Differentiable and Robust Optimization Algorithms

Thomas Powers

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Les E. Atlas, Chair

David W. Krout

Jeffrey Bilmes

Program Authorized to Offer Degree:
Electrical and Computer Engineering

University of Washington

Abstract

Differentiable and Robust Optimization Algorithms

Thomas Powers

Chair of the Supervisory Committee:
Professor Les E. Atlas
Electrical and Computer Engineering

Imposing appropriate structure or constraints onto optimization problems is often the key to deriving guarantees or improving generalization of performance aspects like generalization or interpretability. The main contribution of this dissertation is developing algorithms that can leverage underlying submodular or sparse structure to do robust optimization: unfolded discrete and continuous optimization algorithms and robust submodular optimization. While deep neural networks (DNNs) continue to advance the state-of-the-art for many tasks in fields such as speech and audio processing, natural language processing, and computer vision over traditional statistical generative models, many of the most popular architectures are difficult to analyze and require large amounts of data to achieve such good performance. The choice between DNN and generative model need not be binary, however. Using a technique called deep unfolding, inference algorithms for generative models can be turned into DNNs and trained discriminatively. Such models can also leverage sparisty, submodularity, or other regularization frameworks while still being able to make use of domain knowledge integrated into the underlying model. Subset selection problems are important for many applications in machine learning such as feature and data selection, dictionary learning, compression, and sparse recovery. While these problems are generally NP-hard, if the objective function is submodular (or in some cases has submodular structure), a near-optimal solution can be found in polynomial-time. Deep learning and subset selection have overlap when sparse

models are used. For DNNs and other continuous models, sparsity is typically induced via penalty function such as the ℓ_1 norm, whereas sparsity is induced via a hard cardinality constraint for subset selection algorithms. This dissertation explores algorithms around the intersection of subset selection and deep learning. Broadly, there are two sets of algorithms presented in this dissertation. In the first, algorithms are designed to approximately solve both submodular and non-submodular optimization problems. These algorithms are applied to sensor scheduling and placement problems. In the second, deep unfolding is used to turn inference algorithms into DNNs. These unfolded models have a number of advantages over conventional DNN models, and are shown to have competitive or improved performance on a variety of tasks, can have principled initializations, and tend to need less data compared to conventional network architectures.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The process of earning a PhD is long, meandering, and difficult, but also incredibly rewarding and enlightening. I consider myself extremely lucky to have had so many people to help me along the way, without whom I could not have made it this far. First off, I would like to thank my advisors, Les Atlas and David Krout, for both giving me guidance and support while also allowing me to explore topics outside of their core research areas. I would also like to thank the rest of my committee, Jeff Bilmes and Bob Odom. Professor Bilmes has been instrumental in my exploration of submodularity, which constitutes much of my research. Thank you to John Hershey and Jonathan Le Roux for hosting and mentoring me at MERL. Thank you to Abhinav Sethy and Abdo Mohamed for hosting and mentoring me at Amazon. I feel fortunate that I was able to work with and learn from Shane Settle, Rasool Fakoor, Siamak Shakeri, and Amanjit Kainth during my internships as well.

Thank you to Keith Davidson from the Office of Naval Research for funding my work throughout my entire time at UW and for providing valuable and thoughtful feedback.

Thank you to my labmates, including Scott Wisdom, Brad Ekin, Eldridge Alcantara, Jordan Drew, Elliot Saba, Tyler Ganter, Ruobai Wang, Jessica Tran, Majid Mirbagheri, and David Delongewicz for helpful discussions, collaborations, and fun time spent in the lab. I would especially like to thank Scott Wisdom, with whom I collaborated and learned from extensively, for helping me jump into machine learning research. I am grateful to other mentors and colleagues at UW-APL including Jim Pitton, Greg Okopal, Jack McLaughlin, Lane Owsley, Warren Fox, and Bill Kooiman. I am also grateful for advice and help from other students at UW including Kai Wei and Rishabh Iyer. Thank you to Ivan Tashev, Mike Seltzer, Jasha Droppo, and Sidhant Gupta for opportunities to see research outside of

academia.

I would like to thank many of my professors and mentors from WashU for encouraging me to pursue a PhD and giving me early experience and opportunities to do research, including Ed Richter, Chris Gill, and Arye Nehorai.

Thank you to my family and friends for supporting me through this process. Thank you especially to my parents for fostering a love of learning during my childhood. Thank you to my sister Amanda, and my brother David for their love and support. Lastly, thank you to my wife Heather for your constant support, love, and patience through this process. I could not have done it without you.

DEDICATION

To my wonderful wife Heather.

Chapter 1

INTRODUCTION

As deep neural networks (DNNs) continue to improve state-of-the-art performance on tasks in many fields like speech and audio, computer vision, and natural language processing, many of the most popular architectures remain difficult to analyze; they are essentially "black boxes" built from layers of linear operations and non-linear activation functions. One of the most popular recurrent neural network architectures—the long short-term memory network (LSTM)—is a classic example of such an architecture [26]. So too are deep feedforward networks, for which architectural choices like non-linear activation functions, connectivity patterns, dimensionality, and number of layers are often chosen heuristically. Despite having non-convex loss surfaces with bad critical points [31] and trained with relatively simple algorithms like first-order gradient descent, DNNs have been able to outperform more interpretable models with more sophisticated optimization algorithms in a variety of tasks like speech recognition and image classification [19, 37]. New optimization tricks and architecture heuristics are constantly being developed to further improve performance and increase training speed, such as Adam, batch normalization, dropout, and skip connections [32, 28, 61, 24]. DNNs generally require large amounts of data, though. The choice between DNN and generative model need not be binary, however. Deep unfolding provides a means to train generative models discriminatively and even interpolate between statistical models and DNNs [25, 74]. Such models can leverage sparsity, submodular, or other regularization frameworks as discussed in Chapters 4 and 5. In this dissertation, I explore algorithms around the intersection between subset selection, sparsity, and DNNs.

Subset selection problems are important for many applications in machine learning like feature selection, data selection, dictionary learning, sparse recovery, and compression. Subset

selection is inherently combinatorial, so finding optimal subsets is NP-hard generally and heuristic algorithms are used. However, if the the objective function is submodular, then it is possible to find near-optimal subsets in polynomial time [50]. Moreover, if the objectives are not submodular but have submodular structure to them or are *almost* submodular, then it may still be possible to derive approximation guarantees as discussed in Chapters 3 and 5 [35, 8, 10, 54]. An instance of such a non-submodular problem is robust multi-objective optimization. In this case, the goal is to optimize for the worst-case performance across any number of potential objectives. This is important in applications like multi-target tracking or outbreak detection; if a set of targets are detected with a certain average probability or time to detection of an outbreak is averaged over a number of locations, the worst-case probabilities of detection over the set or detection times over all the locations could be arbitrarily bad. While too hard to even approximate, this worst-case optimization problem can be relaxed slightly and we can derive new theoretical guarantees for a certain class of objective functions [35, 54].

While seemingly separate, deep learning and subset selection overlap when sparse models are used. Sparsity can improve interpretability, as data is fitted using only a few variables, and just as with other forms of regularization, it can often make models more robust to mismatched test data by preventing overfitting. There are several ways to promote sparsity in models depending on the formulation of the optimization problem. One way is to add a sparsity-inducing penalty, also known as a regularizer. Sparsity promoting regularizers like the ℓ_1 norm are often added to the objective function, as is the case in the classic LASSO problem [65]. Another way is to make sparsity a hard constraint by expressly limiting the cardinality of weights with non-zero weighting. These types of hard constraints are easy to incorporate into discrete optimization algorithms that iteratively build a subset from scratch such as Orthogonal Matching Pursuit.

The main contribution of this dissertation is developing algorithms that can leverage underlying submodular or sparse structure to do robust optimization: unfolded discrete and continuous optimization algorithms and robust submodular optimization.

1.1 Outline

Chapter 2 introduces some topics discussed throughout the rest of the dissertation: submodularity, sparse regularization, DNNs, dictionary learning, and deep unfolding. Chapter 3 covers two discrete optimization problems—one submodular and one non-submodular—and a pair of algorithms with approximation guarantees. We apply these algorithms to scheduling of active sonar arrays and a sensor placement problem. Chapter 4 covers several novel DNN architectures that correspond to continuous sequential sparse coding models via deep unfolding. Chapter 5 similarly covers several novel network architectures unfolded from discrete inference algorithms.

1.2 Acknowledgements and relation to published works

Some content in this dissertation has been published in peer-reviewed papers. The titles and coauthors of those paper are listed as follows. Sections 3.1-3.3 include text published in the *International Conference on Information Fusion in 2015 and 2016* with Dr. David W. Krout and Prof. Les Atlas, the *NIPS Workshop on Optimization for Machine Learning in 2016* with Prof. Jeff Bilmes, Dr. Scott Wisdom, Dr. David W. Krout, and Prof. Les Atlas, *IET Radar, Sonar and Navigation* with Dr. David W. Krout, Prof. Jeff Bilmes, and Prof. Les Atlas, and Master's thesis. [57, 54, 55, 58, 53]. Section 4.1 includes text published at NIPS 2016 Workshop on Interpretable Machine Learning for Complex Systems in 2016 with Dr. Scott Wisdom, Dr. James Pitton, and Prof. Les Atlas as co-authors, the IEEE Conference on Acoustics, Speech, and Signal Processing in 2017 with Dr. Scott Wisdom, Dr. James Pitton, and Prof. Les Atlas as co-authors, and the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics in 2017 with Dr. Scott Wisdom, Dr. James Pitton, and Prof. Les Atlas as co-authors [74, 75, 76]. This work was done in close collaboration with Dr. Scott Wisdom and is primarily his work. Section 4.2 includes text from Advances in Neural Information Processing Systems in 2016 with Dr. Scott Wisdom, Dr. John Hershey, Dr. Jonathan Le Roux, and Prof. Les Atlas. All papers published in IEEE conferences are

Chapter 2
BACKGROUND AND RELATED WORK

This chapter covers relevant background material and work related to the later chapters of the dissertation. Section 2.1 defines submodularity, which is a property of set-valued functions, and provides an overview of submodular function maximization. Section 2.2 outlines the use of sparse regularization in statistical models. Section 2.4 outlines dictionary learning. Section 2.3 outlines neural networks and relevant deep learning concepts. Section 2.5 discusses the technique of deep unfolding. Section 2.6 summarizes related work and states the main contributions of this dissertation.

2.1 Submodularity

Submodularity is a property that describes set functions analogous to how convexity describes functions in a continuous space. Rather than exhaustively searching over all combinations of subsets, submodular functions provide a fast and tractable framework to compute a solution [44, 15, 34].

Let the set of available objects, known as the ground set, be denoted as V. Submodularity can be expressed via the notion of diminishing returns, i.e., the incremental gain of the objective diminishes as the context grows. If we define the incremental gain of adding element v to A as $f(v|A) = f(A \cup \{v\}) - f(A)$, then a submodular function is defined as satisfying

$$f(v|A) \geq f(v|B) \quad \forall A \subseteq B \subset V,\ v \notin B. \tag{2.1}$$

Submodularity is very closely tied to structures known as matroids, which generalize the notion of linear independence in vector spaces [12]. One can think of matroids as a generalization of matrices, which extend the definition of rank beyond column vectors of

a matrix to more general independent subsets over a finite ground set. More importantly, submodular function optimization allows for matroid independence constraints to be placed on the problem, which means complicated variable dependence patterns can be encoded into the problem. Though matroids are combinatorial objects that grow exponentially with ground set size, their submodular nature allows approximate solutions in polynomial time. Given a finite set V and a finite set of subsets $\mathcal{I} = \{I_1, I_2, \dots\}$, the pair $(V, \mathcal{I})$ is said to be a matroid when the family of sets $\mathcal{I}$ satisfies the following three properties:

1. $\emptyset \in \mathcal{I}$,

2. $I_1 \subseteq I_2 \in \mathcal{I}$,

3. $I_1, I_2 \in \mathcal{I}, |I_1| < |I_2| \implies \exists v \in I_2 \backslash I_1 : I_1 \cup v \in \mathcal{I}$.

Constrained monotone submodular function maximization algorithms include the forward greedy algorithm [50, 14] and continuous greedy algorithm [72]. Likewise, constrained non-monotone submodular function maximization algorithms include local search algorithm [42], greedy-based algorithms [22], and other algorithms that use some combination of these [48].

For robust submodular optimization, previous work had been limited to rational-valued monotone submodular functions subject to cardinality constraints [35]. We derive a more general framework that could handle arbitrary monotone submodular functions and all submodular function optimization compatible constraints [54, 55].

Recent work has also delved into differentiable submodular functions [11, 68, 56].

2.2 Sparse regularization in statistical modeling

The goal of sparse recovery is to find $\hat{\mathbf{h}} \in \mathbb{R}^N$ minimizes the reconstruction error $\mathbf{A}\hat{\mathbf{h}}$ relative to $\mathbf{x}$ in terms of a loss such as squared error and a sparsity promoting term such as an ℓ_0 or ℓ_1 norm.

The ℓ_0 norm of a vector $\mathbf{h} \in \mathbb{R}^D$ is given by $||h||_0 = \sum_{d=1}^{D} \mathbf{I}[h_d \neq 0]$ where $\mathbf{I}$ is an indicator function that is one if the condition in the brackets is true and zero otherwise. This formulation is typically used to explicitly set the number of non-zero elements in $\mathbf{h}$ to a value K.

$$\min_{\mathbf{h}} \|\mathbf{x} - \mathbf{A}\mathbf{h}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{h}\|_0 \leq K. \tag{2.2}$$

While this problem is non-convex due to the ℓ_0 norm, Iterative Hard-Thresholding (IHT) can be used to converge to a local minimum under certain conditions. The hard-thresholding function hard_K keeps the largest K values in terms of magnitude and sets the rest to zero.

$$\min_{\mathbf{h}} \|\mathbf{x} - \mathbf{A}\mathbf{h}\|_2^2 + \lambda\|\mathbf{h}\|_1. \tag{2.3}$$

One way to solve problem (2.3) is the iterative soft-thresholding algorithm, shown in algorithm 3, where α is the inverse step size. ISTA is a proximal gradient method consisting of repeated application of soft-thresholding. Soft-thresholding is defined as:

$$\text{soft}_b(s_n) = \frac{s_n}{|s_n|}\max(|s_n| - b, 0). \tag{2.4}$$

2.3 Deep neural networks

Neural networks are comprised of layers of differentiable linear and non-linear functions that contain both trainable and non-trainable parameters and have defined forward and backward computations. At the most basic level, on the forward pass, the network transforms input features into a predicted output. On the backward pass, a loss (for which a formulation can be found in Equation (2.5)) is computed from the difference between the predicted output and the ground-truth and used to update the trainable parameters to decrease the loss. The composition and arrangement of the the layers are referred to as the *architecture* of the network.

$$\min_{\theta} \mathcal{L}(\hat{y}, y) + \lambda\Omega(\theta) \quad \text{s.t.} \quad \hat{y} = f_\theta(x) \tag{2.5}$$

One common architecture is the feedforward network. The forward computation of a feedforward network can be found in equation (2.6). The network has K hidden layers, an input and an output layer, each of which have weight matrices $W^{(k)}$, a bias vector $\mathbf{b}^{(k)}$. Except for the output layer, an activation function $\sigma(\cdot)$ is then applied to the output of the affine operation.

$$\mathbf{h}^{(k)} = \sigma(\mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}) \tag{2.6}$$

Another is a recurrent neural network (RNN), which is defined by equation (2.7). It operates on sequential data $\mathbf{x}^{(1:T)}$.

$$\mathbf{h}^{(t)} = \sigma(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}) \tag{2.7}$$

The long short-term memory (LSTM) network developed by Hochreiter and Schmidhuber [26] is given by the following equations:

$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot \tanh(\mathbf{C}^{(t-1)}) \tag{2.8}$$

$$\mathbf{C}^{(t)} = \mathbf{i}^{(t)} \odot \tilde{\mathbf{C}}^{(t)} + \mathbf{f}^{(t)} \odot \mathbf{C}^{(t-1)} \tag{2.9}$$

$$\tilde{\mathbf{C}}^{(t)} = \tanh(\mathbf{W}_c\mathbf{h}^{(t-1)} + \mathbf{U}_c\mathbf{x}^{(t)} + \mathbf{b}_c) \tag{2.10}$$

$$\mathbf{i}^{(t)} = \tanh(\mathbf{W}_i\mathbf{h}^{(t-1)} + \mathbf{U}_i\mathbf{x}^{(t)} + \mathbf{b}_i) \tag{2.11}$$

$$\mathbf{f}^{(t)} = \tanh(\mathbf{W}_f\mathbf{h}^{(t-1)} + \mathbf{U}_f\mathbf{x}^{(t)} + \mathbf{b}_f) \tag{2.12}$$

$$\mathbf{o}^{(t)} = \tanh(\mathbf{W}_o\mathbf{h}^{(t-1)} + \mathbf{U}_o\mathbf{x}^{(t)} + \mathbf{b}_o) \tag{2.13}$$

DNNs make use of many nonlinear activation functions in order to increase the model complexity or meet constraints on the domains of the input and outputs to the network. One common activation function is the softmax, which is a monotone vector valued function that outputs a probability distribution. For each index i in the input, the function is computed as follows:

$$y_i = \frac{\exp(h_i/\tau)}{\sum_j \exp(h_j/\tau)}. \tag{2.14}$$

Another is the rectified linear unit (ReLU) defined as follows:

$$f(x) = max(x, 0). \tag{2.15}$$

The ReLU is preferable when sparse outputs are desired, and has a correspondence to a particular inference algorithm for sparse coding models, which is explored in Section 4.1.

In order to evaluate and improve the network, a loss function must be defined to compare predicted outputs $\hat{\mathbf{y}}$ to the true outputs $\mathbf{y}$. For regression tasks, a common loss function is mean squared error (MSE) given in the following two equation as f_{MSE}.

$$f_{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2 \tag{2.16}$$

Likewise for classification, a common function is categorical cross-entropy given as f_{CE}. In this case, the predictions $\hat{\mathbf{y}}$ and targets $\mathbf{y}$ are one-hot vectors where the non-zero index c indicates which of the C classes has been selected.

$$f_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=0}^{C-1} y_c \log \hat{y}_c \tag{2.17}$$

Using gradient descent algorithms and backpropagation to train networks requires the computation of gradients (or subgradients) of the functions in that define the forward computation of the network with respect to any trainable parameters int he network. Automatic differentiation packages within popular deep learning toolkits has abstracted much of this computation away behind the scenes. In TensorFlow, for example, the code that computes the gradients for basic functions is automatically generated upon installation of the toolkit [1]. This is an especially important topic in Chapter 5 where we discuss training through nondifferentiable functions.

2.4 Dictionary learning

For the dictionary selection problem, the goal is to maximize the reconstruction accuracy of a given signal Z using linear regression on a select few elements of a fixed dictionary

$V = \{x_i\}_{i=1}^n$. The set V is denoted $\mathbf{V}$ in matrix form, and the regression coefficients $\{h_i\}_{i=1}^n 2$ are denoted in vector form as $\mathbf{h}$. Krause and Cevher [8] present a property called *approximate submodularity*, which differs from equation (2.1) by an additive factor ϵ is given by the following equation:

$$f(v|A) \geq f(v|B) - \epsilon \quad \forall A \subseteq B \subset V,\ v \notin B. \tag{2.18}$$

When $\epsilon = 0$, f is submodular. The approximate submodularity of their proposed objective depends upon certain dictionary incoherence assumptions, under which they derive a pair of algorithms, SDS_{OMP} and SDS_{MA}, that yield approximate solutions.

