Controlling Complexity by Sharing Parameters and Minimizing Variation

Philip Bachman

Doctor of Philosophy

School of Computer Science

McGill University

Montreal,Quebec

2015-15-08

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

DEDICATION

This thesis is dedicated to my patient parents.

ACKNOWLEDGEMENTS

First and foremost, I thank my advisor Doina Precup. She's been continually supportive of my efforts, and a patient sounding board for my often rambling thoughts. Along with Doina, I want to thank the kind members of McGill's administrative staff, who have saved me more than once from various bureaucratic pitfalls, often of my own making. Additional thanks go to past and present members of the RL lab who have kept me sane over my long tenure at McGill. Final thanks go to my family who, I hope, will enjoy this ultimate graduation.

ABSTRACT

One common way of describing the tasks addressable by machine learning is to break them down into three categories: supervised learning problems, unsupervised learning problems, and reinforcement learning problems. At least two key steps are required for solving problems in any of these categories: one must construct a class of models broad enough to contain a reasonable solution to the problem at hand, and one must design a search process capable of finding a reasonable solution based on some available data. In realistic settings, a proper balance must be struck between the representational capacity of the model class through which one will search for a solution, and the amount of data available to guide the search. In particular, one must be careful to avoid *overfitting*, i.e. picking a model that performs well on the data available during training, but whose performance degrades significantly when applied to new data. Attempts to mitigate overfitting are typically referred to as regularization. Broadly speaking, this thesis concerns the development of new methods for regularization. We consider regularization in the context of supervised learning, unsupervised learning, and semi-supervised learning – which combines supervised and unsupervised learning. We present five original contributions related to this theme. First, we develop a new method for structure estimation in time-varying graphical models. We regularize the associated temporally-localized estimators by restricting them to a simplified space (learned simultaneously), in which variation among the estimated structures can be captured by only a few parameters. Next, we combine Monte

Carlo integration with gradient approximation via finite differences to construct regularizers that control the first and higher-order derivatives of a broad class of linearly-parametrized functions. Third, we develop a general approach to controlling the robustness of a function approximator subject to uncertainty in both its inputs and its internal structure. This leads to a family of regularizers which encompasses the popular dropout method for training deep architectures. While these regularizers can successfully leverage unstructured input perturbations, their performance would almost certainly increase with perturbations shaped by the input distribution. Hence, the last two contributions were motivated by a desire to learn data-adapted perturbation models. This evolved into a study of methods for regularizing the complexity of distributions learned by a generative model. We develop an approach to training generative models based on unrolling a variational auto-encoder into a Markov chain, and shaping the chain's trajectories using a technique inspired by recent work in Approximate Bayesian computation. Motivated by some difficulties encountered in these efforts, we took a more general look at the relationship between data generation and sequential decision making. We present a general formulation of data generation as sequential decision making. This perspective provides a unifying view of some existing algorithms and opens up the possibility of designing new algorithms based on ideas from reinforcement learning. We illustrate these benefits by developing new algorithms for data imputation based on guided policy search. This thesis contributes both new points-of-view that bring together existing ideas, and algorithms for use with actual data.

ABRÉGÉ

Les tâches d'apprentissage automatique sont communément séparées en trois catégories de problèmes: l'apprentissage supervisé, l'apprentissage non-supervisé et l'apprentissage par renforcement. La résolution d'un problème faisant partie de l'une de ces catégories comporte au moins deux étapes clés: il faut premièrement construire une classe de modèles assez large pour contenir une solution raisonnable au problème posé, et ensuite concevoir un processus de recherche capable de trouver une solution raisonnable basée sur les données disponibles. Un certain équilibre doit être atteint entre la capacité de représentation de la classe de modèles avec laquelle on cherchera une solution et la quantité de données disponibles qui guidera la recherche. Il est particulièrement important d'éviter le surapprentissage, c'est-à-dire le choix d'un modèle qui accomplit bien sa tâche sur l'ensemble d'apprentissage, mais dont la performance se dégrade de façon significative sur de nouvelles données. Les tentatives d'atténuer le surapprentissage sont habituellement appelées régularisation. Le sujet sur lequel se penche cette thèse est le développement de nouvelles méthodes de régularisation. Nous considérons la régularisation dans les contextes d'apprentissage supervisé, non-supervisé et semi-supervisé (ce dernier est une combinaison des deux premiers). Nous présentons cinq contributions originales reliées à ce thème. La première est le développement d'une nouvelle méthode d'estimation de structure dans les modèles graphiques variable dans le temps. En effet, nous régularisons les estimateurs localisés dans le temps qui leur sont associés en les réstraignant à un espace simplifié (appris

