develop generative models for community detection. Their basic idea is to characterize the generation process of graphs and cast community detection as an inference problem [37, 38, 39]. However, the computational complexity of these methods is also high due to complicated inference. Compared with these approaches, vGraph is more scalable and can be efficiently optimized with backpropagation and Gumbel-Softmax [13, 19]. Additionally, vGraph is able to learn and leverage the node representations for community detection.

Node Representation Learning. The goal of node representation learning is to learn distributed representations of nodes in graphs so that nodes with similar local connectivity tend to have sim-

[1]https://github.com/fanyun-sun/vGraph



(a) vGraph (b) Hierarchical vGraph

Figure 1: The diagram on the left represents the graphical model of vGraph and the diagram on the right represents the graphical model of the hierarchical extension. $\boldsymbol{\phi}_n$ is the embedding of node w_n, $\boldsymbol{\psi}$ denotes the embedding of communities, and $\boldsymbol{\varphi}$ denotes the embeddings of nodes used in $p(c|z)$. Refer to Eq. 2 and Eq. 3.

ilar representations. Some representative methods include DeepWalk [24], LINE [26], node2vec [11] and GraphRep [3]. Typically, these methods explore the local connectivity of each node by conducting random walks with either breadth-first search [24] or depth-first search [26]. Despite their effectiveness in a variety of applications, these methods mainly focus on preserving the local structure of graphs, therefore ignoring global community information. In vGraph, we address this limitation by treating the community label as a latent variable. This way, the community label can provide additional contextual information which enables the learned node representations to capture the global community information.

Framework for node representation learning and community detection. There exists previous work [4, 14, 29, 30, 33] that attempts to solve community detection and node representation learning jointly. However, their optimization process alternates between community assignment and node representation learning instead of simultaneously solving both tasks [4, 30]. Compared with these methods, vGraph is scalable and the optimization is done end-to-end.

Mixture Models. Methodologically, our method is related to mixture models, particularly topic models (e.g. PSLA [12] and LDA [2]). These methods simulate the generation of words in documents, in which topics are treated as latent variables, whereas we consider generating neighbors for each node in a graph, and the community acts as a latent variable. Compared with these methods, vGraph parameterizes the distributions with node and community embeddings, and all the parameters are trained with backpropagation.

3 Problem Definition

Graphs are ubiquitous in the real-world. Two fundamental tasks on graphs are community detection and learning node embeddings, which focus on global and local graph structures respectively and hence are naturally complementary. In this paper, we study jointly solving these two tasks. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent a graph, where $\mathcal{V} = \{v_1, \ldots, v_V\}$ is a set of vertices and $\mathcal{E} = \{e_{ij}\}$ is the set of edges. Traditional graph embedding aims to learn a node embedding $\boldsymbol{\phi}_i \in \mathbb{R}^d$ for each $v_i \in \mathcal{V}$ where d is predetermined. Community detection aims to extract the community membership $\mathcal{F}$ for each node. Suppose there are K communities on the graph $\mathcal{G}$, we can denote the community assignment of node v_i as $\mathcal{F}(v_i) \subseteq \{1, ..., K\}$. We aim to jointly learn node embeddings $\boldsymbol{\phi}$ and community affiliation of vertices $\mathcal{F}$.

4 Methodology

In this section, we introduce our generative approach vGraph, which aims at collaboratively learning node representations and detecting node communities. Our approach assumes that each node can belong to multiple communities representing different social contexts [7]. Each node should generate different neighbors under different social contexts. vGraph parameterizes the node-community distributions by introducing node and community embeddings. In this way, the node representations can benefit from the detection of node communities. Similarly, the detected community assignment can in turn improve the node representations. Inspired by existing spectral clustering methods [6], we add a smoothness regularization term that encourages linked nodes to be in the same communities.

4.1 vGraph

vGraph models the generation of node neighbors. It assumes that each node can belong to multiple communities. For each node, different neighbors will be generated depending on the community context. Based on the above intuition, we introduce a prior distribution $p(z|w)$ for each node w and a node distribution $p(c|z)$ for each community z. The generative process of each edge (w, c) can be naturally characterized as follows: for node w, we first draw a community assignment $z \sim p(z|w)$, representing the social context of w during the generation process. Then, the linked neighbor c is generated based on the assignment z through $c \sim p(c|z)$. Formally, this generation process can be formulated in a probabilistic way:

$$p(c|w) = \sum_z p(c|z)p(z|w). \tag{1}$$

vGraph parameterizes the distributions $p(z|w)$ and $p(c|z)$ by introducing a set of node embeddings and community embeddings. Note that different sets of node embeddings are used to parametrize the two distributions. Specifically, let $\boldsymbol{\phi}_i$ denote the embedding of node i used in the distribution $p(z|w)$, $\boldsymbol{\varphi}_i$ denote the embedding of node i used in $p(c|z)$, and $\boldsymbol{\psi}_j$ denote the embedding of the j-th community. The prior distribution $p_{\boldsymbol{\phi},\boldsymbol{\psi}}(z|w)$ and the node distribution conditioned on a community $p_{\boldsymbol{\psi},\boldsymbol{\varphi}}(c|z)$ are parameterized by two softmax models:

$$p_{\boldsymbol{\phi},\boldsymbol{\psi}}(z = j|w) = \frac{\exp(\boldsymbol{\phi}_w^T \boldsymbol{\psi}_j)}{\sum_{i=1}^K \exp(\boldsymbol{\phi}_w^T \boldsymbol{\psi}_i)}, \tag{2}$$

$$p_{\boldsymbol{\psi},\boldsymbol{\varphi}}(c|z = j) = \frac{\exp(\boldsymbol{\psi}_j^T \boldsymbol{\varphi}_c)}{\sum_{c' \in \mathcal{V}} \exp(\boldsymbol{\psi}_j^T \boldsymbol{\varphi}_{c'})}. \tag{3}$$

Calculating Eq. 3 can be expensive as it requires summation over all vertices. Thus, for large datasets we can employ negative sampling as done in LINE [26] using the following objective function:

$$\log \sigma(\boldsymbol{\varphi}_c^T \cdot \boldsymbol{\psi}_j) + \sum_{i=1}^K E_{v \sim P_n(v)}[\log \sigma(-\boldsymbol{\varphi}_v^T \cdot \boldsymbol{\psi}_j)], \tag{4}$$

where $\sigma(x) = 1/(1 + \exp(-x))$, $P_n(v)$ is a noise distribution, and K is the number of negative samples. This, combined with stochastic optimization, enables our model to be scalable.

To learn the parameters of vGraph, we try to maximize the log-likelihood of the observed edges, i.e., $\log p_{\boldsymbol{\phi},\boldsymbol{\varphi},\boldsymbol{\psi}}(c|w)$. Since directly optimizing this objective is intractable for large graphs, we instead optimize the following evidence lower bound (ELBO) [15]:

$$\mathcal{L} = E_{z \sim q(z|c,w)}[\log p_{\boldsymbol{\psi},\boldsymbol{\varphi}}(c|z)] - \mathrm{KL}(q(z|c,w)||p_{\boldsymbol{\phi},\boldsymbol{\psi}}(z|w)) \tag{5}$$

where $q(z|c, w)$ is a variational distribution that approximates the true posterior distribution $p(z|c, w)$, and $\mathrm{KL}(\cdot||\cdot)$ represents the Kullback-Leibler divergence between two distributions.

Specifically, we parametrize the variational distribution $q(z|c, w)$ with a neural network as follows:

$$q_{\boldsymbol{\phi},\boldsymbol{\psi}}(z = j|w, c) = \frac{\exp((\boldsymbol{\phi}_w \odot \boldsymbol{\phi}_c)^T \boldsymbol{\psi}_j)}{\sum_{i=1}^K \exp((\boldsymbol{\phi}_w \odot \boldsymbol{\phi}_c)^T \boldsymbol{\psi}_i)}. \tag{6}$$

where $\odot$ denotes element-wise multiplication. We chose element-wise multiplication because it is symmetric and it forces the representation of the edge to be dependent on both nodes.

The variational distribution $q(z|c, w)$ represents the community membership of the edge (w, c). Based on this, we can easily approximate the community membership distribution of each node w, i.e., $p(z|w)$ by aggregating all its neighbors:

$$p(z|w) = \sum_c p(z, c|w) = \sum_c p(z|w, c)p(c|w) \approx \frac{1}{|N(w)|} \sum_{c \in N(w)} q(z|w, c), \tag{7}$$

where $N(w)$ is the set of neighbors of node w. To infer non-overlapping communities, we can simply take the $\arg\max$ of $p(z|w)$. However, when detecting overlapping communities instead of thresholding $p(z|w)$ as in [14], we use

$$\mathcal{F}(w) = \{\arg\max_k q(z = k|w, c)\}_{c \in N(w)}. \tag{8}$$

That is, we assign each edge to one community and then map the edge communities to node communities by gathering nodes incident to all edges within each edge community as in [1].

Complexity. Here we show the complexity of vGraph. Sampling an edge takes constant time, thus calculating Eq. (4) takes $\mathcal{O}(d(M+1))$ time, where M is the number of negative samples and d is the dimension of embeddings (the node embeddings and community embeddings have the same dimension). To calculate Eq. (6), it takes $\mathcal{O}(dK)$ time where K is the number of communities. Thus, an iteration with one sample takes $\mathcal{O}(\max(dM, dK))$ time. In practice the number of updates required is proportional to the number of edges $\mathcal{O}(|\mathcal{E}|)$, thus the overall time complexity of vGraph is $\mathcal{O}(|\mathcal{E}|d\max(M, K))$.

4.2 Community-smoothness Regularized Optimization

For optimization, we need to optimize the lower bound (5) w.r.t. the parameters in the variational distribution and the generative parameters. If z is continuous, the reparameterization trick [15] can be used. However, z is discrete in our case. In principle, we can still estimate the gradient using a score function estimator [9, 35]. However, the score function estimator suffers from a high variance, even when used with a control variate. Thus, we use the Gumbel-Softmax reparametrization [13, 19] to obtain gradients for the evidence lower bound. More specifically, we use the straight-through Gumbel-Softmax estimator [13].

A community can be defined as a group of nodes that are more similar to each other than to those outside the group [23]. For a non-attributed graph, two nodes are similar if they are connected and share similar neighbors. However, vGraph does not explicitly weight local connectivity in this way. To resolve this, inspired by existing spectral clustering studies [6], we augment our training objective with a smoothness regularization term that encourages the learned community distributions of linked nodes to be similar. Formally, the regularization term is given below:

$$\mathcal{L}_{reg} = \lambda \sum_{(w,c) \in \mathcal{E}} \alpha_{w,c} \cdot d(p(z|c), p(z|w)) \tag{9}$$

where λ is a tunable hyperparameter , $\alpha_{w,c}$ is a regularization weight, and $d(\cdot, \cdot)$ is the distance between two distributions (squared difference in our experiments). Motivated by [25], we set $\alpha_{w,c}$ to be the Jaccard's coefficient of node w and c, which is given by:

$$\alpha_{w,c} = \frac{|N(w) \cap N(c)|}{|N(w) \cup N(c)|}, \tag{10}$$

where $N(w)$ denotes the set of neighbors of w. The intuition behind this is that $\alpha_{w,c}$ serves as a similarity measure of how similar the neighbors are between two nodes. Jaccard's coefficient is used for this metric and thus the higher the value of Jaccard's coefficient, the more the two nodes are encouraged to have similar distribution over communities.

By combining the evidence lower bound and the smoothness regularization term, the entire loss function we aim to minimize is given below:

$$\mathcal{L} = -E_{z \sim q_{\boldsymbol{\phi},\boldsymbol{\psi}}(z|c,w)}[\log p_{\boldsymbol{\psi},\boldsymbol{\varphi}}(c|z)] + \mathrm{KL}(q_{\boldsymbol{\phi},\boldsymbol{\psi}}(z|c,w)||p_{\boldsymbol{\phi},\boldsymbol{\psi}}(z|w)) + \mathcal{L}_{reg} \tag{11}$$

For large datasets, negative sampling can be used for the first term.

4.3 Hierarchical vGraph

One advantage of vGraph's framework is that it is very general and can be naturally extended to detect hierarchical communities. In this case, suppose we are given a d-level tree and each node is associate with a community, the community assignment can be represented as a d-dimensional path vector $\vec{z} = (z^{(1)}, z^{(2)}, ..., z^{(d)})$, as shown in Fig. 1. Then, the generation process is formulated as below: (1) a tree path $\vec{z}$ is sampled from a prior distribution $p_{\phi,\psi}(\vec{z}|w)$. (2) The context c is decoded from $\vec{z}$ with $p_{\psi,\varphi}(c|\vec{z})$. Under this model, the likelihood of the network is

$$p_{\phi,\varphi,\psi}(c|w) = \sum_{\vec{z}} p_{\phi,\psi}(c|\vec{z}) p_{\phi,\psi}(\vec{z}|w). \tag{12}$$

At every node of the tree, there is an embedding vector associated with the community. Such a method is similar to the hierarchical softmax parameterization used in language models [21].

5 Experiments

As vGraph can detect both overlapping and non-overlapping communities, we evaluate it on three tasks: overlapping community detection, non-overlapping community detection, and vertex classification.

5.1 Datasets

We evaluate vGraph on 20 standard graph datasets. For non-overlapping community detection and node classification, we use 6 datasets: Citeseer, Cora, Cornell, Texas, Washington, and Wisconsin. For overlapping communtiy detection, we use 14 datasets, including Facebook, Youtube, Amazon, Dblp, Coauthor-CS. For Youtube, Amazon, and Dblp, we consider subgraphs with the 5 largest ground-truth communities due to the runtime of baseline methods. To demonstrate the scalability of our method, we additionally include visualization results on a large dataset – Dblp-full. Dataset statistics are provided in Table 1. More details about the datasets is provided in Appendix A.

Table 1: Dataset Statistics. $|\mathcal{V}|$: number of nodes, $|\mathcal{E}|$: number of edges, K: number of communities, AS: average size of communities, AN: average number of communities that a node belongs to.

