Nucleotide-Level Modeling of Genetic Regulation Using Dilated Convolutional Neural Networks

Thesis advisor: Alexander Rush Ankit Gupta

Nucleotide-Level Modeling of Genetic Regulation using Dilated Convolutional Neural Networks

Abstract

The expression of genes is the product of a complex regulatory process, whose complete nature remains elusive. In order to better understand gene regulation, this work seeks to improve on efforts to model the locations of regulatory elements of the human genome directly from raw sequences of nucleotides. Past work on building this model has focused on incorporating only small amounts of DNA for this prediction task, making it difficult to model the complex long-term dependencies that arise from DNA's 3-dimensional conformation.

In this work, we model these long-term dependencies using dilated convolutional neural networks, which offer the scaling properties of convolutions while modeling long-term dependencies with the performance of recurrent neural networks (RNNs). We show that this architecture is effective at modeling the locations of transcription factor binding sites, histone modifications, and DNAse hypersensitivity sites. We develop and release a novel dataset for this larger-context modeling task, and show that dilated convolutions perform better than standard deep convolutional neural networks and RNN-based architectures at modeling the locations of regulatory markers in the human genome.

Contents

Listing of figures

To my parents.

Acknowledgments

First and foremost, I thank my advisor, Professor Alexander Rush, for helping me formulate concrete ideas from vague thoughts, teaching me how to be a better student and a better scientist, offering his time and insights generously, and being clear about how I can improve. I have grown immeasurably working with him. I also thank Dr. David Kelley, from Calico Labs, whose expertise both as a computer scientist and biologist has been invaluable and who has helped me explore my scientific interests using machine learning. I also thank Professor David Parkes, who has helped me grow as a computer scientist and was generously willing to be a reader for this thesis. Additionally, I would like to thank the other members of Professor Rush's thesis group - Kevin Yang, Alex Wang, and Jeffrey Ling - who provided feedback and guidance in our weekly meetings, and whose advice I will continue to seek.

Thank you to all of the friends and family who offered help throughout this process. I want to especially thank Tom Silver, Lucy Nam, Jonah Kallenbach, and Shaiba Rather for helping edit drafts, and the countless others who supported me along the way.

Finally, I thank my parents, whose life-long support for everything I do is a gift I cherish.

1

Introduction

The past decade has seen an explosion of structured genomic data through the advent of massively parallel genome sequencing techniques. These tools have allowed researchers to construct nearly complete genomes of several species, ranging from yeast to human. These serve as the basis of experiments to better understand how DNA encodes information in genes that is necessary to power the incredible diversity of life. Inherent in the process of gene expression is an elusive regulatory machine that determines how and when certain genes are expressed in cells. This work seeks to extend the understanding of these regulatory mechanisms by using state of the art techniques from machine learning.

Before we can discuss regulation, we first consider the basic flow of

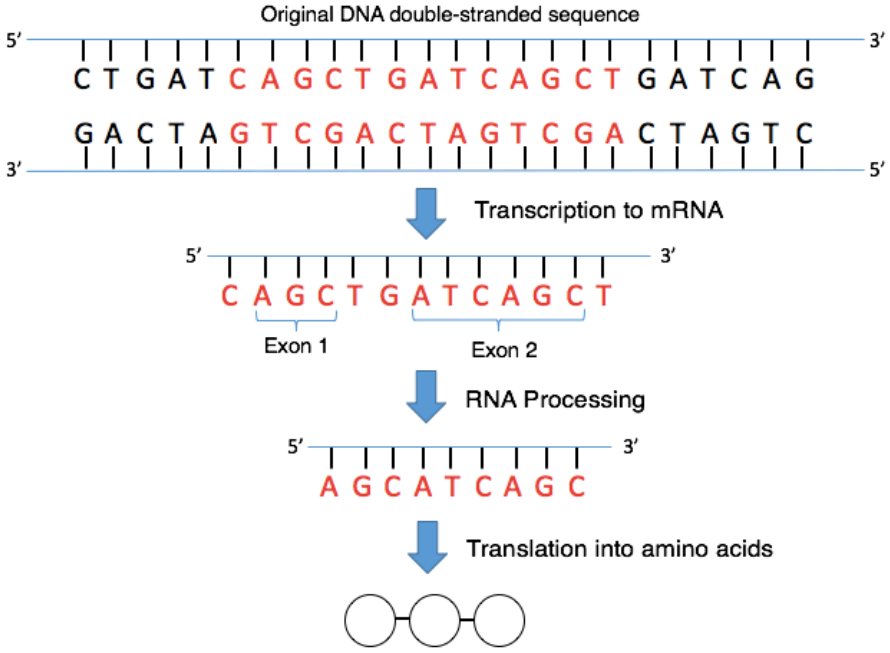


Figure 1.0.1: A visualization of the central dogma of molecular biology. The DNA sequence consists of two strands of nucleotides, where a gene is labeled in red. During the translation phase, RNA polymerase encodes the top strand as a single strand of RNA. Note that in RNA, Uracil (U) is used in place of Thymine (T), though we use T in the above diagram for consistency between the strands. Then, in RNA processing, the non-exon regions of the RNA are removed, and the processing RNA sequence is sent to a ribosome for translation. At the ribosome, every tuple of three RNA nucleotides codes for a particular amino acid, forming an amino acid polypeptide, in this case of length 3. This is also called a protein.

information from DNA in a cell, known as the central dogma of molecular biology (Crick et al., 1970) and visualized in Figure 1.0.1. DNA is a nucleic acid that consists of pairs of nucleotide bases attached in a sequence. This DNA is used to create RNA, a single-stranded nucleic acid, which is in turn used to create proteins, the workhorses of cells. However, this view of the relationship between DNA, RNA, and proteins is simplistic, and decades of genetics research have uncovered a complex array of interactions between these molecules. In particular,

many proteins and even several types of RNA interact with DNA to regulate transcription and RNA processing (Perkins et al., 2005).

These factors are elements of a vast regulatory regime that determines gene expression. Understanding regulation is thus critical to elucidating the wide range of complex interactions that determines the transcription of genes, a process central to cellular function and disfunction. One way to study regulation is to determine the locations of regulatory markers, such as binding sites for proteins, and thus learn how a change to the DNA can change regulatory behavior. In particular, this is done by modeling the mapping from a sequence of DNA to the locations of regulatory markers in that sequence.

However, the structure of DNA makes this task complicated. Because of DNA's three-dimensional conformation, nucleotides that are far apart in the 1-dimensional DNA sequence may actually be close in its 3- dimensional conformation, and thus able to interact. This makes it important to incorporate information about DNA regions that are far away in 1-D space when modeling regulatory markers. However, past models have used relatively small amounts of DNA context, and naively scaling up those techniques to incorporate more context would introduce substantial modeling complexity.

This gives rise to two fundamental questions that we answer through this work:

1. To what extent can we model the locations of regulatory markers from raw DNA sequences directly?

2. How much does increased context size improve the predictive accuracy of these models?

This work addresses the need to model large contexts by using dilated convolutional neural networks to predict the locations of regulatory markers. Dilated convolutions capture both local and distal context without needing additional parameters, allowing them to scale to large context sizes. We compare dilated convolutions to other modeling methods from deep learning, and show that they present advancements over existing models. We show that this enables them to be effective for predicting the locations of regulatory markers.

2

Biological Background

Before precisely describing the modeling task we tackle in this work, we begin by describing the relevant background material from biology that is needed to understand the task, datasets, and modeling constraints. While this is by no means an exhaustive summary of modern molecular biology research, it offers sufficient background to gain the intuitions needed for this task.

It is first prudent to understand the process by which DNA encodes information for gene regulation. Genes are the fundamental unit of function in DNA, as they are discrete regions that encode for functional protein molecules. Gene expression involves two high-level stages, known as transcription and translation, and is visualized in Figure 1.0.1. First, a protein such as RNA polymerase binds to DNA, and transcribes a substring of the DNA into a related molecule called RNA. Then, a number of post-transcriptional modifications occur to the RNA, wherein the string of RNA is potentially spliced to form a

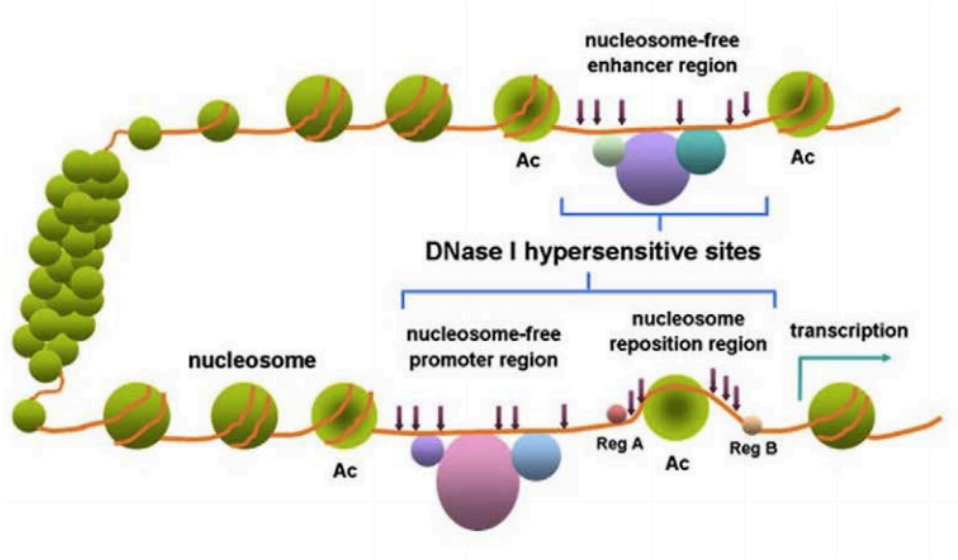


Figure 2.0.1: A visualization of gene regulation, from Wang et al. (2012). The DNA sequence is the yellow strand. Here, we see several aspects of the regulatory code. Transcription begins at the labeled region at bottom-right of the figure. On the left side, DNA is tightly coiled around nucleosomes, which are clusters of histones. Chemical modifications to these, known as histone modifications, can reduce how tightly coiled the DNA is around them. Additionally, on the top and bottom of the figure, there are enhancer and promoter regions respectively. At these areas, DNA is less tightly coiled around histones, and thus these sites are known as DNAse hypersensitivity sites. Additionally, transcription factors can bind on the enhancer or promoter region. A transcription factor that binds at the enhancer is spatially close to the transcriptionally start site, even though is the far in terms of the DNA sequence.

shorter sequence. This modified RNA strand then travels to a ribosome, where other molecules translate it to form amino acid chains, also known as proteins. These proteins are the workhorses of the cell, and so understanding how changes to DNA (or to any post-transcriptional modifications) can change how proteins are formed is critical for a variety of scientific and medical challenges.

However, this is not a complete picture. There is an enormous multitude of other factors, known as regulatory factors, that contribute to this translational process, as visualized in Figure 2.0.1. For example, DNA has a complex

three-dimensional structure which is regulated by a variety of molecules, such as histones, that DNA coils around. As a result, parts of DNA are more accessible to binding proteins than others, and so DNA accessibility is considered to be a regulatory factor. Also, modifications to the histones can affect the conformation of DNA, so histone modifications are also regulatory factors. Furthermore, proteins that bind to DNA and promote or repress transcriptional activity in a nearby region are called transcription factors, which are also regulatory factors. Sometimes, the activity that they affect can be thousands (or millions) or base pairs away, in part because distal elements of the DNA can appear adjacent in the 3-D conformation. We can see this in Figure 2.0.1, because a transcription factor that binds to the site labeled "Nucleosome-free enhancer region" is physically close to the site labeled "Transcription" even though these sites may be thousands of base pairs apart.

As such, understanding the nature of this regulatory machine is critical to uncovering the complex interplay of factors that determines gene expression, and ultimately cellular function. To do this, researchers use high-throughput sequencing technologies called Chip-Seq and DNAse-seq. We discuss these technologies in detail in Appendix A. At a high-level, Chip-Seq combines chromatin immunoprecipitation (ChiP) and massively-parallel sequencing (seq) to find where targeted proteins bind to DNA (Johnson et al., 2007). DNAse-seq is very similar, but it instead finds regions of DNA that can be cleaved by DNAse, which correspond to regions that are highly accessible to binding proteins, where we expect regulatory behavior to occur. The regions of DNA these techniques find, called "reads", are then aligned to a reference genome, giving the locations of the regulatory marker across the entire genome.

Armed with these technologies, we can formulate questions that we can study using machine learning. We can use the sequencing technologies to determine the locations of markers in the genome, particularly transcription factors, accessible regions, and histone modifications. Then, we can use machine learning to train models that can learn the locations of these sites given DNA sequences, and a trained model can be used to predict how the locations of these sites may

change if the DNA changes. In this work, we build a model from a DNA sequence to the locations of all of the markers in that sequence, and in Chapters 3 and 4, we describe machine learning methods to do so.