Similarly, Das and Kempe [10] define their own notion of almost submodularity, the *submodularity ratio*. For a non-negative set function f and with respect to a set U and positive integer k, the submodularity ratio $\gamma_{U,k}$ is given by

$$\gamma_{U,k}(f) = \min_{A \subseteq U, B: |B| \leq k, A \cap B = \emptyset} \frac{\sum_{b \in B} f(b|A)}{f(B|A)}. \tag{2.19}$$

Intuitively, the submodularity ratio measures the relative incremental value of adding each of the elements of B in the context of A for the function f compared to the total incremental value of adding B. The function f is submodular if and only if $\gamma_{U,k} \geq 1\ \forall U, k$. They focus their analysis on a squared multiple correlation objective, $R^2_{Z,S}$ shown in equation (2.20), that measures the fraction of observed variance explained by a chosen subset of dictionary elements $S \subseteq V : |S| \leq k$. They too derive a pair of dictionary selection algorithms, Forward Regression (FR) and Orthogonal Matching Pursuit (OMP), whose approximation guarantees depend on a lower-bound of the submodularity ratio which is the smallest k-sparse eigenvalue $\lambda_{min}(C, k)$ of the covariance matrix C of the observation variables V. The squared multiple correlation objective is given by the following equation:

$$R^2_{Z,S} = \frac{\text{Var}(Z) - \text{E}[(Z - Z')^2]}{\text{Var}(Z)}, \tag{2.20}$$

where $Z' = \sum_{x_i \in S} h_i x_i$.

Matching Pursuit (MP) and OMP are two greedy algorithms that successively select dictionary elements (columns of $\mathbf{A}$ in Problem (2.2)) to approximate a signal [45, 51]. The main difference is in how the two algorithms compute the mixing weights $\mathbf{h}$. MP uses the value of the inner product between the chosen dictionary element i and the signal residual as the value of h_i, whereas OMP recomputes the weights as $\mathbf{h} = \mathbf{A}^{+}\mathbf{x}$ so that each successive projection is mutually orthogonal. MP does have asymptotic convergence guarantees, though the residual might be large after any finite number of iterations. OMP is guaranteed to converge in N iterations, where N is the size of the dictionary.

2.5 Deep unfolding

The choice between deep learning and generative models need not be binary. Recently, Hershey et al. have developed a technique called deep unfolding [25] that fills the continuum between the two extremes and combines the advantages of deep learning (discriminative training through backpropagation, ability to leverage large datasets) and generative models (ability to leverage domain knowledge, optimization bounds, fast training on smaller datasets).

For many generative models, exact inference is computationally intractable, so approximate inference algorithms are used instead that simplify the model assumptions. For instance, inference on probabilistic graphical models like Markov random fields often relies on belief propagation, which imposes extra conditional independence assumptions on the model. The trade-off is between computational complexity and degree of approximation. On the more simplified end are methods like mean field inference that are easy to compute but have limited expressiveness, and at the other end are methods like normalizing flows.

When unfolding an inference algorithm, it is important to distinguish between the optimization problems being solved. First, by running an inference algorithm such as ISTA to solve Problem (2.3), it is optimizing over $\mathbf{h}$ given inputs $\mathbf{x}$ and model parameters θ like the dictionary A. After unfolding the inference algorithm, the predicted state $\mathbf{h}$ is used to compute an error, which is used to learn better model parameters θ as shown in Problem (2.5).

Most of these inference algorithms are iterative, e.g. matrix factorization models use alternating minimization style algorithms to infer parameters and binary Markov random fields use mean field inference. For some problems, inference involves solving a convex optimization problem. While exact, the optimization algorithms can be slow to converge. In sparse coding, one can build a model of the signal as a sparse combination of the elements of some dictionary. Suppose the task is to denoise the signal, and thus this problem corresponds to a regression problem, which involves solving LASSO, which is described in equation (2.3). Since this problem is convex, one could use simple gradient descent to solve it. Unfortunately, gradient descent converges slowly ($1/\sqrt{K}$ where K is the number of gradient steps) because it is non-smooth. Another way to solve the LASSO problem is with the iterative soft-thresholding algorithm (ISTA). ISTA is a proximal gradient method that solves a reformulated, but equivalent, optimization problem (and has a $1/K$ convergence rate). Even with a better convergence rate, ISTA can still take hundreds of iterations to converge. In fact, for sequential ISTA (SISTA), the unfolded networks (SISTA-RNN and DR-NMF) only need to run for a small constant number of iterations to find a better solution than running SISTA without the training as shown in Section 4.1. In this instance, deep unfolding improves the performance of a computationally inexpensive optimization algorithm without increasing the complexity as opposed to more sophisticated optimization techniques. Specifically, many of these algorithms, such as Conjugate Gradient methods, Newton's method, and L-BFGS [7], need to compute expensive second order statistics. Moreover, these more expensive inference algorithms could be unfolded into computational graphs and trained on a dataset.

Unfolding translates inference algorithms into deep networks whose weights still correspond to parameters of the original statistical model. Unfolding mean field updates for a binary Markov random field yields a feedforward sigmoid network [25]. For the sparse coding problem, each iteration of ISTA can be seen as a layer of a feedforward neural network with a rectified linear unit (ReLU) activation function [20]. For sequential sparse coding models, we also derive several recurrent neural network (RNN) architectures: SISTA-RNN, DR-NMF, and uRNN. [74, 76, 59].

Other techniques can be used to analyze the decisions and internal representations of a DNN, such as attention mechanisms or sensitivity analysis [49]. Attention mechanisms attempt to find salient portions of the input that account for the decision being made. Similarly, sensitivity analysis is a classic way of determining what portions of the input account for uncertainty in the output. On the other end of the spectrum, methods like Variational Autoencoders (VAE) propose an alternative way of introducing statistical interpretability by including stochastic nodes that sample from specific distributions [33]. The network then learns a mapping between these sampled values and the distribution of the input data by maximizing the variational lower bound. Compared against DR-NMF, a conventional network with attention may grant similar levels of human interpretability, performing a sensitivity analysis on a conventional DNN architecture might show what inputs the network is particularly sensitive to, and while VAEs provide a general purpose way to combine DNNs and generative models, but all of them still rely on heuristics to improve the architecture of the models.

2.6 Summary and Contributions

By carefully using constraints, regularization tools, and optimization frameworks that are well-matched to a given problem, we can not only derive new and interesting algorithms, we can also characterize their performance by deriving performance guarantees.

The main contribution of this dissertation is developing algorithms that can leverage underlying submodular or sparse structure to do robust optimization: unfolded discrete and continuous optimization algorithms and robust submodular optimization.

Chapter 3

ROBUST MAXIMIZATION OF SUBMODULAR FUNCTIONS

Over the past decade [35, 54], the following constrained robust maximization problem has been considered

$$A^* \in \operatorname*{argmax}_{A \in \mathcal{C}} \min_i f_i(A), \tag{3.1}$$

where $\mathcal{C}$ denotes a family a feasible sets that are compatible with submodular optimization, e.g. matroid, a knapsack, cover, or other submodular constraint, and the functions f_i are monotone submodular functions. Unfortunately, this problem is NP-hard, non-submodular, and inapproximable.

We apply GenSat to scheduling deepwater active sonar arrays and facility location problems. For the sonar application, the goal is to transmit a signal and generate reflections off a target that can be detected by the receiver. In our specific framework, each buoy has a co-located transmitter and receiver that operates monostatically. However, since SFO allows for multiple buoys to be selected, the array functions multistatically in that multiple receivers are operating simultaneously with potentially overlapping coverage regions.

3.1 Ping sequencing for active sonar arrays

In any depth of water, the sound from a buoy at the surface will propagate along the surface for a few kilometers until it attenuates to the point where the sound energy is too low for any reflections off a target to be detected by a receiver. The sound will propagate much further downward due to temperature and pressure gradients, and if the water is deep enough, the sound will arc back up to the surface and back down, forming concentric rings around the transmitter called convergence zones. In shallow water, the sound will simply scatter off the bottom. In the first convergence zone, which is delineated by the blue lines in Figure

3.1, there is enough sound energy that a reflection off a target is potentially detectable by a receiver, so it is called the buoy's coverage region. In the second convergence zone, delineated by the red lines in Figure 3.1, there is still significant signal energy, just not enough to reflect all the way back to the buoy. However, if there is a second buoy in this region, the signal energy will only serve to interfere with the second buoy as it listens for reflections from its own ping, so we call this the buoy's interference region.

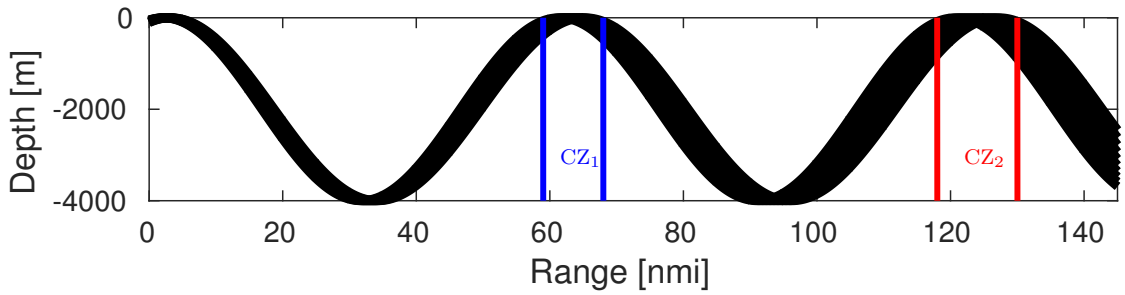
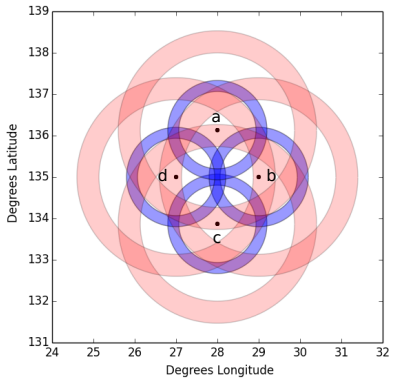


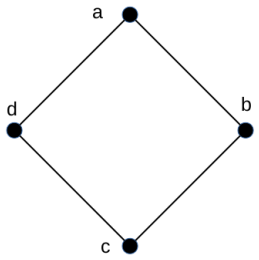
Figure 3.1: Depiction of acoustic propagation path in deep water. The marked boundaries are the two convergence zones: the coverage region (CZ_1) and the interference region (CZ_2).

To elaborate more on interfering buoys, consider the following scenario. There are two pings that are transmitted simultaneously, one from buoy a, and one from buoy b, and the buoys are spaced such that they are in each others interference regions. The feasible time window for buoy b's ping to propagate out to the coverage regions, bounce off a target, and arrive back is coincidental with the time window that buoy a's ping will arrive at buoy b. Hence, it is very difficult to detect the reflection in the presence of the interfering ping, and we say that the two buoys interfere.

An example of a spatial buoy arrangement where some of the buoys interfere can be found in Figure 3.2a. The four buoys are arranged in a diamond pattern with locations represented by black dots. In Figure 3.2a, the blue rings denote the coverage regions for each buoy and the red rings denote the regions where another buoy will interfere with a given buoy. Coverage is defined by the probability of target detection for a buoy. If two interfering buoys transmit



(a) Four buoys (black dots) with coverage regions (blue rings) and interference regions (red rings).



(b) Independence graph for four buoys as nodes with edges that signify pairwise independence.

Figure 3.2: Relationship between the independence graph, coverage regions and interference regions for a four buoy arrangement.

simultaneously, the direct path signal from the first will arrive at the second when the second buoy's reflections would arrive. Figure 3.2a also shows the relationship between the coverage and interference regions for the buoys. In this arrangement, the buoys across from each other, i.e. the top and bottom pair and left and right pair, will interfere with each other, since the buoys in each pair are in the red interference region of the other buoy. However, any other pair of buoys can ping simultaneously [38].

In order to find out the maximum number of buoys that can ping simultaneously, the largest set of nodes is picked such that all the nodes in the set are connected to every node in the set. Note that self-loops are implied, since a buoy does not interfere with itself. The problem of finding the largest subset of fully connected nodes is a well known problem in computer science [5]. Exact methods for solving this problem run in exponential time, but for reasonable graph sizes (tens of vertices), the algorithm runs fairly quickly. Moreover, if the graph meets certain conditions, i.e. if the graph is "planar" or "perfect," finding the largest

clique can be solved in polynomial time [17]. For the arrangement in Figure 3.2a, there is a four-way tie for largest clique, which are the adjacent pairs (top and left buoys, left and bottom buoys, bottom and right buoys, and right and top buoys). The independence graph for this arrangement is depicted in Figure 3.2b. In a real scenario, the coverage regions will not be perfect rings, so one of the pairs might have better coverage than the others. A more complicated interference pattern will emerge as the number of buoys is increased, which is demonstrated in Figure 3.5c.

Our objective function is a variant of probabilistic coverage. It utilizes target state estimates to help determine which buoys are selected. Let V be the set of N buoys $b_i, i = 1...N$. Let $B \subseteq V$ such that B is a clique of G, where G is the independence graph determined by the interference pattern of all the buoys b_i in V. Let the set of all sets of sensors that form cliques on the graph be a partition matroid $\mathcal{I}$. Coverage is a positive, non-decreasing objective, so the goal is to maximize the objective function. Two different problem formulations can be used, which correspond to maximizing the average coverage over a set of target locations [57], Equation (3.2), and maximizing the worst-case coverage over a set of target locations, Equation (3.1). Then the optimal set of buoys for the average-case scenario is given by

$$A^* \in \underset{A \in \mathcal{I}}{\operatorname{argmax}} \frac{1}{M} \sum_{i=1}^{M} f_i(A). \tag{3.2}$$

The worst-case scenario is given by equation (3.1). In both cases, $i = 1, ..., M$ corresponds to the predicted target locations and M is the number of targets, and the functions $f_i : 2^V \to \mathbb{R}$ are given by the equation

$$f_i(A) = 1 - \prod_{b_i \in A} (1 - P_{i,b_i}) \tag{3.3}$$

where P_{i,b_i} is the probability of detection of buoy b_i at location i determined by a table look-up for pre-computed probability of detection maps for each buoy.

Our approaches to both average and worst-case scenarios emphasize tracking in that the objectives prioritize covering areas where known targets are located, but also provide good coverage as well. After the algorithms address coverage of the known targets, they

add as many non-interfering buoys as are available, and thus provide effective simultaneous track and search frameworks. The effective difference between the two is how they weigh the targets in aggregate. Equation (3.2) maximizes the average probability of detection over the targets, and therefore weighs each target equally, regardless of how hard to detect it is. The mean of the probability of detection also lacks any information about the maximum and minimum probability of detection of the set of targets. Equation (3.1), on the other hand, only considers the target with the lowest probability of detection. In sonar and radar, missed detections can be very costly, so it is of interest to maximize the minimum probability of detection of a set of targets.

The average-case scenario objective function given in Problem (3.2) is submodular. Since the values of P_{i,b_i} in Equation (3.3) are probabilities, $f_i(B)$ is a concave over modular function, the average of the functions is also submodular. We use the greedy algorithm from the SFO toolbox to solve the above optimization problem [34], which we refer to as SFO-GREEDY for the experiments in the following section. This submodular objective function is monotonic non-decreasing and subject to a matroid constraint. Submodular maximization for functions of this form have been studied and have certain performance guarantees. For a monotonic non-decreasing objective function subject to a matroid constraint, the solution has a worst case performance bound of $\frac{1}{2}$ using the greedy algorithm (Algorithm 1), and the bound scales with the number of matroids, i.e. for k matroid constraints the worst case performance bound is $\frac{1}{k+1}$ [14]. The greedy algorithm has a computational complexity of $O(nk)$, where n is the ground set size and k is the size of the solution, but can be significantly sped up [47].

3.2 Robust maximization

In this section we consider the constrained robust maximization problem in equation (3.1). To optimize Equation (3.1), we develop a novel algorithm GENSAT, which generalizes the SATURATE algorithm created by Krause et al. [35]. Krause et al. use SATURATE to optimize

Algorithm 1 GREEDY (f, c)

$A \leftarrow \emptyset$
while $\exists a \in V \backslash A$ s.t. $A \cup \{a\} \in \mathcal{C}$
and $f(a|A) > 0$ **do**
    $S \leftarrow \{a \in V \backslash A : \ A \cup \{a\} \in \mathcal{C}\}$
    $s^* \leftarrow \text{argmax}_{s \in S} f^c(a|A)$
    $A \leftarrow A \cup \{s^*\}$
end while
return A

an objective function of the form

$$A^* \in \underset{|A| \leq k}{\text{argmax}} \min_i f_i(A) \tag{3.4}$$

where $f_i(A)$ is a set of monotone submodular functions. SATURATE solves this worst-case optimization problem by proposing an alternative formulation and relaxing the cardinality constraint from $|A| \leq k$ to $|A| \leq \alpha k$. As long as α is large enough, the solution $\hat{A}$ from the SATURATE algorithm guarantees that

$$\min_i f_i(\hat{A}) \geq \max_{|A| \leq k} \min_i f_i(A) \quad \text{and} \quad |\hat{A}| \leq \alpha k.$$

Krause et al. claim that the only way to achieve a non-trivial guarantee is to relax the constraint, which limits both the types of constraints that can be applied to the problem as well as the values the objective functions can take, i.e. integral or rational valued objective functions. Matroid constraints, for instance, have no immediately obvious relaxation, nor do multiple constraints, e.g. a mix of matroid, knapsack, and cover constraints. One way to relax the matroid constraint, however, might be to expand the bases of the matroid, which may lead to undesirable solutions. However, there is another way to achieve non-trivial guarantees, which is to relax the objective itself, leaving the constraints intact, and produce a fractional bound on the objective function, something that is made possible thanks to the use

of submodularity. Whereas relaxing the constraint requires a new bicriterion guarantee be derived (and a new algorithm, potentially), relaxing the problem is more generalizable, in the sense that the proposed algorithm works with any constraint compatible with submodular function optimization including multiple constraints. The proposed algorithm, GenSat, uses this alternative approach to find a solution such that a fraction γ of the submodular functions are above a minimum value β. Moreover, the user can set particular values of β or γ, as long as β is less than the submodular guarantee α and $\gamma < 1$. The derivation for the lower bound is given below.

For a fixed value of c, which can be thought of as the saturation level, we can determine if $f_i(A) \geq c$ from equation (3.1) via submodular maximization of the following surrogate function:

$$f^c(A) = \frac{1}{M} \sum_{i=1}^{M} \min\{f_i(A), c\}. \tag{3.5}$$

$f^c(A)$ is a submodular function, because it is a non-negatively weighted sum of functions $\min\{f_i(A), c\}$ which are submodular [16]. At each iteration of GenSat, we run a submodular maximization algorithm like the one described in Algorithm 1, which selects the element which provides the best incremental gain in the objective and does not violate the constraint.

GenSat is outlined in Algorithm 2. Given monotone submodular functions $(f_1, ..., f_M)$, approximation guarantee α for the matroid constrained submodular maximization problem, and tolerance threshold ϵ, we first set c_{min} and c_{max} to values that ensure the true optimal value lies in the interval. While performing a binary search over c, we test the value of the approximate solution $f^c(\hat{A})$ against the lower bound αc. If the approximate solution is less than the lower bound, we know that the true optimal is less than c, so we limit the search to the lower half of the interval. Likewise, if the lower bound is met, we store the solution (which, as we describe below, is fractionally good w.r.t. the current c) and then continue attempting to find a better one (higher c) by searching over the upper half of the interval. We stop when the range falls within the tolerance.