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	K	AS	AN
Nonoverlapping					
Cornell	195	286	5	39.00	1
Texas	187	298	5	37.40	1
Washington	230	417	5	46.00	1
Wisconsin	265	479	5	53.00	1
Cora	2708	5278	7	386.86	1
Citeseer	3312	4660	6	552.00	1
overlapping					
facebook0	333	2519	24	13.54	0.98
facebook107	1034	26749	9	55.67	0.48
facebook1684	786	14024	17	45.71	0.99
facebook1912	747	30025	46	23.15	1.43
facebook3437	534	4813	32	6.00	0.36
facebook348	224	3192	14	40.50	2.53
facebook3980	52	146	17	3.41	1.12
facebook414	150	1693	7	25.43	1.19
facebook686	168	1656	14	34.64	2.89
facebook698	61	270	13	6.54	1.39
Youtube	5346	24121	5	1347.80	1.26
Amazon	794	2109	5	277.20	1.75
Dblp	24493	89063	5	5161.40	1.05
Coauthor-CS	9252	33261	5	2920.60	1.58
Dblp-full	93432	335520	5000	22.45	1.20

5.2 Evaluation Metric

For overlapping community detection, we use *F1-Score* and *Jaccard Similarity* to measure the performance of the detected communities as in [37, 18]. For non-overlapping community detection, we use *Normalized Mutual Information (NMI)* [28] and *Modularity*. Note that Modularity does not utilize ground truth data. For node classification, *Micro-F1* and *Macro-F1* are used.

5.3 Comparative Methods

For overlapping community detection, we choose four competitive baselines: **BigCLAM** [36], a nonnegative matrix factorization approach based on the Bernoulli-Poisson link that only considers the graph structure; **CESNA** [37], an extension of BigCLAM, that additionally models the generative process for node attributes; **Circles** [20], a generative model of edges w.r.t. attribute similarity to detect communities; and **SVI** [10], a Bayesian model for graphs with overlapping communities that uses a mixed-membership stochastic blockmodel.

To evaluate node embedding and non-overlapping community detection, we compare our method with the five baselines: **MF** [32], which represents each vertex with a low-dimensional vector obtained through factoring the adjacency matrix; **DeepWalk** [24], a method that adopts truncated random walk and Skip-Gram to learn vertex embeddings; **LINE** [26], which aims to preserve the first-order and second-order proximity among vertices in the graph; **Node2vec** [11], which adopts biased random walk and Skip-Gram to learn vertex embeddings; and **ComE** [4], which uses a Gaussian mixture model to learn an embedding and clustering jointly using random walk features.

5.4 Experiment Configuration

For all baseline methods, we use the implementations provided by their authors and use the default parameters. For methods that only output representations of vertices, we apply K-means to the

Table 2: Evaluation (in terms of F1-Score and Jaccard Similarity) on networks with overlapping ground-truth communities. NA means the task is not completed in 24 hours. In order to evaluate the effectiveness of smoothness regularization, we show the result of our model with (vGraph+) and without the regularization.

	F1-score						Jaccard					
Dataset	Bigclam	CESNA	Circles	SVI	vGraph	vGraph+	Bigclam	CESNA	Circles	SVI	vGraph	vGraph+
facebook0	**0.2948**	0.2806	0.2860	0.2810	0.2440	0.2606	**0.1846**	0.1725	0.1862	0.1760	0.1458	0.1594
facebook107	**0.3928**	0.3733	0.2467	0.2689	0.2817	0.3178	**0.2752**	0.2695	0.1547	0.1719	0.1827	0.2170
facebook1684	0.5041	**0.5121**	0.2894	0.3591	0.4232	0.4379	0.3801	**0.3871**	0.1871	0.2467	0.2917	0.3272
facebook1912	0.3493	0.3474	0.2617	0.2804	0.2579	**0.3750**	0.2412	0.2394	0.1672	0.2010	0.1855	**0.2796**
facebook3437	0.1986	0.2009	0.1009	0.1544	0.2087	**0.2267**	0.1148	0.1165	0.0545	0.0902	0.1201	**0.1328**
facebook348	0.4964	0.5375	0.5175	0.4607	**0.5539**	0.5314	0.3586	0.4001	0.3927	0.3360	**0.4099**	0.4050
facebook3980	0.3274	0.3574	0.3203	NA	**0.4450**	0.4150	0.2426	0.2645	0.2097	NA	**0.3376**	0.2933
facebook414	0.5886	0.6007	0.4843	0.3893	0.6471	**0.6693**	0.4713	0.4732	0.3418	0.2931	0.5184	**0.5587**
facebook686	0.3825	0.3900	0.5036	0.4639	0.4775	**0.5379**	0.2504	0.2534	0.3615	0.3394	0.3272	**0.3856**
facebook698	0.5423	0.5865	0.3515	0.4031	0.5396	**0.5950**	0.4192	0.4588	0.2255	0.3002	0.4356	**0.4771**
Youtube	0.4370	0.3840	0.3600	0.4140	0.5070	**0.5220**	0.2929	0.2416	0.2207	0.2867	0.3434	**0.3480**
Amazon	0.4640	0.4680	**0.5330**	0.4730	**0.5330**	0.5320	0.3505	0.3502	0.3671	0.3643	0.3689	**0.3693**
Dblp	0.2360	0.3590	NA	NA	0.3930	**0.3990**	0.1384	0.2226	NA	NA	0.2501	**0.2505**
Coauthor-CS	0.3830	0.4200	NA	0.4070	0.4980	**0.5020**	0.2409	0.2682	NA	0.2972	**0.3517**	0.3432

learned embeddings to get non-overlapping communities. Results report are averaged over 5 runs. **No node attributes** are used in all our experiments. We generate node attributes using node degree features for those methods that require node attributes such as CESNA [37] and Circles [20]. It is hard to compare the quality of community results when the numbers of communities are different for different methods. Therefore, we set the number of communities to be detected, K, as the number of ground-truth communities for all methods, as in [18]. For vGraph, we use full-batch training when the dataset is small enough. Otherwise, we use stochastic training with a batch size of 5000 or 10000 edges. The initial learning rate is set to 0.05 and is decayed by 0.99 after every 100 iterations. We use the Adam optimizer and we trained for 5000 iterations. When smoothness regularization is used, λ is set to 100. For community detection, the model with the lowest loss is chosen. For node classification, we evaluate node embeddings after 1000 iterations of training. The dimension of node embeddings is set to 128 in all experiments for all methods. For the node classification task, we randomly select 70% of the labels for training and use the rest for testing.

5.5 Results

Table 2 shows the results on overlapping community detection. Some of the methods are not very scalable and cannot obtain results in 24 hours on some larger datasets. Compared with these studies, vGraph outperforms all baseline methods in 11 out of 14 datasets in terms of F1-score or Jaccard Similarity, as it is able to leverage useful representations at node level. Moreover, vGraph is also very efficient on these datasets, since we use employ variational inference and parameterize the model with node and community embeddings. By adding the smoothness regularization term (vGraph+), we see a farther increase performance, which shows that our method can be combined with concepts from traditional community detection methods.

The results for non-overlapping community detection are presented in Table 3. vGraph outperforms all conventional node embeddings + K-Means in 4 out of 6 datasets in terms of NMI and outperforms all 6 in terms of modularity. ComE, another framework that jointly solves node embedding and community detection, also generally performs better than other node embedding methods + K-Means. This supports our claim that learning these two tasks collaboratively instead of sequentially can further enhance performance. Compare to ComE, vGraph performs better in 4 out of 6 datasets in terms of NMI and 5 out of 6 datasets in terms of modularity. This shows that vGraph can also outperform frameworks that learn node representations and communities together.

Table 4 shows the result for the node classification task. vGraph significantly outperforms all the baseline methods in 9 out of 12 datasets. The reason is that most baseline methods only consider the local graph information without modeling the global semantics. vGraph solves this problem by representing node embeddings as a mixture of communities to incorporate global context.

Table 3: Evaluation (in terms of NMI and Modularity) on networks with non-overlapping ground-truth communities.

	NMI						Modularity					
Dataset	MF	deepwalk	LINE	node2vec	ComE	vGraph	MF	deepwalk	LINE	node2vec	ComE	vGraph
cornell	0.0632	0.0789	0.0697	0.0712	0.0732	**0.0803**	0.4220	0.4055	0.2372	0.4573	0.5748	**0.5792**
texas	0.0562	0.0684	**0.1289**	0.0655	0.0772	0.0809	0.2835	0.3443	0.1921	0.3926	**0.4856**	0.4636
washington	0.0599	0.0752	**0.0910**	0.0538	0.0504	0.0649	0.3679	0.1841	0.1655	0.4311	0.4862	**0.5169**
wisconsin	0.0530	0.0759	0.0680	0.0749	0.0689	**0.0852**	0.3892	0.3384	0.1651	0.5338	0.5500	**0.5706**
cora	0.2673	0.3387	0.2202	0.3157	**0.3660**	0.3445	0.6711	0.6398	0.4832	0.5392	0.7010	**0.7358**
citeseer	0.0552	0.1190	0.0340	0.1592	**0.2499**	0.1030	0.6963	0.6819	0.4014	0.4657	0.7324	**0.7711**

Table 4: Results of node classification on 6 datasets.

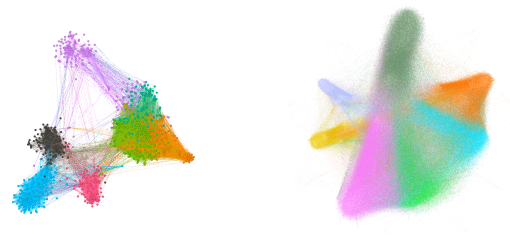
	Macro-F1						Micro-F1					
Datasets	MF	DeepWalk	LINE	Node2Vec	ComE	vGraph	MF	DeepWalk	LINE	Node2Vec	ComE	vGraph
Cornell	13.05	22.69	21.78	20.70	19.86	**29.76**	15.25	33.05	23.73	24.58	25.42	**37.29**
Texas	8.74	21.32	16.33	14.95	15.46	**26.00**	14.03	40.35	27.19	25.44	33.33	**47.37**
Washington	15.88	18.45	13.99	21.23	15.80	**30.36**	15.94	34.06	25.36	28.99	33.33	**34.78**
Wisconsin	14.77	23.44	19.06	18.47	14.63	**29.91**	18.75	**38.75**	28.12	25.00	32.50	35.00
Cora	11.29	13.21	11.86	10.52	12.88	**16.23**	12.79	22.32	14.59	27.74	**28.04**	24.35
Citeseer	14.59	16.17	15.99	16.68	12.88	**17.88**	15.79	19.01	16.80	**20.82**	19.42	20.42

5.6 Visualization

In order to gain more insight, we present visualizations of the facebook107 dataset in Fig. 2(a). To demonstrate that our model can be applied to large networks, we present results of vGraph on a co-authorship network with around 100,000 nodes and 330,000 edges in Fig. 2(b). More visualizations are available in appendix B. We can observe that the community structure, or "social context", is reflected in the corresponding node embedding (node positions in both visualizations are determined by t-SNE of the node embeddings). To demonstrate the hierarchical extension of our model, we visualize a subset of the co-authorship dataset in Fig. 3. We visualize the first-tier communities and second-tier communities in panel (a) and (b) respectively. We can observe that the second-tier communities grouped under the same first-tier communities interact more with themselves than they do with other second-tier communities.

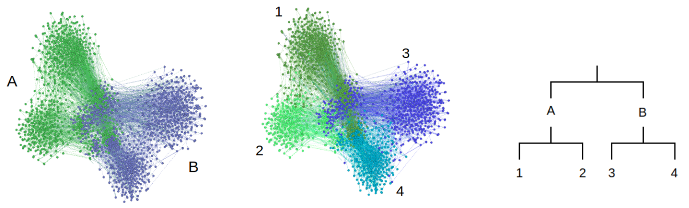
6 Conclusion

In this paper, we proposed vGraph, a method that performs overlapping (and non-overlapping) community detection and learns node and community embeddings at the same time. vGraph casts the generation of edges in a graph as an inference problem. To encourage collaborations between community detection and node representation learning, we assume that each node can be represented by a mixture of communities, and each community is defined as a multinomial distribution over nodes. We also design a smoothness regularizer in the latent space to encourage neighboring nodes to



(a) (b)

Figure 2: In panel (a) we visualize the result on the facebook107 dataset using vGraph. In panel (b) we visualize the result on Dblp-full dataset using vGraph. The coordinates of the nodes are determined by t-SNE of the node embeddings.



(a) (b) (c)

Figure 3: We visualize the result on a subset of Dblp dataset using two-level hierarchical vGraph. The coordinates of the nodes are determined by t-SNE of the node embeddings. In panel (a) we visualize the first-tier communities. In panel (b), we visualize the second-tier communities. In panel (c) we show the corresponding hierarchical tree structure.

be similar. Empirical evaluation on 20 different benchmark datasets demonstrates the effectiveness of the proposed method on both tasks compared to competitive baselines. Furthermore, our model is also readily extendable to detect hierarchical communities.

Acknowledgments

This project is supported by the Natural Sciences and Engineering Research Council of Canada, as well as the Canada CIFAR AI Chair Program.

References

[1] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307):761, 2010.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900. ACM, 2015.

[4] Sandro Cavallari, Vincent W Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. Learning community embedding with community detection and node embedding on graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 377–386. ACM, 2017.

[5] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical review letters*, 94(16):160202, 2005.

[6] Xiaowen Dong, Pascal Frossard, Pierre Vandergheynst, and Nikolai Nefedov. Clustering with multi-layer graphs: A spectral perspective. *IEEE Transactions on Signal Processing*, 60(11):5820–5831, 2012.

[7] Alessandro Epasto and Bryan Perozzi. Is a single embedding enough? learning node representations that capture multiple social contexts. 2019.

[8] Sheng Gao, Ludovic Denoyer, and Patrick Gallinari. Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1169–1174. ACM, 2011.

[9] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[10] Prem K Gopalan and David M Blei. Efficient discovery of overlapping communities in massive networks. *Proceedings of the National Academy of Sciences*, 110(36):14534–14539, 2013.

[11] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[12] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[14] Yuting Jia, Qinqin Zhang, Weinan Zhang, and Xinbing Wang. Communitygan: Community detection with generative adversarial nets. *arXiv preprint arXiv:1901.06631*, 2019.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[16] Nevan J Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron P Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. *Nature*, 440(7084):637, 2006.

[17] Da Kuang, Chris Ding, and Haesun Park. Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 106–117. SIAM, 2012.