3

Machine Learning Background

In this chapter, we give a background of classic machine learning models for discriminative classification. This chapter will offer the tools to understand the key deep learning architectures that we discuss in Chapter 4, which will be central to our models for regulatory marker prediction.

3.1 Logistic Regression

We are working on one of a broad class of machine learning problems known as binary classification, in which we are trying to predict whether or not an input is part of a class. We first describe binary logistic regression, a widely used and effective model for discriminative probabilistic classification.

Let the output y be a binary label, meaning $y \in \{0, 1\}$, and the input $\boldsymbol{x} \in \mathbb{R}^D$. Logistic regression is a discriminative classifier, meaning that we seek to directly

model $p(y|\boldsymbol{x})$, rather than build a generative model of the data. In particular, Logistic Regression uses a linear model and sigmoid activation to represent the probabilities of the outputs. So, we let

$$p(y = 1|\boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{3.1}$$

$$p(y = 0|\boldsymbol{x}; \boldsymbol{w}) = 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) \tag{3.2}$$

where $\sigma(x)$ is the sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

which ensures that the output is a valid probability. As we mentioned earlier, this is a probabilistic classification model, and thus we can fit this via maximum likelihood estimation (MLE). In other words, we want to find the parameters $\boldsymbol{w}, w_0$ that maximize the probability of the outputs. To do this optimization, we define the loss function $L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\})$ to be the negative log-likelihood of the data, and then find the parameters that minimize that loss. For notational simplicity, let $h = \boldsymbol{w}^T\boldsymbol{x} + w_0$.

$$L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\}) = -\sum_{n=1}^{N} \ln p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) \tag{3.4}$$

$$= -\sum_{n=1}^{N} \ln \left(\sigma(h)^{y_i}(1 - \sigma(h))^{1-y_i}\right) \tag{3.5}$$

In order to find the optimal parameters, we calculate the gradient of the loss with respect to the parameters. Then, we use gradient descent (see Chapter 3.2) to update the parameters until convergence. Thus, we get that the gradient of the loss with respect to $\boldsymbol{w}$ is

$$\frac{\partial}{\partial \boldsymbol{w}} L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\}) = \sum_{i=1}^{N} y_i \left(-\boldsymbol{x}_i \frac{\exp -h}{1 + \exp(-h)} \right) + (1 - y_i) \left(\boldsymbol{x}_i \frac{\exp h}{1 + \exp(h)} \right) \tag{3.6}$$

$$= \sum_{i=1}^{N} -y_i \boldsymbol{x}_i p(y_i = 0 | \boldsymbol{x}_i) + (1 - y_i) \boldsymbol{x}_i p(y_i = 1 | \boldsymbol{x}_i) \tag{3.7}$$

Similarly, we get

$$\frac{\partial}{\partial w_0} L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\}) = \sum_{i=1}^{N} -y_i p(y_i = 0 | \boldsymbol{x}_i) + (1 - y_i) p(y_i = 1 | \boldsymbol{x}_i) \tag{3.8}$$

With these gradients, we can use gradient descent, as explained in Chapter 3.2, to optimize the parameters for the model.

3.2 Gradient Descent

Usually, the loss functions we seek to optimize will not have gradients that have closed-form solutions. As a result, we will need to use an iterative optimization algorithm to find a local minimum of our loss function. In particular, we use the fact that if a function $\boldsymbol{F}(\boldsymbol{x})$ is differentiable in a neighborhood of point $\boldsymbol{a}$, then $\boldsymbol{F}$ decreases fastest in the direction of

$$-\frac{\partial \boldsymbol{F}(\boldsymbol{x})}{\partial \boldsymbol{x}}\bigg|_{\boldsymbol{x}=\boldsymbol{a}} = -\boldsymbol{F}'(\boldsymbol{a}) \tag{3.9}$$

In other words, in the opposite direction of the gradient. Thus, to update a set of weights $\boldsymbol{w}$ of a loss function $L(\boldsymbol{w})$ with gradient $\frac{dL}{d\boldsymbol{w}}$, we generally write the update operation on the ith iteration as

$$\boldsymbol{w}^{(i)} = \boldsymbol{w}^{(i-1)} - \eta \frac{dL}{d\boldsymbol{w}} \tag{3.10}$$

where η is the learning rate hyperparameter. We then repeat this iterative process until convergence.

In Equation 3.7, we see that the gradient of the loss can be calculated by summing the contribution to the gradient with respect to each of the N points in the training data. However, doing a single pass over the entire dataset can be slow. To ameliorate this, we can use minibatch stochastic gradient descent, in which rather than summing over the dataset, we look at a minibatch of inputs, in our case ranging up to 256 inputs at a time, and calculate the gradient with respect to just that minibatch. In other words, to calculate the gradient, we sum up the contributions to the true gradient of just that minibatch of inputs.

These stochastic gradients form a estimate of the true gradient, in which we tradeoff some precision for being able to make more updates to our inputs. In practice, this is how discriminative classifiers like logistic regression and neural networks are trained efficiently (Bottou and Bousquet, 2008).

3.3 Multilabel Logistic Regression

There is one small adaptation of the binary logistic regression model that we use in our work. Binary logistic regression, as posed so far, is meant to predict a single binary label. In this work, we are generally more interested in predicting multiple binary labels, as we aim to predict whether *every* possible regulatory marker occurs at a particular location, rather than just one marker. Let there be M binary labels to predict. Thus, we make a small adaptation to the above model by replacing the vector $\boldsymbol{w} \in \mathbb{R}^D$ with a matrix $\boldsymbol{W} \in \mathbb{R}^{D \times M}$. The rest of the model works the same way, with the sigmoids applied element-wise, and the gradients being analogous to the single-label version. Note that this is different from *multiclass* logistic regression, because more than one of these binary labels can be 1 for any input. Essentially, each column in $\boldsymbol{W}$ represents an independent binary logistic regression problem, and combined it is considered to be multilabel binary logistic regression.

The loss function for multi-label logistic regression is the natural extension of

the standard logistic regression loss. In particular, for each of the M output labels, we calculate the loss we derived from maximum likelihood estimation in Equation 3.5, also known as the binary cross entropy loss, where if $x_m \in [0, 1]$ is the prediction value and $z_m \in \{0, 1\}$ is the true label

$$L(x_m, z_m) = -z_m \log(x_m) - (1 - z_m) \log(1 - x_m) \tag{3.11}$$

This is equivalent to Equation 3.5 and is minimized when $x_m = z_m$. This gives us a non-negative loss associated with each of the M predictions. To minimize the total loss over all M predictions, we define our loss to be the mean of the pointwise cross-entropy losses. Thus, we get that the overall loss is

$$L(\{x_m\}_M, \{z_m\}_M) = \frac{1}{M} \sum_{i=1}^{M} L(x_i, z_i) \tag{3.12}$$

This is the loss we will use in our modeling task.

3.4 Multilayer Perceptron

A multilayer perceptron is an extension of logistic regression that is intended for developing adaptive features by repeatedly passing inputs through linear transformations and nonlinearities. In particular, a multilayer perceptron with k hidden layers is the function

$$p(y|\boldsymbol{x}; \boldsymbol{w}, \{\boldsymbol{W}^{(i)}\}_{i=1}^{k}) = \sigma(f_k(f_{k-1}(\ldots.(f_1(\boldsymbol{x})))))\boldsymbol{w} + w_0) \tag{3.13}$$

where

$$f_i(\boldsymbol{x}) = h(\boldsymbol{x}\boldsymbol{W}^{(i)}) \tag{3.14}$$

and h is a non-linear activation function. We commonly use the ReLU activation, which is defined as

$$\text{ReLU}(x) = \max\{0, x\} \tag{3.15}$$

The final layer of a multilayer perceptron should be thought of as simply logistic regression. The k hidden layers learn features that produce linearly separable inputs in the space of the final logistic regression layer. The nonlinearities between each linear transformation allow the model to learn nonlinear decisions boundaries in the original input space. To train a multilayer perceptron, we jointly learn both the parameters to the hidden layers and to the logistic regression layer. We calculate the gradient with respect to each of the parameters $\boldsymbol{W}^{(i)}$, along with $\boldsymbol{w}$ and w_0, using the chain rule of calculus and update the parameters using gradient descent. In practice, these gradients can be efficiently calculated using backpropagation.

3.5 Backpropagation

Backpropagation is the central algorithm used to calculate the gradient with respect to each of the parameters in a neural network. In order to implement backpropagation, it is assumed that each neural network layer (f_i) is able to calculate $f_i(\boldsymbol{x})$ and the gradients $\frac{\partial}{\partial \boldsymbol{W}^{(i)}} f_i(\boldsymbol{x})$ and $\frac{\partial}{\partial \boldsymbol{x}} f_i(\boldsymbol{x})$, with the latter gradient often shortened as $f_i{}'(\boldsymbol{x})$.

Say there are two hidden layers, and we want to calculate the gradients with respect to the parameters in each of those. Then, by the chain rule, the gradients are

$$\frac{\partial}{\partial \boldsymbol{W}^{(1)}} L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\}) = \sigma'(f_2(f_1(\boldsymbol{x}))\boldsymbol{w} + w_0) \cdot \boldsymbol{w} \cdot f_2{}'(f_1(\boldsymbol{x})) \frac{\partial f_1(\boldsymbol{x})}{\partial \boldsymbol{W}^{(1)}} \tag{3.16}$$

$$\frac{\partial}{\partial \boldsymbol{W}^{(2)}} L(\boldsymbol{w}, \{\boldsymbol{x}_i\}, \{y_i\}) = \sigma'(f_2(f_1(\boldsymbol{x}))\boldsymbol{w} + w_0) \cdot \boldsymbol{w} \cdot \frac{\partial f_2(f_1(\boldsymbol{x}))}{\partial \boldsymbol{W}^{(2)}} \tag{3.17}$$

This same process extends for any number of layers and parameters using the

chain rule of calculus. However, the first two multiplications in each of these parameter calculations are shared. The backpropagation algorithm accounts for this by sequentially calculating gradients from the output nodes of the network to the input nodes. Thus, the gradients for all of the parameters can be calculated sequentially without any repeated calculations. Thus, backpropagation can be thought of as an example of dynamic programming, where calculating gradients in a particular order prevents repeated calculation. This allows for the efficient calculation of the gradients with respect to the parameters of very large models.

3.6 Dropout

Dropout (Srivastava et al., 2014) is a form of regularization that prevents overfitting by randomly zeroing a subset of units during each forward propagation of training and zeroing gradients to them during backpropagation. This is analogous to learning an exponential number of sparser networks. At test time, the dropout is set to zero and inputs are scaled, and thus each test-time forward propagation forms an approximate average over the sparse networks. We use this technique to prevent overfitting to the training data.

3.7 Batch Normalization

Addtionally, we use batch normalization (Ioffe and Szegedy, 2015) to accelerate training and add additional regularization. In deep neural networks, the distribution of the inputs of each layer changes during training, and is not consistent across layers. As of result of this, lower learning rates are required and parameter initialization can significantly affect the result. Batch normalization normalizes the input to each layer for each minibatch, allowing for the use of higher learning rates. Additionally, as described in Ioffe and Szegedy (2015), this technique serves as a regularizer, which reduces overfitting by the model. As a result, we can use higher learning rates while reducing generalization error.

4

Neural Architectures

4.1 Convolutions

In the multilayer perceptron, we generally applied a linear transformation to an entire input. This is known as a linear layer or a fully-connected layer. One drawback of this approach is it introduces a large number of parameters that need to be fitted. For example, given an input of size 25000, as is the case in Task 2 of this work, the first dimension of $\boldsymbol{W}^{(1)}$ would be 25000, and the second dimension would be the size of the hidden layer. This alone could introduce a million parameters into our model with a hidden layer of as small as 40. In practice, such a model could easily have tens of millions of parameters, making it prone to overfitting.

Furthermore, the fully-connected hidden layers learn a feature representation over the entire set of inputs. However, in a variety of tasks, including image

recognition, natural language processing, and genetic modeling, we may be first interested in uncovering some local structure in the inputs. For example, in images, we may be interested in capturing clusters of pixels that have similar patterns in different places of the image, or in genetic modeling, learning genetic motifs that may repeat themselves across the genome.

To learn these features with many fewer parameters, we use a convolution (LeCun et al., 1995). Convolutional neural networks (CNNs) apply an affine transformation $\boldsymbol{W}$ to a window of inputs. These are a shared set of weights for the layer. In this work, we are focused on sequential data which require 1-dimensional convolutions, as opposed to images which require 2-dimensional convolutions.