Theorem 1. *Given a value $\beta < \alpha$,* GenSat *finds a solution $\hat{A}$ that guarantees the following*

Algorithm 2 GENSAT $(f_1, ..., f_M, \alpha, \epsilon)$

1: $c_{min} \leftarrow 0, c_{max} \leftarrow \min_i f_i(V)$
2: **while** $(c_{max} - c_{min}) > \epsilon$ **do**
3: $\quad c \leftarrow (c_{max} - c_{min})/2$
4: $\quad f^c(A) \leftarrow \frac{1}{M} \sum_{i=1}^{M} \min\{f_i(A), c\}$
5: $\quad \hat{A} \leftarrow$ GREEDY (f^c, c)
6: $\quad$ **if** $f^c(\hat{A}) < \alpha c$ **then**
7: $\quad\quad c_{max} \leftarrow c$
8: $\quad$ **else**
9: $\quad\quad c_{min} \leftarrow c, \quad A_{best} \leftarrow \hat{A}$
10: $\quad$ **end if**
11: **end while**
12: **return** A_{best}

fraction γ of the M functions $min\{f_i\left(\hat{A}\right), c\} \geq \beta c$:

$$\gamma \geq \frac{\alpha - \beta}{1 - \beta}$$

where α is the approximation guarantee for constrained submodular maximization.

Proof: $f^c(A) \geq c$ only if $\min_i f_i(A) \geq c$. When all $f_i(A) \geq c$, then $f^c(A) = c$. Likewise, when any $f_i(A) < c$ then the $f^c(A) < c$, since the maximum value of $f^c(A)$ is c. The greedy solution $\hat{A}$ for maximizing a monotone submodular function subject to a constraint is $f^c(\hat{A}) \geq \alpha f^c(A^*)$, where α depends on the algorithm chosen and the constraint. If line 6 of Algorithm 2 is true, then $f^c(\hat{A}) < \alpha c$ which implies that $\min_i \ f_i(A^*) < c$. Line 6 being true also implies that c is too large, so $c_{max} \leftarrow c$. At line 12 of Algorithm 2, $(c_{max} - c_{min}) \leq \epsilon$ and the true optimal value $\min_i \ f_i(A^*)$ is in the interval $[c_{min}, c_{max}]$. The submodular approximation guarantee ensures that $f^c(\hat{A}) \geq \alpha f^c(A^*)$ and $f^c(\hat{A}) \geq \alpha c$. Given a value for β, the terms of $f^c(\hat{A})$ can be split into two groups, one that have value less than βc and the other greater than or equal

to βc:

$$f^c(\hat{A}) = \frac{1}{M} \sum_{i:\min\{f_i(A),c\}<\beta c} \min\{f_i(A), c\} + \frac{1}{M} \sum_{i:\min\{f_i(A),c\}\geq\beta c} \min\{f_i(A), c\}.$$

Let γ be the fraction of terms that meet the βc threshold. Then, the two summation terms become $f^c(\hat{A}) = (1-\gamma)\beta c + \gamma c \geq \alpha c$. Rearranged, the expression becomes $\gamma \geq \frac{\alpha-\beta}{1-\beta}$. □

A visualization of the lower bound βc over values of the saturation level c and fractional number of functions $\lfloor \gamma M \rfloor$ for $M = 8$ is shown in Figure 3.3. In this case, $\alpha = 1/2$, which is the submodular guarantee for a monotone submodular function constrained by a single matroid via the greedy algorithm [14]. Notice that the bound becomes trivial, ($\beta c = 0$), when $\gamma \geq \alpha$, i.e. where $\lfloor \gamma M \rfloor \geq 5$.

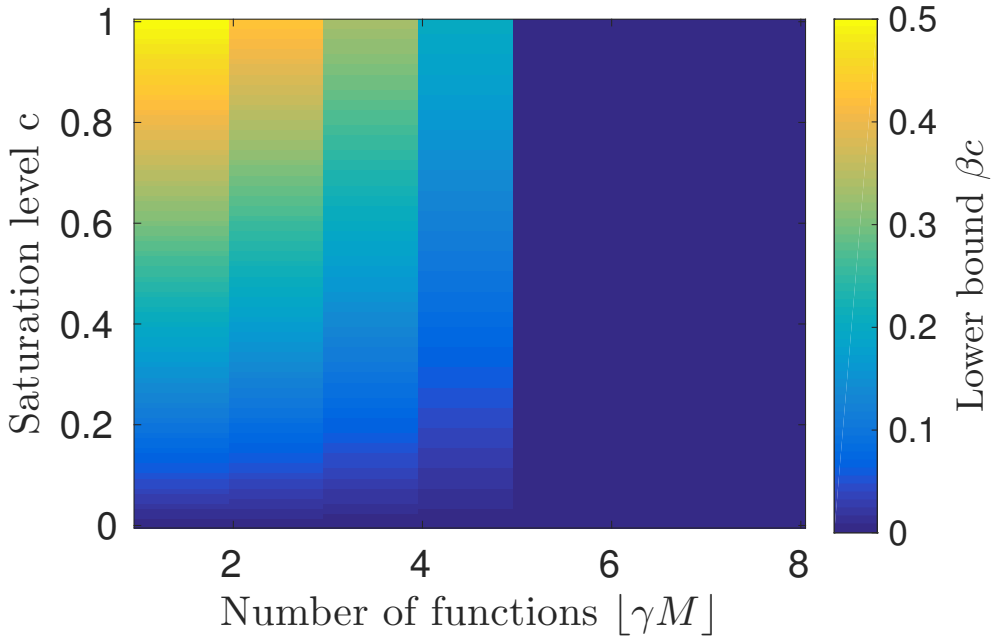


Figure 3.3: Visualization of lower bound βc as the saturation level c and the fractional guarantee $\lfloor \gamma M \rfloor$ vary for $M = 8$ and $\alpha = 1/2$. Note that the bound worsens as we demand that a larger fraction γ of the functions have non-trivial minimum values and if the algorithm returns low values of the saturation level c. Also note that the bound becomes trivial, ($\beta c = 0$), when $\gamma \geq \alpha$, i.e. where $\lfloor \gamma M \rfloor \geq 5$.

3.3 Experiments

In this section, we design four experiments. The first three deal with ping sequence optimization, and the last with facility location. In the first two, we compare the performance of the proposed ping sequence optimization algorithms, SFO-Greedy and GenSat, to the standard round-robin approach, optimized single buoy selection, and exhaustive search in three Monte Carlo simulations. For the first experiment, we have nine buoys in a grid pattern, with each buoy 60 km away from its neighbors. The independence graph for the buoys can be found in Fig. 3.5c, and the interference pattern in Fig. 3.5b. For the second experiment, we have thirty-two buoys in a ring. For all of the experiments, we assume the buoys have a probability of detection of $P = 0.8$ in the coverage region and $P = 0$ everywhere else. We assume here that there is no sensor drift during the experiment. Two targets with random initial location, constrained to be within the buoy array's detection area, and constant velocities are present for each trial. The first experiment consists of ten thousand trials with each trial lasting thirty time-steps or until a target moved out of the array's detection area, and the second experiment consists of one thousand trials with each trial lasting sixty-four time-steps or until a target moved out of the array's detection area.

The results in Table 3.1, Fig. 3.4a, and Fig. 3.4b demonstrate the practical utility of the SFO-Greedy method. In the nine buoy experiment, a round-robin approach to sensor selection detected the targets about 7% of the time on average, whereas the proposed SFO-Greedy approach detected the targets over 30% of the time. In addition, the greedy algorithm performed nearly as well as an exhaustive search over all sets of non-interfering buoys. The results also demonstrate the advantage of using multiple buoys. By allowing simultaneous pinging, we gained 50% better coverage over optimized single buoy selection. The low overall probability of detection in Fig. 3.4a can be attributed to the fact that nine buoys have gaps in their overall coverage, so many simulated targets were impossible to detect given the arrangement. However, in no trial did the round-robin approach beat the proposed algorithm. The worst case bound for the greedy algorithm provides a nice floor

for worst-case behavior, but in practice the algorithm is as good as exhaustively searching over exponential growing sets of non-interfering buoys. Likewise, in the thirty-two buoy experiment, SFO-Greedy significantly outperformed the conventional round-robin method and almost matched exhaustive search for average probability of detection.

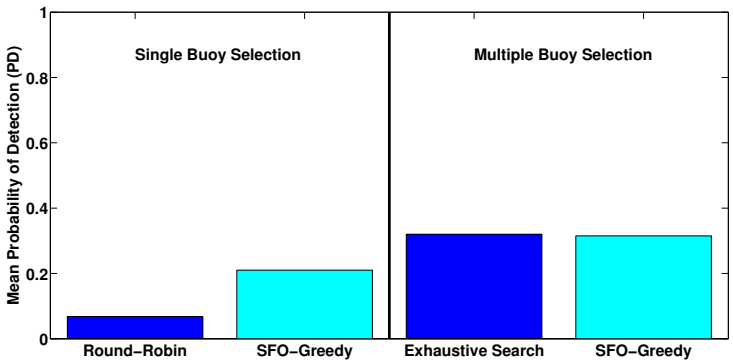
Table 3.1

	Single Buoy		Multiple Buoys	
Method	Round-robin	**SFO-Greedy (Proposed)**	Exhaustive search	**SFO-Greedy (Proposed)**
Mean PD, 9 Buoy Exp	0.068	**0.210**	0.320	**0.315**
Mean PD, 32 Buoy Exp	0.060	**0.505**	0.781	**0.779**

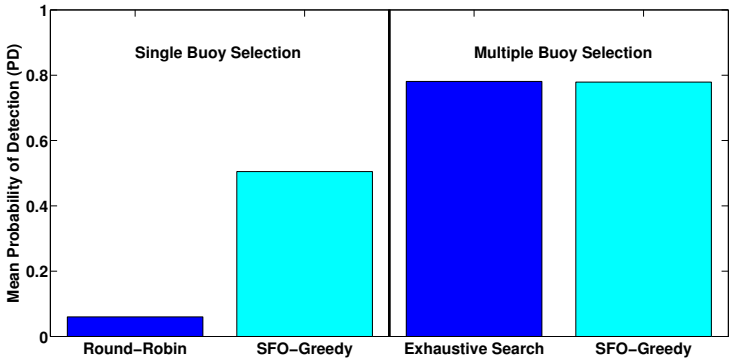
Probability of detection (PD) results for Monte Carlo simulation.

In the second set of experiments, we compare the performance of the proposed ping sequence optimization algorithm Gensat to the previously proposed SFO-Greedy and exhaustive search. Gensat and exhaustive search are optimized with respect to Equation (3.1), while SFO-Greedy is optimized with respect to Equation (3.2). If SFO-Greedy were to be optimized with respect to Equation (3.1), the algorithm would always select the empty set, except in the degenerate case where there is a buoy that can detect every single target.

For the third experiment, there are nine buoys in a grid pattern spaced 128 km away from their neighbors and seven targets. The nine coverage regions and seven target locations are represented by blue rings the red triangles, respectively, in Figure 3.5a. In this experiment, we



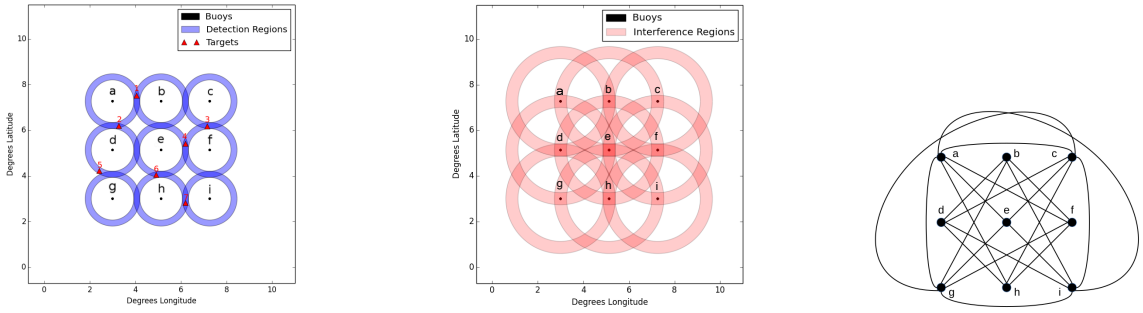
(a) Probability of detection (PD) results for nine buoy Monte Carlo simulation.



(b) Probability of detection (PD) results for thirty-two buoy Monte Carlo simulation.

Figure 3.4: Probability of detection (PD) results for nine and thirty-two buoy Monte Carlo simulations comparing the proposed SFO-Greedy method to a round-robin strategy for single buoy selection, and SFO-Greedy to exhaustive search for multiple buoy selection.

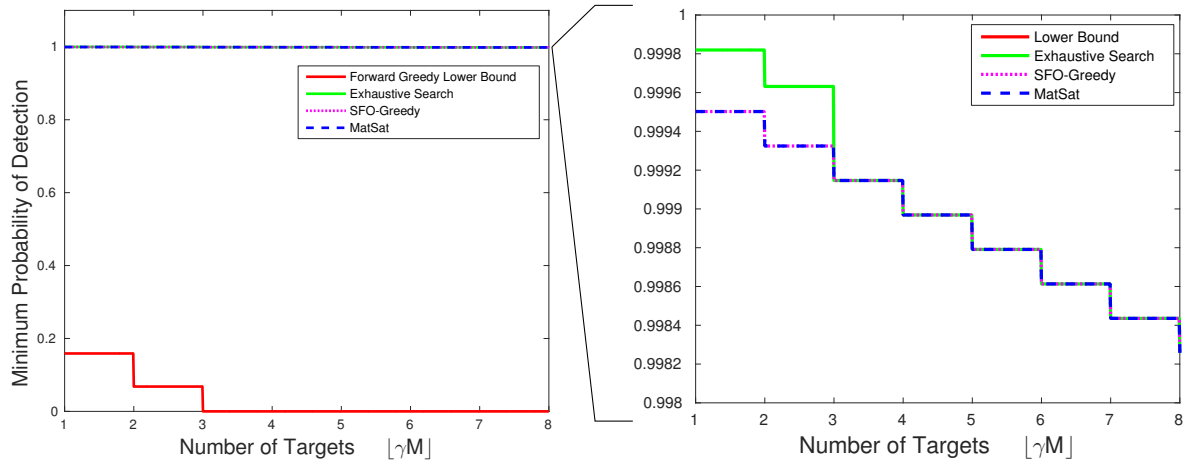
allow the targets to have unequal probabilities of detection. Ordered from top to bottom and left to right, the targets have probability of detection of $P = \{0.6, 0.6, 0.9, 0.2, 0.6, 0.9, 0.9\}$ in each buoy's coverage region and $P = 0$ everywhere else. The independence graph for the buoys can be found in Figure 3.5c, and the interference pattern in Figure 3.5b. By applying Kashiwabara's method for creating a set of matroid constraints to the interference graph in Figure 3.5c, the clique complex can be represented as the intersection of four partition matroids, whose partitions are given as $\{ad, be, cf, hi, g\}$, $\{ab, de, fi, gh, c\}$, $\{bc, ef, dg, a, h, i\}$, and $\{eh, a, b, c, d, e, f, i\}$.



(a) (b) (c)

Figure 3.5: Interference regions and independence graph for a nine buoy experiment. (a) Coverage pattern for the nine buoys in a three by three grid and target locations. (b) Interference pattern for the nine buoys in a three by three grid. (c) Independence graph for the nine buoys in a three by three grid.

The results for experiment three can be found in Table 3.2. GENSAT performs as well as the exhaustive search, which is to say it achieved the optimal solution both in terms of worst case and average coverage objectives. SFO-GREEDY, however, chooses the buoys $\{a, f, h\}$ instead of $\{b, d, f, h\}$, which misses one of the targets and results in suboptimal worst case and average coverage. In the fourth, we test GENSAT on a synthetic sensor placement problem in which the goal is to select a set $\hat{A}$ of sensor locations to maximize



(a) (b)

Figure 3.6: Minimum probability of detection results for an eight target Monte Carlo simulation comparing the proposed GenSat (dashed blue line) and SFO-Greedy (dotted magenta line) methods to exhaustive search (solid green line), and $\frac{1}{k+1}$, $k = 4$, submodular guarantee lower bound (solid red line) for buoy selection. The bottom subfigure (b) zooms in on subfigure (a) to emphasize the difference between GenSat and SFO-Greedy methods and exhaustive search.

the minimum proximity between all locations and sensors, while meeting restrictions on how many sensors can be placed in particular regions. This experiment tests the fractional worst-case performance of Gensat against SFO-Greedy and exhaustive search. In this case, fractional performance means the worst-case probability of detection of a fraction γ of the targets. For this experiment, exhaustive search represents the optimal worst-case probability of detection over all of the targets. This experiment has thirty-two buoys arranged in a ring, eight targets, and runs for two hundred trials. The rest of the experimental parameters are the same as the second experiment.

The results of the fourth experiment can be found in Figure 3.6. Both Gensat and SFO-

Table 3.2: Probability of detection (PD) results for seven targets and nine buoys.

Method	**GENSAT**	SFO-GREEDY	Exhaustive search
Min PD	**0.200**	0.000	0.200
Mean PD	**0.671**	0.586	0.671
Buoys Selected	$\{b, d, f, h\}$	$\{a, f, h\}$	$\{b, d, f, h\}$

GREEDY have significantly higher fractional worst-case probability of detection than the lower bound across all fractions of the eight targets $\lfloor \gamma M \rfloor$. Even when the lower-bound provides a trivial guarantee, $\lfloor \gamma M \rfloor \geq 3$, both GENSAT and SFO-GREEDY match the optimal performance. In fact, the performance of GENSAT and SFO-GREEDY are practically indistinguishable from exhaustive search for any fraction of the targets. Figure 3.6b zooms in on Figure 3.6a to show the minute differences between the approximate solutions of the proposed algorithms and the true optimal. The only differences are at $\lfloor \gamma M \rfloor < 3$, where GENSAT and SFO-GREEDY have slightly lower minimum probabilities of detection. While Figure 3.6b also shows that GENSAT and SFO-GREEDY have equivalent minimum probabilities of detection, GENSAT provides a better theoretical guarantee than SFO-GREEDY. The lower bound is shown by the red line in Figure 3.6a. The bound reflects the submodular guarantee of the forward greedy algorithm used in the experimental code, which is $\frac{1}{k+1}$[14]. Since the clique complex can be represented as the intersection of four matroids, the guarantee is 0.2.

Now we test GENSAT on a synthetic sensor placement problem with a different objective function than in previous experiments (facility location instead of probabilistic coverage). In this case, the goal is to select a set $\hat{A}$ of sensor locations to maximize the minimum proximity between all locations and sensors while meeting restrictions on how many sensors can be

placed in particular regions. The ground set V consists of all possible locations for the sensors. The objective functions in the min are facility location functions given as $f_i = \max_{a \in A}\{w_{ia}\}$, where w_{ia} is the proximity between location $i \in V$ and sensor $a \in A$. The randomly generated symmetric proximity matrix describes the closeness between locations and sensors and has values uniformly sampled from $[0, 1]$. The constraint on the number of sensors that can be placed in a given region is encoded in a randomly generated partition matroid, with four partitions and partition cardinality constraints of three. We use the forward greedy algorithm described in algorithm 1, so $\alpha = 1/2$. We run 100 trials, each with a different similarity matrix and partition matroid, and compare the estimated solution against the true optimal solution $A^\star$ found via exhaustive search over the maximal independent sets of the partition matroid.