[18] Ye Li, Chaofeng Sha, Xin Huang, and Yanchun Zhang. Community detection in attributed graphs: an embedding approach. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[20] Julian McAuley and Jure Leskovec. Discovering social circles in ego networks. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(1):4, 2014.

[21] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[22] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[23] Yulong Pei, Nilanjan Chakraborty, and Katia Sycara. Nonnegative matrix tri-factorization with graph regularization for community detection in social networks. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[24] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[25] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. Gemsec: graph embedding with self clustering. *arXiv preprint arXiv:1802.03997*, 2018.

[26] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[27] Jiliang Tang, Charu Aggarwal, and Huan Liu. Node classification in signed social networks. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 54–62. SIAM, 2016.

[28] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[29] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 539–548. International World Wide Web Conferences Steering Committee, 2018.

[30] Cunchao Tu, Xiangkai Zeng, Hao Wang, Zhengyan Zhang, Zhiyuan Liu, Maosong Sun, Bo Zhang, and Leyu Lin. A unified framework for community detection and network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[31] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[32] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. Community discovery using nonnegative matrix factorization. *Data Mining and Knowledge Discovery*, 22(3):493–521, 2011.

[33] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[34] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM, 2005.

[35] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[36] Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.

[37] Jaewon Yang, Julian McAuley, and Jure Leskovec. Community detection in networks with node attributes. In *2013 IEEE 13th International Conference on Data Mining*, pages 1151–1156. IEEE, 2013.

[38] Hongyi Zhang, Irwin King, and Michael R Lyu. Incorporating implicit link preference into overlapping community detection. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[39] Mingyuan Zhou. Infinite edge partition models for overlapping community detection and link prediction. In *Artificial Intelligence and Statistics*, pages 1135–1143, 2015.

A Datasets

Citeseer, Cora, Cornell, Texas, Washington, and Wisconsin are available online[2]. For Youtube, Amazon, and Dblp, we consider subgraphs with the 5 largest ground-truth communities due to the runtime of the baseline methods.

Facebook[3] is a set of Facebook ego-networks. It contains 10 different ego-networks with identified circles. Social circles formed by friends are regarded as ground-truth communities.

Youtube[4] is a network of social relationships of Youtube users. The vertices represent users; the edges indicate friendships among the users; the user-defined groups are considered as ground-truth communities.

Amazon[5] is collected by crawling amazon website. The vertices represent products and the edges indicate products frequently purchased together. The ground-truth communities are defined by the product categories on Amazon.

Dblp[6] is a co-authorship network from Dblp. The vertices represent researchers and the edges indicate co-author relationships. Authors who have published in a same journal or conference form a community.

Coauthor-CS[7] is a computer science co-authorship network. We chose 21 conferences and group them into five categories: *Machine Learning*, *Computer Linguistics*, *Programming language*, *Data mining*, and *Database*.

[2]https://linqs.soe.ucsc.edu
[3]https://snap.stanfod.edu/data/ego-Facebook.html
[4]http://snap.stanford.edu/data/com-Youtube.html
[5]http://snap.stanford.edu/data/com-Amazon.html
[6]http://snap.stanford.edu/data/com-DBLP.html
[7]https://aminer.org/aminernetwork

B Visualization

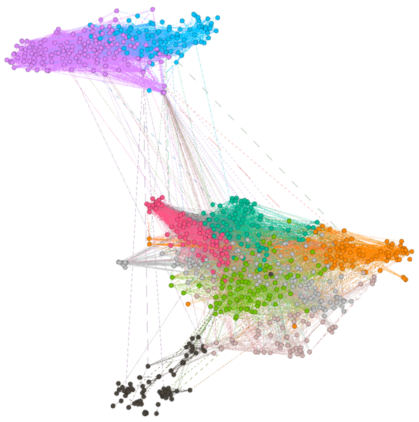


Figure 4: Visualization of the result of vGraph on the facebook1684 dataset. The coordinates of the nodes are determined by t-SNE of the node embeddings.

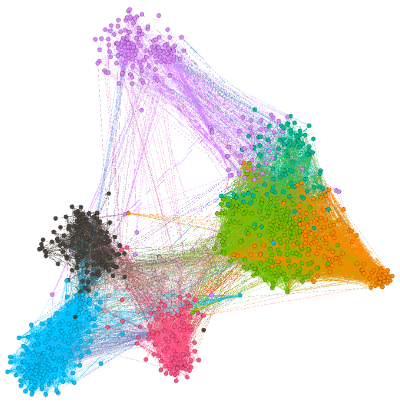


Figure 5: Visualization of the result of vGraph on the facebook107 dataset. The coordinates of the nodes are determined by t-SNE of the node embeddings.

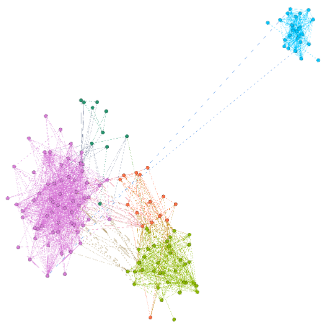


Figure 6: Visualization of the result of vGraph on the facebook414 dataset. The coordinates of the nodes are determined by t-SNE of the node embeddings.

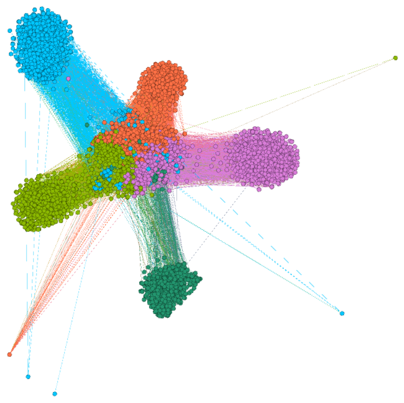


Figure 7: Visualization of the result of vGraph on the Youtube dataset. The coordinates of the nodes are determined by t-SNE of the node embeddings.

5.2.4 Publication: InfoGraph

InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization

A Preprint

Fan-Yun Sun[1,2], **Jordan Hoffmann**[2,3], **Jian Tang**[2,4,5]
[1]National Taiwan University,
[2]Mila-Quebec Institute for Learning Algorithms, Canada
[3]Harvard University, USA
[4]HEC Montreal, Canada
[5]CIFAR AI Research Chair

b04902045@ntu.edu.tw
jhoffmann@g.harvard.edu
jian.tang@hec.ca

Abstract

This paper studies learning the representations of whole graphs in both unsupervised and semi-supervised scenarios. Graph-level representations are critical in a variety of real-world applications such as predicting the properties of molecules and community analysis in social networks. Traditional graph kernel based methods are simple, yet effective for obtaining fixed-length representations for graphs but they suffer from poor generalization due to hand-crafted designs. There are also some recent methods based on language models (e.g. graph2vec) but they tend to only consider certain substructures (e.g. subtrees) as graph representatives. Inspired by recent progress of unsupervised representation learning, in this paper we proposed a novel method called InfoGraph for learning graph-level representations. We maximize the mutual information between the graph-level representation and the representations of substructures of different scales (e.g., nodes, edges, triangles). By doing so, the graph-level representations encode aspects of the data that are shared across different scales of substructures. Furthermore, we further propose InfoGraph*, an extension of InfoGraph for semi-supervised scenarios. InfoGraph* maximizes the mutual information between unsupervised graph representations learned by InfoGraph and the representations learned by existing supervised methods. As a result, the supervised encoder learns from unlabeled data while preserving the latent semantic space favored by the current supervised task. Experimental results on the tasks of graph classification and molecular property prediction show that InfoGraph is superior to state-of-the-art baselines and InfoGraph* can achieve performance competitive with state-of-the-art semi-supervised models.

1 Introduction

Graphs have proven to be an effective way to represent very diverse types of data including social networks [38], biological reaction networks[46], protein-protein interactions [29], the quantum mechanical properties of individual molecules [59, 20], and many more. Graphs provide explicit information about the coupling between individual units in a larger part along with a well defined framework for assigning properties to the nodes and the edges connecting them. There has been a significant amount of previous work done studying many aspects of graphs including link prediction [13, 57] and node prediction [2]. Due to its flexibility, graph-like data structures can capture rich information which is critical in many applications.

At the lowest level, much work has been done on learning node representations– low-dimensional vector embeddings of individual nodes [47, 53, 16]. Another field that has attracted a large amount of attention recently is learning representations of entire graphs. Such a problem is critical in a variety of applications such as predicting the properties of molecular graphs in both drug discovery and material science [7, 6]. There has been some recent progress based on neural message passing algorithms [15, 59], which learn the representations of entire graphs in a supervised way. These methods have been shown achieving state-of-the-art results on a variety of different prediction tasks [25, 59, 15, 6].

However, one of the most difficult obstacles for supervised learning on graphs is that it is often very costly or even impossible to collect annotated labels. For example, in the chemical domain labels are typically produced with a costly Density Functional Theory (DFT) calculation. One option is to use semi-supervised methods which combine a small handful of labels with a larger, unlabeled, dataset. In real-world applications, partially labeled datasets are common, making tools that are able to efficiently utilize the present labels particularly useful.

Coming up with methods that are able to learn unsupervised representations of an entire graph, as opposed to nodes, is an important step in working with unlabeled or partially labeled graphs [36, 18, 39]. For example, there exists work that explores pre-training techniques for graphs to improve generalization [18]. Another common approach to unsupervised representation learning on graphs is through graph kernels [48, 21, 43]. However, many of these methods do not provide explicit graph embeddings which many machine learning algorithms operate on. Furthermore, the handcrafted features of graph kernels lead to high dimensional, sparse or non-smooth representations and thus result in poor generalization performance, especially on large datasets [36].

Unsupervised learning of latent representations is also an important problem in other domains, such as image generation [24, 22] and natural language processing [32]. A recent work introduced Deep Infomax, a method that maximizes the mutual information content between the input data and the learned representation [17]. This method outperforms other methods on many unsupervised learning tasks. Motivated by Deep InfoMax [17], we aim to use mutual information maximization for unsupervised representation learning on the entire graph. Specifically, our objective is to maximize the mutual information between the representations of entire graphs and the representations of substructures of different granularity. We name our model **InfoGraph**.

We also propose a semi-supervised learning model which we name **InfoGraph***. We employ a student-teacher framework similar to Mean-Teacher method [54]. We maximize the mutual information between intermediate representations of the two models so that the student model learns from the teacher model. The student model is trained on the labeled data using a supervised objective function while the teacher model is trained on unlabeled data with **InfoGraph**. Using **InfoGraph***, we achieve performance competitive with state-of-the-art methods on molecular property prediction.

We summarize our contributions as follows:

- We propose InfoGraph, an unsupervised graph representation learning method based on Deep InfoMax (DIM) [17].
- We show that InfoGraph can be extended to semi-supervised prediction tasks on graphs.
- We empirically show that InfoGraph surpasses state-of-the-art performance on graph classification tasks with unsupervised learning and obtains performance comparable with state-of-art methods on molecular property prediction tasks using semi-supervised learning.

2 Related work

Representation learning for graphs has mainly dealt with supervised learning tasks. Recently, however, researchers have proposed algorithms that learn graph-level representations in an unsupervised manner [36, 1].

Concurrently to this work, information maximizing graph neural networks (IGNN) was introduced which uses mutual information maximization between edge states and transform parameters to achieve state-of-the-art predictions on a variety of supervised molecule property prediction tasks [7]. In this work, our focus is on unsupervised and semi-supervised scenarios.

Graph Kernels. Constructing graph kernels is a common unsupervised task in learning graph representations. These kernels are typically evaluated on node classification tasks. In graph kernels, a graph G is decomposed into (possibly different) $\{G_s\}$ sub-structures. The graph kernel $K(G_1, G_2)$ is defined based on the frequency of each sub-structure appearing in G_1 and G_2 respectively. Namely, $K(G_1, G_2) = \langle f_{G_{s_1}}, f_{G_{s_2}} \rangle$, where f_{G_s} is the vector containing frequencies of $\{G_s\}$ sub-structures, and $\langle , \rangle$ is an inner product in an appropriately normalized vector space. Much work has been devoted to deciding which sub-structures are more suitable than others, popular ones are graphlets [48, 52], random walk and shortest path kernels [21, 3], and the Weisfeiler-Lehman subtree kernel [51]. Furthermore, deep graph kernels [63], graph invariant kernels [43], optimal assignment graph kernels [28] and multiscale laplacian graph kernels [27] have been proposed with the goal to redefine kernel functions to appropriately capture sub-structural similarity at different levels. Another line of research in this area focuses on efficiently computing these kernels either through exploiting certain structural dependencies, or via approximations/randomization [11, 9, 37]. Instead of defining hand crafted similarity measures between substructures, InfoGraph adopts a more principled metric – mutual information.

Contrastive methods. An important approach for unsupervised representation learning is to train an encoder to be contrastive between representations that capture statistical dependencies of interest and those that do not. For example, a contrastive approach may employ a scoring function, training the encoder to increase the score on "real" input (a.k.a, positive examples) and decrease the score on "fake" input (a.k.a., negative samples).

Contrastive methods are central many popular word-embedding methods [8, 35, 33]. Word2vec [32] is an unsupervised algorithm which obtains word representations by using the representations to predict context words (the words that surround it). Doc2vec [31] is an extension of the continuous Skip-gram model that predicts representations of words from that of a document containing them. Researchers extended many of these unsupervised language models to learn representations of graph-structured input [1, 36]. For example, *graph2vec* [36] extends Doc2vec to arbitrary graphs. Intuitively, for graph2vec a graph and the rooted subgraphs in it correspond to a document and words in a paragraph vector, respectively. One of the technical contributions of the paper is using the Weisfeiler-Lehman relabelling algorithm [58, 51] to enumerate all rooted subgraphs up to a specified depth. InfoGraph can be interpreted as an extension of graph2vec though there are many major differences such as instead of listing all rooted subgraphs explicitly, we make use of graph neural networks to obtain patch representations of subgraphs.

Deep Graph InfoMax (DGI) [55] also belongs to this category, which aims to train a **node** encoder that maximizes mutual information between node representations and the pooled global graph representation. Although we built upon a similar methodology, our aim is different than theirs as our goal is to obtain embeddings at the whole graph level for unsupervised and semi-supervised learning whereas DGI only evaluates node level embeddings. In order to differentiate our method with Deep Graph Infomax ([55]), we term our model **InfoGraph**.