In a 1-dimensional convolution, the input $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ has **width** N and **depth** D. We can concretely define the 1-dimensional convolution layer with kernel width k, f output filters, and $\boldsymbol{W} \in \mathbb{R}^{kD \times f}$ as

$$Conv(\boldsymbol{X}; \boldsymbol{W}) = \begin{bmatrix} \boldsymbol{x}_{0:k}\boldsymbol{W} \\ \boldsymbol{x}_{1:k+1}\boldsymbol{W} \\ \boldsymbol{x}_{2:k+2}\boldsymbol{W} \\ \vdots \\ \boldsymbol{x}_{n-k:n}\boldsymbol{W} \end{bmatrix} \tag{4.1}$$

where $\boldsymbol{x}_{i:j}$ is the concatenation of vectors $\boldsymbol{x}_i$ through $\boldsymbol{x}_j$ in $\boldsymbol{X}$:

$$\boldsymbol{x}_{i:j} = \begin{bmatrix} \boldsymbol{x}_i & \boldsymbol{x}_{i+1} & \cdots & \boldsymbol{x}_j \end{bmatrix} \tag{4.2}$$

Note that to keep the first dimension of $Conv(\boldsymbol{X})$ the same size as the first dimension of $\boldsymbol{X}$, we pad both sides of the input with zeros such that the convolved data has the same length as the input. Several layers of convolutions with nonlinearities in between form a deep convolutional neural network. This is analogous to the formulation of a multilayer perceptron, except that the connections between layers are local along the width, rather than being fully-connected. Thus, analogous with the notation in Equations 3.13 and 3.14,

we can express a convolutional neural network as

$$p(\boldsymbol{y}|\boldsymbol{X}; \boldsymbol{w}, \{\boldsymbol{W}^{(i)}\}_{i=1}^{k}) = \sigma(f_k(f_{k-1}(....(f_1(\boldsymbol{X})))))\boldsymbol{w} + w_o) \tag{4.3}$$

where

$$f_i(\boldsymbol{X}) = h(Conv(\boldsymbol{X}, \boldsymbol{W}^{(i)})) \tag{4.4}$$

and h is a non-linear activation function, such as a ReLU.

Because of the locally connectivity in the convolutional neural network, each element of the kth convolutional layer is only influenced by a subset of the elements in the input $\boldsymbol{X}$. The **receptive field** of the pth output of f_k is the subset $\mathcal{R}$ of elements $\{\boldsymbol{x}_i\}$ of the original input $\boldsymbol{X}$ that modify its value (Yu and Koltun, 2015). This is best understood visually: the receptive field for an output node in an example three-layer CNN with kernel width 3 is visualized in Figure 4.1.1 as the nodes in bold.

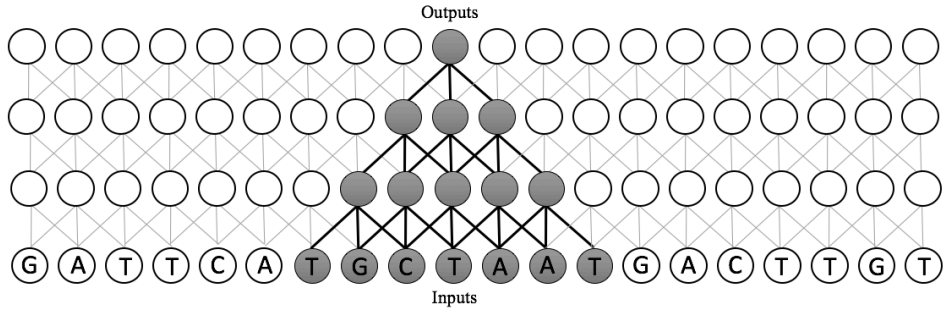


Figure 4.1.1: Here we see the receptive field of a three-layer convolutional neural network with kernel width 3. The input nodes that comprise the receptive field are the elements of the bottom layer in bold. Note that the size of the receptive field is linear in the number of layers and the kernel width.

Furthermore, note that in Equation 4.1, each of the windows were 1 input apart. This is denoted as a stride of 1, and can be tuned. In general, with a stride of s, the above convolution becomes

$$Conv(\boldsymbol{x}; \boldsymbol{W}) = \begin{bmatrix} \boldsymbol{x}_{0:k}\boldsymbol{W} \\ \boldsymbol{x}_{s:k+s}\boldsymbol{W} \\ \boldsymbol{x}_{2s:k+2s}\boldsymbol{W} \\ \vdots \\ \boldsymbol{x}_{n-k:n}\boldsymbol{W} \end{bmatrix} \tag{4.5}$$

If $k = s$, and the input width w is divisible by s, then the output's width will be $\frac{w}{s}$. As such, strided convolutions can be used for downsampling, which makes a long sequence become a shorter one for the next layer of the network. However, if the final output needs to be the same length as the original input, then there needs to also be an upsampling operation. To do this, we can use a deconvolution, or transpose convolution. A deconvolution is defined as the gradient of a convolution. For example, a deconvolution with a stride of 10 and kernel width of 10 would upsample the input by a factor of 10.

Lastly, along with convolutions, we can add max-pooling in between convolutional layers of a neural network. A max-pool is conceptually similar to a convolution, but rather than applying a linear transformation to a window of inputs, it calculates the maximum value of each input in the window along the width axis. This can be used to smooth the activations of a convolutional layer.

However, CNNs have a key problem: the size of the receptive field is linear in the number of layers and the kernel width. This means that in order to have a large receptive field for each output, there must be a proportionally large number of layers or kernel width, and thus many more parameters, which can cause overfitting. With genetic data, it may be useful to incorporate a large amount of DNA for every regulatory marker prediction, so being able to scale the receptive field is crucial. To ameliorate this issue, we discuss two alternative architectures in the next two sections which address this issue in different ways.

4.2 Recurrent Neural Networks

In order to get a larger receptive field, one way is to use a Recurrent Neural Network (RNN), which is an architecture oriented for sequential data analysis. These models have been successfully employed in a variety of sequential modeling tasks, particularly in Natural Language Processing, which we discuss in Chapter 5.

An RNN with a hidden state of size M processes a sequence of inputs $\{\boldsymbol{x}_i\}_N$, where $\boldsymbol{x}_i \in \mathbb{R}^D$, by updating an internal state vector $\boldsymbol{c}_i \in \mathbb{R}^M$ such that

$$\boldsymbol{c}_{i+1} = R(\boldsymbol{c}_i, \boldsymbol{x}_i), \text{ where } R : \mathbb{R}^{M+D} \to \mathbb{R}^M \tag{4.6}$$

This sequence of outputs $\{\boldsymbol{c}_i\}_N$ can then be used analagously to the outputs of a convolutional layer. However, we note that $\boldsymbol{c}_i$ is conditioned on $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_i$, meaning all of the inputs to the left of a particular output are part of its receptive field. This is known as a forward RNN. In a reverse RNN, we do the same process, except we first reverse the order of the input sequence. This means that in a reverse RNN, $\boldsymbol{c}_i$ is conditioned on $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_N$. When we concatenate the results of a forward RNN and a reverse RNN, we get a Bidirectional RNN, in which each output is conditioned on the entire input. This means that the receptive field for each output node contains all of the inputs, as we can see in Figure 4.2.1.

On first glance, this would seem to accomplish our goal. We are now able to condition every output on the entirety of a long input sequence, and we can do this with many fewer parameters than a fully-connected layer. However, a drawback occurs during backpropagation. Since, in the forward propagation step, we process the entire input sequentially (in each direction), during the backpropagation step, gradients are propagated back through the entire set of inputs. This is useful for learning long-term dependencies in sequential tasks. However, when the input sequences are very long, as is the case in genetic data, backpropagating through all of the inputs can lead to vanishing gradients, since

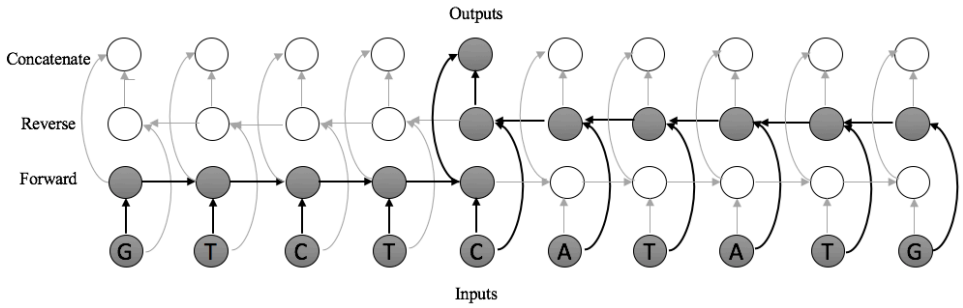


Figure 4.2.1: A Bidirectional LSTM. The first layer is the forward RNN, and the second is the reverse RNN. The outputs of these are concatenated. As a result, the receptive field of each output node consists of the entire set of inputs, which are labeled in bold. This is in contrast with the standard convolution in Figure 4.1.1, where only a small number of inputs affect each output. However, as a result of this structure, errors need to be backpropagated through $O(N)$ nodes for some of the inputs. This can lead to vanishing gradients.

the gradients are being multiplied at each step via the chain rule. To partially deal with this, we use a Long Short-Term Memory (LSTM) gate (Hochreiter and Schmidhuber, 1997), which is a particular transformation function R with which gradients can be propagated through longer sequences than other RNNs without them quickly vanishing.

However, due to the significantly longer length of the input sequences in this task compared to most NLP tasks, we may still suffer from the vanishing gradient problem due to long backpropagations, and we face computational overhead from these long sequences too.

4.3 Dilated Convolutions

So far, we see that standard convolutions offer beneficial short backpropagations but small receptive fields, while RNNs offer large receptive fields but long backpropagations that are susceptible to vanishing gradients. To address these issues, we consider dilated convolutions, which offer wide receptive fields with short backpropagations.

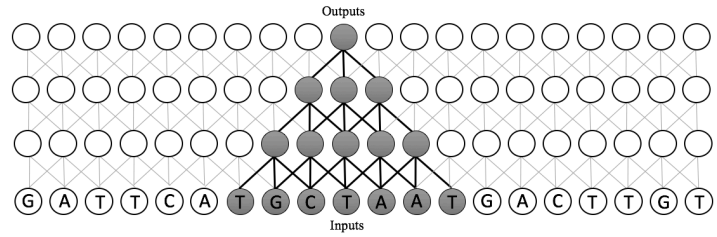
In a standard convolution, each window of elements consists of adjacent elements of the sequence. Dilation allows for control over this, by allowing for the selection of windows of elements that are not necessarily adjacent. With dilation factor $d = 1$, this is just a normal convolution as defined above. With $d > 1$, the window starting at location i of size k is

$$\begin{bmatrix} \boldsymbol{x}_i & \boldsymbol{x}_{i+d} & \boldsymbol{x}_{i+2d} & \cdots & \boldsymbol{x}_{i+(k-1)\cdot d} \end{bmatrix} \tag{4.7}$$

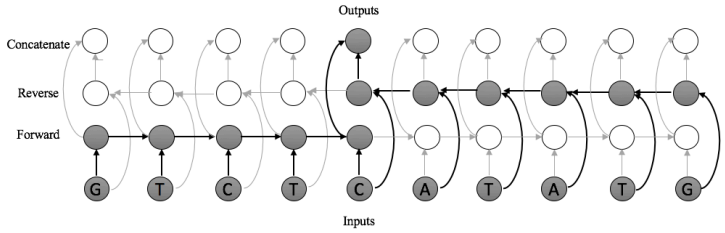
Yu and Koltun (2015) show that by stacking these convolutions with increasingly large d, one can expand the size of the receptive field of each output exponentially with respect to the number of layers. In particular, by having stacked dilated convolutions with dilation rate d doubling at each layer, the output layer has a receptive field size proportional to 2^l, where l is the number of layers. This allows them to have large receptive fields, but still short backpropagations.

To see this, consider Figure 4.3.1, which shows the three models we have discussed: convolution (4.3.1a), RNN (4.3.1b), and dilated convolution (4.3.1c). In this diagram, we can visualize the length of the backpropagation in each model by counting the number of bold lines or arrows along the path between an output node and an input. Here we can see that the standard convolution has short backpropagations to all of the inputs in the receptive field, but has a small receptive field for each output. The RNN has a large receptive field, but has long backpropagations to the inputs that are far from the output along the width. The dilated convolution combines the best of both, with short backpropagations to all of the inputs in a large receptive field.