Results are shown in figure 3.7. The optimal (solid green line) and estimated (dashed blue line) fractional minimum proximity are computed over all fractions γ of the M functions. The fractional minimum proximity is computed as $\min_{j \in \mathcal{J}_\gamma} f_j(A)$, where $\mathcal{J}_\gamma$ is the set of indices of the $\lfloor \gamma M \rfloor$ functions f_i with the greatest value. The lower bound βc (solid red line) from theorem 1 is computed for every fraction γ of the M functions as well. Since this lower bound is bicriterion, the minimum proximity β could be fixed to find the fraction of functions γ that meet the threshold βc. Note that the proposed GenSat algorithm nearly matches the optimal exhaustive search approach, even in the region where the bound becomes trivial (i.e., where $\gamma > \alpha \Rightarrow \lfloor \gamma M \rfloor \geq 13 \Rightarrow \beta = 0$).

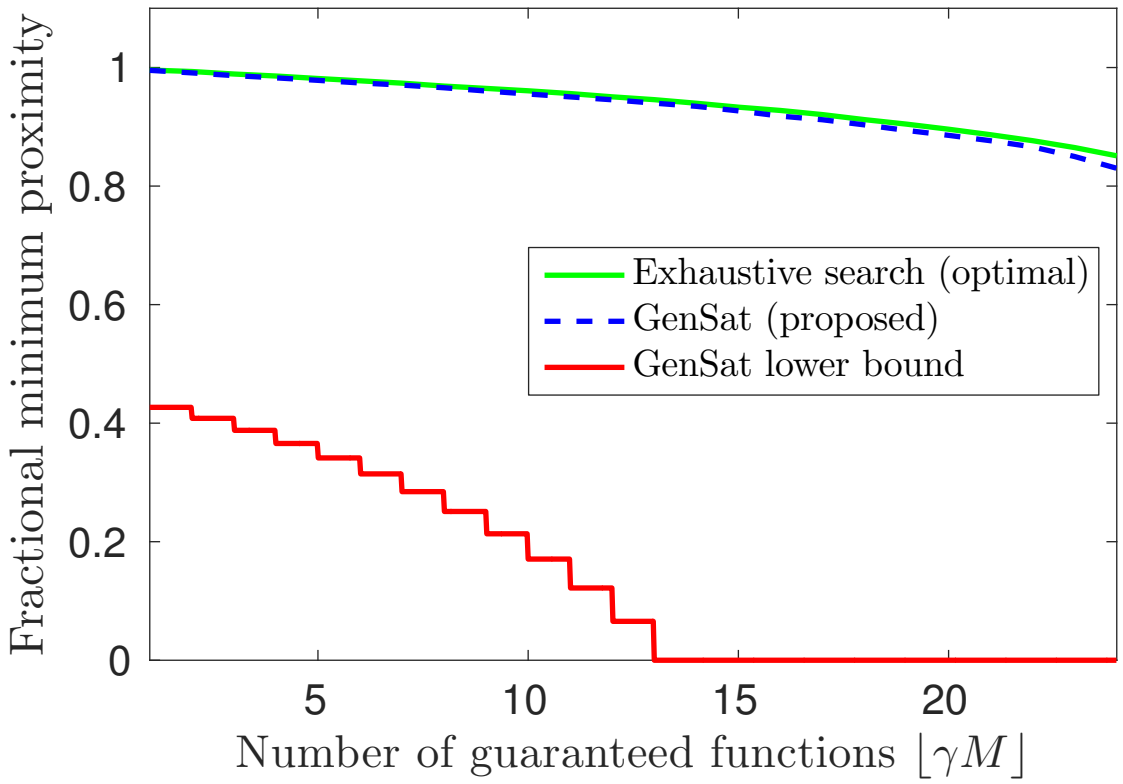


Figure 3.7: Fractional minimum proximity results averaged across 100 realizations of a $M = 24$ sensor placement problem, comparing the proposed GenSat (dashed blue line) method to exhaustive search (solid green line), and submodular guarantee lower bound on GenSat using the forward greedy algorithm (solid red line). GenSat achieves near-optimal performance, even when the bound is trivial.

Chapter 4

DEEP LEARNING THROUGH NOVEL NETWORK ARCHITECTURES

A general problem with deep neural networks is that they are essentially "black boxes" in the sense that the parameters are very difficult to interpret. While they produce state-of-the-art results, the lack of interpretable internal weights makes it very difficult to incorporate domain knowledge to improve the models. On the other end of the machine learning spectrum, there are many generative models that are interpretable, but may make unrealistic assumptions about the distribution of the data, may lead to intractable inference, and often perform worse than deep methods.

4.1 SISTA-RNN and DR-NMF

Our recently proposed SISTA-RNN and DR-NMF models, primarily the work of Dr. Scott Wisdom [74, 75, 76], are two examples of network architectures that result from unfolding the ISTA algorithm, which is an iterative method for solving the LASSO problem [66]. The LASSO objective function in equation (2.3) was proposed to find sparse and interpretable solutions for linear regression.

For this problem, the matrix $\mathbf{V} \in \mathbb{R}^{N \times N}$ is a dictionary whose columns correspond to basis vectors. The vector $\mathbf{z} = \mathbf{V}\mathbf{h}$ is a noisy observation taken through a measurement matrix $\mathbf{A} \in \mathbb{F}^{M \times N}$: $\mathbf{x} = \mathbf{A}\mathbf{z} + \epsilon$, where $M < N$ for a compressed sensing problem.

We extend ISTA to sequential data in two ways: sequential ISTA (SISTA), and warm start ISTA. SISTA, detailed in algorithm 4, assumes that the initial hidden state $h_t^{(0)}$ is correlated with the previous hidden state h_{t-1}^k and therefore that the signal is linearly predictable from the previous observation: $\mathbf{z}_t = \mathbf{F}\mathbf{z}_{t-1} + \mathbf{v}_t$, where $\mathbf{v}_t$ is zero-mean Gaussian noise. This changes

the prior on the hidden states from ISTA. The optimization problem for the SISTA-RNN is defined in problem (4.1). Further details can be found in [75].

The SISTA optimization problem is given by:

$$\min_{\mathbf{h}_{1:T}} \quad \sum_{t=1}^{T} \Big(\frac{1}{2} \big\| \mathbf{x}_t - \mathbf{A}\mathbf{D}\mathbf{h}_t \big\|_2^2 + \lambda_1 \big\| \mathbf{h}_t \big\|_1 + \frac{\lambda_2}{2} \| \mathbf{D}\mathbf{h}_t - \mathbf{F}\mathbf{D}\mathbf{h}_{t-1} \|_2^2 \Big), \tag{4.1}$$

with $\lambda_1 = 2\sigma^2\nu_1$ and $\lambda_2 = 2\sigma^2\nu_2$.

Warm start ISTA, detailed in algorithm 5, has a different model of the dynamics between samples, which also changes the optimization problem, which is in equation (4.2). Rather modeling the next samples as being linearly predictable from the previous observation, we use the warm start optimization technique, where $h_t^{(0)} = h_{t-1}^k$, and thus the optimization problem does not have the squared error term on the linear prediction of the previous observation. Additionally, the cost function for DR-NMF is defined for general β-divergences, which equivalent to the SISTA-RNN when $\beta = 2$, since it corresponds to summed squared error.

$$\underset{\mathbf{W} \in \mathcal{W}, \mathbf{H} \geq 0}{\text{minimize}} \quad D_\beta(\mathbf{X} || \mathbf{W}\mathbf{H}) + \lambda_1 \| \mathbf{H} \|_1 \tag{4.2}$$

Algorithm 3 Basic iterative soft-thresholding algorithm (ISTA)

Input: observations $\mathbf{x}$, dictionary $\mathbf{W}$, initial coefficients $\mathbf{h}^{(0)}$

1: **for** $k = 1$ to K **do**
2: $\quad \mathbf{s} \leftarrow (\mathbf{I} - \frac{1}{\alpha}\mathbf{W}^T\mathbf{W})\mathbf{h}^{(k-1)} + \frac{1}{\alpha}\mathbf{W}^T\mathbf{x}$
3: $\quad \mathbf{h}^{(k)} \leftarrow \text{soft}_{\lambda/\alpha}(\mathbf{s})$
4: **end for**
5: **return** $\mathbf{h}^{(K)}$

Unfolding these two inference algorithms results in two similar architectures, which differ from a conventional stack of RNNs and can be found in figure 4.1. A conventional RNN stack has a lattice structure where the hidden state is a function of the hidden states at the previous layer in the same time frame $\mathbf{h}_t^{(k-1)}$ and the the hidden state at the same layer in the

Algorithm 4 Sequential iterative soft-thresholding algorithm (SISTA)

Input: observation sequence $\mathbf{x}_{1:T}$, measurement matrix $\mathbf{A}$ dictionary $\mathbf{D}$, predictor $\mathbf{F}$, initial coefficients $\hat{\mathbf{h}}_0$

for $t = 1$ to T **do**
  $\mathbf{h}_t^{(0)} \leftarrow \mathbf{D}^{-1}\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}$ ▷ # *Initial estimate for* $\mathbf{h}_t$
  for $k = 1$ to K **do**
    $\mathbf{s} \leftarrow \left[\mathbf{I} - \frac{1}{\alpha}\mathbf{D}^T(\mathbf{A}^T\mathbf{A} + \lambda_2\mathbf{I})\mathbf{D}\right]\mathbf{h}_t^{(k-1)} + \frac{1}{\alpha}\mathbf{D}^T\mathbf{A}^T\mathbf{x}_t$
    $\mathbf{h}_t^{(k)} \leftarrow \text{soft}_{\lambda_1/\alpha}\left(\mathbf{s} + \frac{\lambda_2}{\alpha}\mathbf{D}^{-1}\mathbf{F}\mathbf{D}\hat{\mathbf{h}}_{t-1}\right)$
  end for
  $\hat{\mathbf{h}}_t \leftarrow \mathbf{h}_t^{(K)}$ ▷ # *Assign estimate for* $\mathbf{h}_t$
end forreturn $\hat{\mathbf{h}}_{1:T}$

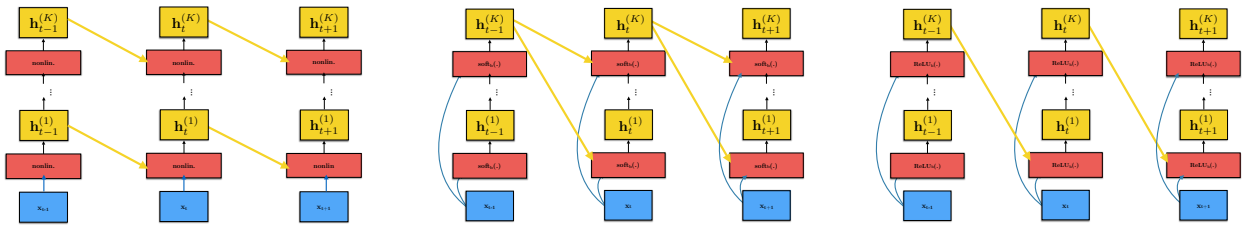
previous time frame $\mathbf{h}_{t-1}^{(k)}$, whereas the hidden states in the SISTA-RNN are a function of the original data $\mathbf{x}_t$, $\mathbf{h}_t^{(k-1)}$, and the last hidden state at the previous time point $\mathbf{h}_{t-1}^{(K)}$. DR-NMF is virtually the same as the SISTA-RNN, but the differences are that only $\mathbf{h}_t^{(0)}$ is a function of $\mathbf{h}_{t-1}^{(K)}$, and the hidden states are non-negative.

$$\mathbf{h}_t^{(k)} = \begin{cases} \sigma_{\mathbf{b}}\left(\mathbf{W}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{V}\mathbf{x}_t\right), & k = 1, \\ \sigma_{\mathbf{b}}\left(\mathbf{W}^{(k)}\mathbf{h}_{t-1}^{(k)} + \mathbf{S}^{(k)}\mathbf{h}_t^{(k-1)}\right), & k = 2..K, \end{cases} \tag{4.3}$$

$$\hat{\mathbf{y}}_t = \mathbf{U}\mathbf{h}_t^{(K)} + \mathbf{c}. \tag{4.4}$$

The SISTA-RNN was applied to sequential sparse image recovery on the Caltech-256 dataset [21]. For the images, the columns of the images function as the time index. The SISTA-RNN achieved the highest PSNR as well as the lowest test set mean squared error on the sparse image reconstruction, outperforming the unsupervised baselines as well as conventional stacked RNNs. Details can be found in [75].

For DR-NMF, full results can be found in [76]. We use the CHiME2 corpus [70], which uses utterances from the Wall Street Journal (WSJ-0) dataset that are convolved with



(a) Conventional stacked RNN (b) SISTA-RNN (c) DR-NMF

Figure 4.1: Comparison between standard stacked RNN architecture (a), SISTA-RNN (b), and DR-NMF (c). Blue nodes are inputs, yellow nodes are hidden states (i.e. estimates of sparse recovery coefficients), and red are the activation functions. Note that the only differences between a conventional stacked RNN, unfolded SISTA-RNN, and DR-NMF on the right is the connection of the input $\mathbf{x}_t$ to every vertical iteration layer for the unfolded models and the broadcasted connections between $\mathbf{h}_{t-1}^{(k)}$ and all $\mathbf{h}_{t-1}^{(i)}$ for SISTA-RNN compared to just $\mathbf{h}_t^{(0)}$ for DR-NMF.

Algorithm 5 Warm start ISTA

Input: observations $\mathbf{x}_{1:T}$, dictionary $\mathbf{W}$, initial coefficients $\mathbf{h}_0^{(K)}$

for $t = 1$ to T **do**
  $\mathbf{h}_t^{(0)} \leftarrow \mathbf{h}_{t-1}^{(K)}$ ▷ *# warm start from* $t-1$
  for $k = 1$ to K **do**
    $\mathbf{s} \leftarrow (\mathbf{I} - \frac{1}{\alpha}\mathbf{W}^T\mathbf{W})\mathbf{h}_t^{(k-1)} + \frac{1}{\alpha}\mathbf{W}^T\mathbf{x}_t$
    $\mathbf{h}_t^{(k)} \leftarrow \mathrm{soft}_{\lambda/\alpha}(\mathbf{s})$
  end for
end for
return $\mathbf{h}^{(K)}$

binaural room impulse responses (RIRs) and mixed with real-world nonstationary noise at six evenly spaced SNRs from −6dB to 9dB. The nonstationary noise was recorded in a home environment, and contains a variety of challenging noise types, including music, radio, television, children, and appliances. The RIRs were recorded in the same environment. The training set consists of 7138 utterances, the development set consists of 2460 utterances, and the test set consists of 1980 utterances, which are equally distributed across the six SNRs. All audio is sampled at 16kHz and we use only the left channel. Separation performance is measured using signal-to-distortion ratio (SDR) from the BSS Eval Matlab toolbox [71, 69].

We compare DR-NMF against sparse NMF (SNMF) using well-done multiplicative updates (MU), which is used to initialize the DR-NMF networks, and stacks of LSTM networks. SNMF uses 200 MU iterations at test time. The depth K of the LSTM stacks are chosen to match the depth of the DR-NMF networks, and the hidden node count N of the LSTMS are chosen to match the counts P of trainable parameters of DR-NMF networks.

The STFT uses 512-sample square-root Hann windows for analysis and synthesis with a hop of 128 samples, and therefore the feature dimension of input spectrogram frames is $F = 257$. For training deep models, the input spectrograms are split into sequences no longer than 500 frames. Deep network training uses the Adam optimizer [32] with a batch size of 32

and default parameters, except for the learning rate. LSTM network training uses gradient clipping to 1 and a learning rate of 10^{-4}, and DR-NMF network training uses no gradient clipping and a learning rate 10^{-3}. During training, model weights with the lowest validation loss are saved. Early stopping on the validation loss with a patience of 50 epochs determines training convergence. All deep networks are implemented in Keras [9] using Theano [64] as a backend. For SNMF, we use the Matlab implementation of well-done multiplicative updates [40]. Code to replicate our results is available online[1].

The results are shown in table 4.1. When 100% of the training set is used, an LSTM achieves the best mean SDR of 12.35 dB on the CHiME2 development set. However, even though it does not achieve the best SDR, the a DR-NMF model achieves the best validation loss of 0.0266. This mismatch indicates a discrepancy between SDR, which is computed in the time domain, and the mean squared reconstruction loss, which is computed in the magnitude spectrogram domain.

When only 10% of the training set is used, DR-NMF networks achieve superior performance in terms of both SDR and validation loss, reaching 11.12 dB SDR on the development set, compared to the best LSTM score of 10.59 dB.

Examining the learning curves, we saw that large LSTM networks quickly overfit (i.e., the validation loss starts to increase while the training loss continues to decrease) when only provided with 10% of the training set, while DR-NMF networks are consistently robust to overfitting. This suggests that DR-NMF networks exhibit stronger generalization performance compared to LSTMs and thus perform better when provided with less training data. Also, DR-NMF networks achieve the lowest validation loss in all cases.

Unfolding can also be used to interpolate between architectures that strictly correspond to the original model and those that are more like black boxes. Inference algorithms like ISTA have fixed parameters across layers, and therefore the SISTA-RNN and DR-NMF models must constrain their dictionary parameters to be tied across layers. In contrast, a conventional

[1] `https://github.com/stwisdom/dr-nmf`

Table 4.1: Results in terms of validation loss (dev. loss) and signal-to-disortion ratio (SDR) in dB on the CHiME2 development and test sets using 100% (center section) or 10% (right section) of the training data. K is the total number of layers or iterations, N is the LSTM hidden state dimension or the number of NMF basis vectors, and P is the total number of trainable parameters.

Architecture				**100% training set**			**10% training set**		
Model	K	N	P	Dev. loss	Dev. SDR	Eval. SDR	Dev. loss	Dev. SDR	Test SDR
SNMF MU	200	200	50k	0.0987	7.14	8.18	0.1319	6.51	7.47
SNMF MU	200	2000	500k	0.0846	7.71	8.61	0.0890	7.43	8.37
LSTM	2	54	100k	0.0408	11.51	12.53	0.0512	10.34	11.35
LSTM	2	244	1M	0.0339	11.95	12.90	0.0481	10.59	11.57
LSTM	5	70	250k	0.0426	10.90	11.94	0.0542	10.22	11.26
LSTM	5	250	2.5M	0.0344	**12.35**	**13.30**	0.0566	10.25	11.32
DR-NMF	2	200	100k	0.0320	10.94	11.86	0.0362	10.39	11.33
DR-NMF	2	2000	1M	0.0295	11.21	12.11	0.0354	10.54	11.43
DR-NMF	5	200	250k	0.0286	11.14	12.04	0.0332	10.88	11.80
DR-NMF	5	2000	2.5M	**0.0266**	11.31	12.19	**0.0316**	**11.12**	**11.99**

stacked RNN has different recurrence matrices at different layers. Untying the parameters in an unfolded network allows for flexibility to create more expressive models at the price of strict model correspondence.

