# Sparse Robust Recovery and Learning

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th>Citation</th>
<th>Gwon, Youngjune Lee. 2015. Sparse Robust Recovery and Learning. Doctoral dissertation, Harvard University, Graduate School of Arts &amp; Sciences.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:17467249">http://nrs.harvard.edu/urn-3:HUL.InstRepos:17467249</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Sparse Robust Recovery and Learning

A dissertation presented
by
Youngjune Lee Gwon
to
The School of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Harvard University
Cambridge, Massachusetts
April 2015
Sparse Robust Recovery and Learning

Abstract

Sparse linear models pose dual views toward data that are embodied in compressive sensing and sparse coding. Despite mathematical equivalence, compressive sensing and sparse coding are two different classes of application for sparse linear models. Compressive sensing draws recoverable, low-dimensional compressed representations from a blind linear projection on data. Sparse coding enables the discovery of structural patterns underlying data in forced decomposition with a given dictionary of basis vectors. Sparsity is the common constraint that makes exact recovery possible for compressive sensing and allows forced decomposition to unveil meaningful features for sparse coding.

In this thesis, I build on compressive sensing and sparse coding to explore the problems for reconstructive and discriminative applications in sensing, wireless networking, and machine learning. Specifically, I aim to develop recovery and feature learning methods robust to complex data transformations and alterations. With a wideband spectrum sensing application for cognitive radios, I empirically demonstrate the resiliency of the proposed sparse recovery technique to linear and nonlinear distortions present in the mix of heavily subsampled RF measurements. I push beyond best-known efficiency for distributed compressive sensing and show feasibility of scaling the spectrum sensing application with constant communications cost.

I also focus on learning sparse feature representations for discriminative machine learning tasks. I build a classification pipeline based on both single-layer and multilayer sparse coding trained on various modalities of data including text, image, and time series. To take advantage of possible higher-level construct of features in data, I propose a deep architecture on multilayer sparse coding, namely Deep Sparse-coded Network (DSN). When I train DSN with layer-by-layer dictionary learning followed by the proposed DSN backpropagation algorithm for image and time-series classification, it leads to performance better than deep stacked autoencoder neural network.
Abstract

In addition, I present Nearest Neighbor Sparse Coding (NNSC), an enhancement for sparse coding by imposing the nearest neighbor constraint in the sparse feature domain. Despite inferior reconstructive error, NNSC improves the classification performance of classical sparse coding.
Contents

Title Page i
Abstract iii
Citations to Previously Published Work xii
Acknowledgments xiii
Dedication xv

1 Introduction 1
  1.1 Sparse Modeling of Data and Applications . . . . . . . . . . . . . . . . . . 3
  1.2 Our Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  1.3 Summary of Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  1.4 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2 Background 9
  2.1 The Two Models . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.2 The Sparse Decomposition Problem . . . . . . . . . . . . . . . . . . . . 10
  2.3 Approaches to Solve Sparse Decomposition Problem . . . . . . . . . . . . 11
    2.3.1 LASSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.3.2 Basis pursuit . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.3.3 Matching pursuit . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
  2.4 Compressive Sensing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  2.5 Sparse Coding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3 Compressive Spectrum Sensing for Cognitive Radios 18
  3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
    3.1.1 Challenges . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
    3.1.2 Problem statement . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
    3.1.3 Related work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  3.2 Minimizing Communications Cost of Sensor Measurements . . . . . . . . . 23
## Contents

3.2.1 Exploiting signal sparsity with simple compressive encoding .... 24  
3.2.2 Combining multiple measurements ............................. 24  
3.3 System Operation .................................................... 25  
3.3.1 Preparation phase for basis estimation .......................... 25  
3.3.2 Recovering original data ......................................... 26  
3.3.3 Algorithms .......................................................... 28  
3.3.4 Comparison with joint decoding ................................ 28  
3.4 Evaluation ............................................................ 29  
3.4.1 Experimental methodology ........................................ 29  
3.4.2 Evaluation scenarios and metric .................................. 31  
3.4.3 Decoding accuracy and complexity ............................... 31  
3.4.4 Effect of iterative refinements .................................... 33  
3.4.5 Asymptotic error performance analysis .......................... 33  
3.5 Summary .............................................................. 36  

4 Feature Learning via Sparse Coding for Discriminative Tasks 37  
4.1 Introduction ............................................................ 37  
4.1.1 Problem description ................................................. 38  
4.2 Semi-supervised Classification Pipeline ............................. 39  
4.2.1 Overview ............................................................ 39  
4.2.2 Data processing with patches ...................................... 40  
4.2.3 Sparse coding and dictionary learning ............................ 41  
4.2.4 Classifier training .................................................... 42  
4.3 Experimental Evaluation with Text and Image Datasets .............. 43  
4.3.1 Text-based regional classification with Twitter dataset .......... 43  
4.3.2 CIFAR-10 image classification ..................................... 47  
4.4 Summary .............................................................. 53  

5 Multilayer Sparse Coding and Deep Sparse-coded Network (DSN) 55  
5.1 Motivation .............................................................. 55  
5.2 Notation .............................................................. 57  
5.3 Architectural Component .............................................. 58  
5.4 Algorithms ............................................................ 58  
5.4.1 Pretraining with layer-by-layer sparse coding and dictionary learning .................................................. 58  
5.4.2 Training classifiers at output layer ................................. 61  
5.4.3 Backpropagation ..................................................... 62  
5.5 Experimental Results .................................................. 65  
5.5.1 Classifying CIFAR-10 images ...................................... 65  
5.5.2 Inferring origin traffic patterns in Wi-Fi .......................... 70  
5.6 Summary .............................................................. 76
## Contents

6  Nearest Neighbor Sparse Coding 77
   6.1 Introduction .............................................. 77
   6.2 Nearest Neighbor Sparse Coding (NNSC) .................... 78
   6.3 Theoretical analysis ....................................... 80
   6.4 Evaluation ................................................ 83
      6.4.1 Feature learning, training, and testing with NNSC .... 84
      6.4.2 Experiments with simulated data ....................... 84
      6.4.3 Performance on GEOTEXT dataset ...................... 87
      6.4.4 Performance on CIFAR-10 dataset .................... 87
   6.5 Summary ................................................ 87

7  Conclusion ................................................. 89

Bibliography ................................................. 92
List of Figures

1.1 Typical machine learning approaches assume that training examples are close realizations of the original data. ........................................... 5
1.2 In this thesis, we explore recovery and learning algorithms that are resilient to linear and nonlinear alterations caused by various sources. .......... 5

2.1 Two sparse computing models: compressive sensing (top) requires a sparse recovery technique to restore the original data while sparse coding (bottom) uses essentially the same computational tool to obtain sparse representation of the data ........................................... 10

3.1 A wideband spectrum (e.g., $B_{tot} \approx 100s$ of MHz to a few GHz) under analysis is partitioned for collective measurement efforts by multiple sensor nodes. Spectrum analysis on the measured data takes place at the distant system backend. ........................................... 21
3.2 In-network transmission of measurements ........................................... 23
3.3 An example for $P$-way mixing of compressive measurements (note: $x_i \in \mathbb{C}^N$, $y_i \in \mathbb{C}^M$, and the $P$-way sum $y \in \mathbb{C}^M$) ........................................... 25
3.4 Experimental setup for evaluation ........................................... 30
3.5 Total number of measurements used in spectrum analysis vs. accuracy of decoding measured by error metric $\xi$. The proposed schemes run $T = 2$ refinement stages. Joint decoding for 8-way mix requires the number of measurements on par with the proposed 8-way combining scheme for similar error performance, but joint decoding is much harder computationally and incurs significantly longer decoding time. ........................................... 32
3.6 Comparison of the proposed $P$-way combining ($P = 2, 4, 8$) vs. compressive sensing only. ........................................... 33
3.7 Snapshots of recovered spectrum under uncompressed, compressed without combining, and compressed with 8-way combined methods ................. 34
3.8 Error improvement over number of refinement stages performed for $P = 8$. .......... 34
3.9 Asymptotic error ........................................... 35

4.1 Semi-supervised approach for classification task ........................................... 40
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Full system pipeline for Twitter text-based regional classification in US</td>
<td>46</td>
</tr>
<tr>
<td>4.3</td>
<td>Sample images of CIFAR-10 classes (photo courtesy of Alex Krizhevsky for University of Toronto)</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>Autoencoder neural network</td>
<td>48</td>
</tr>
<tr>
<td>4.5</td>
<td>Generalized image data processing pipeline that yields pooled sparse code</td>
<td>49</td>
</tr>
<tr>
<td>4.6</td>
<td>Forming classifier feature for CIFAR-10</td>
<td>50</td>
</tr>
<tr>
<td>4.7</td>
<td>CIFAR-10 1-vs-all classification accuracy comparison for AE-0.15, LARS-0.2, and OMP-15. The “dense” configuration indicates the use of $6 \times 6$ overlapping patches, whereas we have used $4 \times 4$ nonoverlapping patches otherwise.</td>
<td>51</td>
</tr>
<tr>
<td>4.8</td>
<td>CIFAR-10 1-vs-all classification accuracy comparison for AE-0.2, LARS-0.1, and OMP-20. Here, we use slightly less sparse (more nonzeros) coder configurations. Lower $\lambda$ for LARS means less penalty for the $\ell_1$-regularized term, resulting lower reconstructive error than LARS-0.2. Similarly, AE and OMP are configured to induce more nonzeros. We observe the overall improvements in the classification performance than sparser coder configuration used in Figure 4.7.</td>
<td>52</td>
</tr>
<tr>
<td>5.1</td>
<td>Sparse coding and pooling units are coupled to form a stackable layering module</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>Multilayer sparse coding architecture with 2 hidden layers</td>
<td>59</td>
</tr>
<tr>
<td>5.3</td>
<td>After pooled sparse code $z$ is corrected to $z^<em>$, the corrected values are put back to the corresponding locations at the original sparse codes that have been max pooled to $z$. Putback corrects sparse codes $y$’s to $y^</em>$’s.</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Sparse coding and max pooling at hidden layer 1 for single CIFAR-10 image</td>
<td>67</td>
</tr>
<tr>
<td>5.5</td>
<td>Sparse coding and max pooling at hidden layer 2 for single CIFAR-10 image</td>
<td>68</td>
</tr>
<tr>
<td>5.6</td>
<td>CIFAR-10 1-vs-all classification accuracy comparison for deep SAE, multilayer sparse coding with LARS-0.1 and OMP-20</td>
<td>69</td>
</tr>
<tr>
<td>5.7</td>
<td>Seven Wi-Fi node scenario in OPNET</td>
<td>72</td>
</tr>
<tr>
<td>5.8</td>
<td>Seven Wi-Fi node scenario in OPNET</td>
<td>75</td>
</tr>
<tr>
<td>6.1</td>
<td>$k$-NNs in sparse feature domain and majority voting</td>
<td>79</td>
</tr>
<tr>
<td>6.2</td>
<td>NNSC produces a new sparse code $y^\dagger$ for $x$ closer to its ground-truth class cluster in the feature domain</td>
<td>82</td>
</tr>
<tr>
<td>6.3</td>
<td>Feature learning via NNSC. Initial dictionary learning is done by classical sparse coding. An alternating procedure to find NNSC codes and dictionary updates follows and repeats until convergence.</td>
<td>83</td>
</tr>
<tr>
<td>6.4</td>
<td>Linear SVM classifiers are trained using features extracted from classical sparse coding and NNSC.</td>
<td>83</td>
</tr>
<tr>
<td>6.5</td>
<td>Classification testing</td>
<td>84</td>
</tr>
<tr>
<td>6.6</td>
<td>Mean normalized reconstructive errors for NNSC-OMP and OMP</td>
<td>85</td>
</tr>
<tr>
<td>6.7</td>
<td>Binary classification accuracy for GMM-generated dataset</td>
<td>86</td>
</tr>
</tbody>
</table>
List of Figures

6.8 4-class classification accuracy for GMM-generated dataset . . . . . . . . . 86
List of Tables

3.1 Comparison of required total measurements for given error budget . . . . . 36

4.1 Classification accuracy comparison . . . . . . . . . . . . . . . . . . . . . 46

4.2 Average 1-vs-all classification accuracy under normal feature extraction
$(w = 4, \text{nonoverlapping patches})$ . . . . . . . . . . . . . . . . . . . . . . 53

4.3 Average 1-vs-all classification accuracy under dense feature extraction $(w = 6, \text{overlapping patches with } s = 2)$ . . . . . . . . . . . . . . . . . . . . . . 53

5.1 Average 1-vs-all classification accuracy comparison . . . . . . . . . . . . 69

5.2 Origin flow patterns used for evaluation . . . . . . . . . . . . . . . . . . . . 71

5.3 Summary of Wi-Fi parameter configuration . . . . . . . . . . . . . . . . . . 72

5.4 Node roles in seven Wi-Fi node scenario . . . . . . . . . . . . . . . . . . . 72

6.1 Regional and state classification accuracy of OMP-20, LARS-0.1, and their
NNSC enhancements for GEO\textsc{EXT} . . . . . . . . . . . . . . . . . . . . . . 86

6.2 1-vs-all classification accuracy of OMP-20, LARS-0.1, and NNSC for CIFAR-
10 under dense feature extraction $(w = 6, \text{overlapping patches with } s = 2)$ . 87
Citations to Previously Published Work

Chapter 3 is based on


A portion of Chapter 4 will appear in


A portion of Chapter 5 has appeared in

Acknowledgments

Foremost, I thank my advisor HT Kung.

My odds of getting this PhD weren’t too good in the beginning. I came in Harvard under a great amount of pressure, knowing too well that anything other than earning a PhD couldn’t be my option after I left a good career and took my family from a settled life in California. I went to meet HT when my doubt started to kick in around mid-November of 2010. Having only communicated through a few emails, HT already had a good research problem in mind for me. After a week, we met again, and HT invited me to his group meeting. As far as I remember, I was with HT since then.

HT taught me research in a close distance. Some might call it old-fashioned or classical. I’d call it effective, nonetheless. He usually spared a few hours for me exclusively, and we sat down and went over all the details together. I can’t thank HT enough for all the hours of thinking and the patience that he has had for me. After all, I’ve benefited so much from HT’s wisdom and high scholastic qualities. I am extremely fortunate to receive his teachings and will remember them to pass on.

I’d like to thank Greg Morrisett. Sheer excellence of Greg has always inspired me. When in need, Greg made himself easily accessible. Greg gave me valuable counsel for both my research and life at Harvard. I also thank Yue Lu, whose technical brilliance has helped shape my work in part. I already see a great scholar in Yue and am grateful that I’ve worked with him. Graciously, both Greg and Yue instantly agreed to serve my qualifying exam and dissertation committees.

I owe a great deal of thanks to Siamak Dastangoo, my mentor. In addition to serving in my dissertation committee, Sia spent so many hours to improve my research. Sia arranged all the great things happened for me at MIT Lincoln Laboratory.
I’m deeply indebted to Margo Seltzer. I give my most sincere thanks to Margo, for I’ve received an immeasurable amount of guidance and care from her. I wouldn’t be able to complete this thesis without her help.

Guyeon Wei and Matt Welsh also supported me throughout the years. I thank them for the practical guidances. I’d like to extend my gratitude to Ken Hetling and Carl Fossa who gave me a unique opportunity to stay on with Lincoln Laboratory. I also thank Gioia Sweetland for her excellent administrative support.

My years at Harvard have been particularly enjoyable because of my fellow students. I thank my predecessors Dario Vlah and Chit-Kwan Lin. Dario helped me greatly to settle in and collaborated on several papers. My thanks also go to Tsung-Han Lin, Kevin Chen, Michael Crouse, Surat Teerapittayanon, Bradley McDanel, and Marcus Comiter.

My special thanks go to Miriam Cha, my research partner. She stood by and helped me through my toughest hours. Miriam made some of my most enjoyable hours at Harvard and Lincoln Laboratory. Without her, later parts of my PhD wouldn’t be possible.

Finally, I’d like to thank my family for their love and support. I thank my mother and father for their unfailing love. My brother and his family have cheered me from distance. I thank my in-laws in Korea, especially my mother-in-law who has always been inspirational and supportive. I thank my son Obin and my daughter Ojou. They remain sweet and lovable despite all the circumstances. Most of all, I thank my wife Heekyung for her love and enduring support. And to you my soul mate I give this thesis.
To my family
Chapter 1

Introduction

Sparse modeling of data has enabled powerful tools in signal processing and machine learning. A common task in sensing is to draw characteristic information from the raw sensory input and store in an efficient form for further analysis. For decades, information acquisition at the front-end has been dictated by the Nyquist-Shannon sampling theorem. The guarantee of full data recovery is a comforting notion although we often have to pay a straightforwardly expensive price proportional to the highest frequency element of the data rather than its actual information rate.

Recently, a class of linear dimensionality reduction techniques [1–5] has led to refreshing insights. They exploit a distance-preserving mapping that projects the raw input onto a low-dimensional space. In compressive sensing [6], such mapping remarkably turns out to be a random subsampling matrix with a high probability to satisfy column incoherency. To reconstruct the original data, compressive sensing decode searches for the sparest solution from an underdetermined linear system of compressed measurements in a salient procedure known as “sparse recovery.” The fuller exploration of this concept\(^1\) is beyond the scope of this dissertation, but we will suggest some novel usage of compressive sensing model for networked wireless systems where we dramatically cut the storage overhead, processing power, and communications cost.

\(^1\)Reconstructive usage is just one aspect of compressive sensing. Compressed domain signal processing [7] is a powerful application scenario that operates directly on compressive measurements. Because of reduced dimensionality, compressed domain algorithms are more efficient in computation and storage.
Chapter 1: Introduction

The overall aim of machine learning is to develop methods that can learn from complex data and make good predictions for the unseen. Unfortunately, it is hard to make an accurate prediction using the raw data directly. For instance, determining which modulation scheme is used for a signal would be nearly impossible from inspecting only received sample magnitudes in time. Also, if we want to map images to their identifier ("label") using only pixel intensity values, we are likely doomed for a failure from poor handling of high dimensionality (images comprise hundreds of thousands to millions of pixels) and high variability (e.g., brightness, contrast, translation, angle). Similar difficulties will hold true for other modalities such as text, audio, and video.

In representation learning [8], we seek for a transformation or mapping that takes in the raw data and computes “feature” to infer more effective factors for discriminative task. Compared to using the raw data, such feature domain analysis will make distinguishing characteristics among different classes of data more apparent. One of the most successful feature learning methods to date is sparse coding [9], an unsupervised method based on sparse modeling of data. Interestingly, sparse coding and aforementioned compressive sensing are highly related. Sparse recovery in compressive sensing decode is merely a different realization of the same mathematical tool embodied in sparse coding.

Sparsity sought in reconstructive compressive sensing is the key required for correctness of the data recovered. This works as if there is no ground-truth for the ill-posed linear system, but it will be sufficient to rely on the search for the sparsest solution as the only guide, optimistically. Sparsity in representation learning, on the other hand, is a forced decomposition for data analysis—we want to explain the data from only a small number of opinions collected, and these opinions are the only ones that matter to describe the input and to instruct classifiers with.

In this thesis, I investigate the problem of applying sparse modeling for reconstructive and discriminative applications in sensing, wireless networking, and machine learning. Specifically, I will develop robust recovery and feature learning methods undergoing complex data transformations. The work of my thesis addresses an important issue to advance the current state of sparse signal processing and machine learning. I will empirically demonstrate an enhanced performance in reconstructive accuracy, communications cost, and classification resulted not only from the proposed algorithms and system design, but
1.1 Sparse Modeling of Data and Applications

Fundamentally, sparse modeling [10,11] poses dual views toward data. We want to describe (or decompose) the data as a linear combination of basis vectors. Consider a matrix $D \in \mathbb{R}^{N \times K}$ of basis vectors $\{d_1, d_2, \ldots, d_K\}$ with each $d_i \in \mathbb{R}^N$ as a column. Given data $x \in \mathbb{R}^N$, we search for $y \in \mathbb{R}^K$ such that $x \approx Dy = \sum_{i=1}^{K} y_i d_i$ in the following optimization

$$
\hat{y} = \arg \min_y \frac{1}{2} \|x - Dy\|_2^2 + \lambda \psi(y)
$$

(1.1)

where the elements $\{y_1, y_2, \ldots, y_K\}$ of $y$ are mostly zero. With $N < K$, an overcomplete $D$ makes the solution $\hat{y}$ not unique. This underdeterminedness should be resolved by regularization in the second term of Equation (1.1). The two strategies for sparsity-inducing $\psi(.)$ are the $\ell_0$- or $\ell_1$-norm of $y$, $\psi(y) = \|y\|_0$ or $\|y\|_1$.

In general, finding the $\ell_0$-minimum solution for Equation (1.1) is known NP-hard. One alternative approach is to make a convex relaxation and solve for the $\ell_1$-minimum solution by quadratic or linear programming. Recently, greedy-$\ell_0$ algorithms such as matching pursuit are used to find a solution in much faster time.