Semi-supervised Learning. A comprehensive overview of semi-supervised learning (SSL) methods is out of the scope of this paper. We refer readers to [67, 5, 42]. Here, we discuss 2 state-of-the-art methods applicable for regression tasks which solely involve adding an additional loss term to the training of a neural network, and otherwise leave the training and model unchanged from what would be used in the fully-supervised setting. **Virtual Adversarial Training (VAT)** [34], a method that approximates a tiny perturbation r_{adv} to add to input data which most significantly affect the output of the prediction function. We did not include VAT in our experiments as small perturbation on molecular graphs can lead to drastically different results in real world scenarios. **Mean Teacher** [54] adds a loss term which encourages the distance between the original network's output and the teacher's output to be small. The teacher's predictions are made using an exponential moving average of parameters from previous training steps. Inspired by the "student-teacher" framework in Mean Teacher model, our semi-supervised model (**InfoGraph***) deploys two separate encoders but instead of explicitly encouraging the output of the student model to be similar to the teacher model's output, we enable the student model to learn from the teacher model by maximizing mutual information between intermediate representations learned by two models.

3 Methodology

Most recent work on graphs focus on supervised learning tasks or learning node representations. However, many graph analytic tasks such as graph classification, regression, and clustering require representing entire graphs as fixed-length feature vectors. Though graph-level representations can be obtained through the node-level representations implicitly, explicitly extracting the graph can be more straightforward and optimal for graph-oriented tasks.

Another scenario that is important, yet attracts comparatively less attention in the graph related literature is semi-supervised learning. One of the biggest challenges in prediction tasks in biology [62, 64] or molecular machine learning [10, 15, 19] is the extreme scarcity of labeled data. Therefore, semi-supervised learning, in which a large number of unlabeled samples are incorporated with a small number of labeled samples to enhance accuracy of models, will play a key role in these areas.

In this section, we first formulate an unsupervised whole graph representation learning problem and a semi-supervised prediction task on graphs. Then, we present our method to learn graph-level representations. Afterwards we present our proposed model for the semi-supervised learning scenario.

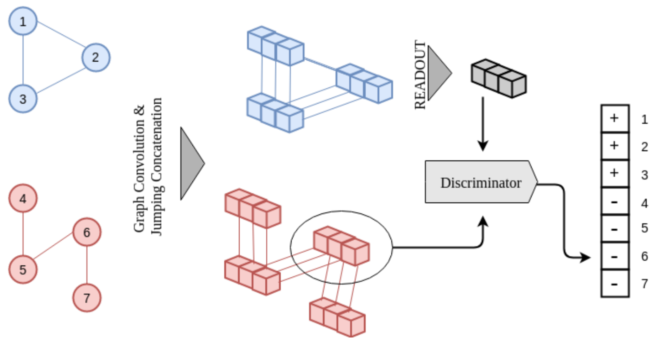


Figure 1: Illustration of InfoGraph. An input graph is encoded into a feature map by graph convolutions and jumping concatenation. The discriminator takes a (global representation, patch representation) pair as input and decides whether they are from the same graph. InfoGraph uses a batch-wise fashion to generate all possible positive and negative samples. For example, consider the toy example with 2 input graphs in the batch and 7 nodes (or patch representations) in total. For the global representation of the blue graph, there will be 7 input pairs to the discriminator and same for the red graph. Thus, the discriminator will take 14 (global representation, patch representation) pairs as input in this case.

3.1 Problem Definition

Unsupervised Graph Representation Learning. Given a set of graphs $\mathbb{G} = \{G_1, G_2, ...\}$ and a positive integer δ (the expected embedding size), our goal is to learn a δ-dimensional distributed representation of every graph $G_i \in \mathbb{G}$. We denote the number of nodes in G_i as $|G_i|$. We denote the matrix of representations of all graphs as $\Phi \in \mathbb{R}^{|\mathbb{G}|\times\delta}$.

Semi-supervied Graph Prediction Tasks. Given a set of labeled graphs $\mathbb{G}^L = \{G_1, \cdots, G_{|\mathbb{G}^L|}\}$ with corresponding output $\{o_1, \cdots, o_{|\mathbb{G}^L|}\}$, and a set of unlabeled samples $\mathbb{G}^U = \{G_{|\mathbb{G}^L|+1}, \cdots, G_{|\mathbb{G}^L|+|\mathbb{G}^U|}\}$, our goal is to learn a model that can make predictions for unseen graphs. Note that in most cases $|\mathbb{G}^U| \gg |\mathbb{G}^L|$.

3.2 InfoGraph

We focus on graph neural networks (GNNs)—a flexible class of embedding architectures which generate node representations by repeated aggregation over local node neighborhoods. The representations of nodes are learned by aggregating the features of their neighborhood nodes, so we refer to these as patch representations. GNNs utilize a READOUT function to summarize all the obtained patch representations into a fixed length graph-level representation.

Formally, the k-th layer of a GNN is

$$h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, \text{AGGREGATE}^{(k)}\left(\left\{\left(h_v^{(k-1)}, h_u^{(k-1)}, e_{uv}\right) : u \in \mathcal{N}(v)\right\}\right)\right), \tag{1}$$

where $h_v^{(k)}$ is the feature vector of node v at the k-th iteration/layer (or patch representation centered at node i), e_{uv} is the feature vector of the edge between u and v, and $\mathcal{N}(v)$ are neighborhoods to node v. $h_v^{(0)}$ is often initialized as node features. READOUT can be a simple permutation invariant function such as averaging or a more sophisticated graph-level pooling function [65, 66].

We seek to obtain graph representations by maximizing the mutual information between graph-level and patch-level representations. By doing so, the graph representations can learn to encode aspects of the data that are shared across all substructures. Assume that we are given a set of training samples $\mathbf{G} := \{G_j \in \mathbb{G}\}_{j=1}^N$ with empirical probability distribution $\mathbb{P}$ on the input space. Let ϕ denote the set of parameters of a K-layer graph neural network. After the first k layers of the graph neural network, the input graph will be encoded into a set of patch representations $\{h_i^{(k)}\}_{i=1}^N$. Next, we summarize feature vectors at all depths of the graph neural network into a single feature vector that captures patch information at different scales centered at every node. Inspired by [61], we use concatenation. That is,

$$h_\phi^i = \text{CONCAT}(\{h_i^{(k)}\}_{k=1}^K) \tag{2}$$

$$H_\phi(G) = \text{READOUT}(\{h_\phi^i\}_{i=1}^N) \tag{3}$$

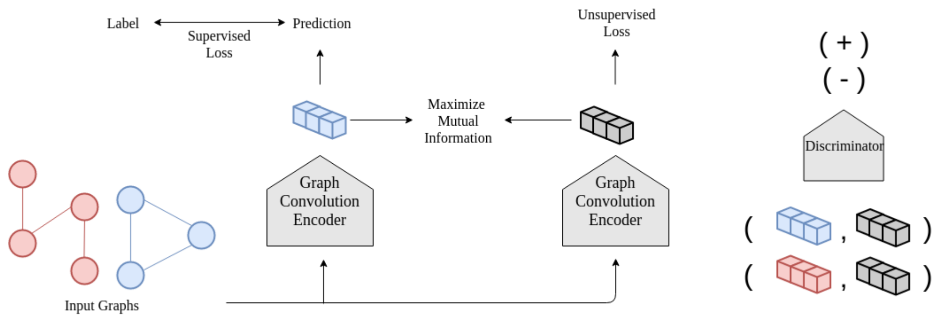


Figure 2: Illustration of the semi-supervised version of InfoGraph (InfoGraph*). There are two separate encoders with the same architecture, one for the supervised task and the other trained using both labeled and unlabeled data with an unsupervised objective (eq. (4)). We encourage the mutual information of the two representations learned by the two encoders to be high by deploying a discriminator that takes a pair of representation as input and determines whether they are from the same input graph.

where h_ϕ^i is the summarized patch representation centered at node i and $H_\phi(G)$ is the global representation after applying READOUT. Note that here we slightly abuse the notation of h.

We define our mutual information (MI) estimator on global/local pairs, maximizing the estimated MI over the given dataset $\mathbf{G} := \{G_j \in \mathbb{G}\}_{j=1}^N$:

$$\hat{\phi}, \hat{\psi} = \arg\max_{\phi,\psi} \sum_{G \in \mathbf{G}} \frac{1}{|G|} \sum_{u \in G} I_{\phi,\psi}(\vec{h}_\phi^u; H_\phi(G)). \tag{4}$$

$I_{\phi,\psi}$ is the mutual information estimator modeled by discriminator T_ψ and parameterized by a neural network with parameters ψ. We use the Jensen-Shannon MI estimator (following the formulation of [41]),

$$I_{\phi,\psi}(h_\phi^i(G); H_\phi(G)) := \\ \mathbb{E}_{\mathbb{P}}[-\text{sp}(-T_{\phi,\psi}(\vec{h}_\phi^i(x), H_\phi(x)))] - \mathbb{E}_{\mathbb{P}\times\tilde{\mathbb{P}}}[\text{sp}(T_{\phi,\psi}(\vec{h}_\phi^i(x'), G_\phi(x)))] \tag{5}$$

where x is an input sample, x' (negative sample) is an input sampled from $\tilde{\mathbb{P}} = \mathbb{P}$, a distribution identical to the empirical probability distribution of the input space, and $\text{sp}(z) = \log(1 + e^z)$ is the softplus function. In practice, we generate negative samples using all possible combinations of global and local patch representations across all graph instances in a batch.

Since $H_\phi(G)$ is encouraged to have high MI with patches that contain information at all scales, this favours encoding aspects of the data that are shared across patches and aspects that are shared across scales. The algorithm is illustrated in Fig. 1.

It should be noted that our model is similar to Deep Graph Infomax (DGI) [55], a model proposed for learning unsupervised node embeddings. However, there are important design differences due to the different problems that we are focusing on. First, in DGI they use random sampling to obtain negative samples due to the fact that they are mainly focusing on learning node embeddings on a graph. However, contrastive methods require a large number of negative samples to be competitive [17], thus the use of batch-wise generation of negative samples is crucial as we are trying to learn graph embeddings given many graph instances.Second, the choice of graph convolution encoders is also crucial. We use GIN [60] while DGI uses GCN [26] as GIN provides a better inductive bias for graph level applications. Graph neural network designs should be considered carefully so that graph representations can be discriminative towards other graph instances. For example, we use sum over mean for READOUT and that can provide important information regarding the size of the graph.

3.3 Semi-supervised InfoGraph

Based on the previous unsupervised model, a straightforward way to do semi-supervised property prediction on graphs is to combine the purely supervised loss and the unsupervised objective function which acts as a regularization term. In doing so, the model is trained to predict properties for the labeled dataset while keeping a rich discriminative intermediate representation learned from both the labeled and the unlabeled dataset. That is, we try to minimize the following objective function:

$$\mathrm{L_{total}} = \sum_{\mathrm{i=1}}^{|\mathbb{G}^{\mathrm{L}}|} \mathrm{L_{supervised}}(\mathrm{y}_\phi(\mathrm{G_i}), \mathrm{o_i}) + \lambda \sum_{\mathrm{j=1}}^{|\mathbb{G}^{\mathrm{L}}|+|\mathbb{G}^{\mathrm{U}}|} \mathrm{L_{unsupervised}}(\mathrm{h}_\phi(\mathrm{G_j}); \mathrm{H}_\phi(\mathrm{G_j})) \tag{6}$$

where $\mathrm{L_{supervised}}(y_\phi(G_i), o_i)$ is defined as the loss function of graph G_i that measures the discrepancy between the classifier output $y_\phi(G_i)$ and the true output o_i. $\mathrm{L_{unsupervised}}(\mathrm{h}_\phi(\mathrm{G_j}); \mathrm{H}_\phi(\mathrm{G_j}))$ is the unsupervised InfoGraph loss term as defined in eq. (4) that can be optimized using both labeled and unlabeled data. The hyper-parameter λ controls the relative weight between the purely supervised and the unsupervised loss. The intuition behind this is that the model will benefit from learning a good representation from the large amount of unlabeled data while learning to predict the corresponding supervised label.

However, supervised tasks and unsupervised tasks may favor different information or a different semantic space. Simply combining the two loss functions using the same encoder may lead to "negative transfer" [1] [44, 50]. Inspired by the "student-teacher" framework in many semi-supervised learning methods [30, 54], we derive a simple way to alleviate this problem. For example, in the Mean Teacher model [54], the teacher model is an exponential moving average of student model's parameters from previous training steps and they encourage student model's output to be similar to teacher model's output. Similarly, we deploy two encoder models on labelled data and unlabelled data respectively. We believe that the supervised encoder (student) can benefit if it incorporates information learned from the unsupervised encoder (teacher) as it is trained on many more data points. However, encouraging the output of two encoders to be similar does not make sense as the two encoders are trained with different objectives. Therefore, instead of adding a loss term that encourages the output of the two networks to be similar, we add a loss term that encourages the representations learned by the two encoders to have high mutual information. Now, let φ denote the set of parameters of another K-layered graph neural network, identical to the one parameterized by ϕ, and let λ be a tunable hyper-parameter. The total loss function can be defined as follows:

$$\mathrm{L_{total}} = \sum_{\mathrm{i=1}}^{|\mathbb{G}^{\mathrm{L}}|} \mathrm{L_{supervised}}(\mathrm{y}_\phi(\mathrm{G_i}), \mathrm{o_i}) + \sum_{j=1}^{|\mathbb{G}^{L}|+|\mathbb{G}^{U}|} \mathrm{L_{unsupervised}}(h_\varphi(G_j); H_\varphi(G_j)) \tag{7}$$

$$- \lambda \sum_{j=1}^{|\mathbb{G}^{L}|+|\mathbb{G}^{U}|} \frac{1}{|G_j|} \sum_{u \in G_j} I(\vec{h}_\phi^u; \vec{h}_\varphi^u). \tag{8}$$

In our semi-supervised experiments, we refer to the naive method using the objective function given in eq. (6) as InfoGraph. We refer to the method that uses two separate encoders and employ the objective function given in eq. (8) as InfoGraph*. InfoGraph* is fully summarized in Figure 2.