4.2 Unitary recurrent neural networks

While deep feed-forward and recurrent neural networks continue to improve state-of-the-art results in many fields, recurrent neural networks (RNNs) suffer from the vanishing and

exploding gradient problem. When training RNNs with backpropogation, the gradient needs to flow through as many layers as length of the input sequence, so the gradient includes a running product of the recurrence matrix, and thus the magnitude of the gradient is largely determined by norm of the recurrence matrix. One way to keep the gradients from vanishing or exploding is to make the recurrence matrix unit norm, the most general class of which are unitary matrices. Unlike previous restricted-capacity unitary recurrent neural networks (uRNNs), we proposed a network architecture that optimizes the recurrence matrix directly on the unitary manifold, which we call the full-capacity uRNN.

Other approaches to address this problem include gradient clipping, orthogonal initialization of recurrence matrices, long short-term memory (LSTM) networks, and gated recurrent unit (GRU) networks. Using unitary recurrence matrices has been proposed to address the vanishing and exploding gradient problem [2], resulting in a complex-valued network with a rectified linear unit (ReLU) activation function. However, this approach in many cases restricts the class of unitary matrices that can be used in the network.

We have two main contributions:

1) We provide a theoretical argument to determine the smallest dimension N for which any parameterization of the unitary recurrence matrix does not cover the entire set of all unitary matrices. The argument relies on counting real-valued parameters and using Sard's theorem to show that the smooth map from these parameters to the unitary manifold is not onto. Thus, we can show that a previously proposed parameterization [2] cannot represent all unitary matrices larger than 7×7. Thus, such a parameterization results in what we refer to as a *restricted-capacity* unitary recurrence matrix.

2) To overcome the limitations of restricted-capacity parameterizations, we propose a new method for stochastic gradient descent for training the unitary recurrence matrix, which constrains the gradient to lie on the differentiable manifold of unitary matrices. This approach allows us to directly optimize a complete, or *full-capacity*, unitary matrix. Neither restricted-capacity nor full-capacity unitary matrix optimization require gradient clipping. Furthermore, full-capacity optimization still achieves good results without adaptation of the learning rate

during training.

Full-capacity uRNNs outperform restricted-capacity uRNNs and LSTMs on benchmark tasks including synthetic system identification, long-term memorization, frame-to-frame prediction of speech spectra, and pixel-by-pixel classification of handwritten digits.

Our derivation of this model differs from the derivations of the SISTA-RNN and DR-NMF models in that we did not unfold an inference model to arrive at this architecture. That being said, it can be viewed in the same light since the uRNN is equivalent to a single step of the SISTA algorithm with a unitary constraint on the dictionary.

4.2.1 Estimating the representation capacity of structured unitary matrices

The uRNN proposed by Arjovsky et al. [2] consists of the following nonlinear dynamical system that has real- or complex-valued inputs $\mathbf{x}_t$ of dimension M, complex-valued hidden states $\mathbf{h}_t$ of dimension N, and real- or complex-valued outputs $\mathbf{y}_t$ of dimension L:

$$\begin{aligned} \mathbf{h}_t =& \sigma_{\mathbf{b}}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{x}_t\right) \\ \mathbf{y}_t =& \mathbf{U}\mathbf{h}_t + \mathbf{c}, \end{aligned} \tag{4.5}$$

where $\mathbf{y}_t = \mathrm{Re}\{\mathbf{U}\mathbf{h}_t + \mathbf{c}\}$ if the outputs $\mathbf{y}_t$ are real-valued. The element-wise nonlinearity σ is

$$[\sigma_{\mathbf{b}}(\mathbf{z})]_i = \begin{cases} (|z_i| + b_i)\frac{z_i}{|z_i|}, & \text{if } |z_i| + b_i > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

Note that this non-linearity consists in a soft-thresholding of the magnitude using the bias vector $\mathbf{b}$. Hard-thresholding would set the output of σ to z_i if $|z_i| + b_i > 0$. The parameters of the uRNN are as follows: $\mathbf{W} \in U(N)$, unitary hidden state transition matrix; $\mathbf{V} \in \mathbb{C}^{N\times M}$, input-to-hidden transformation; $\mathbf{b} \in \mathbb{R}^N$, nonlinearity bias; $\mathbf{U} \in \mathbb{C}^{L\times N}$, hidden-to-output transformation; and $\mathbf{c} \in \mathbb{C}^L$, output bias.

Arjovsky et al. [2] propose the following parameterization of the unitary matrix $\mathbf{W}$:

$$\mathbf{W}_u(\theta_u) = \mathbf{D}_3\mathbf{R}_2\mathcal{F}^{-1}\mathbf{D}_2\mathbf{P}\mathbf{R}_1\mathcal{F}\mathbf{D}_1, \tag{4.7}$$

where $\mathbf{D}$ are diagonal unitary matrices, $\mathbf{R}$ are Householder reflection matrices [27], $\mathcal{F}$ is a discrete Fourier transform (DFT) matrix, and $\mathbf{P}$ is a permutation matrix. The resulting matrix $\mathbf{W}_u$ is unitary because all its component matrices are unitary. This decomposition is efficient because diagonal, reflection, and permutation matrices are $\mathcal{O}(N)$ to compute, and DFTs can be computed efficiently in $\mathcal{O}(N \log N)$ time using the fast Fourier transform (FFT). The parameter vector θ_u consists of $7N$ real-valued parameters: N parameters for each of the 3 diagonal matrices where $D_{i,i} = e^{j\theta_i}$ and $2N$ parameters for each of the 2 Householder reflection matrices, which are real and imaginary values of the complex reflection vectors $\mathbf{u}_i$: $\mathbf{R}_i = \mathbf{I} - 2\frac{\mathbf{u}_i \mathbf{u}_i^H}{\langle \mathbf{u}_i, \mathbf{u}_i \rangle}$.

Using the following theorem that can be used to determine when any particular unitary parameterization does not have capacity to represent all unitary matrices, specifically that (4.7) does not have the capacity to cover all $N \times N$ unitary matrices for $N > 7$. We state and prove this theorem in [59].

Theorem 2. *If a family of $N \times N$ unitary matrices is parameterized by P real-valued parameters for $P < N^2$, then it cannot contain all $N \times N$ unitary matrices.*

We now apply Theorem 2 to the parameterization (4.7). Note that the parameterization (4.7) has $P = 7N$ real-valued parameters. If we solve for N in $7N < N^2$, we get $N > 7$. Thus, the parameterization (4.7) cannot represent all unitary matrices for dimension $N > 7$.

4.2.2 Optimizing full-capacity unitary matrices on the Stiefel manifold

Now, we show how to get around the limitations of restricted-capacity parameterizations and directly optimize a full-capacity unitary matrix. We consider the Stiefel manifold of all $N \times N$ complex-valued matrices whose columns are N orthonormal vectors in $\mathbb{C}^N$ [63]. Mathematically, the Stiefel manifold is defined as

$$\mathcal{V}_N(\mathbb{C}^N) = \left\{ \mathbf{W} \in \mathbb{C}^{N \times N} : \mathbf{W}^H \mathbf{W} = \mathbf{I}_{N \times N} \right\}. \tag{4.8}$$

For any $\mathbf{W} \in \mathcal{V}_N(\mathbb{C}^N)$, any matrix $\mathbf{Z}$ in the tangent space $\mathcal{T}_{\mathbf{W}} \mathcal{V}_N(\mathbb{C}^N)$ of the Stiefel manifold satisfies $\mathbf{Z}^H \mathbf{W} - \mathbf{W}^H \mathbf{Z} = 0$ [63]. The Stiefel manifold becomes a Riemannian manifold when

its tangent space is equipped with an inner product. Tagare [63] suggests using the canonical inner product, given by

$$\langle \mathbf{Z}_1, \mathbf{Z}_2 \rangle_c = \mathrm{tr}\left(\mathbf{Z}_1^H (\mathbf{I} - \frac{1}{2}\mathbf{W}\mathbf{W}^H)\mathbf{Z}_2 \right). \tag{4.9}$$

Under this canonical inner product on the tangent space, the gradient in the Stiefel manifold of the loss function f with respect to the matrix $\mathbf{W}$ is $\mathbf{A}\mathbf{W}$, where $\mathbf{A} = \mathbf{G}\mathbf{W}^H - \mathbf{W}\mathbf{G}^H$ is a skew-Hermitian matrix and $\mathbf{G}$ with $G_{i,j} = \frac{\delta f}{\delta W_{i,j}}$ is the usual gradient of the loss function f with respect to the matrix $\mathbf{W}$ [63]. Using these facts, Tagare [63] suggests a descent curve along the Stiefel manifold at training iteration k given by the matrix product of the Cayley transformation of $\mathbf{A}^{(k)}$ with the current solution $\mathbf{W}^{(k)}$:

$$\mathbf{Y}^{(k)}(\lambda) = \left(\mathbf{I} + \frac{\lambda}{2}\mathbf{A}^{(k)} \right)^{-1} \left(\mathbf{I} - \frac{\lambda}{2}\mathbf{A}^{(k)} \right) \mathbf{W}^{(k)}, \tag{4.10}$$

where λ is a learning rate and $\mathbf{A}^{(k)} = \mathbf{G}^{(k)}\mathbf{W}^{(k)H} - \mathbf{W}^{(k)}\mathbf{G}^{(k)H}$. Gradient descent proceeds by performing updates $\mathbf{W}^{(k+1)} = \mathbf{Y}^{(k)}(\lambda)$. Tagare [63] suggests an Armijo-Wolfe search along the curve to adapt λ, but such a procedure would be expensive for neural network optimization since it requires multiple evaluations of the forward model and gradients. We found that simply using a fixed learning rate λ often works well. Also, RMSprop-style scaling of the gradient $\mathbf{G}^{(k)}$ by a running average of the previous gradients' norms [67] before applying the multiplicative step (4.10) can improve convergence. The only additional substantial computation required beyond the forward and backward passes of the network is the $N \times N$ matrix inverse in (4.10).

4.2.3 uRNN Experiments

All models are implemented in Theano [64], based on the implementation of restricted-capacity uRNNs by [2], available from `https://github.com/amarshah/complex_RNN`. All code to replicate our results is available from `https://github.com/stwisdom/urnn`. All models use RMSprop [67] for optimization, except that full-capacity uRNNs optimize their recurrence matrices with a fixed learning rate using the update step (4.10) and optional RMSprop-style

gradient normalization. Here, we highlight the performance of the full-capacity uRNNs against restricted-capacity uRNNs and LSTMs for the copy memory and pixel-by-pixel handwritten digit recognition tasks. The rest of the experimental results can be found in [59].

The experimental setup follows the copy memory problem from [2], which itself was based on the experiment from [26]. We consider alternative hidden state dimensions and extend the sequence lengths to $T = 1000$ and $T = 2000$, which are longer than the maximum length of $T = 750$ considered in previous literature.

In this task, the data is a vector of length $T+20$ and consists of elements from 10 categories. The vector begins with a sequence of 10 symbols sampled uniformly from categories 1 to 8. The next $T - 1$ elements of the vector are the ninth 'blank' category, followed by an element from the tenth category, the 'delimiter'. The remaining ten elements are 'blank'. The task is to output $T + 10$ blank characters followed by the sequence from the beginning of the vector. We use average cross entropy as the training loss function. The baseline solution outputs the blank category for $T + 10$ time steps and then guesses a random symbol uniformly from the first eight categories. This baseline has an expected average cross entropy of $\frac{10 \log(8)}{T+20}$.

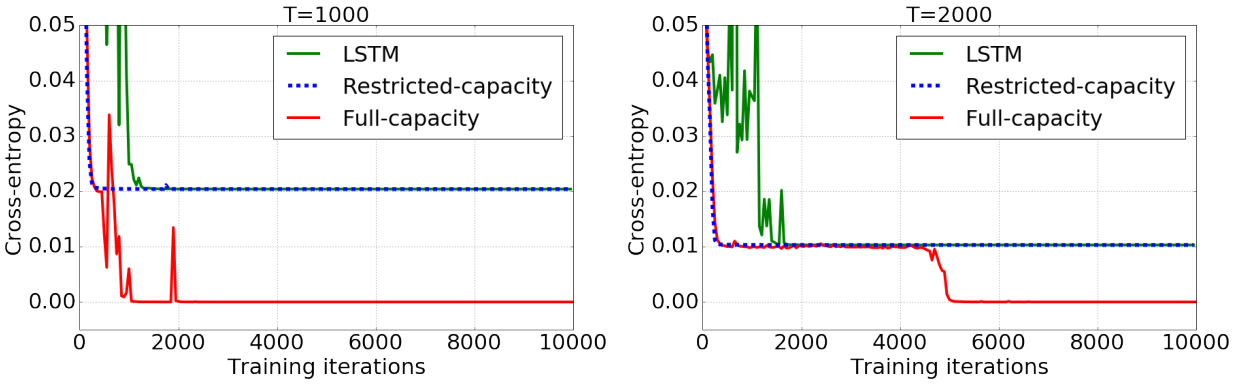


Figure 4.2: Results of the copy memory problem with sequence lengths of 1000 (left) and 2000 (right). The full-capacity uRNN converges quickly to a perfect solution, while the LSTM and restricted-capacity uRNN with approximately the same number of parameters are unable to improve past the baseline naive solution.

The full-capacity uRNN uses a hidden state size of $N = 128$ with no gradient normalization. To match the number of parameters ($\approx$ 22k), we use $N = 470$ for the restricted-capacity uRNN, and $N = 68$ for the LSTM. The training set size is 100000 and the test set size is 10000. The results of the $T = 1000$ experiment can be found on the left half of Figure 4.2. The full-capacity uRNN converges to a solution with zero average cross entropy after about 2000 training iterations, whereas the restricted-capacity uRNN settles to the baseline solution of 0.020. The results of the $T = 2000$ experiment can be found on the right half of Figure 4.2. The full-capacity uRNN hovers around the baseline solution for about 5000 training iterations, after which it drops down to zero average cross entropy. The restricted-capacity again settles down to the baseline solution of 0.010. These results demonstrate that the full-capacity uRNN is very effective for problems requiring very long memory.

First, we compare the performance of full-capacity uRNNs to restricted-capacity uRNNs and LSTMs on two tasks with synthetic data. The first task is synthetic system identification, where a uRNN must learn the dynamics of a target uRNN given only samples of the target uRNN's inputs and outputs. The second task is the copy memory problem, in which the network must recall a sequence of data after a long period of time.

For the task of system identification, we consider the problem of learning the dynamics of a nonlinear dynamical system that has the form (4.5), given a dataset of inputs and outputs of the system. We will draw a true system $\mathbf{W}_{sys}$ randomly from either a constrained set $\mathcal{W}_u$ of restricted-capacity unitary matrices using the parameterization $\mathbf{W}_u(\theta_u)$ in (4.7) or from a wider set $\mathcal{W}_g$ of restricted-capacity unitary matrices that are guaranteed to lie outside $\mathcal{W}_u$. We sample from $\mathcal{W}_g$ by taking a matrix product of two unitary matrices drawn from $\mathcal{W}_u$.

We use a sequence length of $T = 150$, and we set the input dimension M and output dimension L both equal to the hidden state dimension N. The input-to-hidden transformation $\mathbf{V}$ and output-to-hidden transformation $\mathbf{U}$ are both set to identity, the output bias $\mathbf{c}$ is set to $\mathbf{0}$, the initial state is set to $\mathbf{0}$, and the hidden bias $\mathbf{b}$ is drawn from a uniform distribution in the range $[-0.11, -0.09]$. The hidden bias has a mean of -0.1 to ensure stability of the system outputs. Inputs are generated by sampling T-length i.i.d. sequences of zero-mean,

diagonal and unit covariance circular complex-valued Gaussians of dimension N. The outputs are created by running the system (4.5) forward on the inputs.

We compare a restricted-capacity uRNN using the parameterization from (4.7) and a full-capacity uRNN using Stiefel manifold optimization with no gradient normalization as described in Section 4.2.2. We choose hidden state dimensions N to test critical points predicted by our arguments in Section 4.2.1 of $\mathbf{W}_u(\theta_u)$ in (4.7): $N \in \{4, 6, 7, 8, 16\}$. These dimensions are chosen to test below, at, and above the critical dimension of 7.

For all experiments, the number of training, validation, and test sequences are 20000, 1000, and 1000, respectively. Mean-squared error (MSE) is used as the loss function. The learning rate is 0.001 with a batch size of 50 for all experiments. Both models use the same matrix drawn from $\mathcal{W}_u$ as initialization. To isolate the effect of unitary recurrence matrix capacity, we only optimize $\mathbf{W}$, setting all other parameters to true oracle values. For each method, we report the best test loss over 100 epochs and over 6 random initializations for the optimization.

The results are shown in Table 4.2. "$\mathbf{W}_{sys}$ init." refers to the initialization of the true system unitary matrix $\mathbf{W}_{sys}$, which is sampled from either the restricted-capacity set $\mathcal{W}_u$ or the wider set $\mathcal{W}_g$.

Notice that for $N < 7$, the restricted-capacity uRNN achieves comparable or better performance than the full-capacity uRNN. At $N = 7$, the restricted-capacity and full-capacity uRNNs achieve relatively comparable performance, with the full-capacity uRNN achieving slightly lower error. For $N > 7$, the full-capacity uRNN always achieves better performance versus the restricted-capacity uRNN. This result confirms our theoretical arguments that the restricted-capacity parameterization in (4.7) lacks the capacity to model all matrices in the unitary group for $N > 7$ and indicates the advantage of using a full-capacity unitary recurrence matrix.

We now apply restricted-capacity and full-capacity uRNNs to real-world speech data and compare their performance to LSTMs. The main task we consider is predicting the log-magnitude of future frames of a short-time Fourier transform (STFT). The STFT is

Table 4.2: Results for system identification in terms of best normalized MSE. $\mathcal{W}_u$ is the set of restricted-capacity unitary matrices from (4.7), and $\mathcal{W}_g$ is a wider set of unitary matrices.

$\mathbf{W}_{sys}$ init.	Capacity	$N = 4$	$N = 6$	$N = 7$	$N = 8$	$N = 16$
$\mathcal{W}_u$	Restricted	4.81e−1	**6.75e−3**	3.53e−1	3.51e−1	7.30e−1
$\mathcal{W}_u$	Full	**1.28e−1**	3.03e−1	**2.16e−1**	**5.04e−2**	**1.28e−1**
$\mathcal{W}_g$	Restricted	**3.21e−4**	**3.36e−1**	3.36e−1	2.69e−1	7.60e−1
$\mathcal{W}_g$	Full	8.72e−2	3.86e−1	**2.62e−1**	**7.22e−2**	**1.00e−6**

a commonly used feature domain for speech enhancement, and is defined as the Fourier transform of short windowed frames of the time series. In the STFT domain, a real-valued audio signal is represented as a complex-valued $F \times T$ matrix composed of T frames that are each composed of $F = N_{win}/2 + 1$ frequency bins, where N_{win} is the duration of the time-domain frame. Most speech processing algorithms use the log-magnitude of the complex STFT values and reconstruct the processed audio signal using the phase of the original observations.

The frame prediction task is as follows: given all the log-magnitudes of STFT frames up to time t, predict the log-magnitude of the STFT frame at time $t + 1$.We use the TIMIT dataset [18]. According to common practice [23], we use a training set with 3690 utterances from 462 speakers, a validation set of 400 utterances, an evaluation set of 192 utterances. Training, validation, and evaluation sets have distinct speakers. Results are reported on the evaluation set using the network parameters that perform best on the validation set in terms of the loss function over three training trials. All TIMIT audio is resampled to 8kHz. The STFT uses a Hann analysis window of 256 samples (32 milliseconds) and a window hop of

128 samples (16 milliseconds).