For our first application, we develop a sparse recovery technique resilient to linear and nonlinear distortions in the mix of heavily subsampled RF measurements [12], which we push beyond best-known efficiency for compressive and distributed sensing. Compressive sensing is an inverse usage case for sparse data modeling in Equation (1.1). This is interesting since, for compressive sensing, $y$ is the data given, and $x$ the compressed measurements. We exemplify wideband spectrum sensing capabilities highly desired in the emerging dynamic spectrum access paradigm [13] for cognitive radios [14].

We also focus on learning sparse feature representations for discriminative machine learning applications. One of our main applications is text-based geographic classification [15]. We build a classification pipeline based on single-layer sparse coding trained on Twitter microblog messages. Our use of sparse coding as feature extractor for text data...
in geographic classification achieves a high performance and is novel as the majority of
known approaches is based on latent geographic topic model [16], supervised LDA [17],
and text regression [18].

Another application of our importance in this thesis is image classification. Visual ob-
ject recognition is a challenging problem where most state-of-the-art machine learning al-
gorithms are developed for. It is especially difficult for a learning algorithm to capture
most relevant features for discrimination and generalize using the raw data (pixel intensi-
ties) from training images to all. Thus, it is natural to take advantage of possible higher-
level construct of features for images. With this in mind, we experiment with both single-
layer and multilayer sparse coding. Particularly, we model after conventional multilayer
architectures such as stacked autoencoder neural network [19] connected in a feedforward
manner and optimized via backpropagation. In the context of feature learning, we pretrain
our multilayer sparse coding with greedy layer-by-layer dictionary learning before tightly
optimizing the whole sparse-coded network (feedforward) with a novel backpropagation
algorithm.

The last of our discriminative applications is time-series inference. Here, our primary
objective is to learn invariance to discriminate different classes of network traffic data un-
dergone complex alterations. We work on the problem of identifying the origin pattern of
a Wi-Fi traffic flow as is generated at the source Wi-Fi node [20]. This inverse problem is
challenging because of intrusive inner-workings of carrier sense multiple access (CSMA) in
the 802.11 MAC protocol. In the presence of competing flows, CSMA can significantly al-
ter the original traffic pattern of a flow, inducing random delays and causing loss of frames.
Our conjecture is to build robust machine learning schemes out of sparse coding that can
harness complexity and nonlinearity. In order to achieve good performance, however, it
turns out that we require adept domain knowledge or application-specific insights to con-
figure the overall learning pipeline and fine-tune all model parameters to their meticulous
detail.
Chapter 1: Introduction

1.2 Our Approach

We remind that one of our goals is sparse robust recovery and learning that are particularly resilient to complex alterations impacting the original data. Such alterations include noise and distortions that could be linear or nonlinear in nature. For clarity, Figure 1.1 illustrates an assumption on training examples used in a typical machine learning pipeline. Here, the origin data are available as-is for training a learning algorithm.

In Figure 1.2, we illustrate the approach taken by this thesis. As we work with multiple modalities of data, we assume various kinds of noises, corruptions, and erasures that may impact our observation or measurement in a data-specific manner. Sometimes, there are additional sources of noise introduced through sensing, device hardware, and external factors. As a whole, our method is subject to data undergone complex transforms.

Figure 1.1: Typical machine learning approaches assume that training examples are close realizations of the original data.

Figure 1.2: In this thesis, we explore recovery and learning algorithms that are resilient to linear and nonlinear alterations caused by various sources.
Chapter 1: Introduction

Our foremost design element is sparsity. As an example, for spectrum sensing in UHF white spaces in Chapter 3, we leverage sparsity of each radio channel from learning. We perform principal component analysis (PCA) [21], a generic unsupervised learning algorithm, to acquire the Karhunen-Loève (KL) basis [22]. This has two benefits. First, the KL basis enables an optimal use of compressive sensing for each channel. More importantly, we use the KL basis for regularization to separate in-network mixing of multiple compressed measurements from different nodes, which are massively underdetermined and cannot be solved by an ordinary sparse recovery technique. Despite substantial saving in sensor network backhaul bandwidth, in-network mixing operations are the source of additional distortions. We show how our proposed sparse recovery algorithm can overcome these distortions and iteratively reconstruct the original data in multiple stages.

As another example, in our approach for the Wi-Fi traffic inference problem, we model the pattern of a network traffic flow as a finite time series data (of burst runs and gaps). Sparse coding is applied to extract characteristic burst (or gap) sizes as a feature. We have first tried with single-layer sparse coding and concluded that it can learn simple, local features within a relatively short time interval. Aggregation through pooling and stacked sparse coding layers have significantly increased our feature coverage. We have possibly enabled to learn a hierarchy of features that proves to be beneficial from our results. Composing a high-level feature unique for each (flow) pattern is critical for classification tasks in this problem, and we have shown that it helps increase the chance to discover the characteristic burst sizes after backoff, flow control, and frame drop caused by CSMA.

1.3 Summary of Contributions

We summarize the main contributions of this thesis.

- **Sparse robust recovery.** This thesis extends sparse recovery used in compressive sensing for efficient denoising to recover the original data in fine granularity under extremely heavy subsampling. We formulate a distributed sensing problem to learn wideband spectrum occupancy information with largely unsynchronized, commodity narrowband sensors. Each sensor samples a partition of the spectrum and transmits
Chapter 1: Introduction

the measurement to the system backend over a communications network. We have proposed a scheme for in-network combining of compressed measurements from different sensors that minimizes the system communications cost and a novel recovery algorithm that undoes the combining and compression. Notably, the mixed compressed measurements could introduce both linear and nonlinear distortions. We have enabled the optimal compressive sensing with the Karhunen-Loève basis specific to each channel, which is learned via PCA;

- **Network traffic inference.** This thesis shows how the classical problem in networking can benefit from machine learning. We develop unsupervised methods to learn invariance from wireless traffic flow patterns that may survive through complex, nonlinearities introduced by the Wi-Fi CSMA. Using the learned invariance as characteristic features, we can infer the origin traffic pattern of a flow from packet sampling. We reaffirm that efficient discovery of invariance (characteristic burst and gap sizes) is critical for improving classification performance;

- **Deep architecture for sparse coding.** Despite superior performance of shallow sparse coding in feature learning, it is considered nontrivial to build a deep architecture for sparse coding. This thesis experiments with the potential of multilayer sparse coding and proposes Deep Sparse-coded Network (DSN) based on multilayer sparse coding and max pooling. With layer-wise pretraining followed by a novel backpropagation algorithm under the sparse coding framework, DSN significantly improves both classification accuracy and training speed compared to the conventional, backpropagation-trained deep learning schemes such as deep stacked autoencoder;

- **Nearest neighbor sparse coding (NNSC).** We have formulated the nearest neighbor constraint for better discriminative sparse coding and proposed enhanced algorithms for both ℓ₁-regularized and greedy-ℓ₀ sparse coders.

Based on these main ideas, substantial performance improvements in sensing, networking, and machine learning applications are possible.

2Alternatively, we can apply dictionary learning algorithms widely used in sparse coding.
Chapter 1: Introduction

1.4 Organization

We organize the rest of this thesis as follows. Chapter 2 provides a background on sparse modeling of data and related computational algorithms. We will also review compressive sensing and sparse coding. Chapter 3 presents compressive spectrum sensing for cognitive radios. In Chapter 4, we transition to discriminative data analysis and describe our feature learning framework based sparse coding. We will also explain a semi-supervised classification pipeline built on single-layer sparse coding and present empirical evaluation using text and image datasets. Chapter 5 extends our feature learning with multilayer sparse coding. We introduce a deep architecture for sparse coding, highlight its advantage, and discuss training methods. Chapter 6 presents nearest neighbor sparse coding, a simple but powerful enhancement to sparse coding for better discriminative feature learning. Chapter 7 will summarize the thesis with the conclusion and future work.
Chapter 2

Background

This chapter will provide a background on sparse modeling of data and related computational methods [11]. We will start by presenting two versatile usage models, namely compressive sensing and sparse coding. They are both known to enable novel applications in signal processing, inference, and pattern recognition. More importantly, they rely on fundamentally the same computing technique, which we will interchangeably refer as “sparse approximation,” “sparse decomposition,” “sparse recovery,” and “sparse computing” throughout the thesis. These are all popular keywords in machine learning, and we remark that our choice for calling one over the other is based on application context. Recovery would be more suitable for compression and reconstructive applications whereas coding or decomposition is for inference and feature analysis with data. We will also describe important sparse modeling problems and discuss solution approaches that subsequent chapters of this thesis depend on.

2.1 The Two Models

Let us consider the two models presented in Figure 2.1. The top model is known as compressive sensing [1, 2, 6] that encodes input data by multiplying them with a matrix. The result of the multiplication gives compressed measurements that are reduced in dimensionality. With a technique known as sparse recovery, the compressive encoding can be reversed to restore the original input. The model at the bottom is sparse coding [9]. In sparse coding,
Chapter 2: Background

Figure 2.1: Two sparse computing models: compressive sensing (top) requires a sparse recovery technique to restore the original data while sparse coding (bottom) uses essentially the same computational tool to obtain sparse representation of the data.

an input is encoded by the same technique used in the decode for compressive sensing. The encoded input values are known as sparse code that can be used as characteristic features of data. If used as a compression scheme, sparse coding decodes the original input by performing matrix multiplication. In later sections of this chapter, we will revisit each model with more rigorous mathematical review and application insights.

2.2 The Sparse Decomposition Problem

Sparse modeling is a selection problem formulated in an underdetermined linear system of equations. The system is ill-posed with more unknowns than equations that lead to infinitely many solutions. Therefore, additional constraints will be required to find a well-defined solution. Suppose a matrix $\mathbf{D} \in \mathbb{R}^{N \times K}$ of $K$ basis vectors $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_K\}$ with each $\mathbf{d}_i \in \mathbb{R}^N$ as a column. With $K > N$, we call the matrix $\mathbf{D}$ overcomplete. Given data $\mathbf{x} \in \mathbb{R}^N$, the aim of sparse modeling is to decompose $\mathbf{x}$ into a small number of basis vectors in $\mathbf{D}$. That is, we approximate $\mathbf{x}$ in a sparse linear combination (weighted sum) of the basis vectors. If we denote $\mathbf{y} \in \mathbb{R}^K$ the solution, then we have $\mathbf{x} \approx \mathbf{Dy} = \sum_{i=1}^{K} y_i \mathbf{d}_i$, where the elements $\{y_1, y_2, \ldots, y_K\}$ of $\mathbf{y}$ are mostly zero. Sometimes, it is beneficial to be explicit about sparsity of $\mathbf{y}$. We call $\mathbf{y}$ $S$-sparse when exactly $S$ elements of $\mathbf{y}$ are nonzero.
Chapter 2: Background

Formally, a solution for the sparse decomposition problem can be found by the following optimization

\[ \hat{y} = \arg \min_y \|x - Dy\|_2^2 + \lambda \psi(y) \]  

(2.1)

We remind that the overcomplete \( D \) makes the solution \( \hat{y} \) not unique for a general case. Underdeterminedness is resolved by the second term in Equation (2.1) known as regularization. In particular, sparse modeling regularizes the optimization by imposing a sparsity constraint on \( y \). There are two strategies for sparsity-inducing \( \psi(.) \) in the equation, the \( \ell_0 \)-norm \( \|y\|_0 \) and the \( \ell_1 \)-norm \( \|y\|_1 \). Because the \( \ell_0 \)-norm of a vector is the number of its nonzero elements, it can precisely accomplish the regularization purpose of the sparse decomposition problem.

However, finding the minimum \( \ell_0 \)-norm solution for Equation (2.1) is known to be NP-hard. Often, the \( \ell_1 \)-minimization approaches are preferred instead. The \( \ell_1 \) approach can be thought as convex relaxation of the \( \ell_0 \) approach. There are numerous tractable algorithms to solve the \( \ell_1 \)-regularized sparse decomposition problem such as linear programming [23] and Least Absolute Shrinkage Selection Operator (LASSO) [24]. Recently, it is known that the \( \ell_0 \) approaches are not entirely impractical. One can use a greedy algorithm such as Orthogonal Matching Pursuit (OMP) [25] to solve for an exactly \( S \)-sparse \( y \) efficiently.

When the solution found by the \( \ell_1 \)-minimization is sufficiently sparse, it will be the same as the solution found by the original, NP-hard \( \ell_0 \) problem. Despite higher computational cost than OMP, convex optimization over the \( \ell_1 \)-norm is known to yield a better solution in terms of reconstructive accuracy. It is also proven more robust to noise, especially when a very small number of measurements are used.

2.3 Approaches to Solve Sparse Decomposition Problem

We briefly review three computational approaches to solve the optimization problem in Equation (2.1).
Chapter 2: Background

2.3.1 LASSO

Least absolute shrinkage and selection operator, or simply LASSO, was introduced in 1996 by Tibshrani [24]. Originally, LASSO was conceived in the search for an alternative regularization method for least squares along ridge regression.\(^1\) At the core, LASSO is the \(\ell_1\)-regularized least squares problem

\[
\min_y \| x - Dy \|_2^2 + \lambda \| y \|_1
\]

where \(\lambda\) is the regularization parameter that penalizes over sparsity of the solution. The computation of the LASSO solutions is by quadratic programming (QP) [27]. A QP problem can be solved by standard numerical algorithms. A well-known, better approach among them is the least angle regression (LARS) [28] that simultaneously computes the solutions under different values of the regularization parameter. Throughout the thesis, we treat LASSO and LARS the same. That is, unless otherwise stated, we use the LARS algorithm to solve for the \(\ell_1\)-regularized sparse coding.

2.3.2 Basis pursuit

Chen, Donoho & Saunders [29] describe an interior point method to solve a slightly modified optimization in the following

\[
\min_y \| y \|_1 \text{ s.t. } x = Dy.
\]

The approach is known as basis pursuit. Computationally, basis pursuit is via linear programming, which is quite different from Tibshirani’s original method for LASSO or from LARS. However, the equivalence of solutions computed by LASSO/LARS and basis pursuit was first recognized by Sardy et al. [30]. Main idea behind basis pursuit is to perform a stepwise Newton’s method on the Karush-Kuhn-Tucker (KKT) conditions of the linear program for Equation (2.3). The main advantage of basis pursuit is fast convergence to a

\(^{1}\)Ridge regression [26] regularizes on the \(\ell_2\)-norm of the solution in Equation (2.1). Unfortunately, the \(\ell_2\)-regularized \(y\) will not be sparse.
solution with suitable precision although it requires to initialize the system to a good region. Another disadvantage is a computational complexity of $O(K^3)$ [31]. (Note $K$ is the number of the unknowns, \textit{i.e.}, dimensionality of the solution $y$.)

### 2.3.3 Matching pursuit

The $\ell_0$-regularized sparse decomposition is hard because exhaustive combinatorial search for the sparsest $y$ is too expensive and beyond polynomial time. Matching pursuit [32] chooses to work around the $\ell_0$-norm with greedy iterative algorithms. Instead of costly global search, when sparsity level $S$ is given as input, a greedy-$\ell_0$ algorithm takes locally optimal updates to construct an $S$-sparse solution in exactly $S$ iterations by

$$
\min_y \| x - Dy \|_2^2 \text{ s.t. } \| y \|_0 \leq S. \quad (2.4)
$$

Algorithm 1 presents Orthogonal Matching Pursuit (OMP) [25], one of the most efficient greedy algorithms that works well for sparse data modeling. Each iteration of OMP selects a basis vector. The inner product between each basis vector and the current residual vector indicates their similarity, and the basis vector resulted the largest inner product is selected to minimize the residual error (\textit{i.e.}, achieving the maximum correlation). With the support set of selected basis vectors, the least squares approximate the solution vector at the $i$th iteration $y^i$. The residual vector is updated using $y^i$ just computed, and the process runs for exactly $S$ iterations.

---

**Algorithm 1** Orthogonal Matching Pursuit (OMP)

**input** Input vector $x$, matrix of bases $D$, sparsity requirement $S$

**output** $S$-sparse output vector $y$

1: Initialize residual vector $r^0 = x$
2: \textbf{for} $i = 1 \text{ to } S$
3: \hspace{1em} Select basis vector $d_{k_i}$ s.t. $k_i = \arg \max_k |\langle d_k, r^{i-1} \rangle|$
4: \hspace{1em} Approximate by least squares $y^i = \arg \min_y \| x - \sum_{j=1}^{i} d_k y_j \|_2$
5: \hspace{1em} Compute new residual $r^i = x - Dy^i$
6: \textbf{end}
Chapter 2: Background

2.4 Compressive Sensing

A conventional approach in sensing and data acquisition requires to measure (or sample) the source at its Nyquist rate before applying any compression or signal processing techniques. By exploiting a sparse structure either exposed naturally or hidden in data, compressive sensing makes the source measurement possible at a substantially lower rate than the Nyquist rate. Anomalous sensor reading that creates an instantaneous spike is an example of naturally exposed sparsity, whereas a modulated RF signal not sparse in the time domain that consists only of several tones in the frequency domain gives an example of hidden sparsity. Compressive sensing can be applied to these examples for efficiently capturing all essential information from the sources. Key intuition of compressive sensing is that the actual information rate of the source falls much below twice of the highest frequency of the source recommended by Nyquist-Shannon sampling theorem [33].

With compressive sensing, we can directly capture all significant coefficients of the data without prior analysis to remove redundancy (e.g., transform coding [34]). More importantly, the compressive sensing encode does not depend on the knowledge about how to sparsify the data. On the contrary, conventional compression schemes such as entropy coding are all about leveraging the sparsifying bases of the data, which will be learned by conducting a comprehensive statistical analysis for the data. Compressive sensing combines the measurement and compression of the data into one non-analytical, low-complexity encoding process governed by matrix-vector multiplications. In particular, compressive sensing uses an overcomplete sensing matrix of randomly generated numbers. With high probability, such random projection is believed to preserve incoherency of the data in its sparse domain. This is why recovery of the original data is possible for compressive sensing.

Now, suppose data\(^2\) \(x \in \mathbb{C}^N\) and a sensing matrix \(\Phi \in \mathbb{R}^{M \times N}\), where \(M \ll N\) is the number of measurements. Compressive sensing encodes \(x\) into compressive measurements \(y = \Phi x\). Note that this is an \(N\) to \(M\) compression. For cases where \(x\) can be sparsified through a transform \(\Psi \in \mathbb{C}^{N \times N}\), the compressive sensing encode still takes place without the knowledge of \(\Psi\). The sparsification of \(x\) is given by \(s = \Psi x\). In \(s\), there are only

\(^2\)Compressive sensing works for both real- and complex-valued input data.
Chapter 2: Background

$S \ll N$ nonzero elements—i.e., $x$ is $S$-sparse in $\Psi$. Compressive sensing forms the underdetermined linear system of equations, which can search for a solution by the following optimization

$$\min \limits_{s} \|s\|_0 \text{ s.t. } y = \Phi x = (\Phi \Psi^{-1}) s.$$  

(2.5)

This is the familiar $\ell_0$-regularized regression problem, which is known intractable. One possible approach is to rely on convex relaxation that regularizes the $\ell_1$-norm of $s$

$$\min \limits_{s} \|s\|_1 \text{ s.t. } y = \Phi x = (\Phi \Psi^{-1}) s,$$

(2.6)

which we can decode by basis pursuit via linear programming. On another, Needell & Tropp [35] proposed CoSaMP (dubbed compressive sensing matching pursuit), which is much faster than basis pursuit with the running time of $O(N \log^2 N)$. Interestingly, CoSaMP claims the guarantees equivalent to the best optimization-based approaches.

Once $s$ is recovered, we can restore $x$ from the reverse sparsification $x = \Psi^{-1} s$. Restricted Isometry Property (RIP) [36] of $\Phi$ can guarantee a unique solution with high probability through the $\ell_1$-minimum decoding of $s$ [2]. It is known that the product $\Phi \Psi^{-1}$ satisfies RIP with high probability if $\Phi$ satisfies RIP, and $\Psi$ is orthogonal. Robust uncertainty principle [37] states that $M$ needs to be at least $cS \log \frac{N}{S}$ for some small constant $c > 0$ to make the exact recovery possible. The quality of recovered values depends on $M$ and is incremental. Using a smaller $M$ (than required for exact solution) does not disqualify the decoded result entirely. If not all, a portion of larger components in the recovered solution would still be accurate. (How many of them are accurate will depend on the actual $M$ used.) The majority of decoding error is contributed by the unrecovered components, and the sum of their magnitudes quantifies the error. If desired accuracy is not met, one can increase $M$ accordingly. Fixing $M$, however, by no means fixes the decoding accuracy because the use of a better sparsifying basis for the data can still improve the performance. Since the choice of sparsifying basis is critical, learning an optimal basis would be the key to enhance the compressive sensing decode.

Compressive sensing finds a broad range of applications in signal processing [38], computational mathematics [39], anomaly detection [40, 41], group testing [42], and imag-
Chapter 2: Background

ing [43]. One of the earliest applications is reflection seismology that uses sparse reflected signals from band-limited data for tracking changes between subsurface layers [44]. Imaging techniques such as coded aperture and computational photography have a strong affinity with compressive sensing. Also, native hardware implementations of compressive sensing are becoming more readily available.