4 Experiments

We evaluate the effectiveness of the graph-level representation learned by InfoGraph on downstream graph classification tasks and on semi-supervised molecular property prediction tasks.

4.1 Datasets

For graph classification, we conduct experiments on 6 well-known benchmark datasets: MUTAG, PTC, REDDIT-BINARY, REDDIT-MULTI-5K, IMDB-BINARY, and IMDB-MULTI ([63]).

For semi-supervised learning tasks, we use the publicly available QM9 dataset [49]. All molecules in the dataset consist of Hydrogen (H), Carbon (C), Oxygen (O), Nitrogen (N), and Flourine (F) atoms and contain up to 9 non Hydrogen atoms. In all, this results in about 134,000 drug-like organic molecules that span a wide range of chemical compositions and properties. A total of 12 interesting and fundamental chemical properties are pre-computed for each molecule.

[1]We slightly abuse this term in this paper as it usually refers to transferring knowledge from a less related source and thus may hurt the target performance.

For a more detailed description of these datasets and properties, see the supplementary material.

4.2 Baselines

For graph classification, we used 6 state-of-the-art graph kernels for comparison: Random Walk (RW) [14], Shortest Path Kernel (SP) [3], Graphlet Kernel (GK) [52], Weisfeiler-Lehman Sub-tree Kernel (WL) [51], Deep Graph Kernels (DGK) [63], and Multi-Scale Laplacian Kernel (MLG) [27]. Aside from graph kernels, we also compare with 3 unsupervised graph-level representation learning methods: node2vec [16], sub2vec [1], and graph2vec [36]. Node2vec is a neural embedding framework that learns feature representations of individual nodes in graphs and eventually aggregates node embeddings to obtain graph embeddings.

For semi-supervised tasks, aside from comparing the results with the fully supervised results, we also compare our results with a state-of-the-art semi-supervised method: Mean-Teachers [54].

4.3 Experiment Configuration

For graph classification tasks, we adopt the same procedure of previous works [40, 56, 63, 66] to make a fair comparison and used 10-fold cross validation accuracy to report the classification performance. Experiments are repeated 5 times. We report results from previous papers with the same experimental setup if available. If results are not previously reported, we implement them and conduct a hyper-parameter search according to the original paper. For node2vec [16], we took the result from [36] but we did not run it on all datasets as the implementation details are not clear in the paper. For Deep Graph Kernels, we report the best result out of Deep WL Kernels, Deep GK Kernels, and Deep RW Kernels. For sub2vec, we report the best result out of its two variants: *sub2vec-N* and *sub2vec-S*. For all methods, the embedding dimension is set to 512 and parameters of downstream classifiers are independently tuned using cross validation on training folds of data. The best average classification accuracy is reported for each method. The classification accuracies are computed using LIBSVM [4], and the C parameter was selected from $\{10^{-3}, 10^{-2}, \ldots, 10^{2}, 10^{3}\}$.

The QM9 dataset has 130462 molecules in it. We adopt similar experimental settings as traditional semi-supervised methods [54, 30, 34]. We randomly chose 5000 samples as labeled samples for training and another 10000 as validation samples, 10000 samples for testing, and use the rest as unlabeled training samples. Note that we use the exact same split when running the supervised model and the semi-supervised model. We use the validation set to do model selection and we report scores on the test set. All targets were normalized to have mean 0 and variance 1. We minimize the mean squared error between the model output and the target, although we evaluate mean absolute error.

4.4 Model Configuration

For the unsupervised experiments, we use the Graph Isomorphism Network (GIN) [60]. GNN layers are chosen from $\{4, 8, 12\}$. Models are trained using SGD with the Adam optimizer [23] with an initial learning rate chosen from the set $\{10^{-2}, 10^{-3}, 10^{-4}\}$. The number of epochs are chosen from $\{10, 20, 100\}$. The batch size is set to 128.

For the semi-supervised experiments, we adopt a similar model configuration as in [15]. The number of set2set computations is set to 3. Models were trained using SGD with the Adam optimizer[23] with an initial learning rate 0.001. We train for 500 epochs with a batch size 20. As recommended in [42], we use the exact same underlying model architecture when comparing semi-supervised learning approaches as our goal is not to produce state-of-the-art results, but instead to provide a rigorous comparative analysis in a common framework. For the supervised case, the weight decay is chosen from $\{0, 10^{-3}, 10^{-4}\}$. For InfoGraph and InfoGraph*, λ is chosen from $\{10^{-3}, 10^{-4}, 10^{-5}\}$. Hyper-parameters are chosen based on the validation error.

The discriminator scores global-patch representation pairs by passing two representations to different non-linear transformations and then takes the dot product of the two transformed representations. Both non-linear transformations are parameterized by 3-layered feed-forward neural networks with jumping connections. Following each linear layer is a ReLU activation function. We use Pytorch [45] and the Pytorch Geometric [12] libraries for all our experiments.

5 Results

The results of evaluating unsupervised graph level representations using downstream graph classification tasks are presented in Table 1. We show results from six methods including three state-of-the-art graph kernel methods: WL [51], DGK [63], and MLG [27]. While these kernel methods perform well on individual datasets, none of them are competitive across all of the datasets. Additionally, MLG suffers from a long run time and take more than 24 hours to

Dataset	MUTAG	PTC-MR	RDT-B	RDT-M5K	IMDB-B	IMDB-M
(No. Graphs)	188	344	2000	4999	1000	1500
(No. classes)	2	2	2	5	2	3
(Avg. Graph Size)	17.93	14.29	429.63	508.52	19.77	13.00
			Graph Kernels			
RW [14]	83.72 ± 1.50	57.85 ± 1.30	OMR	OMR	50.68 ± 0.26	34.65 ± 0.19
SP [3]	85.22 ± 2.43	58.24 ± 2.44	64.11 ± 0.14	39.55 ± 0.22	55.60 ± 0.22	37.99 ± 0.30
GK [52]	81.66 ± 2.11	57.26 ± 1.41	77.34 ± 0.18	41.01 ± 0.17	65.87 ± 0.98	43.89 ± 0.38
WL [51]	80.72 ± 3.00	57.97 ± 0.49	68.82 ± 0.41	46.06 ± 0.21	72.30 ± 3.44	46.95 ± 0.46
DGK [63]	87.44 ± 2.72	60.08 ± 2.55	78.04 ± 0.39	41.27 ± 0.18	66.96 ± 0.56	44.55 ± 0.52
MLG [27]	87.94 ± 1.61	$\mathbf{63.26 \pm 1.48}$	> 1 Day	> 1 Day	66.55 ± 0.25	41.17 ± 0.03
			Other Unsupervised Methods			
node2vec [16]	72.63 ± 10.20	58.58 ± 8.00	-	-	-	-
sub2vec [1]	61.05 ± 15.80	59.99 ± 6.38	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54	36.67 ± 0.83
graph2vec [36]	83.15 ± 9.25	60.17 ± 6.86	75.78 ± 1.03	47.86 ± 0.26	71.1 ± 0.54	$\mathbf{50.44 \pm 0.87}$
InfoGraph	$\mathbf{89.01 \pm 1.13}$	61.65 ± 1.43	$\mathbf{82.50 \pm 1.42}$	$\mathbf{53.46 \pm 1.03}$	$\mathbf{73.03 \pm 0.87}$	49.69 ± 0.53

Table 1: Classification accuracy on 6 datasets. The result in **bold** indicates the best reported classification accuracy. The top half of the table compares results with various graph kernel approaches while bottom half compares results with other state-of-the-art unsupervised graph representation learning methods. '$>$ 1 day' represents that the computation exceeds 24 hours. 'OMR' is out of memory error.

Target	Mu (0)	Alpha (1)	HOMO (2)	LUMO (3)	Gap (4)	R2 (5)	ZPVE(6)	U0 (7)	U (8)	H (9)	G(10)	Cv (11)
MAE	0.3201	0.5792	0.0060	0.0062	0.0091	10.0469	0.0007	0.3204	0.2934	0.2722	0.2948	0.2368

Semi-Supervised	Error Ratio											
Mean-Teachers	1.09	1.00	**0.99**	1.00	**0.97**	**0.52**	**0.77**	1.16	0.93	**0.79**	0.86	**0.86**
InfoGraph	1.02	0.97	1.02	**0.99**	1.01	0.71	0.96	0.85	0.93	0.93	0.99	1.00
InfoGraph*	**0.99**	**0.94**	**0.99**	**0.99**	1.00	0.68	0.81	**0.71**	**0.87**	0.98	**0.85**	0.89

Table 2: Results of semi-supervised experiments on QM9 dataset. The result in **bold** indicates the best performance. The top half of the table shows the mean absolute error (MAE) of the supervised model. The bottom half shows the error ratio (with respect to supervised result) of the semi-supervised models using the same underlying model. Lower scores are better and values less than 1.0 indicate better performance than the supervised baseline.

run on the two larger benchmark datasets. We find that InfoGraph outperforms all of these baselines on 4 out of 6 of the datasets. In the other 2 datasets, InfoGraph still has very competitive performance.

The results of the semi-supervised learning experiments on the molecular property prediction task are presented in Table 2. We observe that by simply combining the supervised objective with the unsupervised infomax objective (InfoGraph) obtains better performance compared to the purely supervised models on 7 out of 12 of the targets. However, in 1 out of 12 targets it does not obtain better performance and in 4 out of 12 targets, it results in poorer performance. This "negative transfer" effect may be caused by the fact that the supervised objective and the unsupervised objective favor different information or different latent semantic space. This effect is alleviated with InfoGraph*, our modified version of InfoGraph for semi-supervised learning. InfoGraph* improves over the supervised model in 11 out of 12 targets and gets a tie on 1 target. InfoGraph* obtains the best result on 7 targets while the Mean Teacher method obtains the best results on 6 targets (with one overlap). However, the Mean Teacher model yields worse performance on 2 targets when compared to the supervised result.

6 Conclusion and Future work

In this paper, we propose InfoGraph to learn unsupervised graph-level representations and InfoGraph* for semi-supervised learning. We conduct experiments on graph classification and molecular property prediction tasks to evaluate these two methods. Experimental results show that InfoGraph and InfoGraph* are both very competitive with state-of-the-art methods. There are many research works on semi-supervised learning on image data, but few of them focus on semi-supervised learning for graph structured data. In the future, we aim to explore semi-supervised frameworks designed specifically for graphs.

Acknowledgments

We would like to thank Shengchao Liu and Weihua Hu for the extremely helpful discussions and comments.

References

[1] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 170–182. Springer, 2018.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. Journal of machine Learning research, 3(Jan):993–1022, 2003.

[3] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In Fifth IEEE international conference on data mining (ICDM'05), pages 8–pp. IEEE, 2005.

[4] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3):27, 2011.

[5] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. Semi-supervised learning, vol. 2. Cambridge: MIT Press. Cortes, C., & Mohri, M.(2014). Domain adaptation and sample bias correction theory and algorithm for regression. Theoretical Computer Science, 519:103126, 2006.

[6] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. Chemistry of Materials, 31(9):3564–3572, 2019.

[7] Pengfei Chen, Weiwen Liu, Chang-Yu Hsieh, Guangyong Chen, and Shengyu Zhang. Utilizing edge features in graph neural networks via variational information maximization. arXiv preprint arXiv:1906.05488, 2019.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th international conference on Machine learning, pages 160–167. ACM, 2008.

[9] Gerben KD de Vries. A fast approximation of the weisfeiler-lehman graph kernel for rdf data. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 606–621. Springer, 2013.

[10] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems, pages 2224–2232, 2015.

[11] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In Advances in Neural Information Processing Systems, pages 216–224, 2013.

[12] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.

[13] Sheng Gao, Ludovic Denoyer, and Patrick Gallinari. Temporal link prediction by integrating content and structure information. In Proceedings of the 20th ACM international conference on Information and knowledge management, pages 1169–1174. ACM, 2011.

[14] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In Learning theory and kernel machines, pages 129–143. Springer, 2003.

[15] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1263–1272. JMLR. org, 2017.

[16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864. ACM, 2016.

[17] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. arXiv preprint arXiv:1808.06670, 2018.

[18] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Pre-training graph neural networks. arXiv preprint arXiv:1905.12265, 2019.

[19] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. arXiv preprint arXiv:1707.07328, 2017.

[20] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. arXiv preprint arXiv:1802.04364, 2018.

[21] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In Proceedings of the 20th international conference on machine learning (ICML-03), pages 321–328, 2003.

[22] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. arXiv preprint arXiv:1802.05983, 2018.

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

[25] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. arXiv preprint arXiv:1802.04687, 2018.

[26] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[27] Risi Kondor and Horace Pan. The multiscale laplacian graph kernel. In Advances in Neural Information Processing Systems, pages 2990–2998, 2016.

[28] Nils M Kriege, Pierre-Louis Giscard, and Richard Wilson. On valid optimal assignment kernels and applications to graph classification. In Advances in Neural Information Processing Systems, pages 1623–1631, 2016.

[29] Nevan J Krogan, Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta, Aaron P Tikuisis, et al. Global landscape of protein complexes in the yeast saccharomyces cerevisiae. Nature, 440(7084):637, 2006.

[30] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised learning. arXiv preprint arXiv:1610.02242, 2016.

[31] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In International conference on machine learning, pages 1188–1196, 2014.

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.

[34] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. IEEE transactions on pattern analysis and machine intelligence, 41(8):1979–1993, 2018.

[35] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In Advances in neural information processing systems, pages 2265–2273, 2013.

[36] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen andvYang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. CoRR, abs/1707.05005, 2017.

[37] Marion Neumann, Novi Patricia, Roman Garnett, and Kristian Kersting. Efficient graph kernels by randomization. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 378–393. Springer, 2012.

[38] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. Physical review E, 69(2):026113, 2004.

[39] Hai Nguyen, Shin-ichi Maeda, and Kenta Oono. Semi-supervised learning of hierarchical representations of molecules using neural message passing. arXiv preprint arXiv:1711.10168, 2017.

[40] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In International conference on machine learning, pages 2014–2023, 2016.

[41] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In Advances in neural information processing systems, pages 271–279, 2016.

[42] Avital Oliver, Augustus Odena, Colin A Raffel, Ekin Dogus Cubuk, and Ian Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In Advances in Neural Information Processing Systems, pages 3235–3246, 2018.