The LSTM requires gradient clipping during optimization, while the restricted-capacity and full-capacity uRNNs do not. The hidden state dimensions N of the LSTM are chosen to match the number of parameters of the full-capacity uRNN. For the restricted-capacity uRNN, we run models that match either N or number of parameters. For the LSTM and restricted-capacity uRNNs, we use RMSprop [67] with a learning rate of 0.001, momentum 0.9, and averaging parameter 0.1. For the full-capacity uRNN, we also use RMSprop to optimize all network parameters, except for the recurrence matrix, for which we use stochastic gradient descent along the Stiefel manifold using the update (4.10) with a fixed learning rate of 0.001 and no gradient normalization.

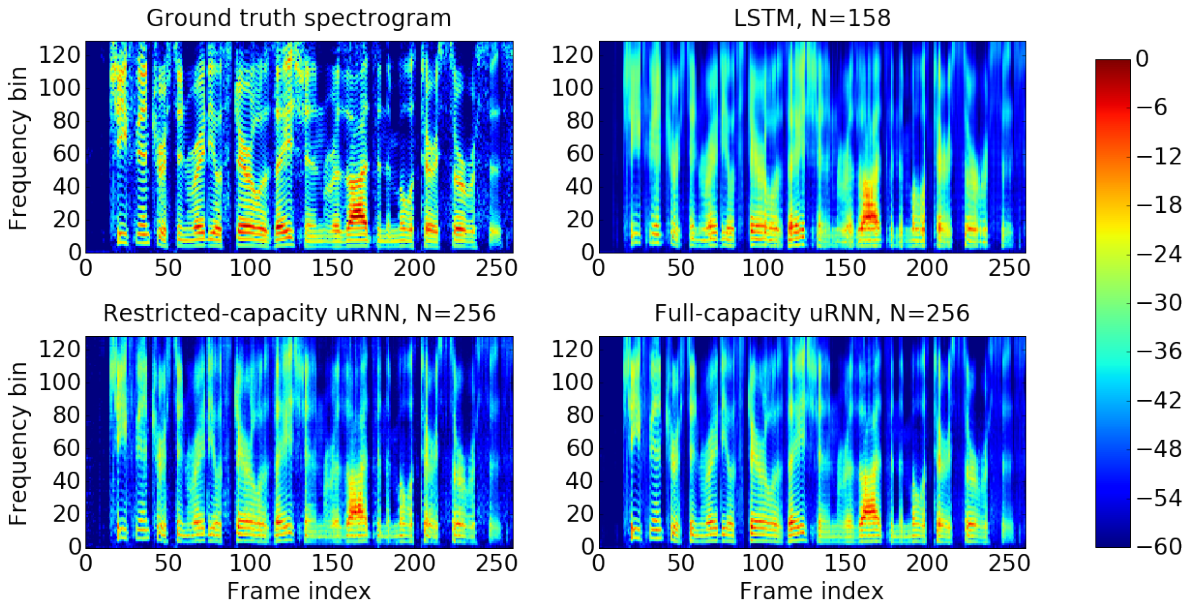


Figure 4.3: Ground truth and one-frame-ahead predictions of a spectrogram for an example utterance. For each model, hidden state dimension N is chosen for the best validation MSE. Notice that the full-capacity uRNN achieves the best detail in its predictions.

Results are shown in Table 4.3, and Figure 4.3 shows example predictions of the three types of networks. Results in Table 4.3 are given in terms of the mean-squared error (MSE) loss function and several metrics computed on the time-domain signals, which are reconstructed

from the predicted log-magnitude and the original phase of the STFT. These time-domain metrics are segmental signal-to-noise ratio (SegSNR), short-time objective intelligibility (STOI), and perceptual evaluation of speech quality (PESQ). SegSNR, computed using [6], uses a voice activity detector to avoid measuring SNR in silent frames. STOI is designed to correlate well with human intelligibility of speech, and takes on values between 0 and 1, with a higher score indicating higher intelligibility [62]. PESQ is the ITU-T standard for telephone voice quality testing [60, 29], and is a popular perceptual quality metric for speech enhancement [43]. PESQ ranges from 1 (bad quality) to 4.5 (no distortion).

Note that full-capacity uRNNs generally perform better than restricted-capacity uRNNs with the same number of parameters, and both types of uRNN significantly outperform LSTMs.

As another challenging long-term memory task with natural data, we test the performance of LSTMs and uRNNs on pixel-by-pixel MNIST and permuted pixel-by-pixel MNIST, first proposed by [39] and used by [2] to test restricted-capacity uRNNs. For permuted pixel-by-pixel MNIST, the pixels are shuffled, thereby creating some non-local dependencies between pixels in an image. Since the MNIST images are 28×28 pixels, resulting pixel-by-pixel sequences are $T = 784$ elements long. We use 5000 of the 60000 training examples as a validation set to perform early stopping with a patience of 5. The loss function is cross-entropy. Weights with the best validation loss are used to process the evaluation set. The full-capacity uRNN uses RMSprop-style gradient normalization.

Learning curves are shown in Figure 4.4, and a summary of classification accuracies is shown in Table 4.4. For the unpermuted task, the LSTM with $N = 256$ achieves the best evaluation accuracy of 98.2%. For the permuted task, the full-capacity uRNN with $N = 512$ achieves the best evaluation accuracy of 94.1%, which is state-of-the-art on this task. Both uRNNs outperform LSTMs on the permuted case, achieving their best performance after fewer traing epochs and using an equal or lesser number of trainable parameters. This performance difference suggests that LSTMs are only able to model local dependencies, while uRNNs have superior long-term memory capabilities. Despite not representing all

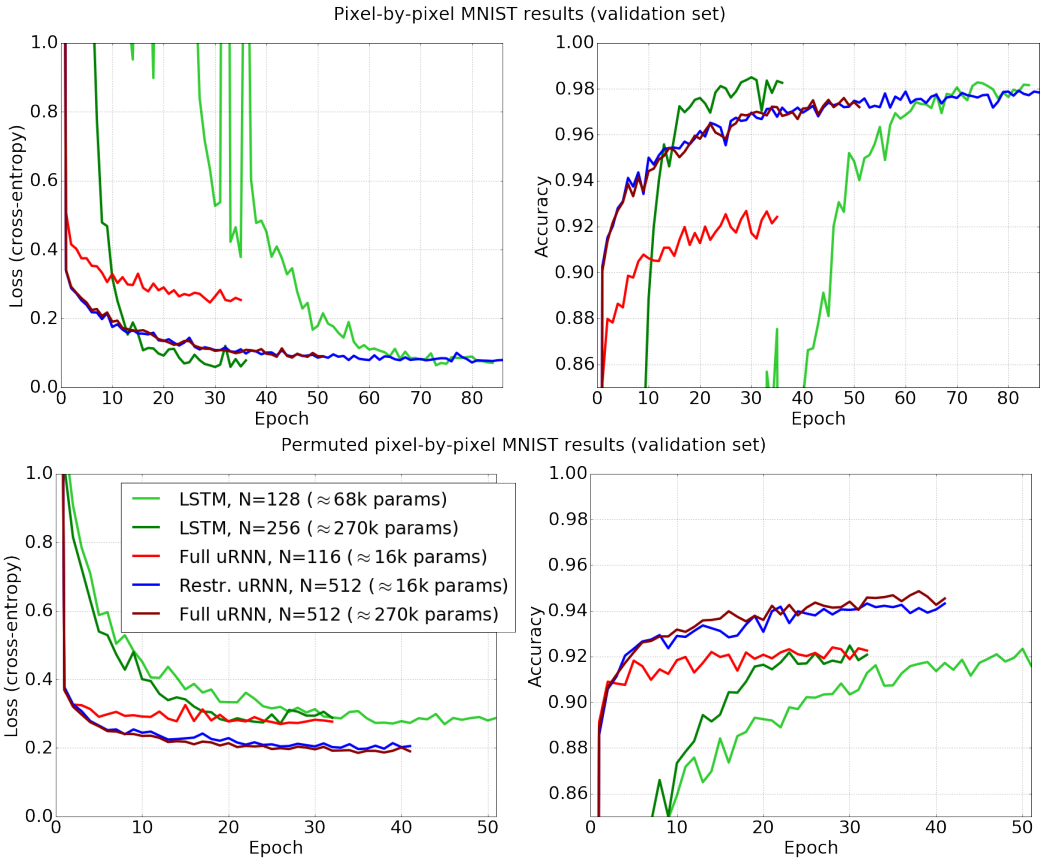


Figure 4.4: Learning curves for unpermuted pixel-by-pixel MNIST (top panel) and permuted pixel-by-pixel MNIST (bottom panel).

unitary matrices, the restricted-capacity uRNN with $N = 512$ still achieves impressive test accuracy of 93.3% with only 1/16 of the trainable parameters, outperforming the full-capacity uRNN with $N = 116$ that matches number of parameters. This result suggests that further exploration into the potential trade-off between hidden state dimension N and capacity of unitary parameterizations is necessary.

Table 4.3: Log-magnitude STFT prediction results on speech data, evaluated using objective and perceptual metrics (see text for description).

Model	N	# parameters	Valid. MSE	Eval. MSE	SegSNR (dB)	STOI	PESQ
LSTM	84	≈83k	18.02	18.32	1.95	0.77	1.99
Restricted-capacity uRNN	128	≈67k	15.03	15.78	3.30	0.83	2.36
Restricted-capacity uRNN	158	≈83k	15.06	**14.87**	3.32	0.83	2.33
Full-capacity uRNN	128	≈83k	**14.78**	15.24	**3.57**	**0.84**	**2.40**
LSTM	120	≈135k	16.59	16.98	2.32	0.79	2.14
Restricted-capacity uRNN	192	≈101k	15.20	15.17	3.31	0.83	2.35
Restricted-capacity uRNN	256	≈135k	15.27	15.63	3.31	0.83	2.36
Full-capacity uRNN	192	≈135k	**14.56**	**14.66**	**3.76**	**0.84**	**2.42**
LSTM	158	≈200k	15.49	15.80	2.92	0.81	2.24
Restricted-capacity uRNN	378	≈200k	15.78	16.14	3.16	0.83	2.35
Full-capacity uRNN	256	≈200k	**14.41**	**14.45**	**3.75**	**0.84**	**2.38**

Table 4.4: Results for unpermuted and permuted pixel-by-pixel MNIST. Classification accuracy is reported for trained model weights that achieve the best validation loss.

	Model	N	# params	Validation accuracy	Evaluation accuracy
Unpermuted	LSTM	128	≈ 68k	98.1	97.8
	LSTM	256	≈270k	**98.5**	**98.2**
	Restricted-capacity uRNN	512	≈ 16k	97.9	97.5
	Full-capacity uRNN	116	≈ 16k	92.7	92.8
	Full-capacity uRNN	512	≈270k	97.5	96.9
Permuted	LSTM	128	≈ 68k	91.7	91.3
	LSTM	256	≈270k	92.1	91.7
	Restricted-capacity uRNN	512	≈ 16k	94.2	93.3
	Full-capacity uRNN	116	≈ 16k	92.2	92.1
	Full-capacity uRNN	512	≈270k	**94.7**	**94.1**

Chapter 5

UNFOLDING DISCRETE OPTIMIZATION ALGORITHMS

Unfolding need not be limited to continuous optimization problems. Discrete optimization algorithms can potentially benefit from discriminative training as well. Previous work has established connections between submodular function optimization and dictionary selection problems [10, 8], and others have investigated learning submodular functions from data [11]. More recent work has linked supervised learning to several discrete optimization algorithms including iterative hard-thresholding and greedy algorithms [68, 56]. In all of this previous work, dictionaries are either assumed or chosen beforehand. By unfolding the algorithms, we can not only learn dictionaries from data, but learn *through* the algorithm.

5.1 Unfolded matching pursuit

For dictionary learning problems, each step of a greedy algorithm like matching pursuit (MP) can be seen as a layer in a feedforward network with residual input connections and shared weights, or as a recurrent network where the time dimension corresponds to the MP iterations and the same data is fed in at each time step. Much like many conventional architectures, each iteration of the unfolded MP architecture performs a linear operation which identifies how well dictionary elements explain the remaining residual of the input data and a non-linear activation function that greedily selects the best dictionary element. Finally, the network outputs a vector indicating the selected dictionary elements, the corresponding mixing weights, and the approximation to the input data. In the case of Orthogonal Matching Pursuit (OMP), all the mixing weights need to be recomputed each iteration so that they remain mutually orthogonal.

The choice of activation function is important to the training stability and performance

of the algorithm. The ideal choice is the argmax function, which potentially creates some difficulty in training the network. In order to optimize the dictionary parameters in the network, the gradient must be defined for this non-differentiable function, for which there are several strategies. One choice is to define a constant subdifferential for the argmax function, which is the default approach in some automatic differentiation toolkits such as in Tensorflow [1]. Specifically, the Jacobian J for the argmax function is almost completely zero, except for the index J_{ii}.

For sparse dictionary learning problems, the goal is to reconstruct the signal as accurately as possible while having minimum support on the dictionary. Many approaches use a convex relaxation (ℓ_1-norm) of the minimum support. The ℓ_1-norm is known to promote sparse solutions. Since it is a continuous relaxation, though, it is difficult to control the sparsity precisely, which is important for applications like compression where there can be hard constraints like storage and transmission rates. Such precision is trivial when using discrete algorithms.

The resulting unfolded networks are novel architectures that can leverage the advantages of discriminative training and can retain theoretical guarantees of the original algorithm. For sparse dictionary learning problems, I propose the Learned Unfolded Matching Pursuit (LUMP) and Learned Orthogonal Matching Pursuit (LOMP) algorithm. Both are derived from the classic Matching Pursuit and Orthogonal Matching Pursuit algorithms [45, 51]. They are greedy algorithms that choose the element that best explains the residual signal. The difference is that OMP recalculates the mixing weights after each iteration so that the combination of dictionary elements remain mutually orthogonal. In addition, these algorithms are linked to submodular optimization through the OMP algorithm from Das and Kempe [10], defined in algorithm 6. Das and Kempe prove that OMP achieves the a lower-bound based on the submodularity ratio of the objective function or the eigenvalues for the covariance matrix of dictionary.

Unfolding OMP results in a feedforward network that is k layers deep, where k is the desired cardinality of the solution set. The inputs to each layer i are a vector $\mathbf{s}$ encoding the

Algorithm 6 ORTHOGONAL MATCHING PURSUIT OMP (Z, V)

$A \leftarrow \emptyset$
while $|A| < k$ **do**
    $S \leftarrow V \backslash A$
    $s^* \leftarrow \text{argmax}_{s \in S} \text{abs}(\text{Cov}(\text{Res}(X, A), s))$
    $A \leftarrow A \cup \{s^*\}$
end while
return A

selected elements, the corresponding weights $\mathbf{h}$, and the signal $\mathbf{x}$. In each layer, the unselected dictionary elements are run through an inner product with the residual after which the activation function is applied. The output of the activation function is the selected element, which is added to $\mathbf{s}$. The corresponding regression coefficient is added to $\mathbf{h}$ from which $\mathbf{r}^{(i+1)}$ can computed. The update equations are defined in equation (5.1), and a depiction of the computational graph is in figure 5.1.

$$\begin{aligned} \mathbf{r}^{(i)} &= \mathbf{x} - \mathbf{h}^{(i)\top}\mathbf{V} \\ \mathbf{s}^{(i+1)} &= \mathbf{s}^{(i)} + \text{argmax}((\mathbf{1} - \mathbf{s}^{(i)}) \odot \mathbf{V}\mathbf{r}^{(i)}) \\ \mathbf{h}^{(i+1)} &= \mathbf{h}^{(i)} + \mathbf{s}^{(i)} \odot (\mathbf{1} - \mathbf{s}^{(i)}) \odot \mathbf{V}\mathbf{r}^{(i)} \end{aligned} \tag{5.1}$$

$$\begin{aligned} \mathbf{r}^{(i)} &= \mathbf{x} - \mathbf{h}^{(i)\top}\mathbf{V} \\ \mathbf{s}^{(i+1)} &= \mathbf{s}^{(i)} + \text{argmax}((\mathbf{1} - \mathbf{s}^{(i)}) \odot \mathbf{V}\mathbf{r}^{(i)}) \\ \mathbf{h}^{(i+1)} &= \mathbf{V}^{+}_{\mathbf{s}^{(i+1)}}\mathbf{x} \end{aligned} \tag{5.2}$$

Choosing the activation function for the network is an important decision. In order to correspond to MP/OMP, it needs to perform a discrete selection of dictionary elements, which

can be done using an argmax. The argmax is not differentiable, potentially causing an issue for gradient descent. One technique to solve this problem is using a subgradient method to compute the Jacobian. Since the argmax is a piecewise constant algorithm, the result will be a zero almost everywhere. More often, the Jacobian is set to be an identity matrix, commonly referred to as a Straight Through Estimator (STE) [3]. This is one technique we use in the following experiments to optimize LUMP and LOMP.

A second way is to train through the selection is through continuous relaxation. By using a strongly convex regularizer and computing the Fenchel conjugate of the max function, a smooth convex function can be derived and used in place of the argmax [46]. Using negative entropy, the resulting argmax substitute is the softmax function. Softmax can be implemented with a temperature parameter τ, as demonstrated in equation (2.14), that controls the peakiness of the function. With low temperature ($\tau < 1$), the softmax closely approximates the argmax, but propagation of the gradient during training will be too limited because the softmax saturates so quickly. As with the subgradient method, by selecting the single max element, the network might not be able to sufficiently update parameters and learn a better dictionary. With a higher temperature, the softmax can make more of a *soft* selection of dictionary elements, which would come at the cost of losing fidelity with the argmax.

The third way to optimize the network parameters through the argmax function is a combination of the first two. Because the softmax, especially at low temperatures, closely resembles the argmax but is smooth and differentiable, the network can use the outputs of the argmax to compute the loss and then the gradient can be backpropagated through the softmax during training [30]. In other words, on the forward pass, the new best state $\hat{\mathbf{s}}^{(i)}$ is computed with the argmax, so the states $\{\mathbf{s^{(i)}}\}$ are discrete, but on the backward pass the gradients are computed through a softmax with temperature τ. We use the subgradient and combination approaches in the following experiments. A depiction of LUMP and LOMP as a computational graph can be found in Figures 5.1 and 5.2 respectively.