2.5 Sparse Coding

Sparse coding has originated from computational neuroscience. Olshausen and Field in 1996 proposed sparse coding to explain the human visual cortex computational model in understanding natural images [45]. Since then, sparse coding has been widely received in machine learning, statistics, neuroscience, and signal processing. Recently, it has gained popularity in computer vision, achieving state-of-the-art performance on face recognition [46], texture segmentation [47], and object recognition [48, 49].

Sparse coding is a general class of unsupervised methods to learn efficient representations of data, expressed as a linear combination of basis vectors spanning the columns of a given dictionary matrix. Sparse coding provides a primitive to extract features of raw data. Sparse coding takes in patches, unit input vectors drawn from the data. Given a patch $x \in \mathbb{R}^N$ and dictionary $D^{N \times K}$, sparse coding solves for a representation $y \in \mathbb{R}^K$ that fits $x = Dy$. As reviewed earlier, LASSO/LARS or OMP can be used as an encoder for sparse coding.

Dictionary learning is essential for sparse coding. During unsupervised feature learning, sparse coding and dictionary learning alternate. We perform sparse coding by holding the dictionary $D$ constant and compute the sparse codes for the input training examples. After sparse coding finishes, dictionary updates follow. Consider the following pseudo-code for an alternating algorithm to estimate $D$

Repeat until convergence:

(Sparse coding) $y^{(i)} = \text{sparse\_coding}(x^{(i)}, D, \lambda \text{ or } S)$ $\forall i$

(Dictionary updates) $D = \arg \min_D \|X - DY\|_F^2 \text{ s.t. } \|d_k\|_2^2 = 1 \forall k$
We note that unsupervised feature learning via sparse coding and dictionary learning is a fully data-driven process. Like sparse coding encode, there are numerous algorithms for dictionary learning such as K-SVD [50].

Sparse coding can be thought as a generalization of K-means clustering [21] that hard-assigns each training example to one cluster. Each dictionary atom is equivalent to a cluster centroid, and sparse coding computes a distributed representation (i.e., soft clustering) that describes each \( x^{(i)} \) using a small number of dictionary atoms. Unlike PCA [21], which is also an unsupervised learning method, sparse coding can learn an overcomplete set of basis vectors. Sparse coding can model inhibition between the bases by regularizing the number of activations. This is similar for the case of biological neurons, which makes sparse coding a plausible model to explain the human visual cortex.
Chapter 3

Compressive Spectrum Sensing for Cognitive Radios

This chapter presents an application system for wideband spectrum sensing desirable in the emerging dynamic spectrum access (DSA) paradigm for cognitive radios. We formulate a distributed sensing problem to learn wideband spectrum occupancy information with largely unsynchronized, commodity narrowband sensors. Each sensor samples a partition of the spectrum and transmits the measurement over a communications network (sensor backhaul) to the system backend where the data recovery takes place.

The purpose of recovery is to compute the frequency response of the entire spectrum equivalent to an FFT analysis based on Nyquist sampling. This chapter describes a salient approach for compressive data gathering that combines multiple compressed measurements from different sensors in-network. In particular, the combining operation is the arithmetic addition free of overhead.

The main contribution of this chapter is a novel sparse recovery algorithm that works in two phases undoing the additive mix and refining the data recovery. Overall, compressive sensing and in-network combining result in heavily subsampled RF measurements pushed beyond compressive sensing. Our recovery algorithm, however, is found robust to such complex transformations on the data. Together with reduced communications cost for loosely coordinated, distributed sensor nodes, our sparse recovery enables a significant improvement for sensor network whose backhaul is bandwidth-limited.
3.1 Introduction

The most precious resource in building a wireless network is spectrum. Over the past decade, cognitive radios [14] have arisen commercially, suggesting a new means to alleviate the spectrum shortage problem. The fundamental premise of cognitive radios is the use of spectrum sensing mechanism to dynamically discover vacancies in the spectrum usage. Under Dynamic Spectrum Access (DSA) [13], an unlicensed radio (called secondary user) is granted an opportunistic access of a licensed spectrum, provided that the secondary user has a proper sensing mechanism to detect the licensees of the spectrum (i.e., the primary users) and yield discreetly.

With spectrum sensing as a key requirement for DSA, much of cognitive radio research has centered around the development of an accurate sensing method. Technically, a spectrum analyzer can provide the ideal sensing capability required by DSA. It measures a radio channel and computes the energy of signals present across the channel frequency range. Spectrum analyzers can provide additional channel characteristic information such as dominant frequency and harmonic distortion, which are not readily discernible in a simple time domain-based approach such as energy detection. Commercial spectrum analyzers are specialized hardware equipped with fast RF swept-tuners and a dedicated Fast Fourier Transform (FFT) to compute the frequency response in real-time.

In this chapter, we present a spectrum analyzer system built on networked sensor nodes. We aim to reconstruct accurate power spectral densities of a wideband spectrum spanning hundreds of MHz to a few GHz in bandwidth from aggregating measurements by individual sensors. The sensors are loosely coordinated and participate in asynchronous data collection. We assume that the sensors are simple, general-purpose commodity devices and may not be dedicated for spectrum analysis (they could be user devices operating for own good).

Our goal is different from recent work such as Microsoft SpecNet [51] that connects conventional spectrum analyzer equipments to the Internet for shared use. The main contribution of SpecNet is an API to operate conventional spectrum analyzer equipments remotely. We focus on partitioning RF measurements, compressive sampling, and combining the compressed measurements for efficient transport to the distant system backend for re-
covery and analysis. In particular, we introduce a novel sparse recovery algorithm that restores the original data by systematically undoing the in-network mix of multiple, compressed channel measurements.

### 3.1.1 Challenges

In digital signal processing, the Nyquist-Shannon sampling theorem [33] dictates the ideal recovery of a measured signal, which needs to be sampled at least twice as frequently of its bandwidth. For example, a signal with 1-MHz bandwidth has the Nyquist rate \( f_s^* = 2 \times 1 \, \text{MHz} \), which means that any two successive measurements should be spaced by time \( T_s^* = 1/f_s^* = 1/(2 \, \text{MHz}) = 0.5 \mu\text{sec} \) to prevent a loss of information. That is, the Nyquist sampling on the signal results in a raw data rate of \( 2 \times 10^6 \times \text{sample size} \) bits/sec. The majority of broadband wireless standards today uses digital I-Q (in-phase and quadrature) modulation that represents each signal sample with complex numbers. Assuming 20-bit complex samples, uninterrupted plain Fourier analysis of a 1-MHz spectrum needs to keep up with a 40-Mbps data stream, which is somewhat surprisingly high.

Fast Fourier Transform (FFT), an efficient numerical algorithm to compute Discrete Fourier Transform (DFT), is at the heart of spectrum analysis. The computational complexity of an \( N \)-point FFT is \( O(N \log N) \). FFT takes a batch of \( N \) samples in the time domain and produces \( N \) frequency components each of which is used to compute the power spectral density of the spectrum. Spectrum analyzers denote resolution bandwidth \( rbw = B/N \) the spacing between two consecutive frequency components in the analysis. Resolution bandwidth determines the fidelity of an analysis—i.e., smaller the resolution bandwidth, finer the granularity. For the 1-MHz example, a fine resolution bandwidth of few kHz will require \( N \approx 512 \) (FFT commonly performed in a power of 2). The real challenge would be fine-grained analysis of a wideband spectrum in several orders of magnitude larger than the 1-MHz example.

### 3.1.2 Problem statement

Imagine sensor nodes in a wireless network connected through a base station or an ad hoc fashion as illustrated in Figure 3.1. There is a backend of the system that runs spec-
Chapter 3: Compressive Spectrum Sensing for Cognitive Radios

Figure 3.1: A wideband spectrum (e.g., $B_{\text{tot}} \approx 100s$ of MHz to a few GHz) under analysis is partitioned for collective measurement efforts by multiple sensor nodes. Spectrum analysis on the measured data takes place at the distant system backend.

Spectrum under analysis with total bandwidth $B_{\text{tot}}$

![Diagram](image)

Sensors can use multi-hop relay to send measurements

Sensor measurements are transmitted wirelessly and aggregated

System backend for spectrum analysis

Sensor measurements are forwarded over network to spectrum analyzer

Network node (Base station)

Network

The system partitions the spectrum into $J$ subchannels with bandwidths, $B_1, \ldots, B_J$, such that $B_{\text{tot}} = \sum_{i=1}^{J} B_i$. For simplicity, assume equipartition of the spectrum $B_i = B \forall i$ (i.e., $B_{\text{tot}} = JB$), and adjacent subchannels do not overlap. A communication protocol $\mathcal{P}$ is used between the system and a node to administer sensing tasks, or between nodes to forward data. The system assigns a sensing task $(f_l, B_l)$ to sensor node $l$. This means that sensor $l$ should tune to $f_l$, the center frequency of the assigned subchannel, and start sampling according to bandwidth $B_l$. Sensor $l$ periodically yields a measurement vector $x_l \in \mathbb{C}^N$.

In this system model, the sensor measurements are transmitted to the base station first and forwarded to the backend. If there are exactly $J$ sensor nodes with each measuring a unique subchannel, we have a total of $JN$ measurements for the entire spectrum per measurement cycle. Our objective is to minimize the total number of measurements transmitted to the system, possibly well under $JN$ samples per each cycle required for straightforward FFT analysis on the Nyquist rate, yet without giving up much fidelity.
Chapter 3: Compressive Spectrum Sensing for Cognitive Radios

There are two types of nodes in the system, sensor or network node. Sensor nodes transmit their measurements either directly to the base station as depicted by ‘white’ circles in Figure 3.2(a) or to a network node that can relay the measurements for others as in Figure 3.2(b). Networking functionality determines the type of a node. Sensor nodes are end-nodes not configured to forward other nodes’ data. On the contrary, network nodes can forward the measurements received from others, using multi-hop relays to the base station as represented by ‘black’ circles in Figure 3.2(b). Network nodes help save the uplink bandwidth to the base station by combining or encoding multiple sets of measurements received from other nodes that eventually reduces the total transmission to the base station.

The communication cost of spectrum analysis for the described system is equivalent to the total number of measurements transmitted to the base station either directly from sensor nodes or through network nodes. For a given reconstruction algorithm, we can express the problem concisely in the following optimization

$$\min_{\theta} \sum_{i=1}^{J} \dim(y_i = \theta(x_i)) \quad \text{s. t.} \quad \|X(f_k) - \hat{X}(f_k)\|_2 \leq \epsilon,$$

where \(\theta(.)\) is a dimensionality reduction function we seek, which makes \(\dim(y_i) \ll \dim(x_i)\). \(X(.)\) is the actual (ground-truth) frequency response of \(x\), and \(\hat{X}(.)\) the estimate based on \(y_i\)’s. Note that \(f_k\) is the \(k\)th frequency component index in the combined \(J \times N\)-point FFT, where \(0 \leq k < J \times N - 1\). The minimization is subject to a performance constraint that the error of the reconstructed frequency response should be bound by some small constant \(\epsilon\). We consider the FFT-analysis on Nyquist sampling the ground-truth frequency response to compare against.

3.1.3 Related work

This chapter addresses similar problems considered by distributed compressed sensing [52], compressive wireless sampling [53], data gathering for sensor networks [54], and collaborative spectrum sensing [55, 56]. In particular, Polo et al. [56] introduced an early wideband spectrum sensing framework that uses multiple, distributed cognitive radios to simultaneously sense a block of radio subchannels with compressive measurements. Wang
Chapter 3: Compressive Spectrum Sensing for Cognitive Radios

Figure 3.2: In-network transmission of measurements

et al. [57] extended the framework, describing a complete system and decoding schemes that rely on the combined use of analog-to-information conversion (AIC) hardware and the SOMP algorithm [52]. Neither work focused on manipulating compressed measurements directly for further networking efficiency, which was critical for our case. The essence of our approach is to relieve sensor nodes from any significant computations other than linear projection. Network nodes operate in a carefree manner to combine compressed measurements by arithmetic additions only. Our algorithm recovers an accurate estimate of the original data from a mix of multiple compressed measurements by separating and refining each set in a novel iterative procedure.

In recent work, Iyer et al. [51] proposed SpecNet, a network of conventional spectrum analyzers, sharing and using expensive lab equipments in new ways through an API such as remote spectrum measurement, learning transmitter statistics of a wireless channel in distant geographic regions. SpecNet, however, is not a spectrum analyzer per se. Its primarily focus is on the development of front-end user interface for remote spectrum measurements, which is different from what we aim to achieve.

3.2 Minimizing Communications Cost of Sensor Measurements

The underlying principle for reducing communications cost is to keep simple encoding schemes for multiple, parallel measurements by the sensors and take advantage of a mea-
Chapter 3: Compressive Spectrum Sensing for Cognitive Radios

sured signal’s sparsity by compressive sensing. This section explains the two approaches integrated in our system.

3.2.1 Exploiting signal sparsity with simple compressive encoding

Compressive sensing can exploit a concealed sparse structure of the measured signal with an optimistic assumption that there will be a suitable basis available for recovery. We treat learning of the basis vectors separately for now, but it can be done offline. Such optimistic use of compressive sensing provides us a key advantage for data collection using many nodes in a wireless sensor network. Moreover, simple encoding process can gracefully embrace our case where commodity sensors and network nodes have limited computational capabilities. In contrast, conventional compression schemes such as joint entropy coding require solving a complex optimization problem at nodes.

3.2.2 Combining multiple measurements

In addition to compressive sensing, we can combine multiple compressed measurements into one. Figure 3.3 illustrates how compressive measurements from $P$ sensors can be combined in-network. Here, we stress that combing operation is no more than the arithmetic sum, keeping only track of which sensor measurements are added. The intuition behind our simple additive combining is to minimize the burden of a network node. We define the $P$-way combined compressive measurements $y$ from $P$ different sensors

$$y = y_1 + \ldots + y_P = \sum_{i=1}^{P} y_i,$$

where $y_i = \Phi x_i \in \mathbb{C}^M$ is the compressive measurement for uncompressed $x_i \in \mathbb{C}^N$ from sensor $i$ using sensing matrix $\Phi \in \mathbb{R}^{M \times N}$ with $M \ll N$.

The top black node in Figure 3.3 will produce the final sum $y$ and transmit it to the base station. The $P$-way additive mixing holds the number of measurements transmitted to the system base station constant, regardless of the number of sensors (up to $P$) whose measurements are combined. The overall saving in bandwidth is roughly by a factor of $P$. 

24
Figure 3.3: An example for $P$-way mixing of compressive measurements (note: $x_i \in \mathbb{C}^N$, $y_i \in \mathbb{C}^M$, and the $P$-way sum $y \in \mathbb{C}^M$)

3.3 System Operation

3.3.1 Preparation phase for basis estimation

Compressive sensing requires a basis for recovery, and we consider the optimal basis decoding via principal component analysis (PCA). In a separate learning process for each subchannel, the system estimates its optimal basis offline. Fundamentally, PCA performs Karhunen-Loève (KL) expansion on the sample covariance matrix $C_i = \frac{1}{n-1} \sum_{j=1}^{n} x_i^{(j)}x_i^{(j)^T}$ for channel $i$, assuming we have collected sufficiently large $n$ measurements for the channel. Eigendecomposition of the sample covariance matrix $C_i = Q_i \Lambda_i Q_i^T$ gives $Q_i$, the optimal basis for compressive sensing decoding for channel $i$. That is, we can recover $x_i$ from its compressed measurements $y_i = \Phi x_i$, using optimal basis $Q_i$ by

$$y_i = \Phi x_i = (\Phi Q_i) s_i \quad (3.2)$$
where $s_i$, the sparse representation (the KL transform) of $x_i$ in basis $Q_i$, is recovered first. The recovery of $x_i$ follows by the inverse KL transform $x_i = Q_i s_i$.

For stable estimation, preparation phase can use a long measurement period. Reestimating the optimal basis does not need to be scheduled too frequently as we can expect that the locations of dominant eigenvectors (i.e., leading principal components) remain relatively stationary over time for modulated RF signals. Nonetheless, we can opt for efficient, incremental updates to the optimal basis by popular rank-1 algorithms such as K-SVD [50].

### 3.3.2 Recovering original data

The backend of our system receives an arbitrary mix of $P$ compressed measurements from disjoint subchannels. We explain logical steps to undo the mix and recover accurate estimates for the original data.

**Initial approximation via least squares.** The basis estimation via PCA identifies dominant eigenvectors (i.e., principal components) for each channel. When compressed measurements arrive as a $P$-way sum, the system first sets up the overdetermined linear system of equations

$$y = \Phi \begin{bmatrix} \tilde{Q}_1 & \tilde{Q}_2 & \cdots & \tilde{Q}_P \end{bmatrix} \begin{bmatrix} \tilde{s}_1 \\ \tilde{s}_2 \\ \vdots \\ \tilde{s}_P \end{bmatrix}$$

(3.3)

where $y$ is the $P$-way sum of compressive measurements, and $\tilde{Q}_i$ is zero-padded truncation of $Q_i$ such that nonzero values of $\tilde{Q}_i$ are from only $\alpha_i$ leading principal components corresponding to the largest $\alpha_i$ leading eigenvalues. The overdetermined system in Eq. (3.3) is possible because of the basis learning by PCA. We compute the initial approximation by the least squares, a method well-known for overdetermined equations.

**Iterative refinement via compressive sensing decoding.** The initial approximation is coarse. Its primary purpose is to separate the sum commensurate with the amount of energy
reflected in each channel’s leading principal components. We refine the initial approximation by an iterative process based on compressive sensing decoding. We can relax each $s_i$ at a time by isolation. This can be done by backsubstituting $(P-1) \tilde{s}_i$’s with their initial approximation and subtract them from the $P$-way sum $y$. We apply compressive sensing decoding on the isolated $s_i$ and take the decoded result as the refined estimate. For example, we relax $s_1$ by solving

$$y = \Phi [Q_1 \tilde{Q}_2 \cdots \tilde{Q}_P]$$

(3.4)

where the rest $\hat{s}_i$’s are the initial approximates used for backsubstitution. After refining $s_1$, we refine the next, say $s_2$. For $s_2$, we backsubstitute $s_1$ with $\hat{s}_1$, which is the best available estimate from the first refinement replacing its initial approximate instead, and $\tilde{s}_3, \ldots, \tilde{s}_P$ with $\hat{s}_3, \ldots, \hat{s}_P$ yet from the initial approximation:

$$y = \Phi [Q_1 Q_2 \tilde{Q}_3 \cdots \tilde{Q}_P]$$

(3.5)

In our refinement, we have managed to convert the overdetermined system in Equation (3.3) to $P$ underdetermined systems such as in Equations (3.4) or (3.5), each of which is recoverable by compressive sensing decode. After iterating through all $P$ $s_i$’s, another stage of $P$ refinements can take place. The new stage uses the results from the previous stage.
3.3.3 Algorithms

We formally present the initial approximation and iterative refinement algorithms that are used to recover original data from the $P$-way sum of compressed measurements.

Algorithm 2 Initial approximation

**require** Optimal bases $Q = \{Q_1, \ldots, Q_P\}$ for $P$ channels learned via PCA

**input** $P$-way sum $y$ of compressed measurements

**output** Initial approximate $\hat{s}^0 = [\hat{s}^0_1 \ldots \hat{s}^0_P]^\top$ for underdetermined system $y = \Phi Q s$

1: Obtain $\tilde{Q} = [\tilde{Q}_1 || \tilde{Q}_2 || \ldots || \tilde{Q}_P]$ by zero-padded truncating $\{Q_1, \ldots, Q_P\}$ with only $\alpha_i$ leading principal components unchanged for each channel;

2: Construct $\tilde{Q} = [\tilde{Q}_1 || \tilde{Q}_2 || \ldots || \tilde{Q}_P]$;

3: Solve for $\hat{s}^0 = [\hat{s}^0_1 \ldots \hat{s}^0_P]^\top$ via least squares on overdetermined system $y = \Phi \tilde{Q} \hat{s}$.