[43] Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[44] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 22(10):1345–1359, 2009.

[45] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[46] Georgios A Pavlopoulos, Maria Secrier, Charalampos N Moschopoulos, Theodoros G Soldatos, Sophia Kossida, Jan Aerts, Reinhard Schneider, and Pantelis G Bagos. Using graph theory to analyze biological networks. BioData mining, 4(1):10, 2011.

[47] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710. ACM, 2014.

[48] Nataša Pržulj. Biological network comparison using graphlet degree distribution. Bioinformatics, 23(2):e177–e183, 2007.

[49] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. Scientific data, 1:140022, 2014.

[50] Michael T Rosenstein, Zvika Marx, Leslie Pack Kaelbling, and Thomas G Dietterich. To transfer or not to transfer. In NIPS 2005 workshop on transfer learning, volume 898, page 3, 2005.

[51] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. Journal of Machine Learning Research, 12(Sep):2539–2561, 2011.

[52] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In Artificial Intelligence and Statistics, pages 488–495, 2009.

[53] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In Proceedings of the 24th international conference on world wide web, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[54] Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In Advances in neural information processing systems, pages 1195–1204, 2017.

[55] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. arXiv preprint arXiv:1809.10341, 2018.

[56] Saurabh Verma and Zhi-Li Zhang. Hunt for the unique, stable, sparse and fast feature learning on graphs. In Advances in Neural Information Processing Systems, pages 88–98, 2017.

[57] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. Community discovery using nonnegative matrix factorization. Data Mining and Knowledge Discovery, 22(3):493–521, 2011.

[58] Boris Weisfeiler and Andrei A Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia, 2(9):12–16, 1968.

[59] Tian Xie and Jeffrey C Grossman. Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. Physical review letters, 120(14):145301, 2018.

[60] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.

[61] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. arXiv preprint arXiv:1806.03536, 2018.

[62] Yan Yan, Shangzhao Qiu, Zhuxuan Jin, Sihong Gong, Yun Bai, Jianwei Lu, and Tianwei Yu. Detecting subnetwork-level dynamic correlations. Bioinformatics, 33(2):256–265, 2017.

[63] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1365–1374. ACM, 2015.

[64] Rendong Yang, Yun Bai, Zhaohui Qin, and Tianwei Yu. Egonet: identification of human disease ego-network modules. BMC genomics, 15(1):314, 2014.

[65] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In Advances in Neural Information Processing Systems, pages 4800–4810, 2018.

[66] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[67] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In Proceedings of the 20th International conference on Machine learning (ICML-03), pages 912–919, 2003.

A Datasets

A.1 Graph Classification Datasets

MUTAG contains 188 mutagenic aromatic and heteroaromatic nitro compounds with 7 different discrete labels. PTC is a dataset of 344 different chemical compounds that have been tested for carcinogenicity in male and female rats. This dataset has 19 discrete labels. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets. Each graph corresponds to an ego-network for each actor/actress, where nodes correspond to actors/actresses and an edge is drawn between two actors/actresses if they appear in the same movie. Each graph is derived from a pre-specified genre of movies, and the task is to classify the genre graph it is derived from. REDDIT-BINARY and REDDIT-MULTI5K are balanced datasets where each graph corresponds to an online discussion thread and nodes correspond to users. An edge was drawn between two nodes if at least one of them responded to another's comment. The task is to classify each graph to the community or subreddit that it belongs to.

A.2 QM9

For a description of the properties in the QM9 dataset see section 10.2 of [15].

5.3 Segmentation Code

The level-set method based segmentation code I developed was of interest to a variety of other researchers at Harvard University. For two of these projects, I worked in collaboration with other groups at Harvard to deploy my segmentation code on their datasets.

5.3.1 Segmentation of Compressing Structures

Katia Bertoldi's group was looking at the properties of compressed structures that were initially composed of a grid of disc-shaped holes. During compression, the holes gained an orientation, either parallel or perpendicular to the axis of compression. However, studying the geometric properties of the holes during the compression was not a trivial segmentation task. As the material compressed, the holes would get smaller, at some point becoming nearly invisible. Additionally, the center of the holes would come in contact before either side, resulting in shapes that appeared to be two objects rather than one. The lighting and the material made it increasingly difficult to use off-the-shelf segmentation tools. However, by modifying my tool, I was able to create extremely accurate segmentations for nearly the entire duration of the video. By initializing each hole with four different seeds, I performed a level set segmentation up until some maximum time, T. Then, using a lookup table, I reassembled the space belonging to each hole. Then, for hole i, I assemble a list of all coordinates belonging to that hole, $\vec{x}_i = ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n))$. Then, for this list, I perform k-means clustering with four seeds. This method allows me to track an object that splits while still calling it a single object.

It's a magical world, Hobbes, ol' buddy... Let's go exploring!

Bill Watterson

6 Conclusion

The data renaissance in the sciences has many exciting possibilities for changing the way science is done. One often overlooked aspect of the digital age is the ability to unearth and make available huge collections of old work. For data-driven disciplines, observational sciences, etc., there is treasure hidden away in older books, papers, and documents. As these documents are made available on the Internet, I am excited and optimistic about how these can be used. Suddenly, papers from over 200 years ago can be revisited, and in some cases the data can be directly extracted and used to answer new questions.[33,34,75,123] For example, in Hoffmann *et al.* (2018)[75], we used resources dating back to the late 1800s. In addition to gathering existing data, new tools have made the rate of data collection unparalleled. For example, biological datasets from bench-top experiments can rapidly exceed 10 terabytes (see Chapter 1) and astronomical, materials, and high-energy physics datasets from large collaborations have reached the petabyte regime.[88,59]

With the influx of new, colossally large datasets there has been an accompanying increase in tools created to process them. Many high-level code bases have been developed to help process these large datasets that have become very commonly used.[32,1,169,105] In an academic setting, new computational tools are being developed and released on a daily basis. Many of them, especially in the biosciences, result from current tools not working well on new or slightly different datasets. For example, the number of image segmentation tools that have been released is impressive. However, recently there has been growing momentum to create unified toolkits that are able to work on a wide range of image types.[137] The creation of more general tools (which often use more domain-specific knowledge such as encoding symmetries) is something that I think is very exciting, and something that is becoming increasingly popular within the machine learning community.

For the work in Chapter 1, on the development of *Gryllus bimaculatus*, though the scientific goal was one in biology, many exciting results were only possible through improvements in computation and imaging.[113,114] Specifically, one of the main tools was in image segmentation.[137,27] Any success in this project relied heavily on our ability to segment and then track 3-D light sheet datasets.[137,163] It is important to track nuclei through many frames and through divisions. To get reliable tracks through many frames, even 95% accuracy is not good enough. Numerical post-processing methods, like deconvolution[113], greatly assist in making the data more clear, but these methods are computationally demanding and still not perfect. In the coming years, I predict that advances in microscopy coupled with improvements in post-processing and segmentation of biological datasets will result in exciting answers to many questions including those of cell fate and nuclear movement. While advances in optics are coming (relatively) more slowly, tools used to correct for imaging distortion[113] and image segmentation[14,119,137,29,93] are rapidly improving. Pushing the tracking between frames closer to 100% accuracy and also more accurately accounting for divisions allow new classes of questions to be answered, especially in developmental regimes that are past what we consider. For example, shortly after the *axial expansion* process described in Chapter 1 completes, certain nuclei coalesce into the embryonic rudiment. At this stage, there are over 2000 nuclei that are quite close together. However, the ability to track nuclei through this stage opens many exciting doors.

Another exciting direction is that relatively little is known about how the development of many insects defers from that of *Drosophila*. Work from the Keller Laboratory[148,4] on *Drosophila* is similar to the work presented in this chapter, but there are notable differences. While the work on model organisms has lead to many important breakthroughs, understanding what aspects we learn from these organisms generalize to other organisms is also an exciting prospect. In the coming years, I think that more general developmental principles will become better understood and I am excited to see a more holistic picture emerge.

Compared to many other disciplines of science in this thesis, research in (developmental) biology is inherently slower and more costly—both in time, but also in terms of resources. I think that mathematical modeling and biology are forming a new symbiosis, as we are past the "spherical cow" phase, where systems are often heavily simplified to be analytically and computationally tractable. The ability to, from data, develop models that are capable of making specific, falsifiable hypothesis is particularly encouraging and something I think as beneficial for both fields.

Many institutions have undertaken massive efforts to digitize their work, but collecting all of this information can still be a monumental task. One example of success in this direction is by Church *et al.* where they used over 10,000 previously published insect egg descriptions to uncover scaling relations and show that size and shape of eggs can be better understood by considering the environment where those eggs are laid.[34]

Another exciting direction in this "data renaissance" is the formation of new machine learning tools. These tools have allowed scientists to revisit previously studied systems with a new toolkit. For example, the work in Chapter 3, on crumpled sheets, built on an existing dataset from previous work.[65] In fact, the published project did not start by looking at the geometric relationship between ridges and valleys. Instead, the original goal was to predict the regions where new creases form. However, for many reasons this proved to be a much harder problem than anticipated. For example, alignment issues and experimental noise make it difficult to definitively label newly-formed creases. Additionally, the nature of this problem makes the dataset much smaller, since the dataset was originally collected for a different project and therefore not specifically tailored for the new questions of interest. In fact, the ideal datasets for these two projects are largely orthogonal since in the existing dataset, subsequent scans are highly correlated.[65] The fact that each crumple is very similar to the previous scan means that the effective size of the dataset is much smaller than the total number of creases. This makes many data-intensive methods less effective.

My collaborators and I looked at the scaling of the mileage in the crease network at smaller scales and whether we could predict the relative enrichment and depletion of creases. Future work exploring this direction will be an exciting path forward, especially if results could be adapted to other materials as well.

I am curious to what extent many disordered systems will be at least partially illuminated in the coming years by recently developed statistical methods. For example, the work by D. Cubuk *et al.* leading to the softness parameter is a nice success story in this line of work.[37] Andrea Liu's group has been extending the idea of using a support vector machine (SVM)[36,126] to quantify the distance to phase-transition boundaries in different systems, including problems in seemingly diverse fields such as geology and biology.[117,131,171,92] One appealing result of this work was that by constructing specific input features (radial distribution functions), they were able to compute a distance from a transition-separating hyperplane. This lends itself to a degree of interpretability that is very desirable in physics.

I'm optimistic about the symbiotic relationship between machine learning and the natural sciences in the future. One avenue that I find particularly exciting is learning compressed representations of physical objects that is able to obey constraints imposed by nature. Exciting prospects for learning compressed, searchable/optimizable representations such as solar cells and materials for batteries, amongst many others. Early work in this direction has begun and I am excited to see how it continues.[60,103]

A concrete avenue for future work relating to the generation of random crystal subjects is underway within the field of random structure generation.[109] Many scientific fields, such as random structure generation, that rely on *ad hoc* rules seem to have tremendous potential for advancement (similar to replacing the value-function in chess engines).[133] Generally, I think that there has recently been much exciting work at the intersection of AI and material science along these directions.[130,38,69,77]

Introducing physics-inspired biases into machine learning methods is a growing trend, as it provides ways to be more data-efficient and also (hopefully) generalize better to new tasks.[159,35] The inductive bias in machine learning (specifically of deep learning) has largely been the antithesis of how scientists think about the natural world.[19] In many image based learning situations, the entire input frame is considered to be a large state that as a whole is encoded/otherwise understood. There is rarely an effective bias towards trying to decompose a single, large, complex system into a series of smaller systems.[155,168] Aside from learning different objects that can be moved or interact differently, the dynamics governing motion is not treated to be a concept that can be simplified. However, we know this to be the case in the natural world made explicit by Newton's laws of motion.[100]

I think that many different researchers are beginning to push in this direction, developing tools that are more in tune with how we understand the world around us.[158,124] Tools for improved generalization in machine learning and few-shot learning are both active fields of research.[53,12,149,116] I believe that for both these cases, there will be concrete progress in the near future.

Data representation has proven to be a key problem in the success of machine learning. Data that can be represented as a graph is becoming increasingly important for many tasks, including predicting properties of social networks, predicting properties of molecules (and the closely related field of drug discovery and protein interaction networks), to name a few.[167,82,8,9,11,13] Many aspects of the natural world can be effectively modeled by a graph, and this representation affords many desirable properties, such as adding in an explicit relationship between interacting parts that needs to be learned.[158]

The data that we have today was unimaginable a decade ago and many of the tools we use today had not been dreamed up a decade ago. I am excited to see what things are like a decade from now.

6.1 Final thoughts

Science is inherently collaborative, perhaps more-so now than ever before. I am lucky to have been surrounded by an excellent set of collaborators that made my time working on these projects very fun. I am still excited about all of the projects discussed in this thesis, something I hope bodes well for the future.

> **Pippin:** I didn't think it would end this way.
> **Gandalf:** End? No, the journey doesn't end here. [Defending] is just another path, one that we all must take.[147]

And with that, this thesis ends—happily ever after.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Achlioptas, P., Diamanti, O., Mitliagkas, I., & Guibas, L. (2017). Learning representations and generative models for 3D point clouds. *arXiv:1707.02392*.

[3] AlQuraishi, M. (2019). End-to-end differentiable learning of protein structure. *Cell systems*, 8(4), 292–301.

[4] Amat, F., Lemon, W., Mossing, D. P., McDole, K., Wan, Y., Branson, K., Myers, E. W., & Keller, P. J. (2014). Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data. *Nature methods*.

[5] Anderson, D. (1972a). The development of hemimetabolous insects. *Developmental Systems: Insects*, 1, 95–163.

[6] Anderson, D. (1972b). The development of holometabolous insects. *Developmental Systems: Insects*, 1, 165–242.

[7] Andresen, C. A., Hansen, A., & Schmittbuhl, J. (2007). Ridge network in crumpled paper. *Physical review E*, 76(2), 026108.

[8] Anonymous (2020a). Combining graph and sequence information to learn protein representations. In *Submitted to International Conference on Learning Representations.* under review.

[9] Anonymous (2020b). Conditional generation of molecules from disentangled representations. In *Submitted to International Conference on Learning Representations.* under review.

[10] Anonymous (2020c). Deep learning for symbolic mathematics. In *Submitted to International Conference on Learning Representations.* under review.

[11] Anonymous (2020d). Directional message passing for molecular graphs. In *Submitted to International Conference on Learning Representations.* under review.