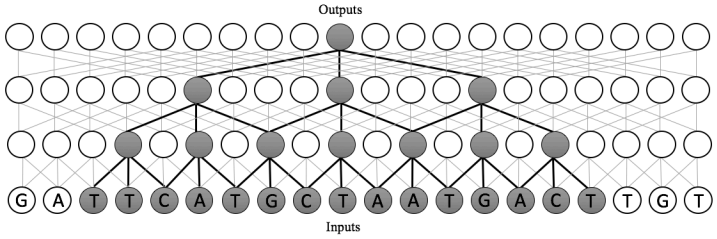
While the more localized structure of a standard CNN may be effective for tasks like image classification in which local information is most important, in genetic data, there are important long-term dependencies across the genome, some spanning several thousand or million base pairs. As a result, this dilated structure allows the model to learn dependencies across an exponentially large number of inputs for each output, while having the same number of parameters



(a) Convolution



(b) Bidirectional RNN



(c) Dilated Convolution

Figure 4.3.1: A comparison of the three models. The inputs to all of these models is a sequence of nucleotides. The standard convolution has short backpropagations to all of the inputs in the receptive field, but has a small receptive field. The RNN has a large receptive field, but has long backpropagations to much of it. The dilated convolution has both needed properties, with short backpropagations to all of the inputs in a large receptive field.

and short backpropagation as a similarly deep CNN. We take advantage of this structure when modeling genetic regulation.

5

Related Work

Convolutions have been successful in a variety of computer vision and natural language tasks. In particular, multilayer convolutions have been used for image classification, such as on the ImageNet task (Krizhevsky et al., 2012). Networks of this form have been applied to a other tasks too, including video classification (Karpathy et al., 2014) and object detection (Girshick et al., 2014). Furthermore, Yu and Koltun (2015) show that using dilated convolutions, the network can learn multi-scale context, allowing it make pixel-level predictions for image segmentation with both local and global information.

In natural language, early work by Bengio et al. (2003) established the basis for using neural networks for language prediction, and Collobert and Weston (2008) showed how a unified convolutional neural network architecture could be used for a variety of language tasks, including part of speech tagging (POS), semantic role prediction, and named-entity recognition (NER). These techniques

demonstrated that convolutions could work to jointly learn hierarchical features and make predictions in sequential tasks. More recently, recurrent neural network architectures have become popular for several sequential analysis tasks, especially those based on the LSTM gate (Hochreiter, 1998). For example, Huang et al. (2015) use a Bidirectional-LSTM model with a Conditional Random Field (CRF) layer to achieve state of the art accuracy on the NER task, and Klein et al. (2017) use LSTMs for machine translation.

We seek to apply the techniques that have been effective in the domains of computer vision and natural language processing to a novel dense prediction task in biology. To do this, we can treat genetic data in which we are making a prediction at every nucleotide location as a subclass of any sequential tagging problem from computer vision or natural language processing. This is analogous to assigning a label at every pixel in an image or a class to every word or character in a corpus. However, there are several reasons why the techniques from these fields do not directly transfer. For one, biological sequences are often longer than analogous images or sentences. This makes RNN architectures less effective or slow to train. Moreover, there are are complex long-term dependencies, which make traditional convolutional architectures less effective as well, which are better-suited for capturing local structure. We build on work applying dilated convolutions to images (Yu and Koltun, 2015) and natural language (Strubell et al., 2017) by showing their efficacy in modeling regulatory markers.

There have been several approaches to modeling regulatory markers. While deep learning allows researchers jointly learn both a feature representation from the raw nucleotide inputs and a classifier on those features, early methods for DNA regulation analysis separated these two approaches. In particular, a common early approach was to use position-weight matrices (PWM) for the feature representation (Stormo, 2000; Kel et al., 2003). These matrices reflected the frequencies of nucleotides in a region of interest, such as binding sites, and performing an element-wise multiplication of the PWM with a one-hot encoded input sequence gave a score that could be used for classification with techniques like logistic regression.

One of the early approaches that involved deep learning was a single-layer CNN which conducted binary classification on a DNA sequence represented in one-hot format (Alipanahi et al., 2015). This model was not used for the task we work on, but instead to predict the binding affinity of a DNA sequence, which is useful for studying regulation. However, it was one of the first works to use deep learning to make predictions about gene regulation.

Zhou and Troyanskaya (2015) introduce a novel dataset and method for regulatory marker prediction, and we reimplement and extend their results in Task 1 of this work. They focus on the classification of short sequences of DNA. In particular, they were interested in training a model that could predict, given a short sequence of DNA, whether each of a set of regulatory markers was present anywhere in that region. Their model uses a simple 3 layer convolution on this task, and we discuss this in more depth in Chapter 7.1. A more recent approach by Quang and Xie (2016) used a Bidirectional LSTM atop a single-layer CNN, and achieved improvement above the purely convolutional model.

6

Approach

The existing work presents an issue. These methods are only incorporating a small number of nucleotides as input to a neural network, meaning they do not allow the classifier to incorporate signal from regions of the genome that are far from the prediction location in terms of number of nucleotides, but that may be physically close in 3-D space. This is problematic, since incorporating signal from these farther regions may be useful in being able to better model what is occurring at particular genomic location. In other words, we would like to increase the size of the receptive field of the architecture.

In this work, we will reimplement and extend methods for regulatory modeling using the existing labeled dataset that was developed by Zhou and Troyanskaya (2015). Then, we develop a new dataset using the same sources as Zhou and Troyanskaya (2015) that allows for the modeling of much longer-term dependencies in the input space, and nucleotide-level resolution in the outputs.

In general, the models trained on these datasets will involve taking, as input, a DNA sequence represented as a vector of nucleotides, and outputting binary predictions about the presence of regulatory markers in that sequence. For each of these datasets, we compare the performance of models based on dilated convolutions with those based on LSTMs and standard convolutions, as well as simpler baseline models like multilayer percetrons. The goal is to determine how these architectures perform with both small and large amounts of context.

We split this investigation into two tasks. In Task 1, we determine whether dilated convolutions can match the performance of the state of the art LSTM-based architecture on an existing regulatory marker prediction task. In this task, we are given a short sequence of $N = 1000$ nucleotides, and are simply trying to predict whether each of $K = 919$ regulatory markers is present anywhere in that sequence. This task investigates whether dilated convolutions can capture the long-term dependencies that make LSTMs effective on this short-sequence prediction task. It also offers intuitions about the types of regulatory markers that dilated convolutions are effective at predicting, and about the scaling properties of each of the architectures. We formally define this task and discuss our findings in Chapter 7.

In Task 2, we use what we learned from Task 1 to train models on our novel dataset. In particular, this dataset has much larger context sizes, with $N = 25000$ for each prediction. So, this task investigates whether dilated convolutions can take advantage of this additional available context to make better predictions. We measure how the performance of dilated convolutional models compares to LSTMs, which have similarly large receptive fields but suffer from longer backpropagations, and standard convolutions, which have much smaller receptive fields, but can learn the dependencies in that smaller field effectively because of their smaller backpropagations. Fundamentally, this task studies whether increased context size improves the predictive success of this model, and then compares the three ways of incorporating more context: more convolutional layers, using an LSTM, or using dilated convolutions. We formally define this task and discuss our findings in Chapter 8.

In both of these tasks, the architectures output a vector of probabilities, each one corresponding to the probability of the regulatory marker being present at the given location. Given these outputs, we use ROC AUC and PR AUC scores to measure the predictive accuracy of the models. These refer to the area under the Receiver Operating Characteristic curve and the Precision-Recall curve. Zhou and Troyanskaya (2015) and Alipanahi et al. (2015) use ROC AUC as their metric, so we report that score for consistency and reproducibility. However, in cases with highly imbalanced class distributions, PR AUC gives a more meaningful distinction between models, as was suggested by Quang and Xie (2016), so we report that score and use it for our analysis. Furthermore, we break down the relative success of these models at predicting the three types of regulatory markers: transcription factor binding sites (TFBSs), histone modifications, and DNAse hypersensitivity sites. We report the mean AUC scores for each of these subsets. For additional information on these metrics, in Appendix B, we formally define ROC AUC and PR AUC, explain the primary differences between the two, and use that to elucidate why the PR AUC statistic is more meaningful for this task.

7

Task 1: Match LSTM Performance on a Prediction Task with Small Context

7.1 Description and Dataset

In this task, we train a model that can predict the presence of regulatory markers in a DNA sequence given a relatively small number of input nucleotides, meaning a small context size. First, we give a high-level overview of the dataset that Zhou and Troyanskaya (2015) developed, which has since been used for regulatory modeling and which we use for this task. For full details, please refer to the original work directly.

Each element of an input sequence comes from a vocabulary $\mathcal{V} = \{A, C, T, G\}$, which refers to the four nucleotides that comprise DNA. An input is a vector $\boldsymbol{x} \in \mathcal{V}^d$, where d is the length of the sequence. In this task, the

authors divide up the genome into non-overlapping 200 base-pair (bp) regions of DNA. Each of these 200bp regions then has 400bp of the adjacent DNA appended on each side, so that each input is a window of length $d = 1000$. For every input $\boldsymbol{x}$, there is corresponding binary output vector $\boldsymbol{y}$, which indicates whether each of the $k = 919$ regulatory markers is present anywhere in the middle 200bp of $\boldsymbol{x}$. These binary markers are not mutually exclusive. The dataset is thus n pairs of input and output sequences $\{(\boldsymbol{x}, \boldsymbol{y})\}_n$

$$\boldsymbol{x} \in \mathcal{V}^d \tag{7.1}$$

$$\boldsymbol{y} \in \{0, 1\}^k \tag{7.2}$$

where $n = 4400000$. The task is to thus to model $p(\boldsymbol{y}|\boldsymbol{x})$. To get this data, the authors use publicly available regulatory marker data from ENCODE (Consortium et al., 2012), which contains the location of each of the k regulatory markers on the human genome, including the locations of transcription factor binding sites, histone modifications, and DNAse hypersensitivity sites. Note that the task is not to predict *where* in the given DNA sequence the regulatory markers are present - only to predict *whether* they are present anywhere in it.

7.2 Models

We summarize the high-level architecture that we use for this task in Figure 7.2.1 with an example with two convolutional layers, and summarize the models we implement for this task in Table 7.2.1. All of the models have the same input representation, and have fully-connected layers at the end. The differences between the models exists between the Input and Flatten layers of Figure 7.2.1, as each model uses a different type of convolutional layer and one uses a Bidirectional LSTM. Note that in the logistic regression and multilayer perceptron baselines, we directly flatten the inputs and use fully-connected layers, rather than first applying convolutions. We describe the models in Table 7.2.1.

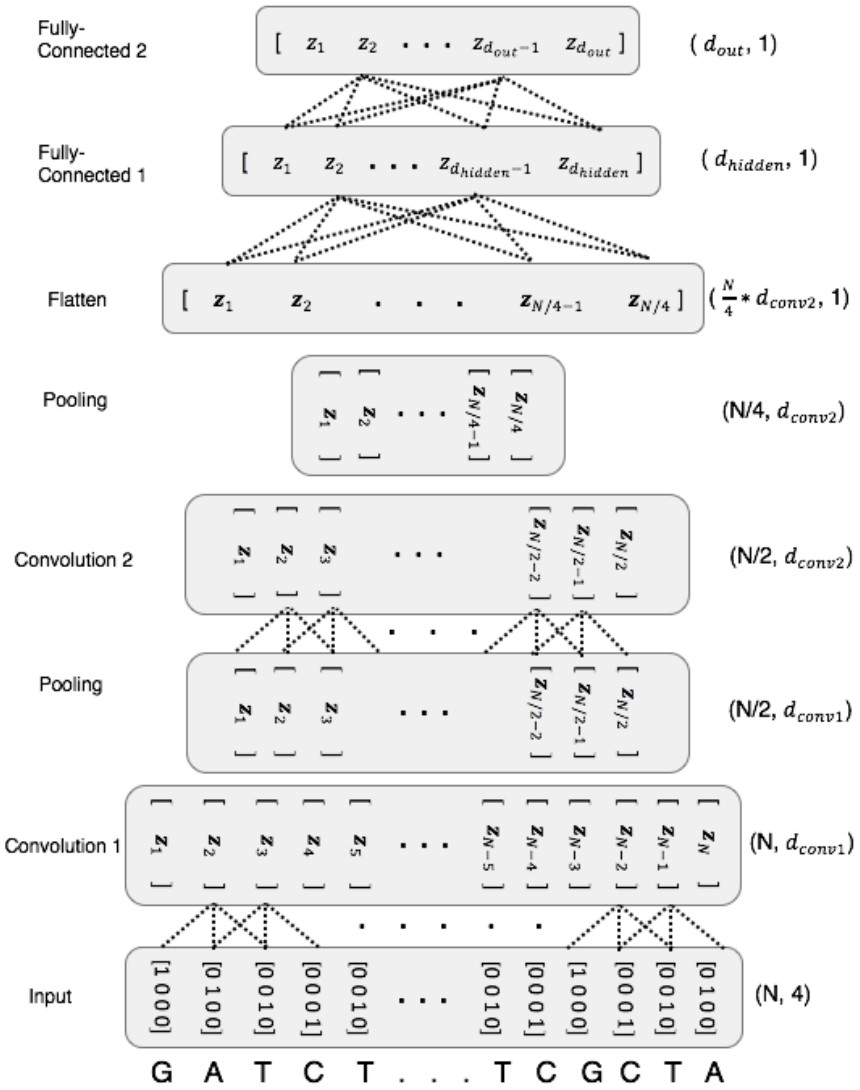


Figure 7.2.1: An overview of the architecture we use in Task 1. The input (bottom) is a sequence of N nucleotides of DNA, represented as one-hot encoded vectors. The shape of the tensor at each layer is listed on the right. This example shows 2 convolutions, each followed by a pooling layer, and finally two fully-connected layers. Note that there are more convolutions and pooling in some of our models, and we also use dilated convolutions, as explained in Chapter 4.3. Additionally, we use a Bidirectional LSTM in between the Input and Flatten layers for one of the baseline models. The fully connected layers apply to the flattened input. This means that there are $d_{out} = 919$ outputs for each sequence of $N = 1000$ inputs.