Algorithm 3 Iterative refinement

**require** Optimal bases $Q = \{Q_1, \ldots, Q_P\}$ for $P$ channels learned via PCA

**input** $P$-way sum $y$ of compressed measurements, initial approximate $\hat{s}^0 = [\hat{s}^0_1 \ldots \hat{s}^0_P]^\top$ for underdetermined system $y = \Phi Q s$, number of iterative states $T$

**output** Refined estimate $\hat{s}^T = [\hat{s}^T_1 \ldots \hat{s}^T_P]^\top$ after $T$ refinement stages

1: $t = 1$;

2: while $t \leq T$

3: Sort $\hat{s}^{t-1}_i$’s by $\ell_1$-norm and obtain $\hat{s}^{t-1}_{(i)}$’s s.t. $||\hat{s}^{t-1}_{(1)}||_1 \geq \ldots \geq ||\hat{s}^{t-1}_{(P)}||_1$;

4: Sort $\hat{y}^{t-1}_i$’s in same order as $\hat{s}^{t-1}_{(i)}$’s to obtain $\hat{y}^{t-1}_{(i)}$’s;

5: for $i = 1$ to $P$

6: Compute $\hat{y}^t_{(i)} = y - \sum_{k=1}^{i-1} Q_{(k)} \hat{s}^t_{(k)} - \sum_{l=i+1}^P Q_{(l)} \hat{s}^{t-1}_{(l)}$;

7: Do compressive sensing decode on $\hat{y}^t_{(i)} = \Phi Q_{(i)} \hat{s}^t_{(i)}$ and obtain $\hat{s}^t_{(i)}$;

8: end

9: Obtain $\hat{s}^t_i$’s in original order from $\hat{s}^t_{(i)}$’s;

10: $t := t + 1$;

11: end

3.3.4 Comparison with joint decoding

Given separate decoding bases for $P$ subchannels, we can also apply a simple joint decoding algorithm to recover the original data from the mixed compressed measurements. We
can stack the unknowns in the $P$-way sum as

$$y = \Phi x_1 + \ldots + \Phi x_P = \Phi \begin{bmatrix} Q_1 & \ldots & Q_P \end{bmatrix} \begin{bmatrix} s_1 \\ \vdots \\ s_P \end{bmatrix}.$$  \hspace{1cm} (3.6)

By letting $Q = [Q_1 \ldots Q_P]$ and $s = \begin{bmatrix} s_1 \\ \vdots \\ s_P \end{bmatrix}$, this is just standard compressive sensing $y = \Phi Q s$, but with important distinction. Due to stacking, joint decoding needs to deal with increased dimensionality $N \times PN$ for $Q$ and $PN \times 1$ for $s$. In other words, joint decoding requires to solve for an increased number of unknowns, $PN$ instead of $N$, in one-shot.

Joint decoding can be implemented easily, but the increased number of unknowns can be a significant drawback. Since the $\ell_1$-minimization decoding for compressive sensing has a known complexity of $O(N^3)$ [6], joint decoding for the $P$-way mix will have a $O(P^3N^3)$ complexity. On the other hand, our proposed recovery algorithms have $O(\alpha^2 P^3 N)$ by from least squares during the initial approximation and $O(TPN^3)$ from compressive sensing decode for $T$ iterations during the refinement stage. We note that $P$ is at least an order of magnitude smaller than $N$. Denoted $\alpha$, the number of leading principal components per subchannel used for initial approximation is also a small value. Therefore, the computational complexity of our algorithms will be dominated by the iterative refinement stage attributed to compressive sensing decode.

### 3.4 Evaluation

#### 3.4.1 Experimental methodology

We have evaluated the empirical performance of the proposed spectrum analyzer system implemented using USRP2 and N200 nodes [58] with GNU radio [59]. We set up 4 USRPs
in an indoor lab and analyzed a 200-MHz UHF spectrum\(^1\). The spectrum is partitioned into \(J = 8\) subchannels (i.e., each with 25-MHz bandwidth) whose center frequencies are 
\[f_c \in \{512.5, 537.5, 562.5, 587.5, 612.5, 637.5, 662.5, 687.5\} \text{ MHz}.
\] We configure each USRP to cycle between two subchannels, responsible for taking measurements at both frequencies. Note that the Nyquist sampling rate of a subchannel is 25 MHz \(\times 2 = 50 \times 10^6\) samples/sec. For each measurement instance, we use \(N = 512\) per subchannel (i.e., the resolution bandwidth of 25 MHz/512 \(\approx 48.8\) kHz). This corresponds to \(L = J \times N = 4096\)-point frequency response for the entire spectrum contributed by all USRPs.

For compressive sensing, we have pre-generated a set of \(\Phi\) matrices and stored in each USRP. For each subchannel, the USRPs can take compressive measurements using configurable \(M\) varied from 26 (\(M/N = 5\%\) or 20x compression) to 308 (60\% or 1.67x). USRPs use the same \(\Phi\) under each configuration. We have saved the ground-truth uncompressed original data for analysis. The uplink transmission to the base station from an USRP is simulated. Our experimental setup is illustrated in Figure 3.4.

The USRPs are not synchronized by a common clock source. Measurement start and end times for each USRP cannot be determined exactly. This is similar to a swept-tuned spectrum analyzer that measures a narrow subchannel one at a time, staying for a short duration before moving to next and cycling the entire spectrum. When changing radio fre-

---

\(^1\)It spans more than 33 UHF channels (6 MHz each) between Ch. 18 and Ch. 52.
quency, the USRPs seem to incur at least a 1-second delay related to reconfiguring the internal voltage controlled oscillator (VCO).

### 3.4.2 Evaluation scenarios and metric

We evaluate the following scenarios:

1. **No combining.** USRPs perform compressive sensing on their assigned subchannels and transmit the compressive measurements directly to the base station. The base station collects 8 separate transmissions for each complete measurement cycle for the entire spectrum;

2. **P-way combined compressive measurements.** Varying $P = 2$, 4, and 8, the compressed measurements from USRPs are combined accordingly. There are $\frac{8}{P}$ transmissions per each measurement cycle.

We compare the decoding performance of the proposed approach (*i.e.*, Algorithms 2 and 3) to no combining as well as joint decoding. We measure the decoding performance using the average normalized frequency error metric defined by

$$
\xi = \frac{1}{L} \sum_{k=1}^{L} \frac{\|X(f_k) - \tilde{X}(f_k)\|_2}{\|X(f_k)\|_2}
$$

(3.7)

where $X(f_k)$ is the frequency response of the spectrum under analysis from the Nyquist sampling (no compression and no manipulation of original data), and $\tilde{X}(f_k)$ the recovered frequency response from a respective scheme. Frequency indices $f_k \in [500, 700) \text{ MHz}$ with $k = 0, \ldots, L - 1$ follow the discrete Fourier analysis convention.

### 3.4.3 Decoding accuracy and complexity

Figure 3.5 presents the number of measurements versus the recovery error (measured by the error metric $\xi$). For the proposed $P$-way combining schemes, we have used $T = 2$ stages of iterative refinements. To highlight the effect of combining, we plot the result for no combining scheme that uses compressive sensing only. Given an error budget, the
Figure 3.5: Total number of measurements used in spectrum analysis vs. accuracy of decoding measured by error metric $\xi$. The proposed schemes run $T = 2$ refinement stages. Joint decoding for 8-way mix requires the number of measurements on par with the proposed 8-way combining scheme for similar error performance, but joint decoding is much harder computationally and incurs significantly longer decoding time.

The proposed 8-way combining scheme can achieve 3- to 5-fold saving in the communication bandwidth compared to no combining. The full comparison of the proposed schemes versus no combining is presented in Figure 3.6. In Figure 3.7, we plot frequency responses (i.e., spectrum analyzer display) of the spectrum reconstructed from uncompressed original time samples (4096 measurements), compressed but no combining scheme (832 measurements, 4.9x compression), and 8-way combined (208 measurements, 20x compression) for visual comparison.

The proposed algorithm is only slightly better than joint decoding in the error performance. However, the proposed algorithm is much better in computational complexity. To decode 8-way combined measurements, joint decoding requires to decode $8 \times 512 = 4096$ variables at once. The proposed algorithm approximates 64 unknowns (i.e., $\alpha_i = 8$ leading principal components for each subchannel) via least squares. Then, we have 8 compressive sensing decodes for only $N = 512$ unknowns for each refinement. With $O(N^3)$ complexity for the $\ell_1$-minimization decoding, joint decoding has a $O(P^3N^3) = O(512N^3)$ while the proposed algorithm has only $O(TPN^3) = O(16N^3)$ excluding least squares in the initial
Figure 3.6: Comparison of the proposed $P$-way combining ($P = 2, 4, 8$) vs. compressive sensing only.

3.4.4 Effect of iterative refinements

Figure 3.8 depicts the error improvement versus the number of refinement stages applied for $P$-way combining with $P = 8$. For this plot, we have fixed the total number of measurements to 200. Note that zero refinement stage means the initial approximation only. There is a diminishing return on the error improvement, and the return becomes saturated faster for larger total number of measurements.

3.4.5 Asymptotic error performance analysis

As a new sparse recovery technique, we evaluate the asymptotic error performance of the proposed scheme for $P$-way mixed measurements. To precisely control sparsity of the input data, we use a synthetic dataset generated from a multivariate Gaussian distribution. We sample a signal $x \in \mathbb{R}^N \sim \mathcal{N}(0, \Sigma)$ for with $N = 512$ and diagonal covariance matrix $\Sigma$. The diagonal entries of the covariance matrix are the eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$ and randomly generated.

$^2$The least squares solve for $\sum_{i=1}^{P} \alpha_i = 64$ unknowns, which is several orders of magnitude fewer than $PN$ and therefore neglected.
Chapter 3: Compressive Spectrum Sensing for Cognitive Radios

Figure 3.7: Snapshots of recovered spectrum under uncompressed, compressed without combining, and compressed with 8-way combined methods

Figure 3.8: Error improvement over number of refinement stages performed for $P = 8$
We denote Gaussian-$\beta$ for a signal model with $\beta\%$ of the total signal energy concentrated over the leading principal components. For example, Gaussian-90 is the signal model that the leading principal components have 90% of the total energy. We use $P = 8$ subchannels and vary $\beta = 70\%$ to 99.99%. For each subchannel, we generate 10,000 signal examples to compute sample covariance matrix to learn the principal components and additional 1,000 examples for testing. Figure 3.9 presents the asymptotic error performance of the proposed technique with 8-way mix of 256 total measurements recovered in $T = 0, 1, 2, 3,$ and 4 refinement stages. We systematically increase $\beta$ the energy concentration in $\alpha = 8$ leading principal components and observe whether the error approaches to zero asymptotically. As observed in Figure 3.9, the proposed sparse recovery technique seems to converge to zero recovery error as all of the signal energy is concentrated in the leading principal components. Lastly, we compare the number of measurements required to meet error budgets of 5%, 10%, and 20% in Table 3.1 for the UHF dataset to Gaussian-80.

\footnote{This means that 256 compressed measurements (i.e., full Nyquist rate would yield 512 uncompressed measurements) from each subchannel are combined 8-way. Since our combining is arithmetic addition, the resulting sum is also a vector of 256 measurements.}
### Table 3.1: Comparison of required total measurements for given error budget

<table>
<thead>
<tr>
<th>Error budget</th>
<th>UHF dataset</th>
<th>Gaussian-80 dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>≤ 5%</td>
<td>482 (8.5x)</td>
<td>Not met</td>
</tr>
<tr>
<td>≤ 10%</td>
<td>328 (12.4x)</td>
<td>476 (8.6x)</td>
</tr>
<tr>
<td>≤ 20%</td>
<td>186 (21.6x)</td>
<td>292 (14x)</td>
</tr>
</tbody>
</table>

### 3.5 Summary

We have described a network-based spectrum analyzer that operates over distant sensor nodes providing the measurements to construct fine-grained spectral information. Overcoming the network communication cost as we scale up the number of sensor nodes has been critical to our approach. To address this, we have devised a recovery algorithm that accompanies a simple, additive in-network combining scheme for compressed measurements from multiple sensors. Our approach makes an important assumption that discrete measurements obtained by sensors bring out sparsity in the frequency domain or in a custom basis. Designing sparsity-preserving discretization schemes is a challenging, ongoing research effort that applications of our work can benefit from, but orthogonal to the main considerations of this paper. Although it may sound surprising, we conclude that it is feasible to scale our network-based spectrum analyzer with constant communication cost.
Chapter 4

Feature Learning via Sparse Coding for Discriminative Tasks

In the previous chapter, we have applied compressive sensing for restoring fine-grained information content comparable to the original data from a mix of compressed measurements. Our focus has been to develop a new sparse recovery technique robust to the combining done at different sensors and the wireless sensor backhaul. Reconstructing the original data to their fine detail is important, yet we are often compelled to infer directly from the sensed data. For example, it would be nice to extend the compressive sensing framework of Chapter 3 for signal classification. In this chapter, we describe our transition toward discriminative tasks with sparse coding.

4.1 Introduction

Learning a representational mapping that can concisely explain the raw data is crucial for discriminative machine learning problems such as object classification. Best current practices exploit sparsity presumably underlying the data in order to bring out similarity or dissimilarity among different classes of data. We use sparse coding as the basic means to extract features of the raw data. As introduced in Chapter 2, the popular choices for sparse coding algorithm are LASSO, basis pursuit, and greedy matching pursuit (e.g., OMP). Despite more expensive computational cost, the traditional choice has been biased toward the
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

\( \ell_1 \)-regularized LASSO (or basis pursuit). Although OMP is very fast, it is known for sub-optimal classification performance on popular benchmarks [60]. Throughout the thesis, we consider both LASSO and OMP for sparse coding. Also, when we say LASSO, we usually mean LARS since LARS is a computationally better version, and the LASSO implementation we use actually implements LARS [61].

4.1.1 Problem description

We start with an assumption that there are abundant unlabeled data examples \( \{x_u^{(1)}, x_u^{(2)}, \ldots, x_u^{(n_u)}\} \), where \( n_u \) is a considerably large number, to learn the representational mapping from. Sparse coding essentially describes a transform \( f : \mathbb{R}^N \rightarrow \mathbb{R}^K \) such that an \( N \)-dimensional input is represented by its \( K \)-dimensional sparse code or feature vector, where there are total \( K (\gg N) \) basis vectors in the learned dictionary. Despite the actual increase in dimensionality, sparse coding achieves the task of efficient representation for the input by using only a small number of learned basis vectors through the \( \ell_0 \)- or \( \ell_1 \)-regularized optimization.

Sparse representation for an input is not the end outcome of a classification task. Given labeled training examples \( \{(x_s^{(1)}, l^{(1)}), (x_s^{(2)}, l^{(2)}), \ldots, (x_s^{(n_s)}, l^{(n_s)})\} \), supervised training of a classifier follows. Here, the number of labeled examples \( n_s \) is substantially smaller than \( n_u \) due to overall scarcity of labeled data. Since it is known more effective to instruct a classifier using sparse representations than the raw data, we express the task of classifier \( h : \mathbb{R}^K \rightarrow \{l_1, \ldots, l_m\} \) that takes the \( K \)-dimensional feature vector as an input and predicts its label in the general \( m \)-class classification problem. (Note that \( l_i \) is a label for the \( i \)th data class.) The trained classifier is then tested with another labeled dataset, which consists of data examples from presumably the same distributions or from real world.

In summary, the discriminative task just described addresses a semi-supervised learning problem [62]. The classification accuracy critically depends on the performance of our unsupervised feature learning [60, 63–66] manifested by the quality of sparse representations that sparse coding can produce. We consider the problem where the labeled training data are limited, and our method still can learn from weakly related, unlabeled data. More importantly, we focus on sparse coding enhancements robust to complex alterations done
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

to data, including nonlinear transformations. Lastly, we aim for a flexible computational algorithm applicable to multiple modalities\(^1\) (types) of input data such as image, text, and time series.

### 4.2 Semi-supervised Classification Pipeline

A classification pipeline incorporates many architectural components and parameters. The key ingredient for our approach is sparse coding, and it is sound to establish a standard pipeline that enables us to systematically alter configurations and optimize the performance, for example, from the choice of sparse coding and dictionary algorithms to input data size, number of dictionary atoms, and sparsity level. In this section, we will present a sparse coding based classification pipeline generally applicable to different modalities of data. We will explore both single-layer and multilayer learning architectures for sparse coding and provide empirically validated results.

#### 4.2.1 Overview

We take a semi-supervised approach that includes both unsupervised and supervised learning stages as illustrated in Figure 4.1. During the unsupervised stage, we learn a representational feature mapping by training the sparse coding dictionary from unlabeled training examples.

1. Draw (uniformly random) patches from unlabeled training examples;
2. Precondition patches with statistical data processing techniques (e.g., DC removal, normalization, whitening, dimensionality reduction);
3. Do sparse coding and dictionary learning with patches as input.

Given the learned dictionary and labeled training examples, the supervised learning stage follows.

\(^1\)For the scope of this thesis, we only consider multiple modalities in one by one. In contrast, multimodal feature learning [67] aims to learn a joint feature mapping from multiple data types.
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

**Unsupervised learning stage**

\[ \begin{align*} \mathbf{X} & \xrightarrow{\text{PCA/ZCA whitening}} \mathbf{D} \xrightarrow{\text{Sparse coding & dictionary learning}} \{\mathbf{D}, \mathbf{Y}\} \end{align*} \]

Patches extracted from unlabeled training examples

Learned dictionary & sparse codes

**Supervised learning stage**

\[ \begin{align*} \{\mathbf{X}, \mathbf{l}\} & \xrightarrow{\text{PCA/ZCA whitening}} \mathbf{Y} \xrightarrow{\text{Max pooling}} \mathbf{Z} \xrightarrow{\text{Classifier training}} \end{align*} \]

Patches of labeled training examples

Sparse codes (feature vectors)

Pooled feature vectors

**Figure 4.1: Semi-supervised approach for classification task**

1. Form feature vectors using sparse codes (*e.g.*, each sparse code from a patch can be a feature vector, or multiple sparse codes are concatenated to form one feature vector);
2. Pool multiple feature vectors;
3. Train classifiers using pooled feature vectors.

The trained classifiers can be tested by another set of labeled examples (*i.e.*, test dataset). Using the provided ground-truth labels, we can compute evaluative metrics for classification performance such as accuracy, recall, and false alarm. We can also run the trained pipeline using real-world examples.

### 4.2.2 Data processing with patches

Sparse coding takes an input vector of finite size. If data examples for classification task have a manageable size, we can probably feed the entire example as an input for sparse coding. However, this is not usually the case (*e.g.*, large image file, lengthy audiovisual clip, entire book), and we need to break down to appropriate sizes. We use patch \( \mathbf{x} \in \mathbb{R}^N \) extracted from the training examples for a unit input to sparse coding. The use of small patches as a processing unit is natural for data with spatial correlation such as images. Text
and time-series data are no different and can be processed in patches.

**Statistical preprocessing for sparse coding input.** We can speed up the unsupervised learning process or enhance its quality by preprocessing patches. Principal component analysis (PCA) is a dimensionality reduction technique for possible aid. Normalizing each patch by removing the mean and scaling by the standard deviation is known to be helpful. Another procedure called *whitening* makes input data less redundant such that dictionary learning can be more effective. Combining all of the above into one integrated procedure, we precondition our patches by PCA whitening [68] described as follows.

1. (Mean removal) \( \mathbf{x}^{(i)} := \mathbf{x}^{(i)} - \mu_x \)
2. (Sample covariance matrix estimation) \( \mathbf{C} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \)
3. (PCA) \( [\mathbf{U}, \mathbf{\Lambda}] = \text{eig}(\mathbf{C}) \)
4. (PCA whitening) \( \mathbf{x}_{\text{PCAwhite}} = (\mathbf{\Lambda} + \epsilon \mathbf{I})^{-1/2} \mathbf{U}^\top \mathbf{x} \)

For patches of image data, ZCA\(^2\) whitening [69] can be used instead for better results. ZCA whitening computes \( \mathbf{x}_{\text{ZCAwhite}} = \mathbf{U}(\mathbf{\Lambda} + \epsilon \mathbf{I})^{-1/2} \mathbf{U}^\top \mathbf{x} \) in Step 4.

### 4.2.3 Sparse coding and dictionary learning

Mathematically, sparse coding describes the transformation \( \mathbf{D} : \mathbb{R}^N \rightarrow \mathbb{R}^K \), producing a sparse feature vector \( \mathbf{y} \in \mathbb{R}^K \) given input \( \mathbf{x} \in \mathbb{R}^N \), dictionary \( \mathbf{D} \in \mathbb{R}^{N \times K} \), and sparsity level \( \lambda \) or \( S \). Sparse coding is the basic means to extract features from data. As we have previously discussed, there are numerous choices for sparse coding. We use both LARS and OMP. Recall that we specify the desired sparsity level for LARS (or LASSO) using the regularization parameter \( \lambda > 0 \). Meaning of \( \lambda \) is penalty over the \( \ell_1 \)-norm of the solution searched. Hence, higher \( \lambda \), sparser (*i.e.*, less nonzeros) the resulting sparse code will be. For OMP, we specify the sparsity level \( S \), which is more explicit as OMP will return the exactly \( S \)-sparse solution. We will denote different configurations of LARS and OMP by LARS-\( \lambda \) and OMP-\( \rho \), where \( \rho = \frac{S}{K} \times 100 \) (%).

\(^2\)ZCA stands for zero-phase component analysis.
We elect K-SVD [50] for our default dictionary learning algorithm. Note that $K$ is the total number of dictionary atoms trained by K-SVD. For our experiments throughout the thesis, we have used the Technion implementations by Rubinstein [70].

During supervised classifier training, we need to define the final feature used as input for classifiers. One obvious choice would be using sparse code $y$ as is. However, using $y$ to train classifiers or predict class label is undesirable in general. Since $y$ reflects a feature vector for only one local patch from an image, audio file, or text document, it is often a poor representative for whole. Also, if a single data example results in many sparse codes, what should one do when predicted class labels from $y$’s do not match?