[12] Antoniou, A., Edwards, H., & Storkey, A. (2019). How to train your MAML. In *International Conference on Learning Representations*.

[13] Assouel, R., Ahmed, M., Segler, M. H., Saffari, A., & Bengio, Y. (2018). DE-Factor: Differentiable edge factorization-based probabilistic graph generation. *arXiv:1811.09766*.

[14] Badrinarayanan, V., Kendall, A., & Cipolla, R. (2015). Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv:1511.00561*.

[15] Baird, H. S. (1992). Document image defect models. In *Structured Document Image Analysis* (pp. 546–556). Springer.

[16] Baker, J., Theurkauf, W. E., & Schubiger, G. (1993). Dynamic changes in microtubule configuration correlate with nuclear migration in the preblastoderm drosophila embryo. *The Journal of Cell Biology*, 122(1), 113–121.

[17] Baldi, P., Sadowski, P., & Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1).

[18] Bar-Sinai, Y., Hoyer, S., Hickey, J., & Brenner, M. P. (2019). Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31), 15344–15349.

[19] Bengio, Y. (2017). The consciousness prior. *arXiv:1709.08568*.

[20] Bernard, F., Lepesant, J.-A., & Guichet, A. (2018). Nucleus positioning within drosophila egg chamber. In *Seminars in Cell & Developmental Biology*, volume 82 (pp. 25–33).: Elsevier.

[21] Bhimji, W., Farrell, S. A., Kurth, T., Paganini, M., Prabhat, & Racah, E. (2017). Deep neural networks for physics analysis on low-level whole-detector data at the LHC. *arXiv:1711.03573*.

[22] Blum, L. C. & Reymond, J.-L. (2009). 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131, 8732.

[23] Brock, A., Donahue, J., & Simonyan, K. (2018). Large scale GAN training for high fidelity natural image synthesis. *arXiv:1809.11096*.

[24] Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2016). Generative and discriminative voxel modeling with convolutional neural networks. *arXiv:1608.04236*.

[25] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., & Lerchner, A. (2018). Understanding disentangling in beta-VAE. *arXiv:1804.03599*.

[26] Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Żołna, K., Aytar, Y., Budden, D., Vecerik, M., Sushkov, O., Barker, D., Scholz, J., Denil, M., de Freitas, N., & Wang, Z. (2019). A framework for data-driven robotics. *arXiv:1909.12200*.

[27] Çiçek, O., Abdulkadir, A., Lienkamp, S. S., Brox, T., & Ronneberger, O. (2016). 3D U-Net: Learning dense volumetric segmentation from sparse annotation. *Springer Lecture Notes in Computer Science*, (pp. 424–432).

[28] Chandrasekaran, A., Kamal, D., Batra, R., Kim, C., Chen, L., & Ramprasad, R. (2019). Solving the electronic structure problem with machine learning. *npj Computational Materials*, 5(1), 22.

[29] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). DeepLab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4), 834–848.

[30] Chen, R. T. Q., Li, X., Grosse, R., & Duvenaud, D. (2018). Isolating sources of disentanglement in variational autoencoders. *Advances in Neural Information Processing Systems*.

[31] Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., Ferrero, E., Agapow, P.-M., Zietz, M., Hoffman, M. M., Xie, W., Rosen, G. L., Lengerich, B. J., Israeli, J., Lanchantin, J., Woloszynek, S., Carpenter, A. E., Shrikumar, A., Xu, J., Cofer, E. M., Lavender, C. A., Turaga, S. C., Alexandari, A. M., Lu, Z., Harris, D. J., DeCaprio, D., Qi, Y., Kundaje, A., Peng, Y., Wiley, L. K., Segler, M. H., Boca, S. M., Swamidass, S. J., Huang, A., Gitter, A., & Greene, C. S. (2018). Opportunities and obstacles for deep learning in biology and medicine. *bioRxiv*.

[32] Chollet, F. et al. (2015). Keras.

[33] Church, S., Donoughe, S., de Medeiros, B., & Extavour, C. (2018). A database of egg size and shape from more than 6,000 insect species. *Scientific Data*, 346(6210), 763–767.

[34] Church, S. H., Donoughe, S., de Medeiros, B. A., & Extavour, C. G. (2019). Insect egg size and shape evolve with ecology but not developmental rate. *Nature*, 571(7763), 58.

[35] Cohen, T. S. & Welling, M. (2016). Steerable CNNs. *arXiv:1612.08498*.

[36] Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3), 273–297.

[37] Cubuk, E., Schoenholz, S., Rieser, J., Malone, B., Rottler, J., Durian, D., Kaxiras, E., & Liu, A. (2015). Identifying structural flow defects in disordered solids using machine-learning methods. *Physical Review Letters*, 114(10).

[38] Cubuk, E. D., Sendek, A. D., & Reed, E. J. (2019a). Screening billions of candidates for solid lithium-ion conductors: A transfer learning approach for small data. *The Journal of Chemical Physics*, 150(21), 214701.

[39] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., & Le, Q. V. (2019b). Autoaugment: Learning augmentation strategies from data. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[40] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

[41] D'Arcy, W. T. (1917). *On Growth And Form*. Cambridge University Press.

[42] Dawes, R., Dawson, I., Falciani, F., Tear, G., & Akam, M. (1994). Dax, a locust Hox gene related to fushi-tarazu but showing no pair-rule expression. *Development*, 120(6), 1561–1572.

[43] De Simone, A., Spahr, A., Busso, C., & Gönczy, P. (2018). Uncovering the balance of forces driving microtubule aster migration in c. elegans zygotes. *Nature Communications*, 9(1), 938.

[44] Deneke, V. E., Puliafito, A., Krueger, D., Narla, A. V., De Simone, A., Primo, L., Vergassola, M., De Renzis, S., & Di Talia, S. (2019). Self-organized nuclear positioning synchronizes the cell cycle in drosophila embryos. *Cell*, 177(4), 925–941.

[45] Dieleman, S., Willett, K. W., & Dambre, J. (2015). Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2), 1441–1459.

[46] Donoughe, S. & Extavour, C. G. (2016). Embryonic development of the cricket gryllus bimaculatus. *Developmental Biology*, 411(1), 140–156.

[47] Donoughe, S., Kim, C., & Extavour, C. G. (2018). High-throughput live-imaging of embryos in microwell arrays using a modular specimen mounting system. *Biology Open*, 7(7), bio031260.

[48] Dutta, S., Djabrayan, N. J.-V., Torquato, S., Shvartsman, S. Y., & Krajnc, M. (2019). Self-similar dynamics of nuclear packing in the early drosophila embryo. *Biophysical journal*, 117(4), 743–750.

[49] Eastham, L. (1927). A contribution to the embryology of pieris rapae. *Quart. J. micr. Sci*, 71, 353–394.

[50] Edgar, B. A., Kiehle, C. P., & Schubiger, G. (1986). Cell cycle control by the nucleocytoplasmic ratio in early drosophila development. *Cell*, 44(2), 365–372.

[51] Eslami, S. M. A., Jimenez Rezende, D., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., Reichert, D. P., Buesing, L., Weber, T., Vinyals, O., Rosenbaum, D., Rabinowitz, N., King, H., Hillier, C., Botvinick, M., Wierstra, D., Kavukcuoglu, K., & Hassabis, D. (2018). Neural scene representation and rendering. *Science*, 360(6394), 1204–1210.

[52] Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Zidek, A., Nelson, A., Bridgland, A., Penedones, H., Petersen, S., Simonyan, K., Crossan, S., Jones, D., Silver, D., Kavukcuoglu, K., Hassabis, D., & Senior, A. (2018). De novo structure prediction with deep-learning based scoring. *Thirteenth Critical Assessment of Techniques for Protein Structure Prediction*.

[53] Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv:1703.03400*.

[54] Fischer, R. (1999). Nuclear movement in filamentous fungi. *FEMS Microbiology Reviews*, 23(1), 39–68.

[55] Foe, V., Odell, G., & Edgar, B. (1993). Mitosis and morphogenesis in the drosophila embryo: Point and counterpoint. In *The Development of Drosophila melanogaster* chapter 3, (pp. 149–300). Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press.

[56] Foe, V. E. & Alberts, B. M. (1983). Studies of nuclear and cytoplasmic behaviour during the five mitotic cycles that precede gastrulation in drosophila embryogenesis. *Journal of Cell Science*, 61(1), 31–70.

[57] Foe, V. E., Field, C. M., & Odell, G. M. (2000). Microtubules and mitotic cycle phase modulate spatiotemporal distributions of f-actin and myosin ii in drosophila syncytial blastoderm embryos. *Development*, 127(9), 1767–1787.

[58] Furtney, J. (2012–2019). scikit-fmm. https://pythonhosted.org/scikit-fmm/.

[59] Gaillard, M. (2017). CERN data centre passes the 200-petabyte milestone. https://home.cern/news/news/computing/cern-data-centre-passes-200-petabyte-milestone.

[60] Gebauer, N. W. A., Gastegger, M., & Schütt, K. T. (2019). Symmetry-adapted generation of 3D point sets for the targeted discovery of molecules. *arXiv:1906.00957*.

[61] Gomez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernandez-Lobato, J. M., Sanchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., & Aspuru-Guzik, A. (2018). Automatic chemical design using a data-driven continuous representation of molecules. *American Chemical Society Central Science*.

[62] Gönczy, P., Pichler, S., Kirkham, M., & Hyman, A. A. (1999). Cytoplasmic dynein is required for distinct aspects of mtoc positioning, including centrosome separation, in the one cell stage caenorhabditis elegans embryo. *The Journal of Cell Biology*, 147(1), 135–150.

[63] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research* (pp. 1319–1327). Atlanta, Georgia, USA: PMLR.

[64] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *arXiv:1406.2661*.

[65] Gottesman, O., Andrejevic, J., Rycroft, C. H., & Rubinstein, S. M. (2018). A state variable for crumpled thin sheets. *Communications Physics*, 1(1), 70.

[66] Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., & Schölkopf, B. (2019). Recurrent independent mechanisms: A new architecture for improving generalization. *arXiv preprint arXiv:1909.10893*.

[67] Guest, D., Cranmer, K., & Whiteson, D. (2018). Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, 68(1), 161–181.

[68] Gundersen, G. G. & Worman, H. J. (2013). Nuclear positioning. *Cell*, 152(6), 1376–1389.

[69] Hanakata, P. Z., Cubuk, E. D., Campbell, D. K., & Park, H. S. (2018). Accelerated search and design of stretchable graphene kirigami using machine learning. *Physical Review Letters*, 121(25).

[70] Hatanaka, K. & Okada, M. (1991). Retarded nuclear migration in drosophila embryos with aberrant f-actin reorganization caused by maternal mutations and by cytochalasin treatment. *Development*, 111(4), 909–920.

[71] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M. M., Mohamed, S., & Lerchner, A. (2017). beta-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*.

[72] Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., & Bengio, Y. (2018). Learning deep representations by mutual information estimation and maximization. *arXiv:1808.06670*.

[73] Ho, K., Dunin-Borkowski, O. M., & Akam, M. (1997). Cellularization in locust embryos occurs before blastoderm formation. *Development*, 124(14), 2761–2768.

[74] Hoffmann, J., Bar-Sinai, Y., Lee, L. M., Andrejevic, J., Mishra, S., Rubinstein, S. M., & Rycroft, C. H. (2019a). Machine learning in a data-limited regime: Augmenting experiments with synthetic data uncovers order in crumpled sheets. *Science Advances*, 5(4).

[75] Hoffmann, J., Donoughe, S., Li, K., Salcedo, M. K., & Rycroft, C. H. (2018). A simple developmental model recapitulates complex insect wing venation patterns. *Proceedings of the National Academy of Sciences*, 115(40), 9905–9910.

[76] Hoffmann, J., Maestrati, L., Sawada, Y., Tang, J., Sellier, J. M., & Bengio, Y. (2019b). Data-driven approach to encoding and decoding 3-D crystal structures. *arXiv:1909.00949*.

[77] Hoyer, S., Sohl-Dickstein, J., & Greydanus, S. (2019). Neural reparameterization improves structural optimization. *arXiv:1909.04240*.

[78] Ingraham, J., Riesselman, A., Sander, C., & Marks, D. (2019). Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations*.

[79] Ji, J.-Y., Crest, J., & Schubiger, G. (2005). Genetic interactions between cdk1-cyclinb and the separase complex in drosophila. *Development*, 132(8), 1875–1884.

[80] Ji, J.-Y., Haghnia, M., Trusty, C., Goldstein, L. S., & Schubiger, G. (2002). A genetic screen for suppressors and enhancers of the drosophila cdk1-cyclin b identifies maternal factors that regulate microtubule and microfilament stability. *Genetics*, 162(3), 1179–1195.

[81] Ji, J.-Y., Squirrell, J. M., & Schubiger, G. (2004). Both cyclin b levels and dna-replication checkpoint control the early embryonic mitoses in drosophila. *Development*, 131(2), 401–411.

[82] Jin, W., Barzilay, R., & Jaakkola, T. (2018). Junction tree variational autoencoder for molecular graph generation. *Thirty-fifth International Conference on Machine Learning*.

[83] Johannsen, O. A. & Butt, F. H. (1941). *Embryology of insects and myriapods*. McGraw-Hill Book Co.

[84] Kasthuri, N., Hayworth, K. J., Berger, D. R., Schalek, R. L., Conchello, J. A., Knowles-Barley, S., Lee, D., Vázquez-Reina, A., Kaynig, V., Jones, T. R., et al. (2015). Saturated reconstruction of a volume of neocortex. *Cell*, 162(3), 648–661.

[85] Kessel, E. L. (1939). The embryology of fleas. *Smithsonian Miscellaneous Collections*, 98.

[86] Kim, J. H., Jin, P., Duan, R., & Chen, E. H. (2015). Mechanisms of myoblast fusion during muscle development. *Current Opinion in Genetics & Development*, 32, 162–170.

[87] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12 (pp. 1097–1105). USA: Curran Associates Inc.

[88] Laanait, N., Romero, J., Yin, J., Young, M. T., Treichler, S., Starchenko, V., Borisevich, A., Sergeev, A., & Matheson, M. (2019). Exascale deep learning for scientific inverse problems. *arXiv:1909.11150*.

[89] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436.