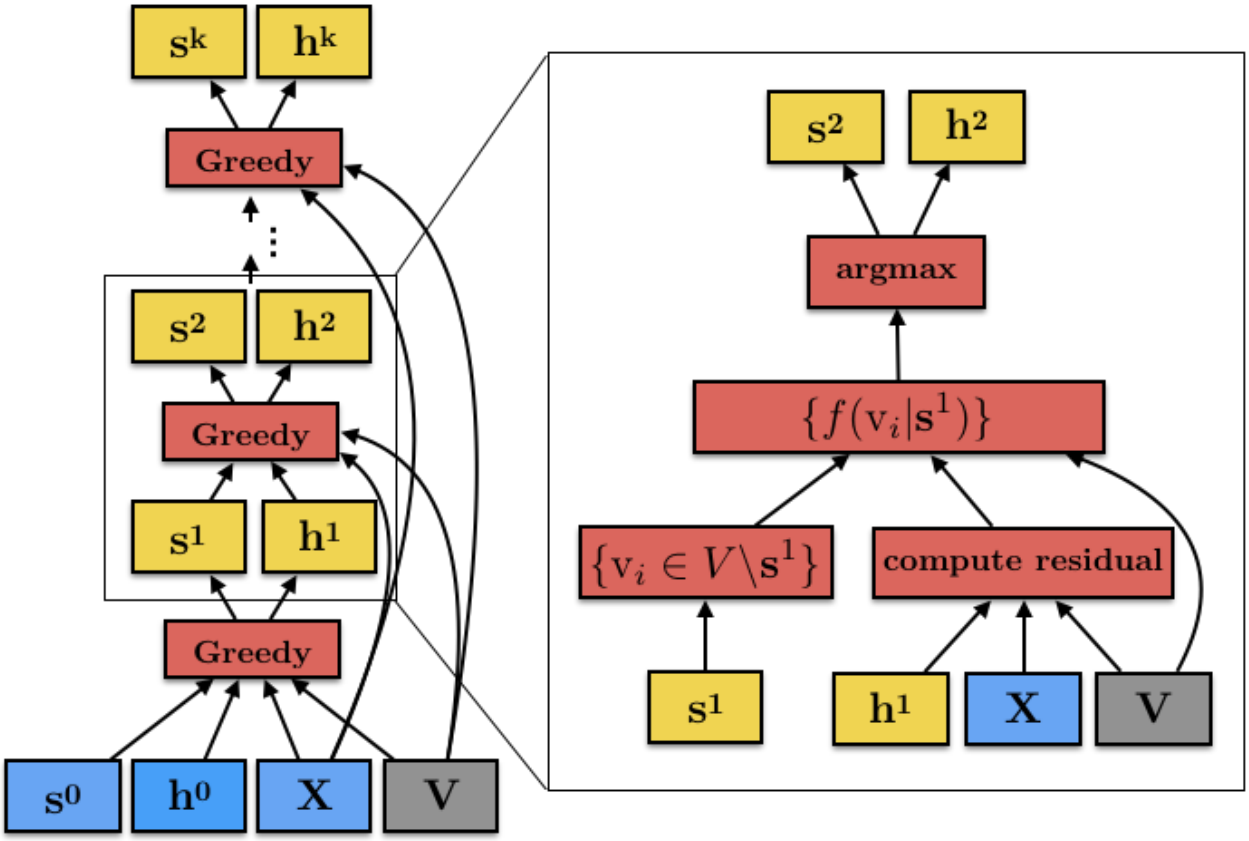


Figure 5.1: Computational graph for Learned Unfolded Matching Pursuit (LUMP)

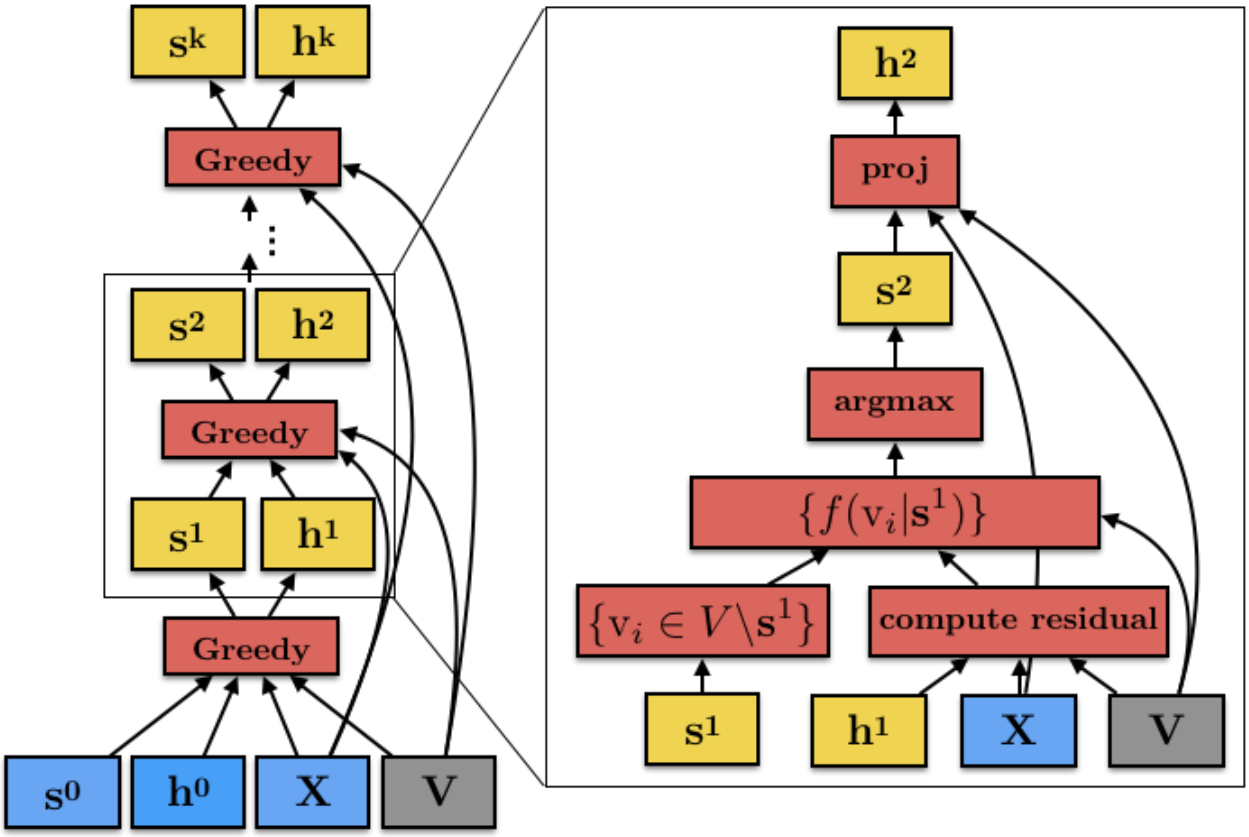


Figure 5.2: Computational graph for Learned Orthogonal Matching Pursuit (LOMP)

5.1.1 Experiments

First, we test LUMP as an autoencoder on MNIST data [41]. The goal in this experiment is to find a compressed representation of the data and to be able to reconstruct the matrix with minimal loss. We reshape the 28×28 images into a vector of length 784. We then compare LUMP against a feedforward autoencoder (FFAE). The best performing LUMP model has a dictionary of size 2048, which results in 1.6M parameters, and uses Adam with a learning rate of 1e-4, and a decay of 1e-5. The dictionary is initialized with exemplars from the training set. LUMP has a cardinality constraint of $K = 20$. The best performing FFAE model of similar size and compression has 5 hidden layers of sizes 512-128-20-128-512, respectively, and uses Adam with a learning rate of 1e-3 and a decay of 1e-5. From a compression point of view, the models are reducing the size of the images by a factor of 39. The loss function for both is normalized mean squared error, and both models were trained for 500 epochs with early stopping and a patience of 10 epochs.

Table 5.1: MNIST autoencoder performance

Method	# params	Validation MSE	Test MSE	Test PSNR
OMP	1.6M	-	6.08e-5	26.4
FFAE	940k	2.55e-5	2.55e-5	17.6
LUMP	1.6M	2.42e-5	2.41e-5	13.2

The results in Table 5.1 show that LUMP is achieves a lower mean squared error than a similar sized feedforward autoencoder, and significantly improves during training. LUMP took 50 epochs to converge, whereas FFAE took 500 epochs to converge. LUMP does take approximately 16 times longer to train per epoch, and is very sensitive to certain

hyperparameters–specifically dictionary size and learning rate. In general, the dictionary needs to be overcomplete and the learning rate low enough for LUMP to converge somewhat smoothly during training. Unfortunately, the large size of the vectorized images necessitates the model to have a huge number of parameters, which slows down the training. If the learning rate is slightly too small or large, LUMP will have problems with vanishing and exploding gradients.

Perceptually, LUMP creates good quality reconstructions of the images as seen in Figure 5.3. Compared to the outputs of FFAE, LUMP's reconstructed images are less blurry despite the naturally smoothing nature of mean squared loss. Training also significantly improves LUMP's output compared to just using the initial dictionary learned through Least Angle Regression (LARS) using the scikit-learn toolkit [13, 52].

Next we break up the MNIST images into independent rows and test the proposed LUMP and LOMP networks as autoencoders on the 28-dimensional data. In addition to OMP and FFAE, we add Learned Iterative Soft-Thresholding Algorithm (LISTA) as a baseline [20]. In this scenario, we compress each vector into 5 coefficients which is a compression ratio of 5.6.

The results can be found in Table 5.2, and a sample of outputs in Figure 5.4. LOMP and LUMP perform significantly better than the baselines, achieving orders of magnitude better MSE, and significant increases in Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM), which are used to measure the perceptual quality of the images relative to the reference data [73]. Consistent with the metrics, the outputs from the baselines in Figure 5.4 all feature some kind of distortion. The outputs from OMP in Figure 5.4a are recognizable, but are are gray in the background and blur in the rows where the digit is present. The FFAE samples in Figure 5.4b are somewhat legible, and it does a better job with capturing the darker background than OMP, but the digits are heavily blurred, and LISTA has apparent issues encoding certain rows. Both LOMP and LUMP achieve the highest possible SSIM score, meaning the images are practically indistinguishable from the reference images, and this is validated by the sample images in Figure 5.4d-5.4f.

Table 5.2: Row-wise MNIST autoencoder performance

Method	# params	Validation MSE	Test MSE	Test PSNR	Test SSIM
OMP	24K	-	0.026	15.8	0.69
FFAE	9K	0.037	0.026	15.8	0.74
LISTA	10K	0.0072	0.052	12.8	0.54
LUMP	24K	0.0010	0.00046	33.4	**1.0**
LOMP	24K	0.00028	**0.00029**	**35.3**	**1.0**

Finally, we test LUMP and LOMP as autoencoders on the CIFAR10 dataset [36]. As with the MNIST dataset, we split each image into rows that are encoded independently. We compress each row into 5 coefficients for a compression ratio of 6.4. For comparison, we use OMP, FFAE, and LISTA as baselines. As an additional means to evaluate the quality of the compressed images, we train a classifier for CIFAR10 on the original images. We then evaluate with each of the compressed outputs.

The results are shown in Table 5.3, and a sample of outputs in Figure 5.5. On this more visually complex dataset, LOMP again achieves the best performance across all metrics. Just as with the MNIST experiment, LOMP trains better in terms of the MSE loss and the perceptual metrics PSNR and SSIM. Unlike the previous experiment, however, LUMP and OMP achieve very similar performance on the metrics in Table 5.3. In this case, LUMP was able to compensate for the lack of orthogonal projection by learning a dictionary that was better tuned to the data. While the metrics are even, the type of noise and distortion added to the reconstructed signals is very different. The errors produced when using OMP looks like additive white Gaussian noise, whereas LUMP produces smoother but blurry images as

with the other neural network baselines. In training the dictionary, LUMP seems to settle into a middle ground between the OMP and FFAE solutions.

The CIFAR10 classifier is a convolutional neural network (CNN) with four two-dimensional convolutional layers with ReLU activation functions (the first two with 32 filters of size 3×3, the second with 64 3×3 filters), max-pooling layers after the second and fourth convolutional layers, a linear layer with dimension 512 and a ReLU activation, and a output layer of dimension 10 with a softmax activation. The classifier was trained on the CIFAR10 reference images with categorical cross-entropy (CE) loss and we used the Adam optimizer as well as dropout after the max-pooling layers. As shown in Table 5.4, it achieves a CE loss of 0.78 and classification accuracy of 0.73 on the reference data. Consistent with MSE, PSNR, and SSIM, LOMP maintains the highest relative performance at a CE of 1.08 and 0.64 accuracy. LUMP and OMP similarly have accuracies of 0.54 and 0.55, though LUMP does achieve a lower CE of 1.36 compared to 1.51, and the LISTA and FFAE compressed images baselines decrease the classification performance significantly. Figure 5.6 shows the confusion matrices when evaluating the classifier with the images from each of the methods. LOMP and LUMP have a fairly similar distribution to the reference images, but more often wrongly classify trucks as automobiles and mix up cats and dogs. To a greater degree, LUMP also mixes up the animal classes amongst other and the vehicles with other vehicles. LISTA has a similar but more extreme error pattern to LUMP in addition to over classification of all classes and airplanes and ships. OMP tends to wrongly classify many images as deer and frogs as well as classifying automobiles as trucks. FFAE seems to classify most images into a few select classes, which makes sense as most of the fine-grained detail of the images is lost after compressing them.

The addition of the orthogonal projection is the only difference between MP and OMP, but it is an important one. On top of the convergence guarantee provided by OMP, training the dictionary through the orthogonal projection results in a less coherent dictionary–a desirable property for dictionary learning algorithms. The worst-case coherence μ for the best performing LOMP model is $\mu = 0.062$ and the average coherence ν is $\nu = 0.00059$ compared to $\mu = 1.0$ and $\nu = 0.54$ for LUMP.

Table 5.3: Row-wise CIFAR10 autoencoder performance

Method	# params	Validation MSE	Test MSE	Test PSNR	Test SSIM
OMP	32K	-	0.0025	25.9	0.97
FFAE	1.1K	0.0068	0.0069	21.6	0.91
LISTA	25K	0.0046	0.0046	23.2	0.94
LUMP	19K	0.0025	0.0025	25.9	0.97
LOMP	32K	0.0015	**0.0016**	**28.1**	**0.98**

Table 5.4: Row-wise CIFAR10 autoencoder classification performance

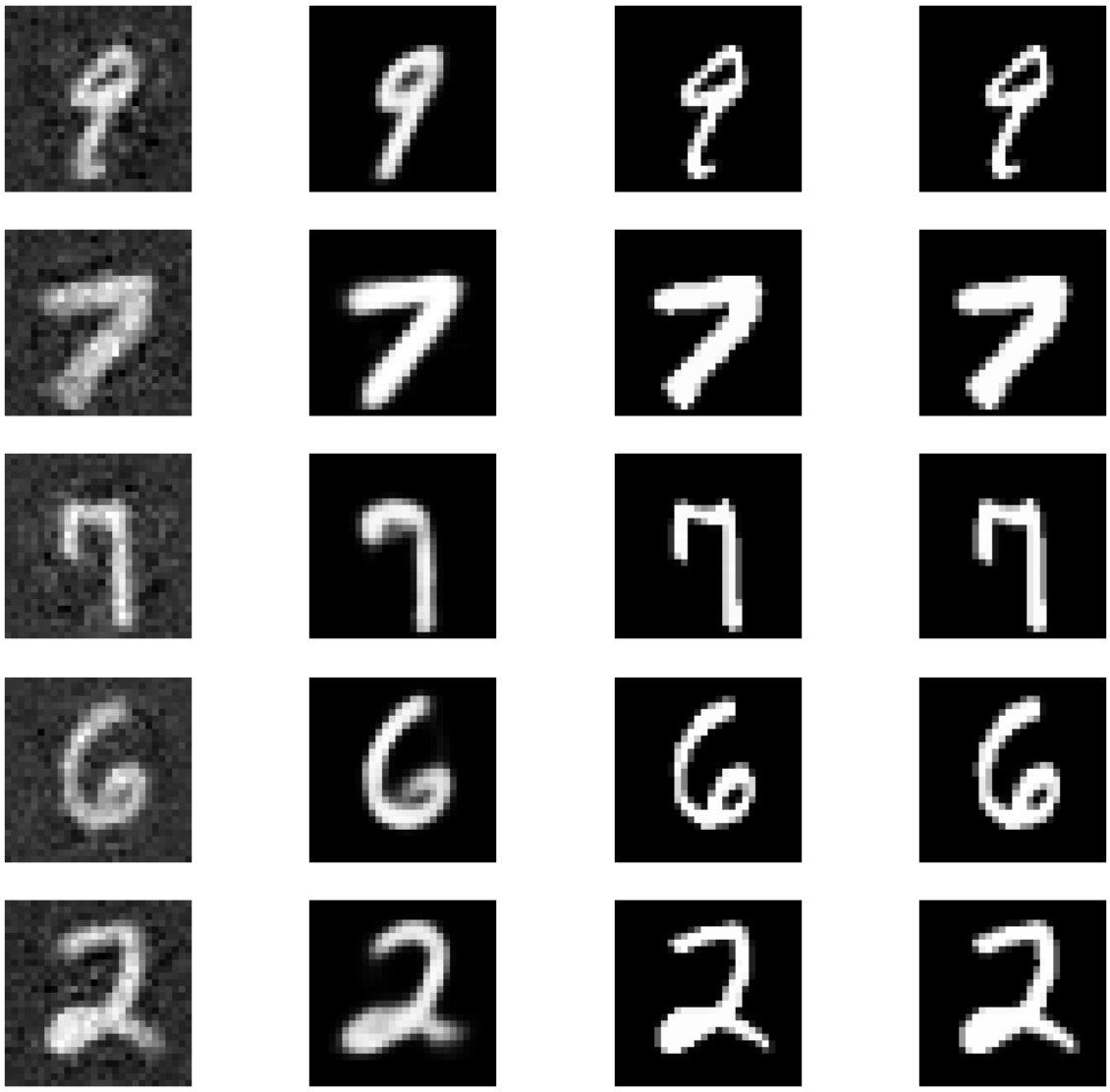
Method	Test Categorical CE	Test Accuracy
OMP	1.51	0.55
FFAE	2.71	0.24
LISTA	2.01	0.37
LUMP	1.36	0.54
LOMP	**1.08**	**0.64**
Reference	0.78	0.73

5.2 Future work

Another interesting method with connections to both unfolded ISTA and matching pursuit algorithms is unfolded iterative hard-thresholding (IHT). IHT is an alternative to ISTA that solves a slightly different optimization problem given in Equation (2.2). where $\|\cdot\|_0$ is the ℓ_0 norm that counts the number of non-zero elements of a vector. The ℓ_0 norm is not convex, but more explicitly controls the support of the solution, and does not suffer from the same bias problem that ISTA has. The hard-thresholding algorithm in [4] is guaranteed to not increase the objective function and converge to a local minimum as long as $\|\mathbf{AV}\|_2 \leq 1$. Besides the norm restriction, the key to the algorithm is estimating the support of h.