To prevent local feature vectors from being used independently, we can aggregate them by a special subsampling technique called “pooling.” Note that pooling is by no means to discard any useful information. Popular in convolutional neural networks [71], pooling summarizes multiple feature vectors into one. An important property of pooled feature vectors is translation invariance [72]. We use max pooling [73] that takes the maximum value for the elements in the same position over $M$ sparse codes and yields pooled sparse code $z$ whose $k$th element is $z_k = \max(y_{1,k}, y_{2,k}, \ldots, y_{M,k})$, where $y_{i,k}$ is the $k$th element from $y_i$, the $i$th sparse code in the pooling group. Note that max pooling by $M$ essentially does $\mathbb{R}^{M \times K} \rightarrow \mathbb{R}^K$.

In summary, we have described the transformation $x \rightarrow y \rightarrow z$ for the case of single-layer sparse coding. Sparse coding computes feature vector $y$ from patch $x$. During the supervised learning stage, multiple feature vectors computed from patches of the same training example are aggregated by max pooling. We use the pooled feature vector $z$ for the input to classifiers.

### 4.2.4 Classifier training

We consider standard off-the-shelf classification algorithms such as logistic regression and support vector machine (SVM) [74,75]. We are given labeled training examples $\{(x^{(1)}, l^{(1)}), (x^{(2)}, l^{(2)}), \ldots, (x^{(n_s)}, l^{(n_s)})\}$ where class label is $l^{(i)} \in \{c_1, \ldots, c_m\}$ for generalized $m$-class classification problem. To train classifiers, we need to decide on how to form feature vectors. We use pooled sparse code $z$ obtained by max pooling $M$ sparse codes $y$’s as our
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

There are numerous approaches to train classifiers, most of which are inherently binary, for multiclass classification. The first approach is to train \( \frac{m(m-1)}{2} \) 1-vs-1 binary classifiers exhaustively. Each trained classifier is dedicated for every possible class pair out of the \( m \)-class pool. An obvious drawback of this approach is poor scalability against the number of classes. It may also require a decision tree to determine optimal test order. The second approach is to train \( m \) 1-vs-all classifiers [76]. For class \( c_i \), training 1-vs-all classifier requires two labeled datasets, one containing training examples from class \( c_i \) only and the other with uniformly mixed examples from all the other classes \( c_j \) \( \forall j \neq i \). The last approach would opt for (natively) multiclass classification algorithms such as multiclass SVM [77] and softmax regression [21]. In this thesis, we will use the second or third approaches for multiclass classification.

4.3 Experimental Evaluation with Text and Image Datasets

In this section, we empirically evaluate the performance of the semi-supervised classification pipeline based on sparse coding using text and image datasets.

4.3.1 Text-based regional classification with Twitter dataset

Problem and dataset. It has been known that extracting linguistic features from text data created online can lead to geographical classification (e.g., region, state, city) and geolocation. This evaluation considers two multiclass classification problems that 1) predict one of the four US regions and 2) predict one of the 48 contiguous US states, using Twitter microblogging text known as tweet. We build a classification pipeline based on single-layer sparse coding trained on tweets and benchmark the classification performance against numerous known approaches including latent geographic topic model, supervised LDA, and text regression.

In this evaluation, we use the CMU GEO TEXT dataset [78], a geo-tagged microblog corpus comprising 377,616 tweets by 9,475 users from 48 contiguous US states and Washington D.C. Each document in the dataset is concatenation of all tweets by one user whose
GPS-assigned geocoordinates are provided. The dataset comes with vocab, the table of 5,216 unique words appeared in the tweets.

**Data processing and training.** We first apply a basic text data processing technique called embedding that converts text into vectors. We use three embedding schemes. Binary bag-of-words (BW) embedding creates a vector $w_{BW} \in \{0, 1\}^V$ from a document containing $W$ words, where $V = |\text{vocab}|$. The $i$th element in $w_{BW}$ is 1 if the word $\text{vocab}[i]$ has appeared in the text. Word-counts (WC) embedding creates a vector $w_{WC} \in \{0, 1, 2, \ldots\}^V$, where the $i$th element is now the number of times the word $\text{vocab}[i]$ has appeared. We also use word-sequence (WS) embedding that creates a vector $w_{WS} \in \{1, \ldots, V\}^W$ whose $i$th element $w_i$ represents the $i$th word in the text (i.e., $\text{vocab}[w_i]$).

We process all user geocoordinates using MATLAB Mapping Toolbox to label each document example with one of regions and states in the US. We cut the dataset into five folds such that $\text{fold} = \text{user.id} \% 5$, exactly following Eisenstein et al. [16] for fair comparison. Folds 1–4 are used for training, and fold 5 for testing. We have converted the entire text in the dataset to vectors using all three embedding schemes. We use patch $x \in \mathbb{R}^N$ with sizes $N = 32, 64$ as an input to sparse coding. Patch is a subvector taken from $w_{BW}, w_{WC},$ or $w_{WS}$ of a document.

We summarize each step to build the pipeline.

1. (Text embedding) perform binary, word-counts, or word-sequence embedding of raw tweet text, using vocab;

2. (Unsupervised feature learning) use unlabeled patches $\{x^{(1)}, x^{(2)}, \ldots\}$ drawn from embedded text data to learn basis vectors in $D$ via sparse coding and dictionary learning;

3. (Feature extraction) do sparse coding with labeled data patches $\{(x^{(1)}, l^{(1)}), (x^{(2)}, l^{(2)}), \ldots\}$, using the learned dictionary $D$ and yield (unpooled) sparse codes $\{y^{(1)}, y^{(2)}, \ldots\}$;

4. (Feature pooling) perform max pooling over a group of $M$ sparse codes from a user to obtain pooled sparse code $z$ such that the $k$th element in $z, z_k = \max(y_{1,k}, y_{2,k}, \ldots, y_{M,k})$ where $y_{i,k}$ is the $k$th element from $y_i$, the $i$th sparse code in the pooling group;
5. (Supervised classifier training) use pooled sparse codes $z$ as features to train a classifier such as multiclass SVM and softmax regression.

In unsupervised learning, we precondition patches with PCA whitening prior to sparse coding. We have tried both LASSO and OMP for sparse coding. The sparse coding parameters have been experimentally determined. We use a dictionary size $K \approx 10N$, regularization parameter $0.1 \leq \lambda \leq 0.2$ for LASSO, and sparsity level $0.05N \leq S \leq 0.4N$ for OMP. We have trained the dictionaries with the K-SVD algorithm. For max pooling, we use pooling factors $M$ in 10s. In supervised learning, we have trained linear multiclass SVM and softmax classifiers, using max pooled sparse codes as features with US region and state labels. That is, we have trained 4-way and 49-way classifiers for the regional and state classification tasks.

We have also applied the following enhancements.

1. **Regional dictionary learning.** In principle, dictionary learning is an unsupervised process. Depending on availability, however, class labels of training examples can help learn better dictionary. For example, we can train dictionaries $D_1$, $D_2$, $D_3$, and $D_4$ separately for each region and concatenate them to yield the final $D = [D_1 \| D_2 \| D_3 \| D_4]$. The resulting basis vectors in the dictionary are better representatives for respective regions.

2. **Feature augmentation.** Sparse code $y$ serves a feature vector for raw patch input $x$. Information-theoretically speaking, $x$ and $y$ are approximately equivalent since one representation is a transform of the other. However, due to reasons such as insufficient samples for dictionary training or noisy $x$, there is a possibility of many-to-one ambiguity. In other words, raw data belonging to different classes may have very similar sparse code. To mitigate this problem, we apply a simple enhancement by appending the raw patch vector $x$ to its sparse code (or, its pooled sparse code $z$) $y$ and use the combined vector as the augmented feature.

The full system pipeline including the enhancements is described in Figure 4.2.

**Results.** Table 4.1 compares the multiclass classification accuracies by various schemes. As we have tested with two different patch sizes $N = 32, 64$ for sparse coding, we re-
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

Figure 4.2: Full system pipeline for Twitter text-based regional classification in US

Table 4.1: Classification accuracy comparison

<table>
<thead>
<tr>
<th>Classification accuracy</th>
<th>US region (4-way)</th>
<th>US state (49-way)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographic topic model [16]</td>
<td>58%</td>
<td>24%</td>
</tr>
<tr>
<td>Mixture of unigrams [79]</td>
<td>53%</td>
<td>19%</td>
</tr>
<tr>
<td>Supervised LDA [17]</td>
<td>39%</td>
<td>4%</td>
</tr>
<tr>
<td>Text regression [18]</td>
<td>41%</td>
<td>4%</td>
</tr>
<tr>
<td>k-NNs with majority class voting [80]</td>
<td>37%</td>
<td>27%</td>
</tr>
<tr>
<td>Our approach 1 (OMP-20, $N = 32$) [15]</td>
<td>63.2%</td>
<td>35.3%</td>
</tr>
<tr>
<td>Our approach 2 (OMP-20, $N = 64$) [15]</td>
<td>66.7%</td>
<td>40.8%</td>
</tr>
<tr>
<td>Our approach 3 (LARS-0.1, $N = 64$)</td>
<td><strong>69.5%</strong></td>
<td><strong>43.2%</strong></td>
</tr>
</tbody>
</table>

port the best result from each case. Our best case configuration is described in the table. We use OMP-$\rho$ notation with $\rho = \frac{S}{K} \times 100$ (%). For US regional classification, we have achieved a 9% gain over geographic topic model, the best among the other schemes. Geographic topic model is a generative graphical model based on complex hybrid of Gaussian, Gamma, Wishart, and Dirichlet priors. Our best result for the 49-way state classification is also superior over the best of other schemes by 14%. It is noteworthy that $k$-NNs with majority voting, one of the simplest classification algorithms, has achieved the best state classification accuracy among the other schemes. We remark that our results here are the first for sparse coding.
Figure 4.3: Sample images of CIFAR-10 classes (photo courtesy of Alex Krizhevsky for University of Toronto)

4.3.2 CIFAR-10 image classification

**Problem and dataset.** For our second application in the discriminative framework, we build an image classification pipeline based on sparse coding. The evaluation here considers single-layer sparse coding and dictionary trained on the CIFAR-10 dataset [81, 82]. Again, we test both LARS and OMP to characterize the performance of the $\ell_1$-regularized and the greedy-$\ell_0$ coders. Additionally, we test a special kind of neural network called autoencoder (AE) [83], which will be explained shortly.

CIFAR-10 is a collection of 60,000 $32 \times 32$ color images. The 10 classes of the dataset are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Sample images are shown in Figure 4.3. CIFAR-10 comes as 6 folds with 5 of them designed for training and 1 for testing. Each fold contains 10,000 images of uniformly mixed classes.

**Autoencoder.** Autoencoder is an *unsupervised*\(^3\) learning algorithm that can learn a feature mapping by setting its output values to be equal to the inputs (hence, encoding *self*). Figure 4.4 presents a standard single-layer autoencoder. The hidden encoding units in the middle are neurons. The values captured by the neurons under an applied input are called *activations*, which are equivalent to a sparse code computed by sparse coding. The figure

\(^3\)Neural network [84] is a *supervised* learning method to define a complex, nonlinear form of hypotheses for data typically over multiple layers of computational units called neurons.
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

Figure 4.4: Autoencoder neural network

also describes the training and usage of autoencoder. Given an input \( x \in \mathbb{R}^N \), backpropagation trains a set of encoding weights that transforms \( x \) into the activations \( a \in \mathbb{R}^H \) and a set of decoding weights that recovers \( \hat{x} \), an estimate of \( x \), from \( a \). Similar to sparse coding, we can use the encoding \( a \) as the feature vector for \( x \) to train classifiers. If we set \( H < N \), the autoencoder network is forced to learn a compressed representation of the input. Also, even when \( H > N \) (i.e., there are more hidden units than input values entered), we can still extract a meaningful feature out of the applied input vector by imposing a sparsity constraint on \( a \) such that only \( S \ll H \) elements of \( a \) are nonzero—i.e., we have a sparse autoencoder with \( S \)-sparse activations \( a \) from the neurons.

Data processing and training. Figure 4.5 illustrates our image data processing pipeline that extracts features from patches of an image via sparse coding. Note that this pipeline is general and can be applied to other images than CIFAR-10. An input image is first planned in pooling groups of patches. Each pooling group consists of \( M \) spatially proximate patches. We use a receptive field of \( w \times w \) pixels to draw a patch. CIFAR-10 images are encoded in RGB color format, giving each pixel a dimension \( d = 3 \). Thus, when fully vectorized, we have a patch \( x \in \mathbb{R}^N \) with its dimensionality \( N = d \cdot w \cdot w \). For dense sparse coding, it is possible to introduce an overlap between successive patches using a stride \( s \). Usually, the patches are preprocessed before sparse coding via PCA- or ZCA-whitening.
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

Sparse coding produces $M$ sparse codes per pooling group. Each sparse code $y$ has dimensionality $K > N$, which matches the number of dictionary atoms in $D$. At the end of the pipeline, pooling over all $y$’s in the pooling group (i.e., max pooling by $M$) forms a pooled sparse code $z$ that keeps the same dimensionality as each $y$.

Instead of using the full CIFAR-10 dataset, we uniformly sample 20,000 images from the five training folds for unsupervised learning. We enforce exactly 2,000 images per class. We use a square receptive field with $w = 4, 6, 4$ resulting the patch dimensionalities $N = 48, 108, 48$ respectively. Because the width of a CIFAR-10 image is 32, which is not a multiple of 6, we have introduced an overlap between the patches extracted from successive receptive fields for the case $w = 6$. Using a stride $s = 2$, we have performed dense feature extraction. On the other hand, for $w = 4$, we use nonoverlapping patches. For sparse coding, we use the dictionary size $K = 8N$ with two different sparsity levels. For LARS, we consider $\lambda = 0.1$ and 0.2. We configure $S \approx 0.15N$ and $0.2N$ for OMP.

\footnote{We have not further tried much larger receptive field sizes discouraged by experiments on CIFAR-10 such as Coates, Lee, and Ng [85].}

Figure 4.5: Generalized image data processing pipeline that yields pooled sparse code
Chapter 4: Feature Learning via Sparse Coding for Discriminative Tasks

For fair comparison, we have to match the number of weights trained in sparse coding and autoencoder. As indicated in Figure 4.4, the autoencoder with $H$ hidden units have $N(H + 1) + H(N + 1) = 2NH + N + H$ total weights to be trained. Sparse coding with $K$ dictionary atoms need to train $NK$ weights. By $2NH + N + H \approx NK$, we configure the autoencoders with $H = 190$ for $w = 4$ and $H = 430$ for $w = 6$ using sparsity levels $S \approx 0.15H$ and $0.2H$. We distinguish them by AE-0.15 and AE-0.2, respectively.

For supervised classifier training, we sample another 10,000 images from the training folds. Again, we enforce exactly 1,000 images per class. To perform the multi-class classification, we have adopted the 1-vs-all classification with linear SVMs. The SVM classifiers are trained by the feature vector $\phi$ equivalent to the entire single CIFAR-10 image. In Figure 4.6, we describe the feature formation using the pooled sparse codes. We divide a CIFAR-10 image into four quadrants. Each quadrant is a pooling group of 16 (for $w = 4, s = 0$) and 36 (for $w = 6, s = 2$) sparse codes. With the resulting pooled sparse codes $\{z_1, z_2, z_3, z_4\}$, we form the final feature vector $\phi_{SC} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$. Thus, the entire single CIFAR-10 image is represented by $\phi_{SC} \in \mathbb{R}^{4K}$ for our sparse coding approaches. This means that input sizes $N = 48$ and 108 correspond to feature vector sizes $4K = 1,536$ and 3,456, respectively. Despite high dimensionality, we note that the feature vectors are relatively sparse—about 35–55% nonzeros after sparse coding and pooling. We have applied
Figure 4.7: CIFAR-10 1-vs-all classification accuracy comparison for AE-0.15, LARS-0.2, and OMP-15. The “dense” configuration indicates the use of $6 \times 6$ overlapping patches, whereas we have used $4 \times 4$ nonoverlapping patches otherwise.

the same quadrant and patching schemes to yield $\phi_{AE} \in \mathbb{R}^{4H}$ for autoencoders.

For testing, we sample 2,000 images from the testing fold with exactly 200 images per class. After undergoing the feature transformation with the testing images, we evaluate accuracy of the 1-vs-all SVM classifiers trained.

**Results.** In Figure 4.7, we present per-class 1-vs-all classification accuracy for AE-0.15, LARS-0.2, and OMP-15. Similarly, Figure 4.8 presents classification accuracy for AE-0.2, LARS-0.1, and OMP-20. We notice about 4–6% accuracy improvements in Figure 4.8. This means that AE-0.2, LARS-0.1, and OMP-20 have outperformed their respective counterparts configured using a stricter sparsity constraint. We find that the features learned by both sparse coding approaches are more effective than autoencoder in classification. When fairly matched in training efforts, both OMP and LARS have consistently outperformed autoencoder. In sparse coding, we find that the $\ell_1$-regularized LARS performs better than OMP. Computation time for LARS is longer than OMP due to searching for fuller regularization path, but LARS proves to find a better sparse representation for classification.
Figure 4.8: CIFAR-10 1-vs-all classification accuracy comparison for AE-0.2, LARS-0.1, and OMP-20. Here, we use slightly less sparse (more nonzeros) coder configurations. Lower $\lambda$ for LARS means less penalty for the $\ell_1$-regularized term, resulting in lower reconstructive error than LARS-0.2. Similarly, AE and OMP are configured to induce more nonzeros. We observe the overall improvements in the classification performance than sparser coder configuration used in Figure 4.7.
Table 4.2: Average 1-vs-all classification accuracy under normal feature extraction ($w = 4$, nonoverlapping patches)

<table>
<thead>
<tr>
<th></th>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder</td>
<td>65.3%</td>
</tr>
<tr>
<td>OMP-15</td>
<td>69.4%</td>
</tr>
<tr>
<td>OMP-20</td>
<td>71.0%</td>
</tr>
<tr>
<td>LARS-0.2</td>
<td>73.2%</td>
</tr>
<tr>
<td>LARS-0.1</td>
<td>74.7%</td>
</tr>
</tbody>
</table>

Table 4.3: Average 1-vs-all classification accuracy under dense feature extraction ($w = 6$, overlapping patches with $s = 2$)

<table>
<thead>
<tr>
<th></th>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autoencoder</td>
<td>69.8%</td>
</tr>
<tr>
<td>OMP-15</td>
<td>75.3%</td>
</tr>
<tr>
<td>OMP-20</td>
<td>76.9%</td>
</tr>
<tr>
<td>LARS-0.2</td>
<td>78.4%</td>
</tr>
<tr>
<td>LARS-0.1</td>
<td>80.1%</td>
</tr>
</tbody>
</table>

4.4 Summary

This chapter describes our transition to discriminative machine learning tasks. We build a classification pipeline on unsupervised feature learning via sparse coding. In our evaluation, we have experimented with text and image datasets. We show that sparse coding trained on unlabeled Twitter microblog text leads to a powerful representational mapping for text-based geographic classification superior to topical models for lexical variation. Such mapping is particularly effective when it is used in a similarity searching. Our classification accuracy for 4-way US Regions is an 11.5% improvement over the best topical model in the literature, and the 19.2% for 48-way US States.

We have also applied the same classification pipeline to CIFAR-10. While we have
validated basic algorithm-independent findings such that the use of more features and dense extraction enhance the classification performance, we confirm the superiority of sparse coding over autoencoder in unsupervised feature learning when the total number of trained weights are matched.
Chapter 5

Multilayer Sparse Coding and Deep Sparse-coded Network (DSN)

In Chapter 4, we have explored feature learning with a single layer of sparse coding followed by nonlinear summarization via max pooling for discriminative applications on text and image data. We have shown that slightly high-level feature representations made possible by sparse coding are indeed effective for classification tasks. This chapter considers multilayer sparse coding to make feature learning possibly hierarchical. We will formalize Deep Sparse-coded Network (DSN) and discuss training methods to enable deep learning when there are two or more sparse coding layers. In particular, we propose a novel backpropagation algorithm that trains the whole DSN as a tightly optimized classifier.

5.1 Motivation

Representational power of single-layer feature learning is somewhat limited for tasks that involve large and complex data objects such as a high-resolution image of human face. A deep architecture in machine learning stacks two or more layers of feature learning units in the hope of unveiling a structure of higher-level features. Best current practices in object recognition use deep architectures based on autoencoder [19], restricted Boltzmann machine (RBM) [86, 87], and convolutional neural network (CNN) [88]. The biggest advantage of deep architectures is learning hierarchical representations for data. In other
words, we understand how to compose the features at each layer using the features of the layer below. Such hierarchical decomposition is particularly useful when we cannot resolve ambiguity of the low-level (or local) features of data. Another benefit of using deep architectures is representational efficiency. Deep architectures can achieve compaction of all characteristic features for the entire image, book, or lengthy multimedia clip to a single vector.