Model	Layers	Layer Type	Parameters
Baseline: Logistic Regression	0	N/A	3,676,919
Baseline: MLP	1	Fully-Connected	4,551,919
Baseline: Conv3	3	Convolution	60,472,479
Baseline: LSTM	2	LSTM	46,926,479
Dilated3	3	Dilated Convolution	22,284,639
Dilated6	6	Dilated Convolution	8,043,759
Deep-Dilated	15	Dilated Convolution	2,073,943

Table 7.2.1: Models for Task 1. Note that the first four models are the baseline models that we compare our results to, with Baseline:Conv3 being the model from Zhou and Troyanskaya (2015), and Baseline:LSTM the model from Quang and Xie (2016). Note that all of our new models, despite having more layers, use fewer parameters than existing models, since they use fewer filters per convolutional layer. To see a more thorough description of each model, including the dilation rates for each layer, see Appendix E.

7.3 Hyperparameters

We modified several sets of hyperparameters for each of the above models. For the above architectures, we tested the following configurations:

Hyperparameter	Options
Learning Rate	Between .0001 and .1
Batch Norm Decay Rate	$\{.9, .99\}$
Dropout	$\{.1, .01\}$

Table 7.3.1: Hyperparameters tested in experiment 1.

Note that the dropout was between every layer, and could be low since we also used batch normalization as a regularizer.

7.4 Results

We report the mean ROC and PR AUC scores for each of the categories of predictions in Table 7.4.1.

Model	ROC AUC			PR AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Baseline: LR	0.716	0.670	0.674	0.042	0.143	0.097
Baseline: FF	0.737	0.693	0.690	0.046	0.181	0.106
Baseline: Conv3	0.869	0.782	0.860	0.205	0.273	0.319
Baseline: LSTM	0.935	0.833	0.904	**0.305**	**0.340**	**0.407**
Dilated3	0.864	0.779	0.852	0.190	0.271	0.299
Dilated6	0.923	0.813	0.895	**0.273**	**0.320**	**0.390**
Deep-Dilated	0.881	0.771	0.873	0.233	0.271	0.350

Table 7.4.1: Results of baseline models and dilated convolution models trained on the dataset from Zhou and Troyanskaya (2015). We show that the dilated convolution models allow for significant improvements in both the ROC AUC and PR AUC scores relative to Conv3 from Zhou and Troyanskaya (2015). Note that all of these data are from reimplementations of the existing and new models in order to have a fair comparison between the results. Here, we note that the Dilated6 model performs better than the standard convolutions on both metrics on all three types of predictions, and similar to the LSTM-based model, though not quite as well. The results on the validation set are in Appendix C and are similar.

Here, we see several important results. First, we note that the dilated convolutional models are able to perform better with both the ROC AUC and PR AUC metrics than both the logistic regression baseline and the Conv3 model from Zhou and Troyanskaya (2015). This suggests that dilated convolutions can incorporate more information than the shallower convolutional models. Moreover, note that the dilated convolutional models had significantly fewer parameters than the Conv3 model, with Dilated6 using 8,043,759 parameters and Conv3 using 60,472,479. Thus, we see that the dilated convolutional model is able to capture the underlying features of the input better than the Conv3 model,

and can do so with an order of magnitude fewer parameters. This is important, as having fewer parameters increases training speed and reduces the amount of overfitting that the model may be susceptible to.

Additionally, Dilated6 is able to achieve close to the performance of the LSTM-based model. However, as noted before, Dilated6 has substantially fewer parameters than the LSTM-based model. This suggests that the dilated convolution is able to similarly model the long-term dependencies that the LSTM is able to capture, despite having many fewer parameters. Because the sequences that the LSTM operates over are relatively short in this task, it makes sense that the LSTM would be the most effective model.

This suggests that dilated convolutions are able to capture meaningful properties of genetic data with few parameters. With this proof of concept, we next test whether the dilated convolutions are able to perform effectively on a more complex task, with both a longer set of inputs (25,000 vs 1,000), and with more outputs (making one set of 919 predictions per nucleotide, rather than per sequence of nucleotides). In particular, since dilated convolutions scale much better in terms of backpropagation distances than LSTMs, we will test to see whether dilated convolutions are able to keep their high level of performance relative to the other models.

8

Task 2: Scale Models to Incorporate More Context

8.1 Description and Dataset

In this section, we describe the process of developing our novel dataset, which allows for the modeling of regulatory markers at nucleotide scale with larger contexts. In particular, we introduce a dataset with two major differences from the previously analyzed data: (1) the size of the input sequences is much larger, with each a $d = 25000$ bp sequence, and (2) the outputs annotate the sequence at nucleotide-resolution. The task is to predict the presence of all regulatory markers at each nucleotide, rather than per 200bp as in Task 1, thus annotating the locations of regulatory markers along the sequence at a finer resolution. Also, we choose not to represent our inputs in one-hot format, and instead as unique

integers to be used as indices in order to train embeddings, or vector-representations, for each of the nucleotides. To construct this dataset, we use the hg19 reference genome to extract sequences, just like Zhou and Troyanskaya (2015), and $k = 919$ regulatory markers from the same ENCODE database. Thus we get a set of n pairs of input and output sequences $\{(\boldsymbol{x}, \boldsymbol{y})\}$

$$\boldsymbol{x} \in \mathcal{V}^d \tag{8.1}$$

$$\boldsymbol{y} \in \{0, 1\}^{d \times k} \tag{8.2}$$

Note that the outputs $\boldsymbol{y}$ have a different shape than in Task 1, because of the nucleotide-resolution we use.

Given these changes to the dataset, we have a few different processing steps. First, our raw data consists of a genome with vocabulary $\mathcal{V} = \{N, A, C, T, G\}$, where the N character represents nucleotides for which the uncertainty in the sequencing process was too high to have a definitive tag. Zhou and Troyanskaya (2015) screen out sequences with that tag in the dataset we used in Task 1. Because we seek to model long-term dependencies, we find it acceptable to have a small number of unknown nucleotides in a long input sequence. To make sure that we do not train or test on sequences with a high percentage of unknown elements, we only keep sequences with at most 10% of the nucleotides as the letter N. Furthermore, there are regions of the DNA where a read from a sequencer would align to several genomic positions, which can impact the accuracy of the training data. As such, we wish to exclude data from these multi-mapped regions, so we also remove sequences with more than 10% of the sequence labeled as part of a multi-mapped sequence. After doing this, there were $n = 93880$ sequences that were $d = 25000$ in length, comprising approximately 2.3 billion nucleotides. These sequences are non-overlapping.

Of the 919 regulatory markers, 690 are transcription factor binding sites (TFBSs), 104 are histone modifications, and 125 are DNAse hypersensitivity sites. All of these data are publicly available through ENCODE (Consortium et al., 2012) in narrowPeak format, and we align the peaks to the human genome.

These are the same factors used by Zhou and Troyanskaya (2015) in the dataset from Task 1.

8.2 Models

We summarize the models we implement for this task in Table 8.2.1, and summarize the high-level architecture of this task in Figure 8.2.1. The differences between the models we implement for this task exists between the Embedding and first linear (fully-connected) layer of Figure 8.2.1. In contrast with Task 1, we are now making dense predictions, meaning one set of predictions for every nucleotide in the input sequence, which leads to a different model structure. This is because in Task 1, $\boldsymbol{y} \in \{0, 1\}^k$, but in this task, $\boldsymbol{y} \in \{0, 1\}^{d \times k}$.

Model	Layers	Conv Layer Type	Parameters
Conv1	1	Convolution	136,935
Conv3	3	Convolution	554,675
Conv7	7	Convolution	654,875
Dilated	6	Dilated Convolution	629,825
Bi-LSTM	4	Convolution, LSTM	397,155
ID-CNN	15	Iterated Dilated Convolution	632,487

Table 8.2.1: Models for Task 2. To see a more thorough description of each model, including the dilation rates for each layer, see Appendix E.

Each of these models varies in the amount and type of convolutional layers it has. After the convolutional layers, these models all have two fully-connected layers that are applied element-wise, meaning they are fully connected across the activations of each individual nucleotide. This is visualized in Figure 8.2.1. In models that have a strided first convolutional layer, we use a deconvolution as the final convolutional layer. For the LSTM-based model, we first have one strided convolutional layer, then the LSTM layers, and then the deconvolution. We

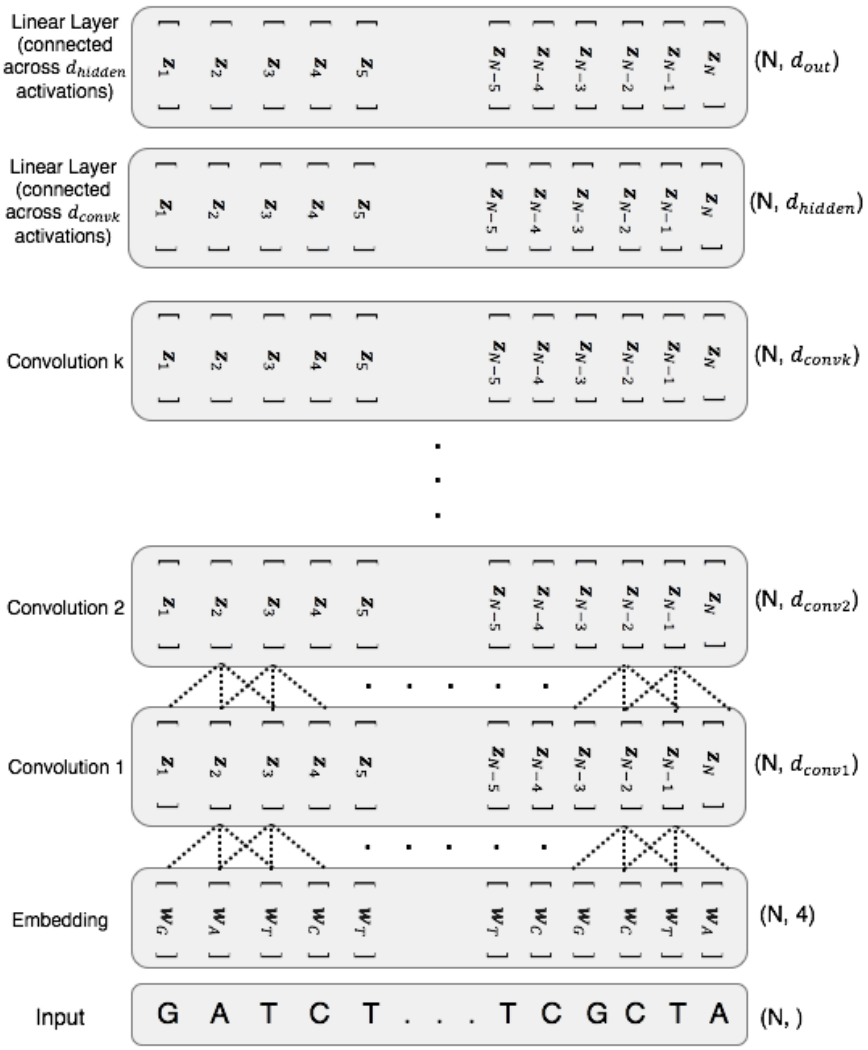


Figure 8.2.1: An overview of the convolutional architecture of Task 2. The input (bottom) is a sequence of N nucleotides. The shape of the tensor at each layer is listed on the right. The first layer embeds each element into a higher-dimensional vector space. After that, there are k convolutions (or dilated convolutions) with ReLU activations. Furthermore, pooling layers with a stride of 1 can be added in between convolutional layers. In the LSTM model, the convolutions are replaced with a bidirectional LSTM. Finally, the last two linear layers do not connect elements that are horizontally separated in the above diagram, but only those within the activations for a particular element. So, they apply a linear map to each of the activations $\mathbf{z}_i \in \mathbb{R}^{d_{convk}}$. In other words, the first linear layer has weights which are $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d_{convk} \times d_{hidden}}$, and the second has weights which are $\boldsymbol{W}^{(2)} \in \mathbb{R}^{d_{hidden} \times d_{out}}$. In practice, $N = 25000$, and $d_{out} = 919$.

summarize the models in Table 8.2.1, and give the full set of details including the strides, dilations, filters in Appendix E.