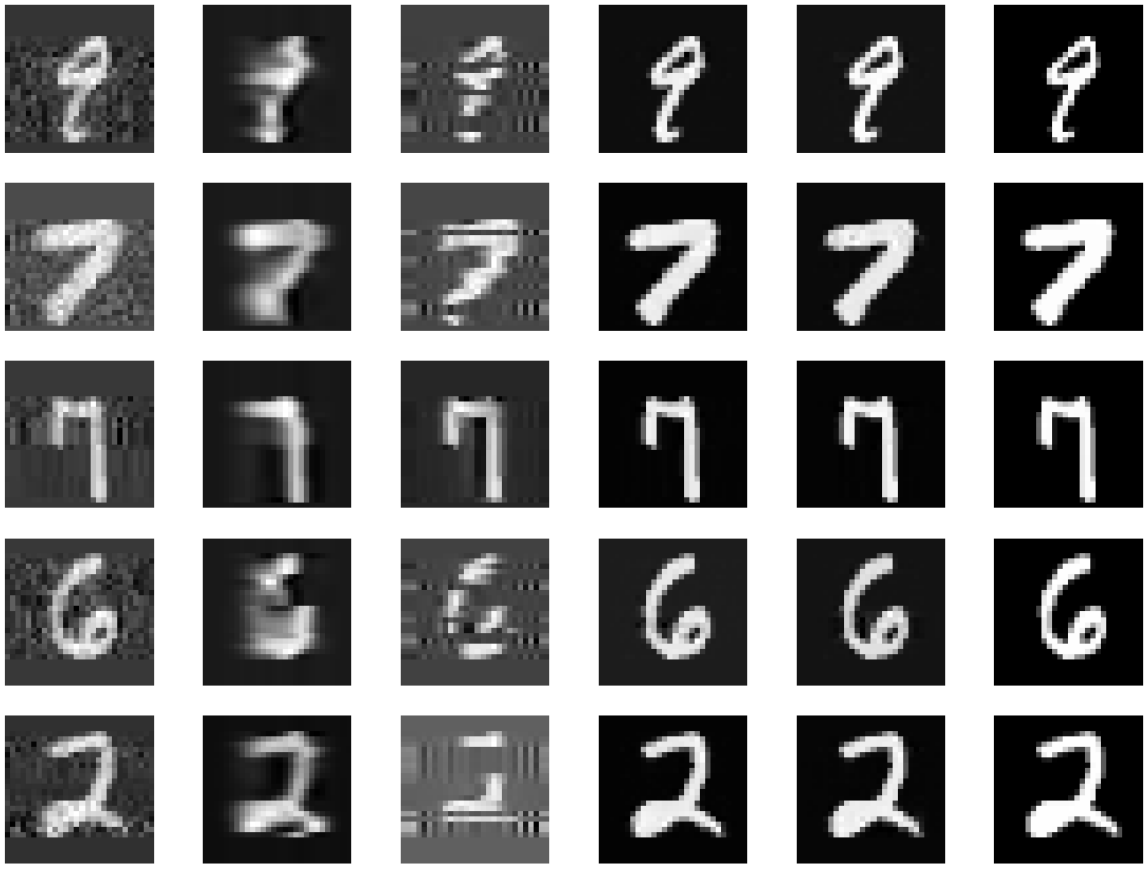
While ISTA solves the convex LASSO problem, it does introduce bias into the solution. Specifically, since ISTA shrinks every element of a vector, the signal power can be underestimated, especially in low noise situations.

Since iterative hard-thresholding selects the top K coefficients, a single layer of IHT is similar to one iteration of the OBLIVIOUS algorithm from [10] that selects the dictionary elements with the highest similarity to the input signal. As is the case with deriving the LUMP algorithm, the main challenge is finding a differentiable approximation to the argmax_K function.



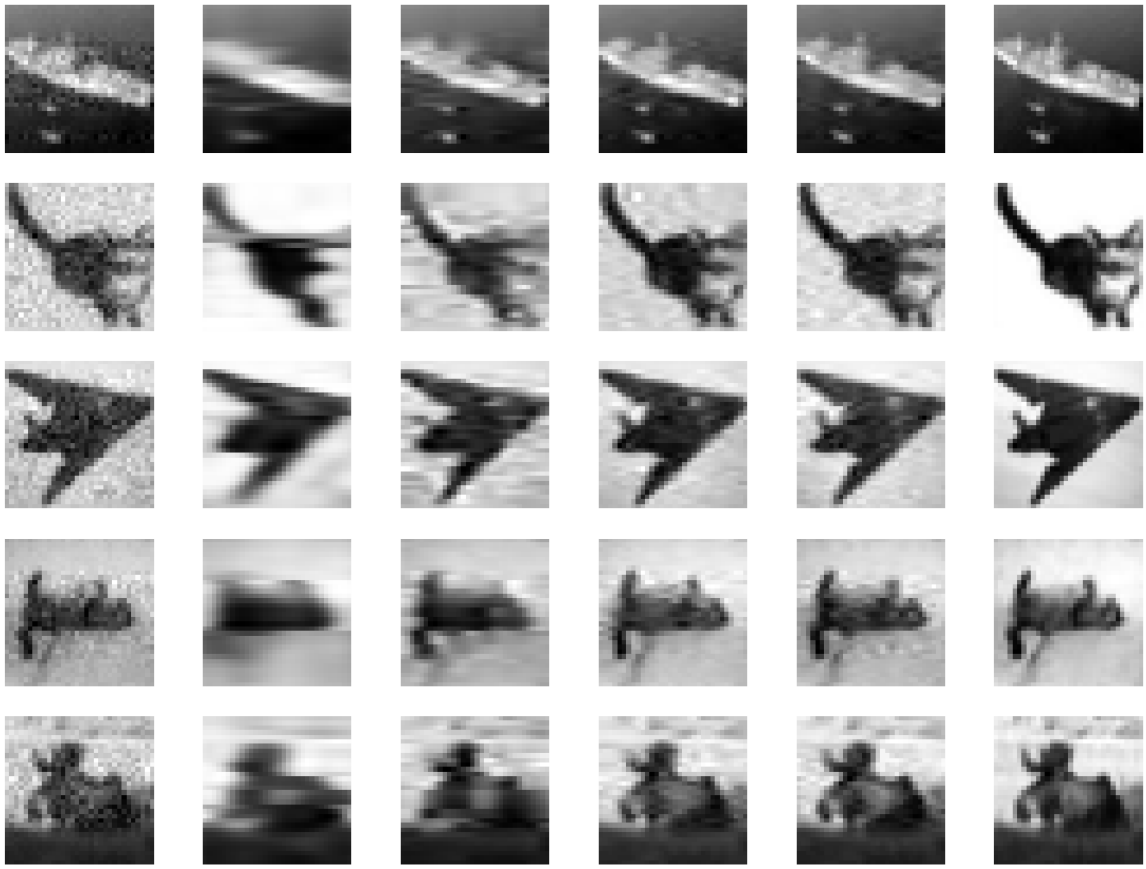
(a) (b) (c) (d)

Figure 5.3: MNIST autoencoder outputs. The images reconstructed by LUMP in (c) are nearly identical to the reference images in (d), sharper than those from the FFAE baseline in (b), and significantly improve over the initial outputs (a) during training. (d) Reference images. (c) Reconstructed images from LUMP after training. (a) Reconstructed images from LUMP before training. (b) Reconstructed images from FFAE.



(a) (b) (c) (d) (e) (f)

Figure 5.4: Row-wise MNIST autoencoder outputs. The images reconstructed by LUMP and LOMP in (d) are nearly identical to the reference images in (f), sharper than those from the FFAE baseline in (b), and significantly improve over the initial outputs (a) and (c) during training. (a) Reconstructed images from OMP. (b) Reconstructed images from FFAE. (c) Reconstructed images from LISTA after training. (d) Reconstructed images from LUMP after training. (e) Reconstructed images from LOMP after training. (f) Reference images.



(a) (b) (c) (d) (e) (f)

Figure 5.5: Row-wise CIFAR10 autoencoder outputs. (a) Reconstructed images from OMP. (b) Reconstructed images from FFAE. (c) Reconstructed images from LISTA after training. (d) Reconstructed images from LUMP after training. (e) Reconstructed images from LOMP after training. (f) Reference images.

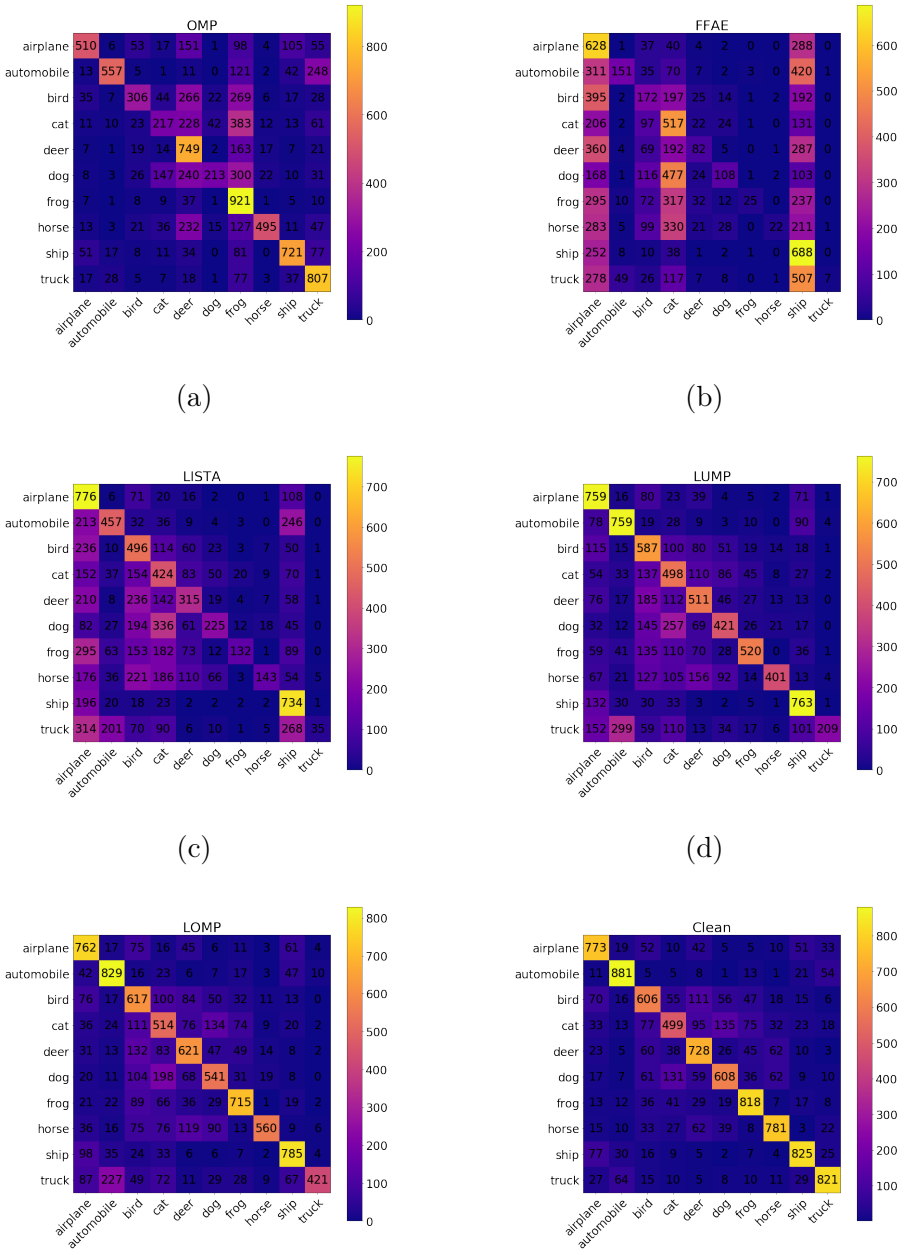


Figure 5.6: CIFAR10 classification confusion matrices. (a) Confusion matrix for OMP. (b) Confusion matrix for FFAE. (c) Confusion matrix for LISTA. (d) Confusion matrix for LUMP. (e) Confusion matrix for LOMP. (f) Confusion matrix for clean reference images.

BIBLIOGRAPHY

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary Evolution Recurrent Neural Networks. In *International Conference on Machine Learning (ICML)*, June 2016.

[3] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] T. Blumensath, M. Yaghoobi, and M. E. Davies. Iterative hard thresholding and l0 regularisation. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 3, pages III–877. IEEE, 2007.

[5] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[6] M. Brookes. VOICEBOX: Speech processing toolbox for MATLAB, 2002. [Online]. Available: http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html.

[7] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[8] V. Cevher and A. Krause. Greedy Dictionary Selection for Sparse Representation. *IEEE Journal of Selected Topics in Signal Processing*, 5(5):979–988, Sept. 2011.

[9] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[10] A. Das and D. Kempe. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. *arXiv:1102.3975 [cs, stat]*, Feb. 2011. arXiv: 1102.3975.

[11] B. W. Dolhansky and J. A. Bilmes. Deep submodular functions: Definitions and learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3404–3412. Curran Associates, Inc., 2016.

[12] J. Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, 1971.

[13] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–451, 2004.

[14] M. Fisher, G. Nemhauser, and L. Wolsey. An analysis of approximations for maximizing submodular set functions-ii. In M. Balinski and A. Hoffman, editors, *Polyhedral Combinatorics*, volume 8 of *Mathematical Programming Studies*, pages 73–87. Springer Berlin Heidelberg, 1978.

[15] S. Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

[16] T. Fujito. Approximation algorithms for submodular set cover with applications. *IEICE Transactions on Information and Systems*, 83(3):480–487, 2000.

[17] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[18] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. DARPA TIMIT acoustic-phonetic continous speech corpus. Technical Report NISTIR 4930, National Institute of Standards and Technology, 1993.

[19] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[20] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proc. ICML*, pages 399–406, Haifa, Israel, 2010.

[21] G. Griffin, A. Holub, and P. Perona. Caltech-256 Object Category Dataset, Mar. 2007.

[22] A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar. *Constrained Non-monotone Submodular Maximization: Offline and Secretary Algorithms*, pages 246–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[23] A. K. Halberstadt. *Heterogeneous acoustic measurements and multiple classifiers for speech recognition.* PhD thesis, Massachusetts Institute of Technology, 1998.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] J. R. Hershey, J. Le Roux, and F. Weninger. Deep Unfolding: Model-Based Inspiration of Novel Deep Architectures. *arXiv:1409.2574*, Sept. 2014.

[26] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, 5(4):339–342, 1958.

[28] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[29] ITU-T P.862. Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs, 2000.

[30] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[31] K. Kawaguchi. Deep learning without poor local minima. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 586–594. Curran Associates, Inc., 2016.

[32] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, Dec. 2014.

[33] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[34] A. Krause. SFO: A toolbox for submodular function optimization. *J. Mach. Learn. Res.*, 11:1141–1144, Mar. 2010.

[35] A. Krause, B. McMahan, G. C., and G. A. Robust submodular observation selection. *Journal of Machine Learning Research (JMLR)*, 9:2761–2801, December 2008.

[36] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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

[38] D. Krout and T. Powers. Sensor management for multistatics. In *Information Fusion (FUSION), 2014 17th International Conference on*, pages 1–6, July 2014.

[39] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv:1504.00941*, Apr. 2015.

[40] J. Le Roux, F. J. Weninger, and J. R. Hershey. Sparse NMF -– half-baked or well done? *MERL Technical Report*, 2015.

[41] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[42] J. Lee, M. Sviridenko, and J. Vondrák. *Submodular Maximization over Multiple Matroids via Generalized Exchange Properties*, pages 244–257. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[43] P. C. Loizou. *Speech Enhancement: Theory and Practice*. CRC Press, Boca Raton, FL, June 2007.

[44] L. Lovász. Submodular functions and convexity. In A. Bachem, B. Korte, and M. Grötschel, editors, *Mathematical Programming The State of the Art*, pages 235–257. Springer Berlin Heidelberg, 1983.

[45] S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Trans. Sig. Proc.*, 41(12):3397–3415, Dec. 1993.

[46] A. Mensch and M. Blondel. Differentiable dynamic programming for structured prediction and attention. *arXiv preprint arXiv:1802.03676*, 2018.

[47] M. Minoux. *Accelerated greedy algorithms for maximizing submodular set functions*, pages 234–243. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[48] B. Mirzasoleiman, A. Badanidiyuru, and A. Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *ICLM'16: Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[49] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[50] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, 14(1):265–294, Dec. 1978.

[51] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*, pages 40–44. IEEE, 1993.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] T. Powers. Constrained robust submodular sensor selection with application to multistatic sonar arrays. Master's thesis, University of Washington, December 2016.

[54] T. Powers, J. Bilmes, D. Krout, and L. Atlas. Constrained robust submodular sensor selection with applications to multistatic sonar arrays. In *2016 19th International Conference on Information Fusion (FUSION)*, pages 2179–2185, July 2016.

[55] T. Powers, J. Bilmes, S. Wisdom, D. Krout, and L. Atlas. Constrained robust submodular optimization. In *NIPS Workshop on Optimization for Machine Learning*, December 2016.

[56] T. Powers, R. Fakoor, S. Shakeri, A. Sethy, A. Kainth, A.-r. Mohamed, and R. Sarikaya. Differentiable greedy networks. *arXiv preprint arXiv:1810.12464*, 2018.

[57] T. Powers, D. Krout, and L. Atlas. Sensor selection from independence graphs using submodularity. In *Information Fusion (Fusion), 2015 18th International Conference on*, pages 333–337, July 2015.

[58] T. Powers, D. W. Krout, J. Bilmes, and L. Atlas. Constrained robust submodular sensor selection with application to multistatic sonar arrays. *IET Radar, Sonar & Navigation*, 11:1776–1781(5), December 2017.

[59] T. Powers, S. Wisdom, J. Hershey, J. Le Roux, and L. Atlas. Full-Capacity Unitary Recurrent Neural Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4880–4888. Curran Associates, Inc., 2016.

[60] A. Rix, J. Beerends, M. Hollier, and A. Hekstra. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In *Proc. ICASSP*, volume 2, pages 749–752, 2001.

[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[62] C. Taal, R. Hendriks, R. Heusdens, and J. Jensen. An algorithm for intelligibility prediction of time-frequency weighted noisy speech. *IEEE Trans. on Audio, Speech, and Language Processing*, 19(7):2125–2136, Sept. 2011.

[63] H. D. Tagare. Notes on optimization on Stiefel manifolds. Technical report, Yale University, 2011.

[64] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv: 1605.02688*, May 2016.

[65] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[66] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[67] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude, 2012. COURSERA: Neural Networks for Machine Learning.

[68] S. Tschiatschek, A. Sahin, and A. Krause. Differentiable submodular maximization. *arXiv preprint arXiv:1803.01785*, 2018.

[69] E. Vincent. BSS Eval toolbox version 3.0. `http://bass-db.gforge.inria.fr/bss_eval`.

[70] E. Vincent, J. Barker, S. Watanabe, J. Le Roux, F. Nesta, and M. Matassoni. The second 'CHiME' speech separation and recognition challenge: An overview of challenge systems and outcomes. In *Proc. ASRU*, pages 162–167, Olomouc, Czech Republic, 2013.

[71] E. Vincent, R. Gribonval, and C. Fevotte. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462 –1469, July 2006.

[72] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 67–74, New York, NY, USA, 2008. ACM.

[73] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[74] S. Wisdom, T. Powers, J. Pitton, and L. Atlas. Interpretable Recurrent Neural Networks Using Sequential Sparse Recovery. In *NIPS Workshop on Interpretable Machine Learning for Complex Systems, arXiv:1611.07252*, Dec. 2016.

[75] S. Wisdom, T. Powers, J. Pitton, and L. Atlas. Building Recurrent Networks by Unfolding Iterative Thresholding for Sequential Sparse Recovery. In *Proc. ICASSP*, New Orleans, LA, USA, Mar. 2017.

[76] S. Wisdom, T. Powers, J. Pitton, and L. Atlas. Deep recurrent nmf for speech separation by unfolding iterative thresholding. In *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 254–258. IEEE, 2017.

VITA

Thomas Powers grew up in Reno, Nevada. He attended Washington University in St. Louis and earned a B.S. in Electrical Engineering and a B.S.A.S. in Systems Science and Engineering with a Second Major in Computer Science in May 2013. He earned his M.S. in Electrical Engineering at the University of Washington in December 2016, for which his thesis is titled "Constrained Robust Submodular Sensor Selection with Applications to Multistatic Sonar Arrays." He participated in the Jelinek Workshop on Speech and Language with the Robust ASR team in the Summer of 2015. He completed internships at Mitsubishi Electric Research Labs (MERL) with the speech group in the summer of 2017 and with the Alexa Brain group at Amazon in the Summer of 2018. After graduating, he will rejoin the Alexa Brain group full-time as an applied scientist.