Previously, we have demonstrated that the classification performance of our pipeline based on single-layer sparse coding is superior to that of single-layer autoencoder. According to the empirical results by Coates, Lee, & Ng [85], sparse coding is also found superior to sparse RBM [89], deep neural network [90], and CNN [91] for classification tasks using the CIFAR-10 and NORB datasets. With these benefits in mind, it is sound to extend a deep architecture for sparse coding. Unfortunately, it does not seem automatic to realize a useful hierarchy by simply stacking sparse coding layers. From sparse coding-related research that tries to build hierarchical representations [92–95], we deduce some plausible explanations for the difficulty. First, the $\ell_1$-regularized sparse coding is computationally expensive for multilayering and associated optimizations. From our experience, it turns out quite cumbersome and challenging to just connect multiple sparse coding units and run data as a feedforward network. Secondly, sparse coding makes an inherent assumption on the input being non-sparse. This makes a straightforward adoption to take the output from one sparse coding unit for an input to another flawed. Lastly, it is difficult to jointly optimize all sparse coding layers since a common notion of deep learning suggests layer-by-layer unsupervised pretraining should be followed by supervised training of the whole system, which is commonly done by backpropagation.

In this chapter, we present a deep architecture based on multilayer sparse coding. We will describe a principled extension of single-layer sparse coding. We build on both LARS and OMP. By interlacing pooling units with sparse coding layers, we achieve nonlinear activation analogous to neural networks. Our pooling units avoid linear cascade of dictionaries to keep the effect of multilayering in tact. Specifically, we elect max pooling to aggregate multiple sparse codes to their maximum elements. Max pooling is known to preserve translational invariance of higher-layer representations [72]. Beyond the layer-by-layer training, we propose a novel backpropagation algorithm that can finetune our entire architecture
holistically. We name our approach Deep Sparse-coded Network (DSN) and empirically validate with Wi-Fi network traffic and CIFAR-10 datasets.

5.2 Notation

We follow the previously established notation for sparse coding as well as adopting the layering convention used in neural networks. We denote input vector $x$, its sparse code $y$, and the pooled sparse code $z$. As in neural network, layer 1 is the input layer, and the final layer is reserved for the output. (The output layer is typically configured as classifiers or regressors depending on specifics of applications.) We will also call a sparse coding layer a “hidden” layer. Unlike convolutional neural network, we do not count pooling units as a separate layer. We use subscripted Roman numerals to indicate sparse coding layers. For example, $D_1$ means the first hidden layer’s dictionary. Note that the first hidden layer is the overall layer 2. Accordingly, $y_1$ is the sparse code computed at the first hidden layer with the input $x$ and $D_1$, and $z_1$ the pooled sparse code over multiple $y_1$’s.
Chapter 5: Multilayer Sparse Coding and Deep Sparse-coded Network (DSN)

5.3 Architectural Component

Figure 5.1 depicts a layering module that can be stacked to form a multilayer sparse coding architecture. Sparse coding and pooling units together constitute the module. Note that the module depicted in the figure is for hidden layers 2 or higher since the first hidden layer takes patches of raw data as its input. At hidden layer $J$ for $J \geq 1^1$, the pooled sparse codes $z_{J-1}$'s from the previous layer are fed as the input. Sparse coding with hidden layer $J$'s dictionary $D_J$ produces $y_J$, and max pooling over $y_J$'s yields a pooled sparse code $z_J$ for the layer. The pooled sparse codes $z_J$'s are passed as the input for hidden layer $J + 1$.

5.4 Algorithms

In this section, we explain training algorithms for multilayer sparse coding using an example architecture illustrated in Figure 5.2. The example architecture has four layers in total. This is a deep architecture because there are two hidden layers of sparse coding, each of which can learn own feature representations by training a dictionary of the corresponding level.

5.4.1 Pretraining with layer-by-layer sparse coding and dictionary learning

The input layer consists of patches consecutively drawn from the raw data. For example, we can draw spatially consecutive patches from an image, whereas we can draw temporally consecutive patches from time series data to make the overall feature learning meaningful. Optionally, we can preprocess the patches by normalization and whitening.

The input layer is organized as pooling groups of $M_1$ patches: \{x^{(1)}, x^{(2)}, \ldots, x^{(M_1)}\}, \{x^{(M_1+1)}, \ldots, x^{(2M_1)}\}, \{x^{(2M_1+1)}, \ldots, x^{(3M_1)}\}, and so forth. Sparse coding and dictionary

\(^1\)Hidden layer $J$ is layer $J + 1$ for the overall architecture.
Figure 5.2: Multilayer sparse coding architecture with 2 hidden layers
learning at hidden layer 1 perform

\[
\begin{align*}
\{ \mathbf{x}^{(1)}_1, \ldots, \mathbf{x}^{(M_1)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(1)}_1, \ldots, \mathbf{y}^{(M_1)}_1 \} \\
\{ \mathbf{x}^{(M_1+1)}_1, \ldots, \mathbf{x}^{(2M_1)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(M_1+1)}_1, \ldots, \mathbf{y}^{(2M_1)}_1 \} \\
\{ \mathbf{x}^{(2M_1+1)}_1, \ldots, \mathbf{x}^{(3M_1)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(2M_1+1)}_1, \ldots, \mathbf{y}^{(3M_1)}_1 \} \\
\vdots
\end{align*}
\]

while computing sparse codes \( \mathbf{y}^{(i)}_1 \)'s and learning the columns of \( D_1 \) jointly. Subsequently, max pooling at hidden layer 1 does

\[
\begin{align*}
\{ \mathbf{y}^{(1)}_1, \ldots, \mathbf{y}^{(M_1)}_1 \} & \xrightarrow{\text{max pool}} \mathbf{z}^{(1)}_1 \\
\{ \mathbf{y}^{(M_1+1)}_1, \ldots, \mathbf{y}^{(2M_1)}_1 \} & \xrightarrow{\text{max pool}} \mathbf{z}^{(2)}_1 \\
\{ \mathbf{y}^{(2M_1+1)}_1, \ldots, \mathbf{y}^{(3M_1)}_1 \} & \xrightarrow{\text{max pool}} \mathbf{z}^{(3)}_1 \\
\vdots
\end{align*}
\]

Hidden layer 1 passes the pooled sparse codes \( \{ \mathbf{z}^{(1)}_1, \mathbf{z}^{(2)}_1, \mathbf{z}^{(3)}_1, \ldots \} \) to hidden layer 2. Sparse coding and dictionary learning continue at hidden layer 2

\[
\begin{align*}
\{ \mathbf{z}^{(1)}_1, \ldots, \mathbf{z}^{(M_2)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(1)}_2, \ldots, \mathbf{y}^{(M_2)}_2 \} \\
\{ \mathbf{z}^{(M_2+1)}_1, \ldots, \mathbf{z}^{(2M_2)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(M_2+1)}_2, \ldots, \mathbf{y}^{(2M_2)}_2 \} \\
\{ \mathbf{z}^{(2M_2+1)}_1, \ldots, \mathbf{z}^{(3M_2)}_1 \} & \xrightarrow{D_1} \{ \mathbf{y}^{(2M_2+1)}_2, \ldots, \mathbf{y}^{(3M_2)}_2 \} \\
\vdots
\end{align*}
\]

Pooling groups at hidden layer 2 consist of \( M_2 \) pooled sparse codes from hidden layer 1.
Max pooling by $M_2$ yields

\[
\{y^{(1)}_I, \ldots, y^{(M_2)}_I\} \xrightarrow{\text{max pool}} z^{(1)}_I \quad \{y^{(M_2+1)}_I, \ldots, y^{(2M_2)}_I\} \xrightarrow{\text{max pool}} z^{(2)}_I \\
\{y^{(2M_2+1)}_I, \ldots, y^{(3M_2)}_I\} \xrightarrow{\text{max pool}} z^{(3)}_I \\
\vdots
\]

The layer-by-layer training completes by producing the layer-wise dictionaries $\{D_I, D_{II}\}$, sparse codes $\{y^{(1)}_I, y^{(2)}_I, \ldots\}$, $\{y^{(1)}_{II}, y^{(2)}_{II}, \ldots\}$, and pooled sparse codes $\{z^{(1)}_I, z^{(2)}_I, \ldots\}$, $\{z^{(1)}_{II}, z^{(2)}_{II}, \ldots\}$.

The role of max pooling is critical for our architecture. It subsamples sparse codes by aggregating to their max elements. More importantly, max pooling serves a nonlinear activation function as in neural network. Without nonlinear pooling units, multilayering has no effect: $x = D_I y_I$ and $y_I = D_{II} y_{II} \Rightarrow x = D_I D_{II} y_{II} \approx D y_{II}$, since linear cascade of dictionaries can be simply $D \approx D_I D_{II}$ regardless of total number of layers.

### 5.4.2 Training classifiers at output layer

By now, we have pretrained each layer’s dictionary greedily. The highest hidden layer’s output, the pooled sparse code $z^{(j)}_{II}$, is already a powerful feature for classification tasks. Consider that the output layer consists of classifiers as in Figure 5.2. We predict a class label $\hat{l} = h_w(\phi)$, where $h_w(\cdot)$ is a standard linear classifier or logistic regression that takes a feature encoding $\phi$ as input. Note that $\phi$ can be formed using the highest hidden layer output $z_{II}$, but the actual setup depends on specifics of the multilayer architecture. For simplicity, we assume $\phi = z_{II}$.

A linear classifier computes $\hat{l} = h_w(z_{II}) = w^\top \cdot z_{II} + w_0$. We train the classifier weight $w = [w_0 \ w_1 \ \ldots \ w_{K_2}]^\top$ using labeled examples $\{(X_1, l_1), \ldots, (X_m, l_m)\}$ in a supervised process by filling the input layer with patches from each example—$X_i = \{x^{(1)}_i, x^{(2)}_i, \ldots\}$ where $x^{(k)}_i$ is the $k$th patch from $i$th example $X_i$—and working up the layers to compute $z_{II}$’s.
5.4.3 Backpropagation

It dates back to Hinton et al. [86] that suggests pretrain a deep architecture with layer-by-layer unsupervised learning and finetune via backpropagation, a supervised training algorithm popularized by neural network. Is backpropagation possible for our multilayer sparse coding architecture?

We now describe a backpropagation algorithm that optimizes the classification performance of our multilayer architecture. We call our approach Deep Sparse-coded Network (DSN). The DSN backpropagation is quite different from conventional neural network or deep learning architectures. Let us express the complete feedforward path of DSN, comprising feature formations via sparse coding and max pooling, and classification at the top

\[
\begin{align*}
    x & \xrightarrow{D_I} y_I \xrightarrow{\text{max pool}} z_I \xrightarrow{D_{II}} y_{II} \xrightarrow{\text{max pool}} z_{II} \xrightarrow{\text{classify}} \hat{l}
\end{align*}
\]

We remind that the goal of backpropagation is to finetune the entire DSN for labeled training examples \( \{(X_1, l_1), \ldots, (X_m, l_m)\} \). Note that we draw patches from each example and apply to the input layer of DSN—i.e., \( X_i = \{x_i^{(1)}, x_i^{(2)}, \ldots, \} \) where \( x_i^{(k)} \) is the \( k \)-th patch drawn from \( i \)-th example \( X_i \), and patches from the same example share the class label. We define the loss or cost function for the final output of DSN as classification error

\[
J(z_{II}) = \frac{1}{2} \| \hat{l} - l \|^2 = \frac{1}{2} \| h_w(z_{II}) - l \|^2
\]  

(5.1)

We must first estimate \( z_{II}^* \) that minimizes \( J(z_{II}) \). To do so, we perform gradient descent learning with \( J(z_{II}) \) that adjusts each element in \( z_{II} \) by

\[
z_{II,k} = z_{II,k} - \alpha \frac{\partial J(z_{II})}{\partial z_{II,k}}
\]

(5.2)

where \( z_{II} = [z_{II,1}, z_{II,2}, \ldots, z_{II,K_2}]^T \), and \( K_2 \) is the number of atoms in dictionary \( D_{II} \) for hidden layer 2. Since an optimal \( z_{II}^* \) is estimated by correcting \( z_{II} \), the partial derivative is
Figure 5.3: After pooled sparse code $z$ is corrected to $z^*$, the corrected values are put back to the corresponding locations at the original sparse codes that have been max pooled to $z$. Putback corrects sparse codes $y$’s to $y^{**}$’s.

with respect to each element $z_{II,k}$

$$
\frac{\partial J(z_{II})}{\partial z_{II,k}} = \left[ h_w(z_{II}) - l \right] \frac{\partial h(z_{II})}{\partial z_{II,k}} = \left[ h_w(z_{II}) - l \right] w_k
$$

(5.3)

Here, a linear classifier has $h_w(z_{II}) = w_0 + w_1 \cdot z_{II,1} + \cdots + w_{K_2} \cdot z_{II,K_2}$. Therefore, the following gradient descent rule estimates $z_{II}^*$

$$
z_{II,k} := z_{II,k} + \alpha \left[ l - h_w(z_{II}) \right] w_k
$$

(5.4)

where $\alpha$ is the learning rate. This update rule is intuitive as it down-propagates the error $\left[ l - h_w(z_{II}) \right]$ proportionately to the exact contribution by each $z_{II,k}$ and adjusts each accordingly.

Using the estimated $z_{II}^*$, we can estimate unpooled, optimal $y_{II}^{**}$’s by a procedure called putback illustrated in Figure 5.3. At hidden layer 2, we have performed max pooling by $M_2$. For putback, we need to keep the original $M_2$ $y_{II}$’s that have resulted each $z_{II}$. As in the figure, putback corrects the original $y_{II}$’s to $y_{II}^{**}$’s using the estimated $z_{II}^*$.

With $y_{II}^{**}$, going down a layer is relatively straightforward. By sparse coding relation,
we simply compute \( z^*_1 = D_1 y^*_1 \). Next, we do another putback at hidden layer 1. Using \( z^*_1 \), we obtain \( y^*_1 \)'s. Note that each pooling group at hidden layer 1 originally has \( M_1 y_1 \)'s that need to be saved in memory.

With \( y^*_1 \)'s, we should now correct \( D_1 \). It does not make sense to correct \( x \) because \( x \) is a given input. Hence, down-propagation of the error for DSN stops here, and we up-propagate corrected sparse codes to finetune the dictionary atoms and classifier weights.

We first define a loss function with respect to \( D_1 \)

\[
J(D_1) = \frac{1}{2} \| D_1 y^*_1 - x \|_2^2
\]

(5.5)

Adjusting \( D_1 \) requires to solve the following optimization problem given examples \( (x, y^*_1) \)

\[
\min_{d_{1,k}} J(D_1) \quad \text{s.t.} \quad \|d_{1,k}\|_2^2 = 1 \quad \forall k
\]

(5.6)

where \( d_{1,k} \) is the \( k \)th dictionary atom in \( D_1 \). Taking the partial derivative with respect to \( d_{1,k} \) yields

\[
\frac{\partial J(D_1)}{\partial d_{1,k}} = (D_1 y^*_1 - x) [y^*_{1,k} - y_{1,k}]
\]

(5.7)

where \( y^*_1 = [y^*_{1,1} \ldots y^*_{1,K_1}]^\top \) and \( y_1 = [y_{1,1} \ldots y_{1,K_1}]^\top \), and \( K_1 \) is the number of dictionary atoms in \( D_1 \). Thus, we obtain the following update rule to adjust \( D_1 \) by gradient descent

\[
d_{1,k} := d_{1,k} - \beta (D_1 y^*_1 - x) [y^*_{1,k} - y_{1,k}]
\]

(5.8)

We denote the corrected dictionary \( D^*_1 \). We redo sparse coding at hidden layer 1 with \( D^*_1 \) followed by max pooling. Similarly at hidden layer 2, we adjust \( D_{II} \) by

\[
d_{II,k} := d_{II,k} - \gamma (D_{II} y^*_II - z^*_I) [y^*_{II,k} - y_{II,k}]
\]

(5.9)

where \( z^*_I \) is the pooled sparse code over \( M_1 y^*_I \)'s from sparse coding redone with \( D^*_I \). Using corrected dictionary \( D^*_II \), we also redo sparse coding and max pooling at hidden layer 2. The resulting pooled sparse codes \( z^*_II \) are the output of the highest hidden layer, which will
be used to retrain the classifier $h_w$. The entire steps just described are a single iteration of the DSN backpropagation. We run multiple iterations until convergence. We present the backpropagation algorithm for general $L$-layer DSN in Algorithm 4.

We summarize the corrections made during down-propagation for the DSN backpropagation

$$z_{II} \xrightarrow{GD} z_{II}^* \xrightarrow{putback} y_{II}^* \xrightarrow{D_{II}} z_{I}^* \xrightarrow{putback} y_{I}^*$$

The corrections by up-propagation follow

$$D_{I} \xrightarrow{GD} D_{I}^* \xrightarrow{SC} y_{I}^* \xrightarrow{max \ pool} z_{I}^* \xrightarrow{GD} D_{II}^* \xrightarrow{SC} y_{II}^* \xrightarrow{max \ pool} z_{II}^* \xrightarrow{GD} h_w$$

where GD stands for gradient descent, and SC sparse coding.

5.5 Experimental Results

5.5.1 Classifying CIFAR-10 images

Schemes. In Chapter 4, we have evaluated single-layer sparse coding in classification task for CIFAR-10 images. Our best single-layer setup is the $\ell_1$-regularized LARS-0.1 that has achieved the classification accuracy of 80.1%. Since we have found that LARS-0.1 is superior to LARS-0.2, and OMP-20 to OMP-15, we consider the following multilayer sparse coding schemes.

1. Two hidden layers of OMP-20 with layer-by-layer pretraining only;
2. Two hidden layers of LARS-0.1 with layer-by-layer pretraining only;
3. Two hidden layers of OMP-20 with both layer-by-layer pretraining and backpropagation;
4. Two hidden layers of LARS-0.1 with both layer-by-layer pretraining and backpropagation.

Additionally, we also test deep stacked autoencoders (SAE).
Algorithm 4 DSN backpropagation

require Pretrained \( \{D_1, D_{II}, \ldots, D_{L-2}\} \) and classifier \( h_w \)

input Labeled training examples \( \{(X_1, l_1), \ldots, (X_m, l_m)\} \)

output Fine-tuned \( \{D_1^*, D_{II}^*, \ldots, D_{L-2}^*\} \) and classifier \( h_w^* \)

1: repeat
2: \hspace{1em} subalgorithm Down-propagation
3: \hspace{2em} for \( J := L - 2 \) to 1
4: \hspace{3em} if \( J == L - 2 \)
5: \hspace{4em} Compute classifier error \( \epsilon^{(i)} = l^{(i)} - h_w(z_{L-2}^{(i)}) \forall i \)
6: \hspace{3em} Compute \( z_{L-2}^{* (i)} \) by \( z_{L-2,k}^{(i)} = z_{L-2,k}^{(i)} + \alpha \cdot \epsilon^{(i)} \cdot w_k \forall i, k \)
7: \hspace{4em} Estimate \( y_{L-2}^{* (i)} \) from \( z_{L-2}^{* (i)} \) via putback \( \forall i \)
8: \hspace{3em} else
9: \hspace{4em} Compute \( z_{J}^{* (i)} = D_{J+1} y_{J+1}^{* (i)} \forall i \)
10: \hspace{4em} Estimate \( y_{J}^{* (i)} \) from \( z_{J}^{* (i)} \) via putback \( \forall i \)
11: \hspace{2em} end
12: \hspace{1em} end
13: \hspace{1em} subalgorithm Up-propagation
14: \hspace{2em} for \( J := 1 \) to 2
15: \hspace{3em} if \( J == 1 \)
16: \hspace{4em} Compute \( D_{1}^{*} \) by \( d_{1,k} := d_{1,k} - \beta (D_{1} y_{1}^{* (i)} - x_{1}^{(i)}) \left[ y_{1,k}^{* (i)} - y_{1,k}^{(i)} \right] \forall i, k \)
17: \hspace{4em} Compute \( y_{1}^{* (i)} \) by sparse coding with \( D_{1}^{*} \) \( \forall i \)
18: \hspace{4em} Compute \( z_{1}^{* (i)} \) by max pooling \( \forall i \)
19: \hspace{3em} else
20: \hspace{4em} Compute \( D_{J}^{*} \) by \( d_{J,k} := d_{J,k} - \gamma (D_{J} y_{J}^{* (i)} - z_{J-1}^{* (i)}) \left[ y_{J,k}^{* (i)} - y_{J,k}^{(i)} \right] \forall i, k \)
21: \hspace{4em} Compute \( y_{J}^{* (i)} \) by sparse coding with \( D_{J}^{*} \) \( \forall i \)
22: \hspace{4em} Compute \( z_{J}^{* (i)} \) by max pooling \( \forall i \)
23: \hspace{3em} end
24: \hspace{2em} end
25: \hspace{1em} end
26: Retrain classifier \( h_w \) with \( \{z_{L-2}^{* (i)}, l^{(i)}\} \forall i \)
27: \hspace{1em} end
28: until converged
Figure 5.4: Sparse coding and max pooling at hidden layer 1 for single CIFAR-10 image.