8.3 Hyperparameters

We modified several sets of hyperparameters for each of the above models and list them in Table 8.3.1. For the convolutions, we adjusted the number of filters in each convolutional layer, represented by the variables $\{f_1, f_2, f_3\}$. f_1 and f_2 are the number of filters in the first and second convolutional layers, respectively. f_3 is the number of filters in the third and beyond layers of the model, as the Conv7 and Dilated models both had more than 3 convolutions. For the LSTM, f_1 referred to the number of filters in the strided convolution, f_2 referred to the state size of each LSTM, and f_3 referred to the number of filters in the deconvolution. Additionally, for the LSTM, we adjusted the stride for the first convolution and deconvolution as a hyperparameter. Each model had one hidden layer after the convolutions, with d_{hidden} units. Note that the dropout was between every layer, and could be low since we also used batch normalization as a regularizer. We break down the hyperparameters we test for the convolution-based models and for the LSTM model separately.

Hyperparameter	Convolution Options	LSTM Options
Learning Rate	Between .001 and .1	Between .001 and .1
Batch Norm Decay	{.9, .99}	{.9, .99}
Dropout	{.1, .01}	{.1, .01}
f_1	{ 64, 128 }	128
f_2	{120, 240}	{20, 40, 80}
f_3	{50, 128}	{50, 128}
Stride	10	{10, 20, 50, 100}
d_{hidden}	{125, 200}	{64, 125, 200}

Table 8.3.1: Hyperparameters tested in experiment 2.

8.4 Results

For the second, larger dataset, we again measure the ROC AUC and PR AUC scores for each of the above models. We report the PR AUC scores in Table 8.4.1, and the ROC AUC scores in Appendix C.

Model	Validation PR AUC			Test PR AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Conv1	0.013	0.053	0.035	-	-	-
Conv3	0.059	0.115	0.100	-	-	-
Conv7	0.167	0.154	0.160	0.167	0.152	**0.171**
Dilated	0.274	0.279	0.167	**0.274**	**0.270**	0.163
Bi-LSTM	0.065	0.238	0.089	0.070	0.216	0.086
ID-CNN	0.166	0.247	0.147	-	-	-

Table 8.4.1: Validation and Test set Precision-Recall Area Under Curve (PR AUC) scores for the models. Note that these are the best validation results across all hyperparameter configurations. For the test set scores, in this table we only report the scores for the best performing standard convolution, dilated convolution, and LSTM models, which we wish to compare to one another. The full results are in Appendix C with both ROC and PR AUC Scores. Here we note that we see substantially higher performance using dilated convolutions on predicting transcription factor binding sites and histone modifictions. However, we see no improvement using dilated convolutions on predicting DNAse hypersensitivity sites.

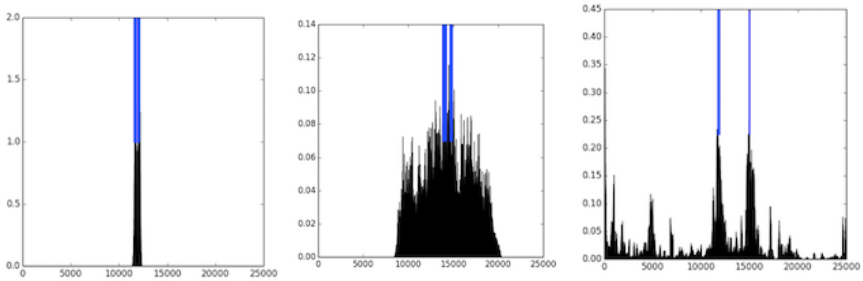
We report models that performed best for each architecture across all of the potential hyperparameters. The scores show that dilated convolutional models get the best results on both TFBS and histone modification prediction, and do only marginally worse than the best non-dilated models on predicting DNAse hypersensitivity sites. While the LSTM-based model performed better than some of the shallower models like Conv3, it did worse than the deep convolutions and dilated convolutions across the board. This is consistent with our expectations about this model not being able to scale as well as the dilated convolutions due to vanishing gradients with long sequences.

There are two major points to focus on here. First, even though the LSTM models were successful in Task 1, where the sequence lengths were shorter, they were not able to scale to larger context sizes. Second, the models that incorporated very small receptive fields, such as Conv1 and Conv3, performed much worse than ones with larger receptive fields. This suggests that future regulatory prediction work should consider datasets that incorporate wider contexts than existing work. Also, we see major differences in the performance on the different regulatory markers, with dilated convolutions helping substantially for TFBS and histone modification prediction, but Conv7 actually marginally outperforming it on DNAse hypersensitivity site prediction. This suggests that increasing the amount of context has different levels of value for the different markers, with it being particularly useful for TFBS and histone modification prediction, but not helpful for DNAse hypersensitivity site prediction.

8.5 Visualization

We also performed experiments to show whether the various models were picking up on features that were from sites far from the targets. The fundamental hypothesis behind using dilated convolutions was that they would be able to capture genetic motifs in locations that are relatively far from the actual prediction locations while being less susceptible to overfitting than deep CNNs. Thus, we now sought to visualize the model that was learned, and understand what inputs it was capturing in making predictions.

To do this, we use backpropagation to generate visualizations of the receptive fields. In particular, we randomly sample an input sequence from the validation set, and backpropagate an error of 1 from every positive output in that sequence for a randomly selected regulatory marker. In Figure 8.5.1, we then plot both the locations of the outputs (in blue), and the norm of the error backpropagated to the inputs (in black), which shows us the impact that the various parts of the input had on making the output predictions. We have additional plots in



(a) Conv7 **(b)** Dilated **(c)** Bi-LSTM

Figure 8.5.1: To generate each of these figures, we backpropagate an error of 1 from each of the outputs where a randomly-selected regulatory marker is present. The location of the regulatory marker is labeled by the blue bars in the top half of each of these graphs. We then visualize the norm of the gradient with respect to the inputs at every location in the input, and that norm is visualized by the black bar graph in the bottom half of each subplot. This black bar graph gives an indication about the actual receptive field that was used to make a decision at the outputs. Figure 8.5.1a shows the norm of the gradient with respect to the inputs of the best Conv7 model. Figures 8.5.1b and 8.5.1c do the same for the Dilated and Bi-LSTM models respectively.

Appendix D.

As expected, the standard convolutions have a narrow receptive field, while the dilated convolutional models have errors that propagate across the input more widely. Moreover, looking specifically at Figure 8.5.1b, we can see that the Dilated model is using a significant amount of the input far from the location of the regulatory marker (which is labeled in blue). We can see this because the norm of the gradient is very high for a large amount of the input in that figure. That is in contrast to the LSTM-based model in Figure 8.5.1c, where we can see that there is a small amount of gradient backpropagated to much of the input, but the norm of that gradient is very low for most of the input space.

In other words, it does not appear that the LSTM models are able to learn the long-term dependencies that the Dilated model is picking up on more significantly. However, we do note that there are smaller bumps in the LSTM image at several locations that are far from the output locations, which are not

present in the Conv7 model in Figure 8.5.1a, or in the shallower convolutional models in Appendix D. As such, it is possible that the LSTM is actually modeling some aspects of the inputs that are farther away, but isn't able to capture enough of that faraway information to do as well as the dilated convolution models.

9 Discussion

From these results, we can draw several conclusions. First, on Task 1, where there are smaller context sizes, dilated convolutions present an improvement over the standard convolutional models, but perform similarly to LSTM-based techniques. This suggests that when sequence lengths are relatively small, the LSTM-based models are still able to backpropagate effectively enough to not have issues with vanishing gradients. However, the improvement of the dilated convolution model over the standard convolutional models suggests that the ability to model wider contexts through convolutions is important to the prediction task. Furthermore, we draw attention to the fact that our dilated convolutional models have approximately an order of magnitude fewer parameters than the existing convolutional or LSTM-based models, suggesting that the dilated structure is able to account for the reduced complexity of the model. Note that in this task, there are fully-connected layers as the final two

layers of the neural network for every model. As a result, the receptive field of every output is the entire input sequence (unlike in Task 2). The difference between the models is thus not the size of the receptive field, but the degree to which the different layers before the fully-connected ones are able to capture meaningful relationships between values in the input, such that the final binary logistic regression layer performs well. We thus find that using dilated convolutions matched or improved this modeling ability with many fewer parameters relative to standard convolutions, and performed similarly to LSTMs. This suggests that they would scale well to larger context sizes, which we tested in Task 2.

In Task 2, our task was one of dense predictions, where we were trying to predict the precise locations of the markers in the input sequence, rather than whether the marker was simply present anywhere in the input, as in Task 1. We also used much larger input sequences. For Task 2, we see that dilated convolutions appear to work substantially better at predicting histone modifications and transcription factor binding sites than all other models, and only marginally worse on predicting DNA accessibility. Interestingly, the best model for DNA accessibility was not the LSTM-based model, but instead the standard deep convolutional model, suggesting that large context may not be necessary for predicting DNA accessibility. Moreover, the dilated convolutional models work substantially better than the LSTM-based model, likely because the much larger sequence length means that the LSTM is much more susceptible to vanishing gradients during backpropagation. This is despite the fact that the LSTM models also have large receptive fields, like the dilated convolution. This suggests that the multi-scale context aggregation that stacked dilated convolutions offer allows the network to learn longer-term dependencies in the network without having the vanishing gradient problem that LSTMs and other recurrent neural networks have. It also suggests that past work that focused on short sequences for inputs, as in Task 1, is not capturing sufficient information from the underlying DNA that is available to understand the locations of the various regulatory markers.

Furthermore, our visualizations in Chapter 8.5 for Task 2 correspond with the performance metrics we measured. In particular, we find that the standard convolutions have a very narrow receptive field, with a relatively small number of inputs being involved for the prediction of every output. On the other hand, the dilated convolutional models are able to scale the size of the receptive field exponentially with the number of layers, and can thus capture complex dependencies across the wide inputs. In contrast, the LSTM models pick up non-zero but limited signal from distal regions of the input sequence, which is attributable to the inability to backpropagate errors across long distances, and thus the inability to learn relationships between faraway inputs. In other words, while LSTMs do technically have large receptive fields, they appear to be less successful at actually getting signal from far away, because of the difficulty of actually learning patterns through that model when the input sequences are long.

With this work, we thus show that dilated convolutions present an improvement over state of the art techniques for modeling the locations of TFBSs and histone modifications. This work shows that the ability of dilated convolutions to incorporate larger input contexts for every prediction without dramatically growing the number of parameters in the model allows them to be useful for the dense prediction of marker locations.

One of the notable observations we had was that for DNAse hypersensitivity sites, there was a small drop in the predictive accuracy of the dilated convolutional model compared to the standard convolutional one. This may arise from some of the underlying biological properties of this data. For example, the area immediately surrounding histones is tightly coiled and thus likely does not contain accessible regions of DNA. As such, it makes sense that being able to look several thousand base pairs away, which dilated convolutions enable, allows for an improvement in predicting histone modifications since we may be able to find patterns that match regulatory regions there. On the other hand, DNAse hypersensitivity sites are inherently areas that are accessible, meaning they likely have recurring motifs such as binding sites or coding regions in that accessible area. This may mean that the relatively small context that standard convolutional

neural networks capture is sufficient for making the predictions, and hence why the dilated convolution does not help.

This suggests that using dilated convolutions, we are able to advance the state of the art in modeling the regulatory code of human DNA, particularly in the prediction of the locations of transcription factor binding sites and histone modifications. This improved accuracy is important for understanding the nature of the relationship between the genome itself and the regulatory factors that interact with it. In particular, armed with this more accurate model of the mapping from DNA sequence to the regulatory factors that interact with it, we can build useful predictive models. For example, we can study how changes to DNA can impact regulation, and seek patterns in the genome that are associated with regulation. We discuss these possibilities in Chapter 10.

10

Conclusion and Future Directions

As a whole, this work shows that dilated convolutions can be used to effectively model the locations of regulatory factors in DNA. This model offers substantial improvements over both standard convolutions and LSTMs, particularly in predicting the locations of histone modifications and transcription factor binding sites. This work establishes a proof of concept on an existing prediction task, and introduces a novel dataset and task that allows for the further exploration of large receptive fields. With this work, we hope to extend the understanding of the complex regulatory machine that enables gene expression, enabling more precise study of the regulatory code, and thus a better set of tools to understand cellular function and disease. With that said, there are several future directions to explore in this work, and we conclude with a summary of them.