[90] Li, Q., Lin, D., & Shi, Z. (2005). Task-oriented sparse coding model for pattern classification. *Advances in Natural Computation*, (pp. 903–914).

[91] Lintott, C., Schawinski, K., Bamford, S., Slosar, A., Land, K., Thomas, D., Edmondson, E., Masters, K., Nichol, R. C., Raddick, M. J., & et al. (2010). Galaxy zoo 1: data release of morphological classifications for nearly 900,000 galaxies. *Monthly Notices of the Royal Astronomical Society*, 410(1), 166–178.

[92] Liu, A. (2019). Machined-learned softness as a structural order parameter for understanding glassy systems. In *APS Meeting Abstracts*.

[93] Maier, A., Syben, C., Lasser, T., & Riess, C. (2019). A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik*, 29(2), 86–101.

[94] Manukyan, L., Montandon, S. A., Fofonjka, A., Smirnov, S., & Milinkovitch, M. C. (2017). A living mesoscopic cellular automaton made of skin scales. *Nature*, 544(7649), 173.

[95] Markow, T. A., Beall, S., & Matzkin, L. M. (2009). Egg size, embryonic development time and ovoviviparity in drosophila species. *Journal of evolutionary biology*, 22(2), 430–434.

[96] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv:1301.3781*.

[97] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *arXiv:1310.4546*.

[98] Morris, N. R. (2000). Nuclear migration: from fungi to the mammalian brain. *The Journal of Cell Biology*, 148(6), 1097–1102.

[99] Nakamura, T., Yoshizaki, M., Ogawa, S., Okamoto, H., Shinmyo, Y., Bando, T., Ohuchi, H., Noji, S., & Mito, T. (2010). Imaging of transgenic cricket embryos reveals cell movements consistent with a syncytial patterning mechanism. *Current Biology*, 20(18), 1641–1647.

[100] Newton, I. (1999). *The Principia: mathematical principles of natural philosophy*. Univ of California Press.

[101] Ng, A. Y. (2004). Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04 (pp. 78–). New York, NY, USA: ACM.

[102] Nogueira, K., Fadel, S. G., Dourado, Í. C., de Oliveira Werneck, R., Muñoz, J. A., Penatti, O. A., Calumby, R. T., Li, L., dos Santos, J. A., & da Silva Torres, R. (2017). Data-driven flood detection using neural networks. In *MediaEval*.

[103] Nouira, A., Sokolovska, N., & Crivello, J.-C. (2018). CrystalGAN: Learning to discover crystallographic structures with generative adversarial networks. *arXiv:1810.11203*.

[104] Oktay, O., Schlemper, J., Folgoc, L. L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Hammerla, N. Y., Kainz, B., et al. (2018). Attention U-Net: Learning where to look for the pancreas. *arXiv:1804.03999*.

[105] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch. *NIPS-W*.

[106] Pearson, J. E. (1993). Complex patterns in a simple system. *Science*, 261(5118), 189–192.

[107] Perez, L. & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv:1712.04621*.

[108] Pfau, D., Spencer, J. S., de G. Matthews, A. G., & Foulkes, W. M. C. (2019). Ab-initio solution of the many-electron schrödinger equation with deep neural networks. *arXiv:1909.02487*.

[109] Pickard, C. J. & Needs, R. (2011). Ab initio random structure searching. *Journal of Physics: Condensed Matter*, 23(5), 053201.

[110] Pietzsch, T., Saalfeld, S., Preibisch, S., & Tomancak, P. (2015). Bigdataviewer: visualization and processing for large image data sets. *Nature methods*, 12(6), 481.

[111] Płachno, B. J. & Świątek, P. (2011). Syncytia in plants: cell fusion in endosperm—placental syncytium formation in utricularia (lentibulariaceae). *Protoplasma*, 248(2), 425–435.

[112] Polishchuk, P. G., Madzhidov, T. I., & Varnek, A. (2013). Estimation of the size of drug-like chemical space based on GDB-17 data. *Journal of computer-aided molecular design*, 27(8), 675–679.

[113] Preibisch, S., Amat, F., Stamataki, E., Sarov, M., Singer, R. H., Myers, E., & Tomancak, P. (2014). Efficient Bayesian-based multiview deconvolution. *Nature methods*, 11(6), 645.

[114] Preibisch, S., Saalfeld, S., Schindelin, J., & Tomancak, P. (2010). Software for bead-based registration of selective plane illumination microscopy data. *Nature methods*, 7(6), 418.

[115] Reinsch, S. & Gonczy, P. (1998). Mechanisms of nuclear positioning. *Journal of Cell Science*, 111(16), 2283–2295.

[116] Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., & Turner, R. E. (2019). Fast and flexible multi-task classification using conditional neural adaptive processes. *arXiv:1906.07697*.

[117] Ridout, S., Rocks, J., & Liu, A. (2019). Machine-learned structure/dynamics relation in sheared jammed packings. In *APS Meeting Abstracts*.

[118] Roman, W. & Gomes, E. R. (2018). Nuclear positioning in skeletal muscle. In *Seminars in Cell & Developmental Biology*, volume 82 (pp. 51–56).: Elsevier.

[119] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (pp. 234–241).: Springer.

[120] Roonwal, M. (1936). Studies on the Embryology of the African Migratory Locust, Locusta migratoria migratorioides R. and F. I. The Early Development, with a New Theory of Multi-Phased Gastrulation among Insects. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences.*

[121] Royou, A., Sullivan, W., & Karess, R. (2002). Cortical recruitment of nonmuscle myosin ii in early syncytial drosophila embryos: its role in nuclear axial expansion and its regulation by cdc2 activity. *The Journal of Cell Biology*, 158(1), 127–137.

[122] Rupp, M., Tkatchenko, A., Müller, K.-R., & von Lilienfeld, O. A. (2012). Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108, 058301.

[123] Salcedo, M. K., Hoffmann, J., Donoughe, S., & Mahadevan, L. (2018). Size, shape and structure of insect wings. *bioRxiv*, (pp. 478768).

[124] Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., & Lillicrap, T. (2017). A simple neural network module for relational reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 4967–4976). Curran Associates, Inc.

[125] Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., et al. (2012). Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7), 676.

[126] Schölkopf, B. (2001). The kernel trick for distances. *Advances in Neural Information Processing Systems 13*, (pp. 301–307).

[127] Scholtz, G. & Wolff, C. (2013). Arthropod embryology: cleavage and germ band development. In *Arthropod biology and evolution* (pp. 63–89). Springer.

[128] Schütt, K. T., Kindermans, P.-J., Sauceda, H. E., Chmiela, S., Tkatchenko, A., & Müller, K.-R. (2017). Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *arXiv:1706.08566*.

[129] Segler, M. H. S., Preuss, M., & Waller, M. P. (2018). Planning chemical syntheses with deep neural networks and symbolic AI. *Nature*, 555, 604 EP –.

[130] Sendek, A. D., Cubuk, E. D., Antoniuk, E. R., Cheon, G., Cui, Y., & Reed, E. J. (2019). Machine learning-assisted discovery of many new solid Li-ion conducting materials. *Chemistry of Materials*, 31(2), 342–352.

[131] Sharp, T. & Liu, A. (2019). Connecting structure and dynamics in a model of confluent cell tissues using machine learning. In *APS Meeting Abstracts*.

[132] Shemer, G. & Podbilewicz, B. (2000). Fusomorphogenesis: cell fusion in organ formation. *Developmental Dynamics*, 218(1), 30–51.

[133] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419), 1140–1144.

[134] Simard, P. Y., Steinkraus, D., Platt, J. C., et al. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3.

[135] Simons, K. T., Kooperberg, C., Huang, E., & Baker, D. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *Journal of molecular biology*, 268(1), 209–225.

[136] Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*.

[137] Sommer, C., Straehle, C., Koethe, U., & Hamprecht, F. A. (2011). Ilastik: Interactive learning and segmentation toolkit. In *2011 IEEE international symposium on biomedical imaging: From nano to macro* (pp. 230–233).: IEEE.

[138] Sonnenblick, B. (1950). The early embryogenesis of drosophila melanogaster. *In Biology of Drosophila*, (pp. 62–163).

[139] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 1929–1958.

[140] Stoppard, T. (1993). *Arcadia: A play in two acts.* Samuel French, Inc.

[141] Strom, N. B. & Bushley, K. E. (2016). Two genomes are better than one: history, genetics, and biotechnological applications of fungal heterokaryons. *Fungal Biology and Biotechnology*, 3(1), 4.

[142] Sun, F.-Y., Hoffmann, J., & Tang, J. (2019a). Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv:1908.01000*.

[143] Sun, F.-Y., Qu, M., Hoffmann, J., Huang, C.-W., & Tang, J. (2019b). vgraph: A generative model for joint community detection and node representation learning. *arXiv:1906.07159*.

[144] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[145] Telley, I. A., Gáspár, I., Ephrussi, A., & Surrey, T. (2012). Aster migration determines the length scale of nuclear separation in the drosophila syncytial embryo. *J Cell Biol*, 197(7), 887–895.

[146] Thompson, D. W. (1942). *On growth and form*. Cambridge [Eng.] New York: The University Press; The Macmillan Company, a new ed. edition.

[147] Tolkein, J. (1994). *The Return of the King*. Houghton Mifflin Company.

[148] Tomer, R., Khairy, K., Amat, F., & Keller, P. J. (2012). Quantitative high-speed imaging of entire developing embryos with simultaneous multiview light-sheet microscopy. *Nature Methods*, 9(7), 755–763.

[149] Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., & Larochelle, H. (2019). Meta-dataset: A dataset of datasets for learning to learn from few examples.

[150] Tshitoyan, V., Dagdelen, J., Weston, L., Dunn, A., Rong, Z., Kononova, O., Persson, K. A., Ceder, G., & Jain, A. (2019). Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature*, 571(7763), 95.

[151] Tu, Z. & Zhu, S.-C. (2002). Image segmentation by data-driven markov chain monte carlo. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5), 657–673.

[152] Turing, A. M. (1990). The chemical basis of morphogenesis. *Bulletin of mathematical biology*, 52(1-2), 153–197.

[153] Turner, N., Goodwine, B., & Sen, M. (2016). A review of origami applications in mechanical engineering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 230(14), 2345–2362.

[154] Valsesia, D., Fracastoro, G., & Magli, E. (2019). Learning localized generative models for 3D point clouds via graph convolution. In *International Conference on Learning Representations*.

[155] van Steenkiste, S., Chang, M., Greff, K., & Schmidhuber, J. (2018). Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *International Conference on Learning Representations*.

[156] Von Dassow, G. & Schubiger, G. (1994). How an actin network might cause fountain streaming and nuclear migration in the syncytial drosophila embryo. *The Journal of Cell Biology*, 127(6), 1637–1653.

[157] Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134, 198–287.

[158] Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., & Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. In I. Guyon,

U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 4539–4547). Curran Associates, Inc.

[159] Weiler, M., Geiger, M., Welling, M., Boomsma, W., & Cohen, T. (2018). 3D steerable CNNs: Learning rotationally equivariant features in volumetric data. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18 (pp. 10402–10413). USA: Curran Associates Inc.

[160] Weininger, D. (1988). Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28(1), 31–36.

[161] Wolf, R. (1980). Migration and division of cleavage nuclei in the gall midge, Wachtliella persicariae II. Origin and ultrastructure of the migration cytaster. *Wilhelm Roux' Archiv für Entwicklungsmechanik der Organismen*, 188(1), 65–73.

[162] Wolff, C., Tinevez, J.-Y., Pietzsch, T., Stamataki, E., Harich, B., Guignard, L., Preibisch, S., Shorte, S., Keller, P. J., Tomancak, P., et al. (2018). Multi-view light-sheet imaging and tracking with the mamut software reveals the cell lineage of a direct developing arthropod limb. *Elife*, 7, e34410.

[163] Wolff, C., Tinevez, J.-Y., Pietzsch, T., Stamataki, E., Harich, B., Preibisch, S., Shorte, S., Keller, P. J., Tomancak, P., & Pavlopoulos, A. (2017). Reconstruction of cell lineages and behaviors underlying arthropod limb outgrowth with multi-view light-sheet imaging and tracking. *bioRxiv*, (pp. 112623).

[164] Wu, J., Zhang, C., Xue, T., Freeman, W. T., & Tenenbaum, J. B. (2016). Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (pp. 82–90).

[165] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv:1901.00596*.

[166] Wulfmeier, M., Abdolmaleki, A., Hafner, R., Springenberg, J. T., Neunert, M., Hertweck, T., Lampe, T., Siegel, N., Heess, N., & Riedmiller, M. (2019). Regularized hierarchical policies for compositional transfer in robotics. *arXiv:1906.11228*.

[167] Xie, T. & Grossman, J. C. (2018). Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties. *Phys. Rev. Lett.*, 120, 145301.

[168] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *arXiv:1502.03044*.

[169] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Confer-*

ence on Hot Topics in Cloud Computing, HotCloud'10 (pp. 10–10). Berkeley, CA, USA: USENIX Association.

[170] Zalokar, M. (1976). Division and migration of nuclei during early embryogenesis of drosophila melanogaster. *J. Microsc. Biol. Cell.*, 25, 97–106.

[171] Zhang, G., Ridout, S., & Liu, A. (2019). Interplay of softness and rearrangements during avalanche propagation. In *APS Meeting Abstracts*.

[172] Zhavoronkov, A., Ivanenkov, Y. A., Aliper, A., Veselov, M. S., Aladinskiy, V. A., Aladinskaya, A. V., Terentiev, V. A., Polykovskiy, D. A., Kuznetsov, M. D., Asadulaev, A., Volkov, Y., Zholus, A., Shayakhmetov, R. R., Zhebrak, A., Minaeva, L. I., Zagribelnyy, B. A., Lee, L. H., Soll, R., Madge, D., Xing, L., Guo, T., & Aspuru-Guzik, A. (2019). Deep learning enables rapid identification of potent ddr1 kinase inhibitors. *Nature Biotechnology*, 37(9), 1038–1040.

[173] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv:1812.08434*.

[174] Zhu, J.-Y., Zhang, Z., Zhang, C., Wu, J., Torralba, A., Tenenbaum, J., & Freeman, B. (2018). Visual object networks: image generation with disentangled 3D representations. In *Advances in Neural Information Processing Systems* (pp. 118–129).