1. Two hidden layers of neurons with layer-by-layer pretraining only;
2. Two hidden layers of neurons with both layer-by-layer pretraining and backpropagation.

For fairness, we match the total number of trained weights for multilayer sparse coding and stacked autoencoder schemes. Note that all six schemes considered here have deep architectures.

**Data processing and training.** As in the single-layer evaluation, we form a feature vector for the entire single CIFAR-10 image. The best results for single-layer sparse coding have been achieved with densely overlapping patches drawn from a receptive field width $w = 6$ pixels having a stride $s = 2$. Each patch has size $N = 108$ under this configuration (because there are $6 \times 6 = 36$ pixels from each R, G, and B color dimension). Also, we preprocess
patches by ZCA-whitening. Keeping the same patch configuration, Figure 5.4 illustrates sparse coding and max pooling at hidden layer 1. A CIFAR-10 image is divided into four quadrants, each of which has four (pooling) groups of 9 patches. Hidden layer 1 uses a dictionary size $K_1 = 8N = 864$ with max pooling by $M_1 = 9$. Hidden layer 1 produces $\{z_{11}^{(1)}, \ldots, z_{11}^{(4)}\}, \{z_{11}^{(5)}, \ldots, z_{11}^{(8)}\}, \{z_{11}^{(9)}, \ldots, z_{11}^{(12)}\},$ and $\{z_{11}^{(13)}, \ldots, z_{11}^{(16)}\}$ (4 pooled sparse codes per quadrant), which will be passed to hidden layer 2. In Figure 5.5, the pooled sparse codes from hidden layer 1 undergo sparse coding and max pooling at hidden layer 2. Hidden layer 2 uses a dictionary size $K_2 = 2K_1 = 1728$ with max pooling by $M_2 = 4$. We form the final per-image feature vector $\phi_{MLSC} = \begin{bmatrix} z_{11}^{(1)} \\ z_{11}^{(2)} \\ z_{11}^{(3)} \\ z_{11}^{(4)} \end{bmatrix}$ for classification purposes. With $N = 108$, the feature vector for our multilayer sparse coding has a size $4K_2 = 6912$. We have used the exactly the same images sampled for our single-layer evaluation and tested for 1-vs-all classification.

\footnote{We choose ZCA-whitening over PCA-whitening encouraged by our single-layer results. We have found that ZCA-whitened patches result in slightly better classification performances for CIFAR-10 images.}
Chapter 5: Multilayer Sparse Coding and Deep Sparse-coded Network (DSN)

Figure 5.6: CIFAR-10 1-vs-all classification accuracy comparison for deep SAE, multilayer sparse coding with LARS-0.1 and OMP-20

Table 5.1: Average 1-vs-all classification accuracy comparison

<table>
<thead>
<tr>
<th>Classification accuracy</th>
<th>69.8%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-layer autoencoder</td>
<td>69.8%</td>
</tr>
<tr>
<td>Deep SAE (2 hidden layers, pretraining only)</td>
<td>71.8%</td>
</tr>
<tr>
<td>Deep SAE (2 hidden layers, pretraining+backpropagation whole net)</td>
<td>78.9%</td>
</tr>
<tr>
<td>Single-layer OMP-20</td>
<td>76.9%</td>
</tr>
<tr>
<td>Multilayer OMP-20 (2 hidden layers, pretraining only)</td>
<td>79.6%</td>
</tr>
<tr>
<td>Multilayer OMP-20 (2 hidden layers, pretraining+DSN backpropagation)</td>
<td>84.3%</td>
</tr>
<tr>
<td>Single-layer LARS-0.1</td>
<td>80.1%</td>
</tr>
<tr>
<td>Multilayer LARS-0.1 (2 hidden layers, pretraining only)</td>
<td>83.1%</td>
</tr>
<tr>
<td>Multilayer LARS-0.1 (2 hidden layers, pretraining+DSN backpropagation)</td>
<td>87.8%</td>
</tr>
</tbody>
</table>
Chapter 5: Multilayer Sparse Coding and Deep Sparse-coded Network (DSN)

Results. Figure 5.6 presents per-class 1-vs-all classification accuracy of various sparse coding and autoencoder schemes. We indicate each scheme with “pre” for pretraining only and “opt” for both pretraining and optimization via backpropagation over the whole network. The per-class accuracy shows similar trends that we have observed for our single-layer results with CIFAR-10. Both multilayer sparse coding approaches are consistently better than SAE, and the best performance is achieved by the $\ell_1$-regularized LARS.

In Table 5.1, we summarize the average 1-vs-all classification accuracy of our multilayer sparse coding schemes against deep SAE. For comparison with shallow architectures, we also report the best single-layer results from Chapter 4. All schemes show improvements from their single-layer configurations. Optimization by backpropagation over the whole network is critical for deep SAE as its accuracy gain from multilayering is only about 2%, but it improves more than 7% in classification accuracy by the layer-by-layer pretraining followed by backpropagation. For sparse coding, we gain about 3% from multilayering and another 4% from the DSN backpropagation. Importantly, multilayer OMP-20 with only pretraining is already 0.7% better than deep SAE with both pretraining and backpropagation.

5.5.2 Inferring origin traffic patterns in Wi-Fi

Problem and dataset. In networking, flow is a sequence of data packets traversing the network under shared context such as TCP connection for an application sharing source and destination port numbers and IP addresses. Accurate knowledge about flow characteristics such as burst sizes and inter-departure or arrival times can be used beneficially for network management. In this evaluation, we train 1-vs-all classifiers with features extracted from received traffic patterns sampled at Wi-Fi nodes to infer the origin traffic pattern as were generated at the source). Carrier Sense Multiple Access (CSMA) for 802.11 MAC protocol poses a challenging problem as it can significantly alter the characteristics of a flow. While competing with other flows to access the shared wireless channel, CSMA accumulates data bursts, introduces random delays, and discards frames, impacting original flow patterns with complex, nonlinear changes.

The traffic flow dataset used in the evaluation is generated by OPNET simulation. First
Table 5.2: Origin flow patterns used for evaluation

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Flow type</th>
<th>Generative triplet \langle t_r, s_r, t_g \rangle</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_1</td>
<td>Constant</td>
<td>(2, 100, 4)</td>
</tr>
<tr>
<td>f_2</td>
<td>Constant</td>
<td>(2, 500, 2)</td>
</tr>
<tr>
<td>f_3</td>
<td>Constant</td>
<td>(5, 200, 5)</td>
</tr>
<tr>
<td>f_4</td>
<td>Constant</td>
<td>(10, 200, 10)</td>
</tr>
<tr>
<td>f_5</td>
<td>Stochastic</td>
<td>\langle \text{Exp}(1), \text{Pareto}(100, 2), \text{Exp}(0.1) \rangle</td>
</tr>
<tr>
<td>f_6</td>
<td>Stochastic</td>
<td>\langle \text{Exp}(0.5), \text{Pareto}(40, 1), \text{Exp}(0.25) \rangle</td>
</tr>
<tr>
<td>f_7</td>
<td>Stochastic</td>
<td>\langle \text{U}(4, 10), \text{Pareto}(100, 2), \text{Exp}(0.5) \rangle</td>
</tr>
<tr>
<td>f_8</td>
<td>Stochastic</td>
<td>\langle \text{N}(10, 5), \text{Pareto}(40, 1), \text{N}(10, 5) \rangle</td>
</tr>
<tr>
<td>f_9</td>
<td>Mixed</td>
<td>(1, \text{Pareto}(100, 2), 1)</td>
</tr>
<tr>
<td>f_{10}</td>
<td>Mixed</td>
<td>(1, \text{Pareto}(100, 2), \text{Exp}(0.25))</td>
</tr>
</tbody>
</table>

of all, we use the runs-and-gaps model [96] to generate data traffic flows. The flow pattern is described as finite-length time-series data written in vectors. The triplet \langle t_r, s_r, t_g \rangle describes the generative pattern of a flow, where \( t_r \) and \( t_g \) are the run and gap lengths in number of unit intervals \( T_s \), and \( s_r \) denotes the size of payload in bytes generated per each unit interval of a run. A flow type can be constant, stochastic, or mixed. A constant flow has deterministic \( t_r, s_r, \) and \( t_g \) values. For example, flow 1 with \( \langle 2, 100, 4 \rangle \) creates the origin flow pattern (time series) \( f_1 = [100 100 0 0 0 0 100 100 0 0 \ldots ] \). For a stochastic flow, \( t_r, s_r, \) and \( t_g \) are random variables. For example, flow 6 with \( \langle \text{Exp}(0.5), \text{Pareto}(40, 1), \text{Exp}(0.25) \rangle \) has exponentially distributed run and gap lengths (with mean of \( \frac{1}{0.5} \) and \( \frac{1}{0.25} \) unit intervals, respectively), and Pareto-distributed payload sizes. An instance of flow 6 could be \( f_6 = [518 97 0 0 0 0 0 32 0 \ldots ] \). Table 5.2 summarizes 10 origin flows in the dataset. Using \( T_s = 10 \) msec, we generate 2,000 instances for each flow. Each instance is a vector of size 500. These instances reflect the origin traffic pattern of a flow passed to OPNET that simulates Wi-Fi transmission and reception.

OPNET provides several built-in models for Wi-Fi. We have chosen IEEE 802.11g and configured the parameters with the values described in Table 5.3. In Figure 5.7, we illustrate our Wi-Fi scenario simulated in OPNET. The roles of Wi-Fi nodes in the simulation are described in Table 5.4. We are interested in predicting the original patterns of the traffic generated by two flow source nodes A and C from packet sampling done at node B, which is an Wi-Fi access point. Nodes D–G are engaged in Internet-style of communications and generate background traffic that competes with the flows from A and C.
Table 5.3: Summary of Wi-Fi parameter configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aSlotTime</td>
<td>Slot time</td>
<td>20 μsec</td>
</tr>
<tr>
<td>aSIFSTime</td>
<td>Short interframe space (SIFS)</td>
<td>10 μsec</td>
</tr>
<tr>
<td>aDIFSTime</td>
<td>DCF interframe space (DIFS)</td>
<td>50 μsec</td>
</tr>
<tr>
<td>aCWmin</td>
<td>Min contention window size</td>
<td>15 slots</td>
</tr>
<tr>
<td>aCWmax</td>
<td>Max contention window size</td>
<td>1023 slots</td>
</tr>
<tr>
<td>tPLCPPreamble</td>
<td>PLCP preamble duration</td>
<td>16 μsec</td>
</tr>
<tr>
<td>tPLCP SIG</td>
<td>PLCP SIGNAL field duration</td>
<td>4 μsec</td>
</tr>
<tr>
<td>tSymbol</td>
<td>OFDM symbol duration</td>
<td>4 μsec</td>
</tr>
</tbody>
</table>

Figure 5.7: Seven Wi-Fi node scenario in OPNET

Table 5.4: Node roles in seven Wi-Fi node scenario

<table>
<thead>
<tr>
<th>Node</th>
<th>Role</th>
<th>Main networking activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Flow source</td>
<td>Transmits $f_i$</td>
</tr>
<tr>
<td>B</td>
<td>Receiver</td>
<td>Intercepts flows as Wi-Fi router/AP</td>
</tr>
<tr>
<td>C</td>
<td>Flow source</td>
<td>Transmits $f_j \quad \forall j \neq i$</td>
</tr>
<tr>
<td>D</td>
<td>Flow source</td>
<td>Multimedia streaming over RTP/UDP/IP</td>
</tr>
<tr>
<td>E</td>
<td>Flow dest.</td>
<td>HTTP with page size $\sim U[10,400]$ B</td>
</tr>
<tr>
<td>F</td>
<td>Flow dest.</td>
<td>ftp file transfer with size 50000 B</td>
</tr>
<tr>
<td>G</td>
<td>Flow dest.</td>
<td>DB access with inter-arrival $\sim \text{Exp}(3)$ sec</td>
</tr>
</tbody>
</table>
When collecting the data, we input the generative instance vectors (describing the origin flow pattern) for OPNET and sample Wi-Fi frames for the duration of $1,000 \times T_s$. Thus, each received flow measurement corresponds to a vector of 1,000 elements whose values indicate run sizes in received bytes (zero means a gap). These measured vectors are received flow patterns given the origin pattern instance input.

**Data processing and training.** We cut the dataset for 5-fold cross validation. We use 4 folds for training and 1 fold for testing. The training folds are used for unsupervised learning and supervised classifier training. We have tried 3 different patch sizes $N = 100, 200, 300,$ and $500$ extracted from the measured flow patterns. We do not preprocess the patches with PCA or ZCA whitening but normalized them by removing patch means before dictionary learning.

For single-layer sparse coding, we use OMP and K-SVD. We have chosen dictionary size $K \approx 10N$ and sparsity level $S = 0.2K$. In general, it is known difficult to learn effective feature representation for time-series data by clustering methods. This is largely due to impracticality of compensating all possible shifts of a time series.\(^3\) For partial remedy, we have performed dense feature extraction with consecutive patches overlapped by 95%. We have also used an overlapping pooling scheme taken from Krizhevsky et al. [88] that further improves on the translational invariance of the features and reduces classifier overfit. A pooling factor of $M = 8$ is used.

Multilayer sparse coding we use has two hidden layers as in Figure 5.2. We configure the first hidden layer identical to the single-layer setup, which produces the pooled sparse codes $z_1 \in \mathbb{R}^K$. At the second hidden layer, we use a dictionary size $K_2 = 4K$, sparsity level $S_2 = 0.2K_2$, and pooling factor $M_2 = 4$ to produce the pooled sparse codes $z_{II} \in \mathbb{R}^{4K}$. We use the second hidden layer output $z_{II}$ to train classifiers and test. Thus, the final feature vectors for our multilayer sparse coding have quadrupled the feature size of the single-layer setup.

We train 1-vs-all linear SVM classifiers. Since there are 10 origin flow patterns, we

\(^3\)Although we have not explored this option, we can use shift-invariant sparse coding algorithms [97]. The optimization for shift-invariance, however, is much more demanding computationally. Nevertheless, shift-invariant dictionaries have proved representational efficiency for time-series data.
need ten 1-vs-all SVMs each of which classifiers one flow against all the others for either single-layer or multilayer feature learning. To benchmark the classification performances of our single-layer and multilayer sparse coding approaches, we accompany three classical schemes, autoregressive moving average with exogenous inputs by recursive least squares (ARMAX-RLS) [98, 99], naïve Bayes [21], and Gaussian mixture model (GMM) [100]. We train the ARMAX model with $n$ previously transmitted ground-truth flow patterns $\{f_{i,k-1}, \ldots, f_{i,k-n}\}$ and $m$ previously received patterns $\{x_{i,k-1}, \ldots, x_{i,k-m}\}$ for origin flow pattern $i$ at time $k$. By trial and error, we choose $n = 2$ and $m = 3$. We train 1-vs-all linear SVM classifiers with the ARMAX least squares estimates $\hat{f}_{\text{ARMAX}}$ given measured input $x$.

For naïve Bayes, we extract the statistic $y = [\hat{\mu}_{\text{run size}}, \hat{\mu}_{\text{gap length}}]$ from measured pattern $x$ by computing mean run burst size and gap length. Using $y$ as a feature with label $l_i$ for flow $i$, we build empirical distributions $p(y|x, l_i)$. Naïve Bayes predicts a class label by $\hat{l} = \arg \max_l p(l|y) \propto p(y|x, l)$.\(^4\)

We compute the same $y$ used in naïve Bayes from flow measurements to train GMM by the EM algorithm [101]. We train 1-vs-all linear SVMs using the posterior probabilities computed from the trained Gaussians. We use the number of Gaussians $K = 100$ for the mixture model. We will report classification accuracy measured by recall (true positive rate) and false alarm (false positive rate) metrics:

\[
\text{Recall} = \frac{\sum \text{True positives}}{\sum \text{True positives} + \sum \text{False negatives}},
\]

\[
\text{False alarm} = \frac{\sum \text{False positives}}{\sum \text{False positives} + \sum \text{True negatives}}.
\]

**Results.** Figure 5.8 presents 1-vs-all classification performances of 1) ARMAX-RLS, 2) naïve Bayes, 3) Gaussian mixtures, 4) single-layer sparse coding, 5) multilayer sparse coding with layer-by-layer pretraining only, and 6) multilayer sparse coding optimized by the DSN backpropagation after pretraining. We have computed the average recall and false alarm rate from the cross-validated results. Overall, sparse coding approaches have re-

\(^4\)Naïve Bayes is a classifier by itself, so we do not need to train any SVMs for naïve Bayes.
Figure 5.8: Seven Wi-Fi node scenario in OPNET
sulted consistently higher recall with lower false alarms than ARMAX and naïve Bayes. A possible explanation for poor ARMAX performance is that the changes introduced to the origin flow pattern by the 802.11 CSMA are of nonlinear nature. The performance of Gaussian mixtures are on par with single-layer sparse coding. This result is reasonable because sparse coding essentially is a generative model based on Gaussian assumptions. If optimized to their limits, Gaussian mixtures are more fundamental alternative to sparse coding for classification.

5.6 Summary

Motivated by superior feature learning performance of sparse coding, we describe a deep architecture on multilayer sparse coding. DSN is a feedforward network having two or more sparse coding layers, each of which is trained to learn its own dictionary. Sparse coding layers interlaced by max pooling that serves as nonlinear activation function in neural networks. This chapter has discussed the rationale of DSN, highlighted the advantage, and discussed training methods including a novel backpropagation algorithm. We have empirically evaluated multilayer sparse coding using CIFAR-10 and Wi-Fi traffic data.
Chapter 6

Nearest Neighbor Sparse Coding

6.1 Introduction

We have seen sparse feature representations beneficial for discriminative learning tasks and demonstrated how sparse coding can be used to build high-performance classification pipelines for different modalities of data such as text, image, and time series. In this chapter, we describe a simple, yet powerful enhancement for sparse coding and dictionary learning. Our idea is based on $k$-Nearest Neighbor ($k$-NN), a data-driven, nonparametric classification algorithm. Key to the enhancement is to promote the sharing of dictionary atoms among the data that fall under the nearest neighbor constraint in their sparse codes.

The effectiveness of nearest neighbor search in the sparse feature domain is largely unknown and undervalued. Boiman et al. [102] have successfully built a high-performance image classifier that examines the nearest neighbors of a sparse code-like codeword for an image encoded using local descriptors. Their classifier is ranked among the top empirical performances on the Caltech-101, Caltech-256, and Graz-01 benchmarks [103, 104]. Approximate nearest neighbor (ANN) [105] collectively refers to a sparse coding-based technique to retrieve highly correlated data objects. Recently, it is shown that the $k$-NN search in sparse codes of microblog text leads to an accurate geolocation system [15] outperforming the best-known latent topic models.

We name our approach Nearest Neighbor Sparse Coding (NNSC). We will discuss theoretical properties of NNSC, present an algorithmic improvement for classical OMP
Chapter 6: Nearest Neighbor Sparse Coding

and LARS, and evaluate the empirical performance with both simulated and real datasets. Despite suboptimal reconstructive accuracy, NNSC can be used to produce more label-consistent sparse representations for OMP and LARS that can substantially improve the classification performance.

6.2 Nearest Neighbor Sparse Coding (NNSC)

The idea of NNSC is simple and intuitive. First of all, NNSC is an outer algorithmic enhancement. NNSC is neither tied to specifics of how sparse coefficients are settled by LARS or OMP, nor how dictionary atoms are adjusted in the inner loop execution of a particular dictionary learning algorithm. Unlike classical sparse coding, NNSC requires labeled examples during the initial dictionary learning. The entire training dataset may not need to be labeled, but more labeled examples will help achieve more stable sparse codes for NNSC.

Assume that we are given a trained dictionary $D$ from classical sparse coding. For an input vector $x$, we first compute a sparse code $y$ via unmodified OMP or LARS. We then search for $Y_{knn}$, the $k$-nearest neighbors ($k$-NNs) of $y$ in the sparse feature domain (i.e., the $K$-dimensional space spanned by $x$’s sparse codes).

Let us express $Y_{knn} = \{y_{(1)}, y_{(2)}, \ldots, y_{(k)}\}$ such that $\text{cossim}(y, y_{(1)}) \geq \text{cossim}(y, y_{(2)}) \geq \cdots \geq \text{cossim}(y, y_{(k)})$, where $\text{cossim}(a, b) = \frac{a \cdot b}{\|a\|\|b\|}$ is cosine similarity$^1$ between two vectors, and $y_{(i)}$ the $i$th nearest neighbor of $y$. For NNSC, cosine similarity measures the proximity between a pair of sparse codes to determine $k$-NNs of a given sparse code. Note that the $k$-NNs are retrieved from sparse codes of the training examples used in the initial dictionary learning. By majority label voting, we deduce $Y_{knn}^\dagger = \{y_{(1)}^\dagger, y_{(2)}^\dagger, \ldots, y_{(k')}^\dagger\}$, the majority subset of $Y_{knn}$. That is, $y_{(i)}^\dagger$’s share the same class label, and the label is determined by the majority class votes in $\{y_{(1)}, y_{(2)}, \ldots, y_{(k)}\}$. Note also that $\text{cossim}(y, y_{(1)}^\dagger) \geq \text{cossim}(y, y_{(2)}^\dagger) \geq \cdots \geq \text{cossim}(y, y_{(k')}^\dagger)$, and $k' \leq k$. This is illustrated in Figure 6.1.