10.1 Further Dataset Development

There are a few opportunities for further dataset development in this task. In the dataset that we release with this work, we analyze a set of 919 regulatory markers that are available via ENCODE. Furthermore, we process this data in several ways, by removing sequences that have a large number of unknown nucleotides, and removing multimapped regions, where the alignment of parallel sequencing methods may have produced faulty results. In the future, we may look to augment this dataset with more regulatory markers, information from newer assemblies of the human genome (notably hg38 as opposed to hg19), and the genomes of species other than humans. While we are often interesting in human genomes for the purposes of disease modeling, many of these genetic effects should be conserved across species, and training a model that jointly learns this behavior for several species could produce improved results.

10.2 Mutation Impact Analysis

Additionally, with a model that is trained to map regions of DNA to the locations of regulatory markers in that region, we can begin to study the impact of mutations. In particular, it would be useful to determine whether small mutations, like changes in single nucleotides called "single nucleotide polymorphisms" (SNPs), can produce changes in the predicted locations of regulatory markers. Then, we can generate random SNPs in an existing region of interest, and determine what impact, if any, each of these have on the predictions. A SNP that does have an impact may be part of a genetic motif, or sequence, that is meaningful for understanding gene regulation. As a result, we can use a trained model to start to make predictions about how changes to DNA can impact regulatory events, such as transcription factor binding. This can serve as a useful tool for geneticists that may be looking for promising candidates to test in a physical experiment.

10.3 Incorporating 3-D Conformational Information

One of the central issues that this work attempts to address is the fact that elements that are distal in the 1-D representation of a DNA strand may be physically close in its 3-D conformation. As a result, modeling longer-term dependencies between regions of DNA is important to being able to build predictive models for the regulatory code. However, there are new experimental methods that have sought to build a better picture of the interactions between regions of DNA. One particularly promising technology is called Hi-C (Belton et al., 2012). This technique uses high-throughput sequencing to measure the pairwise distances between all regions in the genome. In short, the technique involves cutting DNA into fragments, and then ligating the fragments. In the ligation step, fragments that are close to one another will be more likely to ligate together, and then the resulting ligated fragments can be sequenced and aligned to a reference genome. Ultimately, this technique gives a picture of the relative distances between regions of the genome, and thus this additional data could be used to augment the existing dataset. For example, having an additional feature that measures the distance between the ends of an input sequence can give an indication as to how coiled DNA may be in a particular region, and thus improve the model's ability to predict markers such as DNA accessibility.

10.4 Incorporating TFBS Sequential Information

In this work, we were focused on learning the locations of a particular set of regulatory markers. The bottom layers of our models jointly learned parameters that were shared across all of the markers, and the final layer was always a simple logistic regression, where we were training one logistic regression per marker. As a result, though the bottom layers of our models did learn shared properties of regulatory markers as a whole, our model cannot generalize as-is to new regulatory markers.

For example, our data included 690 transcription factors. If a researcher

wished to use our trained model to get insight about a new transcription factor that was not in our training set, however, she would not be able to do so without first running the full ChIP-seq experiment to target that new transcription factor and generating a genome worth of data like that which we downloaded from ENCODE, and then retraining the model. As such, another potential future direction that is specific to transcription factor binding site prediction is to incorporate additional information about the protein itself, such as the polypeptide sequence. For example, one way to structure the task could be: given a protein (as its amino acid sequence) and a DNA sequence, does that protein bind on the DNA? Alternatively, given a protein (as its amino acid sequence) and a DNA sequence that it is known to bind to, where precisely on the DNA does the protein bind? This model could use convolutional or recurrent neural networks, similar to this work, for these tasks, and may be able to use the bottom layers of our models to pre-train the new network. With a model that is trained to perform one of these tasks, a researcher could extend the model's benefits to new transcription factors. All she would need is the amino acid sequence of the new transcription factor, which is relatively easy to get, and it can be a valid input to the model.

10.5 New Architectures

There is also substantial room for modeling improvements. In this work, we focus on dilated convolutions, and compare them to standard convolutions. Additionally, we compare them to LSTMs to determine their relative ability to model long-term dependencies. However, there are additional convolutional and LSTM-based architectures that are worth exploring. With LSTMs, there are stacked LSTM architectures, with several layers of bidirectional LSTMs placed one atop another. These may be able to capture more complex long-term dependencies. Additionally, we could seek to conduct a more significant dimensionality reduction of the input sequence. In our work, we generally first have a strided convolution with stride 10 and kernel width 10, which shortens the

length of the sequence by a factor of 10. Then, the final layer serves as an upsampling operation. However, this technique still leaves a relatively long sequence (2,500 in length) that the LSTMs have to operate over, which leads to issues with vanishing gradients. We can try additional methods to learn lower-dimensional representations of the inputs, perhaps through many layers of strided convolutions, and then have LSTMs operate on the shorter sequences. This will come with a tradeoff, since having more reduction in the input sequence will require having more substantial upsampling in the upper layers of the network.

10.6 Architecture Optimization

One of the issues that Chapter 10.5 sheds light on is the fact that while we use sound methods for learning the optimal weights of a given architecture, our decisions about what architecture to use are based largely on intuitions. In particular, the architecture of the network refers to hyperparameters like those that define the number of layers, the stride, kernel width, and dilation of each layer, and the type of layer to use (convolution, pooling, recurrent, etc.). Recent work has suggested that these architectures can themselves be learned (Cortes et al., 2016). In that work, the authors learn the architecture and the optimal weights for a particular image classification task. This approach is more generally applicable too, and can be used to improve the architectural search for this regulatory marker prediction task as well.

Appendices

A

Sequencing Technologies

There are two major technologies that are used to generate the data used in this work: ChIP-seq and DNAse-Seq. ChIP-seq combines chromatin immunoprecipitation (ChIP) with massively-parallel sequencing (seq) to find where targeted proteins bind to DNA (Johnson et al., 2007). At a high level, this technique works by identifying regions of the genome that are bound to a particular targeted protein, and then sequencing all of those regions and aligning them to a reference genome. To do this, proteins are cross-linked to DNA using a formaldehyde treatment and then the DNA is cut into small segments. An antibody specific to a protein of interest is used to select DNA regions that are bound to that protein. The result of this immunoprecipitation is a large quantity of short DNA fragments from across the genome that are bound to the protein of interest. Any regions of DNA not bound to the protein of interest are washed away. Then, the protein is unlinked, and the previously-bound DNA can be

sequenced and the resultant sequences can be aligned to a reference genome. This will then give a measure of where the targeted protein was bound across the genome.

DNAse-seq is very similar, except it amplifies regions where DNA is more sensitive to being cleaved by DNAse, and these correspond to regions of the genome that are more accessible to proteins, where we may expect regulatory behavior to occur (Song and Crawford, 2010). After amplification, the sequencing step is analogous to ChIP-seq.

Additionally, histones can be modified by a cell in a variety of ways, largely due to the addition or removal of certain chemical groups from the protein. Just like discussed before, these histone modifications can be tightly targeted by antibodies, and thus the sites of histone modifications can be sequenced using ChIP-seq.

B

Discussion of ROC AUC and PR AUC Metrics

Precision, Recall, True Positive Rate, False Positive Rate In binary classification tasks, the classic metrics used to determine the success of a model involve determining the ratios of true positives (TP), false negatives (FN), true negatives (TN), and false positives (FP). These values determine the correspondence between the predicted values and the true values for a set of binary labels. "Positive" refers to a predicted label of "1", and the true/false refers to a match between the prediction and the actual value. From these, we derive **precision** and **recall**, which are defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN} \tag{B.1}$$

Additionally, we define the following ratios: **True Positive Rate** (TPR) and **False Positive Rate** (FPR).

$$\text{TPR} = \frac{TP}{TP + FN}, \ \text{FPR} = \frac{FP}{FP + TN} \tag{B.2}$$

Note that the TPR and recall are the same.

ROC AUC Score While the above precision, recall, TPR, and FPR values give useful measures of the predictive accuracy of a classifier, they have one fundamental issue - they require an apriori threshold value. In other words, since binary classifiers generally output a single floating-point number between 0 and 1 as their prediction, those values have to be thresholded into the two classes in order to determine the value of each of those metrics. Unfortunately, determining a threshold is not always obvious. For example, consider a classifier that outputs a prediction of 0.8 for every true class, and 0.7 for every false class. In this case, with a threshold of 0.75, it would have a precision and recall of 1. However, with a threshold of 0.5, it would have a precision of 0.5 (assuming an even number of true and false classes).

However, a classifier that always outputs a higher prediction for the true class than the false class is still considered a good classifier. To reflect this, we use an multiple-threshold formulation by calculating the Receiver Operating Characteristic (ROC) curve. To graph the ROC curve, we have a 2-dimensional plot where the x-axis is the FPR, and the y-axis is the TPR. Each point on the plot is determined by the TPR and FPR values when thresholded at a particular threshold, and we create the plot by choosing increasing thresholds between 0 and 1 (for example: $0, .01, .02, .03, \ldots, .99, 1$).

Then, we calculate the area under this curve (AUC). The higher this area, the less we trade off the false positive rate for improvements in the true positive rate. We want this to be as high as possible.

PR AUC Score To calculate the PR AUC Score, we follow a very similar process as the ROC AUC Score, except we instead make the precision-recall curve. On this curve, we plot the recall values on the x-axis, and precision values on the y-axis for different thresholds. Then, we calculate the area under the curve, and again we want this to be as large as possible.

Using PR AUC vs ROC AUC One of the central issues that have been raised about past approaches to tasks similar to this is the validity of each of these metrics under very imbalanced class ratios. Quang and Xie (2016) suggest that using the PR AUC may be better suited to this task than ROC AUC. In particular, in cases where the positive class is much rarer than the negative class, it is thought that the PR AUC score is more meaningful. To explain this further, consider the following setup. Let $\hat{Y}$ represent the classification predicted by the model, and let Y represent the true classification. Then, we can summarize the definitions from above as

$$\text{Precision} = P(Y = 1|\hat{Y} = 1) \tag{B.3}$$

$$\text{Recall and TPR} = P(\hat{Y} = 1|Y = 1) \tag{B.4}$$

$$\text{FPR} = P(\hat{Y} = 1|Y = 0) \tag{B.5}$$

Precision asks the question: *Given a positive class prediction, what is the probability this prediction is correct?* The others condition on the true class, rather than the predicted class, and this can cause problems for the ROC AUC in certain cases. Note that we generally want to have high precision, high recall, high TPR, and low FPR. We will now construct an example that has both low FPR and low precision, in the presence of imbalanced classes. We will do this using Bayes Rule:

$$\text{Precision} = P(Y = 1|\hat{Y} = 1) = \frac{P(\hat{Y} = 1|Y = 1)P(Y = 1)}{P(\hat{Y} = 1|Y = 1)P(Y = 1) + P(\hat{Y} = 1|Y = 0)P(Y = 0)} \tag{B.6}$$

Now, let the true class frequency be very low, meaning we can approximate $P(Y = 1) \approx 0$. Then, the numerator goes to 0, and therefore the precision goes to 0. The second term in the denominator contains the FPR. Thus, even if we have a low FPR, we can have a low precision. In other words, even though our model very rarely predicts a false positive class, we can still end up with a situation where most of our positive class predictions are wrong. This makes sense in situations with huge class imbalances, because if there are 100 times the number of negative class data as positive class data, then even a rare negative class error could produce a ton of false positives, relative to the number of true positives.

As such, it is generally thought that PR curves serve as a more meaningful metric when dealing with highly imbalanced class distributions. For this work, we generally report both ROC AUC and PR AUC values for comparison with past work, but do our comparative analysis based on the PR AUC values.

C

Additional Data Tables

Here, we report the full data tables for each of the classes of models described in this work. In the main body of this report, we included the central findings from these tables. Here, we include them in full. Note that we primarily used PR AUC scores for our analysis. That said, for comparison with existing work that uses ROC AUC scores, we include the results with that metric as well.

C.1 Task 1

Here, we report the AUC ROC and PR AUC scores for the models in Task 1. Note that we report the best performing results across all of the hyperparameter configurations for a model.

Model	Validation ROC AUC			Test ROC AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Baseline: LR	0.735	0.656	0.686	0.716	0.670	0.674
Baseline: FF	0.745	0.684	0.705	0.737	0.693	0.690
Baseline: Conv3	0.880	0.780	0.870	0.869	0.782	0.860
Baseline: LSTM	0.947	0.835	0.910	0.935	0.833	0.904
Dilated3	0.877	0.783	0.866	0.864	0.779	0.852
Dilated6	0.939	0.823	0.903	0.923	0.813	0.895
Deep-Dilated	0.895	0.788	0.880	0.881	0.771	0.873

Table C.1.1: Full Validation and Test set ROC Area Under Curve (ROC AUC) scores for the models in Task 1. Note that these are the best results across all hyperparameter configurations.

Model	Validation PR AUC			Test PR AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Baseline: LR	0.079	0.139	0.066	0.042	0.143	0.097
Baseline: FF	0.076	0.194	0.114	0.046	0.181	0.106
Baseline: Conv3	0.250	0.285	0.373	0.205	0.273	0.319
Baseline: LSTM	0.365	0.350	0.437	0.305	0.340	0.407
Dilated3	0.264	0.294	0.352	0.190	0.271	0.299
Dilated6	0.332	0.329	0.429	0.273	0.320	0.390
Deep-Dilated	0.285	0.278	0.386	0.233	0.271	0.350

Table C.1.2: Full Validation and Test set Precision-Recall Area Under Curve (PR AUC) scores for the models in Task 1. Note that these are the best results across all hyperparameter configurations.

C.2 Task 2

Here, we report the AUC ROC and PR AUC scores for the models in Task 2. Note that we report the best performing results across all of the hyperparameter configurations for a model.

Model	Validation ROC AUC			Test ROC AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Conv1	0.712	0.644	0.688	0.726	0.649	0.693
Conv3	0.840	0.747	0.824	0.850	0.753	0.829
Conv7	0.905	0.787	0.874	0.911	0.789	0.878
Dilated	0.895	0.863	0.863	0.898	0.861	0.861
Bi-LSTM	0.854	0.840	0.816	0.864	0.833	0.818
ID-CNN	0.888	0.840	0.857	0.892	0.834	0.857

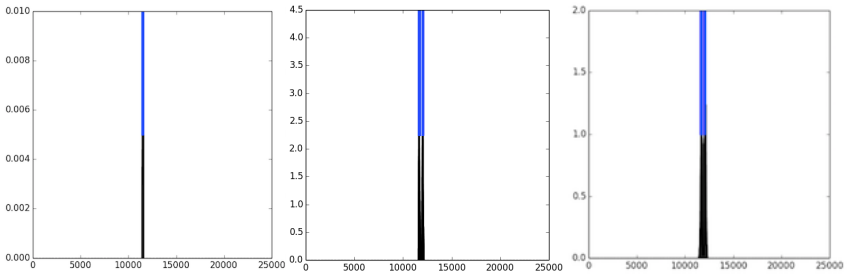
Table C.2.1: Full Validation and Test set ROC Area Under Curve (ROC AUC) scores for the models in Task 2. Note that these are the best results across all hyperparameter configurations.

Model	Validation PR AUC			Test PR AUC		
	TFBS	Hist	DNAse	TFBS	Hist	DNAse
Conv1	0.013	0.053	0.035	0.014	0.055	0.035
Conv3	0.059	0.115	0.100	0.059	0.116	0.098
Conv7	0.167	0.154	0.160	0.167	0.152	0.171
Dilated	0.274	0.279	0.167	0.274	0.270	0.163
Bi-LSTM	0.065	0.238	0.089	0.070	0.216	0.086
ID-CNN	0.166	0.247	0.147	0.171	0.236	0.142

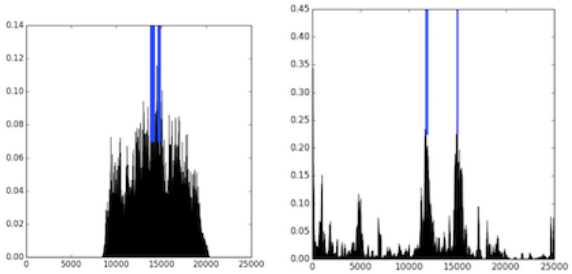
Table C.2.2: Full Validation and Test set Precision-Recall Area Under Curve (PR AUC) scores for the models in Task 2. Note that these are the best results across all hyperparameter configurations.

D

Additional Visualizations



(a) Conv1 **(b)** Conv3 **(c)** Conv7



(d) Dilated **(e)** Bi-LSTM

Figure D.0.1: This shows the receptive field for all the major architectures that we study. In blue is the location of a regulatory marker. In black, we plot a histogram indicating the gradient backpropagated to the input sequence, which gives a measure of the size of the receptive field. Here, we note that increasing the number of convolutional layers from 1 to 7 does not substantially increase the size of the receptive field, as we expected. However, using a Dilated model leads to a big receptive field. The Bidirectional LSTM has a large receptive field, but has very little gradient backpropagated to most of it, meaning that it may not be getting as much signal from faraway regions as the Dilated model.

E

Precise Model Descriptions

E.1 Task 1

- **Baseline: Logistic Regression (LR)**: This is a baseline logistic regression model that consists of reshaping the input into a single (4000,) vector, and then having a single fully-connected layer and sigmoid. This model serves as a baseline to the more complex convolutional models we discuss later. There are a total of 3,676,919 trainable parameters.
- **Baseline: Multilayer Perceptron (MLP)**: This is a baseline multilayer perceptron model that consists of reshaping the input into a single (4000,) vector, and then having a fully-connected layer with 925 units, a ReLU activation, and another fully-connected layer and sigmoid. This model serves as a baseline to the more complex convolutional models we discuss later. There are a total of 4,551,919 trainable parameters.

- **Baseline: 3-Layer Convolution (Conv3)**: This is a reimplementation of the model made in Zhou and Troyanskaya (2015). Though that model was originally written in Torch, we re-implemented it in Tensorflow along with the rest of our models. This model consists of three convolutions with kernel width 8, with the first two followed by max pooling with stride 4 and kernel width 4. After these three convolutional layers, there are two fully-connected layers to make the 919 final predictions. This model has a total of 60,472,479 trainable parameters.

- **Baseline: Convolution and Bidirectional LSTM (LSTM)**: In this model, we reimplement the work from Quang and Xie (2016). The model consists of a single convolution with kernel width 26 and 320 filters, followed by a max pooling layer with a stride of 13 and kernel width of 13. Then, there is a Bidirectional LSTM, and two fully-connected layers. In total, there are 46,926,479 trainable parameters.

- **3-Layer Dilated Convolution (Dilated3)**: This is an extension of the Conv3 model above that first replaces each of the convolutions with a dilated convolution. The dilation increases by a factor of 3 for each layer. Additionally, the number of filters used in this model is much smaller, as the dilated convolutions are able to capture more complex behavior. Thus, the total number of parameters is 22,284,639.

- **6-Layer Dilated Convolution (Dilated6)**: This is an extension of the Dilated3 model above. The primary difference is that it adds 3 additional dilated convolutional layers, before adding the fully-connected layers. In order to account for the increased parameters introduced by the higher number of convolutions, we reduce the number of filters in each convolution. The total number of parameters is 8,043,759.

- **Deep Dilated Convolution (Deep-Dilated)**: In this model, we use a large number of convolutional layers, with only a very small fully connected layer at the end. In particular, we have 15 dilated convolutional

layers, divided into 3 groups of 5 convolutions. Each of these groups has dilated convolutions with dilation rate 1, 2, 4, 8, 16, and each group is followed by a max pooling with kernel width 4 and stride 2. Each of the 15 convolutions has 32 filters (much less than the others models), and there is only a single fully connected layer at the end. As a result, despite the substantial depth of this model, there are only 2,073,943 parameters.

E.2 Task 2

- **1 layer Convolution (Conv1)**: This model begins by embedding the nucleotide in 4 dimensions. Then, it runs a convolution with kernel width of 10 over the sequence with 128 filters, padded to maintain the original width. Finally, it uses a simple multilayer perceptron with 1 hidden layer at each nucleotide to predict all 919 outputs. This model tests whether a simple convolution is able to capture the genetic motifs effectively enough to not need a more complex model. There are a total of 136,935 trainable parameters.

- **3 layer Convolution resizing (Conv3)**: The model is similar to Conv1, except it uses 3 convolutional layers with dropout between consecutive layers. The number of filters in each respective layer is 128, 240, and 50. In an alternate version of this model, we have the first convolution as strided, with a stride of 10, which downsamples the inputs. The second layer is a standard convolution which operates over a sequence that is 1/10th the length of the input layer. The third layer is a deconvolution. This means that the middle convolutional layer goes over a sequence that is 1/10th the size of the original, and so the receptive field of each output neuron is larger. The basis of this downsampling is to pick up on the fact that because of the small vocabulary size of our inputs, doing a strided convolution would allow the middle layers to work in an input space that has learned meaningful motifs from the underlying dataset. There are a total of

554,675 trainable parameters.

- **7 layer Convolution (Conv7)**: This model is similar to Conv3, except there are more convolutional layers in between the strided convolution and deconvolution. This model has approximately the same number of parameters as the dilated convolution below. There are a total of 654,875 trainable parameters.

- **Dilated Convolution (Dilated)**: In this model, there is first a strided convolution, as in Conv7. After that, there are 4 dilated convolutional layers, with dilation rates of 3, 9, 27, and 81 respectively. Lastly, there is a deconvolutional layer like in Conv7. Note that the dilated convolutions all occur over sequences that are 1/10th of the original length because of the first layer being a strided convolution. We have two adaptations of this model. In one, we add max-pooling with a stride of 1 between several of the layers. This max-pooling smoothes the activations. Since the stride is 1, this does not change the dimensions of the tensor. In the second adaptation, we use dilated max-pooling, which is the same as max-pooling except over a dilated window. We add these dilated max-pooling layers after the third and fourth dilated convolutions. This follows a similar intuition as the standard max-pooling setup. However, having dilated max pooling allows each node to perform an OR operation over activations that are separated by the given dilation rate, allowing each node to learn activations from a wider set of inputs. There are a total of 629,825 trainable parameters.

- **Bidirectional LSTM (Bi-LSTM)**: This model consists of a strided 1-layer convolution with kernel width 10, followed by a Bidirectional LSTM, and a deconvolution. This is similar to the bottom layers of the model from Quang and Xie (2016). This model is meant primarily to compare the dilated convolutional models with recurrent neural networks. While RNNs are commonly used for sequential modeling tasks, the vanishing

gradient problem becomes more problematic for them when inputs are very long. Thus, this model serves as a point of comparison with the convolutional techniques. There are a total of 397,155 trainable parameters in this model.

- **Iterated Dilated CNN (ID-CNN)**: In this model introduced by Strubell et al. (2017), there are 3 blocks of 5 dilated convolutions. These blocks share parameters, so even though there are more layers than the other models, the total number of parameters remains small. This model uses a state of the art dilated convolution model that can learn complex mappings of inputs with few parameters. We compare this approach to the more standard dilated convolutional models as well. There are 632,487 trainable parameters in this model.

References

Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 2015.

Jon-Matthew Belton, Rachel Patton McCord, Johan Harmen Gibcus, Natalia Naumova, Ye Zhan, and Job Dekker. Hi–c: a comprehensive technique to capture the conformation of genomes. *Methods*, 58(3):268–276, 2012.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb): 1137–1155, 2003.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

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.

ENCODE Project Consortium et al. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, 2012.

Corinna Cortes, Xavi Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. Adanet: Adaptive structural learning of artificial neural networks. *arXiv preprint arXiv:1607.01097*, 2016.

Francis Crick et al. Central dogma of molecular biology. *Nature*, 227(5258): 561–563, 1970.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

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

David S Johnson, Ali Mortazavi, Richard M Myers, and Barbara Wold. Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316 (5830):1497–1502, 2007.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Alexander E Kel, Ellen Gößling, Ingmar Reuter, Evgeny Cheremushkin, Olga V Kel-Margoulis, and Edgar Wingender. Matchtm: a tool for searching transcription factor binding sites in dna sequences. *Nucleic acids research*, 31 (13):3576–3579, 2003.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10): 1995, 1995.

Diana O Perkins, Clark Jeffries, and P Sullivan. Expanding the ?central dogma?: the regulatory role of nonprotein coding genes and implications for the genetic liability to schizophrenia, 2005.

Daniel Quang and Xiaohui Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, page gkw226, 2016.

Lingyun Song and Gregory E Crawford. Dnase-seq: a high-resolution technique for mapping active gene regulatory elements across the genome from mammalian cells. *Cold Spring Harbor Protocols*, 2010(2):pdb–prot5384, 2010.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Gary D Stormo. Dna binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. Fast and accurate sequence labeling with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

Ya-Mei Wang, Ping Zhou, Li-Yong Wang, Zhen-Hua Li, Yao-Nan Zhang, and Yu-Xiang Zhang. Correlation between dnase i hypersensitive site distribution and gene expression in hela s3 cells. *PloS one*, 7(8):e42414, 2012.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.