For simplicity, this example considers two classes (depicted blue circle and red cross in the figure), and we obtain $Y_{knn}^\dagger = \{y_{(1)}^\dagger = y_{(3)}, y_{(2)}^\dagger = y_{(5)}, y_{(3)}^\dagger = y_{(6)}, y_{(4)}^\dagger = y_{(7)}\}$ with

$^1$Use of the cosine similarity metric is our preferred choice, and other similarity or dissimilarity metrics such as Euclidean distance can be used instead.
Chapter 6: Nearest Neighbor Sparse Coding

Figure 6.1: $k$-NNs in sparse feature domain and majority voting

$k' = 4$.

With $Y_{knn}^\dagger$ learned, we can construct $D^\dagger = \begin{bmatrix} d^\dagger_{(1)} & d^\dagger_{(2)} & \ldots & d^\dagger_{(K')} \end{bmatrix}$. Here, the majority label sparse codes $y^\dagger_{(i)}$’s have dictionary atoms $d^\dagger_{(j)}$’s in common. This is the NNSC support set that consists of $K'$ best basis vectors under the $k$-NN majority voting rule. We solve for NNSC-OMP by

$$y^\dagger = \arg\min_y \|x - D^\dagger y\|_2^2 \text{ s.t. } \|y\|_0 \leq S$$ \hspace{1cm} (6.1)

Similarly for the $\ell_1$-regularized LARS, we have

$$y^\dagger = \arg\min_y \|x - D^\dagger y\|_2^2 + \lambda \|y\|_1$$ \hspace{1cm} (6.2)

During feature learning with NNSC, it is important to update the dictionary according to the changes made to sparse codes by NNSC. Given $n$ training examples $X = \{x^{(1)}, \ldots, x^{(n)}\}$, we compute sparse codes $Y^\dagger = \{y^{(1)}\dagger, \ldots, y^{(n)}\dagger\}$ via NNSC. Then, the NNSC dictionary updates take place by

$$D = \arg\min_D \|X - DY^\dagger\|_F^2 \text{ s.t. } \|d_k\|_2^2 = 1 \ \forall k$$ \hspace{1cm} (6.3)

We can use off-the-shelf dictionary learning algorithms such as K-SVD to update dictionary after NNSC. The NNSC algorithm is presented in Algorithm 5. Note that the algorithm
Chapter 6: Nearest Neighbor Sparse Coding

works for both OMP and LARS. Also, we default to rank-1 updates (used in the inner loop of K-SVD) to adjust D to yield NNSC-trained dictionary $D^\dagger$.

Algorithm 5 Nearest Neighbor Sparse Coding (NNSC)

input Training examples $X = \{x^{(1)}, \ldots, x^{(n)}\}$ with labels $L = \{l^{(1)}, \ldots, l^{(n)}\}$, number of subdictionary atoms $K'$ and $K''$, penalty $\lambda$ or sparsity level $S$

output NNSC-trained dictionary $D^\dagger$ and sparse codes $Y^\dagger$ for $X$

1: Do initial sparse coding and dictionary learning $X \rightarrow \{Y, D\}$
2: For each $y^{(i)} \in Y$, do $k$-NN search to obtain $Y_{knn}^{(i)} = \{y^{(1)}, \ldots, y^{(k)}\}$
3: Predict class of $y^{(i)}$ by majority voting with $Y_{knn}^{(i)}$
4: if Predicted label is correct
5: Form subdictionary $\Delta$ by setting only $K'$ top weighted atoms from $k$-NN sparse codes $y^{(j)}$'s active and the rest zeros
6: else
7: Form subdictionary $\Delta$ by setting only $K''$ top weighted atoms from all ground-truth class sparse codes for $y^{(i)}$ active and the rest zeros
8: end
9: Redo sparse coding $y^{(i)\dagger} = \arg\min_y \|x - \Delta y\|_2^2 + \lambda \|y\|_1$ for LARS
or $y^{(i)\dagger} = \arg\min_y \|x - \Delta y\|_2^2$ s.t. $\|y\|_0 \leq S$ for OMP
10: Using $y^{(i)\dagger}$’s, adjust dictionary $D \rightarrow D^\dagger$ via rank-1 updates

6.3 Theoretical analysis

NNSC computes sparse coefficients from a subset of basis vectors searched under the probabilistic guarantees of $k$-NN classification. The NNSC algorithm uses the nearest neighbor constraint to search for a class-wise support set (i.e., subset of dictionary) desirably shared by sparse codes of the data vectors belonging to the same class. This makes more dictionary atoms shared among sparse codes of the data vectors belonging to the same class, thereby strengthening their coherency. Conversely, as less dictionary atoms are shared between sparse codes of different classes, NNSC enhances the incoherency between classes. However, we have to reason about validity of the assumption that $k$-NN with majority voting indeed gives a correct class label probabilistically. In this section, we give theoretical justifications about why NNSC works. We begin by the following lemma on asymptotic convergence of the nearest neighbor in generalized $N$-dimensional vector space.

80
Lemma 1 (Asymptotic Convergence of the Nearest Neighbor) Let $x$ and $x_1, x_2, \ldots \in \mathbb{R}^N$ be independent identically distributed random variables taking values in a separable metric space $X$. Let $x_{(1)}$ denote the nearest neighbor to $x$ from the set $\{x_1, x_2, \ldots, x_n\}$. Then $x_{(1)} \rightarrow x$ with probability 1.

**Proof.** See Cover & Hart [80]. \square

Corollary 2 For fixed $k$, as $n \rightarrow \infty$, also does the $k$th nearest neighbor $x_{(k)} \rightarrow x$ with probability 1.

According to Lemma 1, the simple majority voting rule that NNSC uses is robust in predicting the label for a given input $x$. In the limit of infinitely large sample size, remarkably the error will become zero with probability 1. Furthermore, by Corollary 2, Cover & Hart assert that not only the nearest neighbor label gives the correct label, but also gives the $k$th nearest neighbor, when $k$ is fixed. These mean that if we can maintain a reasonably large number of examples available for NNSC, we can produce optimally discriminative feature vectors from learning.

Using these results as a ground, we can prove that NNSC indeed produces more discriminative, label-consistent sparse codes than the classical approaches. In classification, separability of classes is examined by whether or not there could be a hyperplane that the data for Class 1 lie on one side, and the data for Class 2 on the opposite side as depicted in Figure 6.2. For simplicity, we explain a case for binary classification and linear separating hyperplane. Here, the ground-truth label for $x$ is Class 2. To classify $x$, we use its feature vector $y$ computed by (classical) sparse coding. The decision boundary relating the hyperplane is described by the tuple $(w, b) \in \mathbb{R}^K$ where $w$ is orthogonal to the hyperplane. Geometric margin $\gamma$ for $y$ measures distance of $y$ from the hyperplane

$$\gamma = \frac{w^T \cdot y + b}{\|w\|}.$$  

Geometric margin can be positive or negative. Since vector $w$ directs toward Class 1\(^2\) (red crosses), $\gamma$ should be positive for the example in Figure 6.2. Note that any $y$ lying on the

\(^2\)This means that Class 1 is assigned with the label +1, and Class 2 -1 for classifier training.
Figure 6.2: NNSC produces a new sparse code $y^{\dagger}$ for $x$ closer to its ground-truth class cluster in the feature domain.

hyperplane should satisfy $w^\top \cdot y + b = 0$.

With $y^{\dagger}$, the sparse code computed by NNSC, whose geometric margin is $\gamma^{\dagger} = \frac{w^\top \cdot y^{\dagger} + b}{\|w\|}$, the proof we need is equivalent to:

$$\text{NNSC makes } \gamma^{\dagger} \leq \gamma$$

if the direction of $w$ is toward the opposite class of $x$. For clarity of explanation, let us ignore the bias $b$ (i.e., $b = 0$). The coefficients of $w$ is chosen by optimizing best margins for data in both classes with respect to the hyperplane. Since margins for Class 1 feature vectors should induce positive values, the support or nonzero positions of features vectors for Class 1 are where $w$ have positive coefficients. On the other hand, margins for Class 2 should be negative, thus nonzero positions for Class 2 feature vectors are where $w$ has negative coefficients accordingly. More generally, by letting $D_+$ and $D_-$ be subdictionaries with dictionary atoms from $D$ common in Class 1 and Class 2 feature vectors, the respective positions of the atoms in $D_+$ and $D_-$ at $D$ indicate signs for the coefficients of $w$. When solving for $y^{\dagger}$, NNSC uses a subset $D^{\dagger}$. By guarantees of the nearest neighbor constraint from Lemma 1 and Corollary 2, we claim $|D_- \cap D^{\dagger}| \geq |D_+ \cap D^{\dagger}|$. Therefore, we
Chapter 6: Nearest Neighbor Sparse Coding

**Sparse coding & initial dictionary learning**

\[
X_t \xrightarrow{\text{LARS/OMP \\ & K-SVD}} (D, Y_t)
\]

**NNSC & dictionary update**

\[
(X_t, D, Y_t, I_t) \xrightarrow{\text{NNSC \\ (Alg. 5)}} Y_t'
\]

\[
(X_t, D, Y_t') \xrightarrow{\text{K-SVD}} (D, Y_t')
\]

Figure 6.3: Feature learning via NNSC. Initial dictionary learning is done by classical sparse coding. An alternating procedure to find NNSC codes and dictionary updates follows and repeats until convergence.

**Classifier training**

\[
(Y_t, l_t) \xrightarrow{\text{SVM}}
\]

\[
(Y_t', l_t) \xrightarrow{\text{SVM}'}
\]

Figure 6.4: Linear SVM classifiers are trained using features extracted from classical sparse coding and NNSC.

conclude \( w^T \cdot y^{\dagger} \leq w^T \cdot y \), which implies \( \gamma^{\dagger} \leq \gamma \). □

### 6.4 Evaluation

In this section, we evaluate the empirical performance of NNSC using simulated data from Gaussian mixture models (GMMs) as well as GEO\textsc{Text} and CIFAR-10 datasets. We use a simple classification pipeline based on single-layer sparse coding that uses classical approaches, LARS and OMP, and their respective NNSC improvements. We use K-SVD to train dictionaries for all classical and NNSC scenarios.
6.4.1 Feature learning, training, and testing with NNSC

Figure 6.3 illustrates the NNSC feature learning. This process requires training examples \( X_t = \{x^{(1)}, \ldots, x^{(n)}\} \) and their class labels \( l_t = \{l^{(1)}, \ldots, l^{(n)}\} \). Based on our previous results, we consider LARS-0.1 and OMP-20. The initial dictionary learning by classical sparse coding yields \( D \) and sparse codes \( Y_t \). NNSC takes in \( \{X_t, D, Y_t, l_t\} \) to produce new sparse codes \( Y_t^\dagger \). It is necessary to fold in \( Y_t^\dagger \) to \( D \). To do so, we consider an alternating procedure comprising NNSC and dictionary update by K-SVD. Similar to classical sparse coding, we repeat the procedure until we see convergence of \( Y_t^\dagger \) and \( D \).

We use sparse codes as feature to train linear SVM classifiers as depicted in Figure 6.4. The first SVM is trained with \( Y_t \), the sparse feature vectors computed by either LARS or OMP, whereas the second SVM (denoted SVM\(^\dagger\) in the figure) is trained with the NNSC sparse codes \( Y_t^\dagger \). Figure 6.5 describes classification testing. Given a test input \( x \), classical sparse coding results in \( y \), which is used to predict the label \( \hat{l} \). For NNSC, \( y^\dagger \) is used to predict the label \( \hat{l}^\dagger \).

6.4.2 Experiments with simulated data

We first evaluate the reconstructive accuracy and classification performance of NNSC against classical sparse coding using simulated data from Gaussian mixture models. We generate two datasets each of which has 12,000 labeled examples. The first dataset has two classes of data, and the second with four classes. We use dimensionality \( N = 64 \) for each
Chapter 6: Nearest Neighbor Sparse Coding

Figure 6.6: Mean normalized reconstructive errors for NNSC-OMP and OMP
datum in the datasets—i.e., Gaussians in our mixture models are 64-variate. We have randomly chosen the mean and standard deviations for the Gaussians in the intervals [-10, 10] and [1, 5], respectively. Each dataset is cut to two for training and testing. The training set has 10,000 examples. To observe the asymptotic behavior of NNSC, we train the dictionary with all 10,000 examples but vary the availability of labels from 1,000 to 10,000 examples. The testing set has \( n = 2,000 \) samples. For K-SVD, we set the number of dictionary atoms \( K = 10N = 640 \). For sparse coding, we consider LARS-0.1 and OMP-20. For NNSC, we use \( k = 100 \) for the \( k \)-NN majority voting.

In Figure 6.6, we plot mean normalized reconstructive error \( \bar{\epsilon} = \frac{1}{n} \sum_{i=1}^{n} \frac{\|x^{(i)} - D y^{(i)}\|_2}{\|x^{(i)}\|_2} \) for classical OMP-20, LARS-0.1, and their NNSC variants. Since NNSC computes sparse codes from a reduced subset \( D^\dagger \), its reconstructive error is generally greater than classical sparse coding. Empirically, we find that the reconstructive accuracy of NNSC is at best equal to classical sparse coding that encodes using the entire \( D \). As we increase the availability of labeled examples, NNSC starts to benefit from more diverse set of dictionary atoms and performs better in reconstruction. On the other hand, both OMP-20 and LARS-0.1 errors remain flat regardless of the availability of labeled examples. This is expected because sparse coding is an unsupervised learning method independent of labeled training.
Chapter 6: Nearest Neighbor Sparse Coding

Figure 6.7: Binary classification accuracy for GMM-generated dataset

Figure 6.8: 4-class classification accuracy for GMM-generated dataset

Table 6.1: Regional and state classification accuracy of OMP-20, LARS-0.1, and their NNSC enhancements for GEOTEXT

<table>
<thead>
<tr>
<th></th>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>US region (4-way)</td>
</tr>
<tr>
<td>OMP-20, ( N = 64 )</td>
<td>66.7%</td>
</tr>
<tr>
<td>OMP-20 with NNSC \hspace{1em}improvement, ( N = 64 )</td>
<td>70.2%</td>
</tr>
<tr>
<td>LARS-0.1, ( N = 64 )</td>
<td>69.5%</td>
</tr>
<tr>
<td>LARS-0.1 with NNSC \hspace{1em}improvement, ( N = 64 )</td>
<td>71.3%</td>
</tr>
</tbody>
</table>

data amount.

With the same dataset, we also test NNSC for the use in discriminative tasks. In Figures 6.7 and 6.8, we present the binary and multi-class classification accuracy of NNSC against classical sparse coding. For clarity of analysis, we have trained linear SVM classifiers using sparse codes without nonlinear pooling. We find that NNSC can enhance the classification performance of classical sparse coding. In particular, the improvement for OMP is more significant than LARS. As the amount of labeled examples available for training increases, all classical sparse coding and NNSC schemes result in better performance. This is because the classifiers become statistically more robust by working through a larger set of examples, reducing the overall risk of overfit. With more training data, the increased number of nearest neighbors further helps NNSC as its lead over classical sparse coding is consistent and becomes clearer in the steady state.
Table 6.2: 1-vs-all classification accuracy of OMP-20, LARS-0.1, and NNSC for CIFAR-10 under dense feature extraction ($w = 6$, overlapping patches with $s = 2$)

<table>
<thead>
<tr>
<th>Classification accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMP-20</td>
</tr>
<tr>
<td>OMP-20 with NNSC improvement</td>
</tr>
<tr>
<td>LARS-0.1</td>
</tr>
<tr>
<td>LARS-0.1 with NNSC improvement</td>
</tr>
</tbody>
</table>

6.4.3 Performance on GEOText dataset

Previously, we have evaluated single-layer OMP-20 and LARS-0.1 using GEOText. We reevaluate single-layer sparse coding with NNSC improvements. Table 6.1 reports the regional and state classification accuracy of classical sparse coding and their NNSC improvements with GEOText. As we have seen with the simulated datasets, we observe a similar trend that the improvement is more significant for OMP than LARS for GEOText. NNSC enhances the classification performance of OMP-20 by 3.5% and 4.3%, whereas the improvements are only 1.8% and 2.2% for LARS-0.1.

6.4.4 Performance on CIFAR-10 dataset

We reevaluate the performance of single-layer sparse coding on CIFAR-10 with NNSC enhancements. We also find that NNSC enhances both OMP-20 and LARS-0.1 for classifying CIFAR-10 images. The improvements are by 3.5% for OMP-20 and 1.5% for LARS-0.1. The results are summarized in Table 6.2.

6.5 Summary

We have presented Nearest Neighbor Sparse Coding (NNSC). NNSC is based on nearest neighbor classification, one of the oldest nonparametric learning algorithms. Despite its simplicity, the asymptotic property of $k$-NN is among the most favorable. We have devised NNSC by exploiting probabilistic guarantees of $k$-NN when combined with majority voting. As an outer algorithmic enhancement, NNSC can improve discriminative feature learning of classical sparse coding. In particular, we have shown that NNSC substantially
improves the classification performance of OMP, a greedy-$\ell_0$ sparse encoder known for suboptimality and occasional instability. We conjecture interesting properties of NNSC.

1. NNSC produces label-consistent sparse codes;
2. NNSC promotes nonnegativity;
3. NNSC reconstructive error at best equals classical sparse coding;
4. NNSC incurs smaller classification errors than classical sparse coding.

With NNSC, we have established a foundation for beneficially using the nearest neighbor argument in the sparse feature domain. However, NNSC opens up more future research opportunities and problems. We believe that NNSC holds a refreshingly new direction for data-driven machine learning.
Chapter 7

Conclusion

This thesis has explored the problem of applying sparse modeling for reconstructive and discriminative applications in sensing, wireless networking, and machine learning. Building on the two sparse computing models, compressive sensing and sparse coding, we have developed sparse recovery and feature learning methods robust to complex data transformations. Despite their mathematical equivalence, compressive sensing and sparse coding instantiate two different classes of application for sparse data modeling. Compressive sensing enables (exactly) recoverable, low-dimensional compressive data representations using a blind linear projection. If we can introduce the projection as early as in the sensing front-end, our sensing systems will truly benefit. On the other hand, sparse coding enables the discovery of localized, structural patterns underlying the raw data in forced decomposition over a given dictionary. Sparsity is the sole constraint that makes forced decomposition unveil meaningful features. By exploiting the degree of overcompleteness for the dictionary, we can improve the chance of linear separability among computed sparse codes that will ultimately reflect the classification performance.

This thesis has emphasized on the empirical evaluation of the proposed approaches. We have demonstrated the enhanced performance that are resulted not only from algorithmic and system design factors, but also from our domain-specific knowledge. Machine learning provides fundamental tools to cope with complexity and ambiguity of data, but it often requires to apply domain- or application-specific insights to set up the overall pipeline and fine-tune the model parameters to meticulous detail.
Chapter 7: Conclusion

We summarize our key lessons.

1. The benefits for multi-way combining of compressed measurements come with the cost of various distortions. To recover from the mixed compressed measurements, we have worked in two levels: coarse separation using the bases learned from PCA at the first level followed by iterative refinement through sparse recovery at the second level. For our spectrum sensing application, the overall heavy subsampling and nonlinearities (from compressive sensing and mixing) could be denoised to give fine-grained spectrum information equivalent to FFT analyzer on Nyquist sampling;

2. We show how the classical problem in networking can benefit from machine learning. We apply the features extracted from Wi-Fi traffic patterns by sparse coding to infer the origin traffic pattern described in terms of characteristic burst sizes and interarrival times. Complex traffic mix introduced by CSMA that affect the origin traffic pattern can be overcome by feature transformation via sparse coding;

3. We have proposed a deep architecture with multilayer sparse coding and max pooling and described training methods to enable the learning of a possible hierarchy in features. Our DSN has significantly improved both classification accuracy and training speed compared to the conventional, backpropagation-trained deep learning schemes;

4. As an outer algorithmic enhancement, NNSC improves on discriminative feature learning of classical sparse coding by imposing the nearest neighbor constraint in the sparse feature domain. Although preliminary, we have found that label-consistent sparse codes produced by NNSC improves the classification performance of classical sparse coding.

In concluding remarks, we believe that algorithms with ease and general applicability to multiple modalities of data can only make significant impact. In this thesis, we have partly shown by applying our methods to wireless signals, text, image, and time-series data. For immediate future work, we plan to extend our framework for multimodal sparse coding. There are significant benefits in computational efficiency of a system that can learn directly from compressed representations. Our future study also includes compressed domain learn-
Chapter 7: Conclusion

...ing such as geographical classification and inference using subsampled microblog text. We believe that NNSC holds many exciting research opportunities and problems. Thus, we will follow up NNSC and its new direction for data-driven machine learning as well.
Bibliography