simultanément), dans lequel la variation entre les structures estimées peut être capturée seulement par quelques paramètres. Nous combinons ensuite l'intégration Monte Carlo avec l'approximation de gradient via des différences finies afin de construire des régularisateurs qui contrôlent les dérivés du premier ordre et de l'ordre supérieur d'une large classe de fonctions linéairement paramétrées. Nous développons troisièmement une approche générale qui contrôle la robustesse d'un approximateur de fonction sujet à l'incertitude autant dans ses entrées que dans sa structure interne. Ceci nous mène à une famille de régularisateurs qui englobent la populaire méthode de "dropout" pour l'apprentissage d'architectures profondes. Tandis que ces régularisateurs peuvent tirer partie des perturbations d'entrée non structurées avec succès, leur performance s'en trouverait aussi presque certainement augmentée lors de perturbations façonnées par la distribution d'entrée. Ces deux dernières contributions sont donc motivées par un désir d'apprendre des modèles de perturbation adaptés aux données. Ceci a par la suite progressé vers une étude sur les méthodes qui régularisent la complexité des distribution apprises par un modèle génératif. Nous développons une approche d'apprentissage de modèles génératifs basée sur le déroulement d'un auto-encodeur variationnel en une chaîne de Markov, celle-ci ayant sa trajectoire ensuite modifiée en utilisant une technique inspirée par de récents travaux dans le calcul bayésien approximatif. Motivés par des difficultés rencontrées lors de ces travaux, nous jetons un regard plus général à la relation entre la génération de données et la prise de décisions séquentielle. Nous présentons donc une formulation générale de génération de données en tant que prises de décisions séquentielles. Cette perspective nous

offre une vision unifiée de certains algorithmes existants et nous permet d'élargir les possibilités de concevoir de nouveaux algorithmes basés sur des concepts d'apprentissage par renforcement. Nous démontrons ces bénéfices en développant de nouveaux algorithmes pour l'entrée de données basée sur une recherche de politique guidée. Cette thèse met donc de l'avant à la fois de nouveaux points de vue rassemblant des concepts déjà existant ainsi que des algorithmes pouvant être utilisés avec de vraies données.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

ABSTRACT iv

ABRÉGÉ vi

LIST OF TABLES xiii

LIST OF FIGURES xiv

1 General Introduction 1

1.1 Overview of Thesis Contributions 3
1.2 Statement of Authorship 9

2 General Background 12

2.1 Supervised Learning 12
2.1.1 Regression 13
2.1.2 Regularization 14
2.1.3 Classification 16
2.1.4 Sophisticated Hypotheses 17
2.2 Unsupervised Learning 18
2.2.1 Dimension Reduction 18
2.2.2 Encoder-Decoder Methods 20
2.2.3 Generative Modelling 22
2.3 Semi-supervised Learning 26

3 Improved Estimation in Time-Varying Models 28

3.1 Motivation 28
3.2 Learning Transformed Representations of Model Parameters . . . 31
3.3 Examples of Transformed Multiple Parametrization 34

3.3.1 Sparse Coding . . . 36
3.3.2 PCA . . . 37
3.4 Learning Transformed Parametrizations of Network Structure . . . 38
3.4.1 Sparse Network Structure Estimation . . . 39
3.4.2 Using Basis Structures to Estimate Network Structures . . . 42
3.4.3 Supervised Basis Structure Learning . . . 46
3.5 Synthetic Network Analysis . . . 48
3.6 BCI EEG Analysis . . . 53
3.7 Discussion . . . 57

4 Sample-based Approximate Regularization . . . 59

4.1 Motivation . . . 59
4.2 The SAR Approach . . . 62
4.3 Smoothness-inducing Regularizers . . . 63
4.4 The SAR Method . . . 64
4.4.1 Approximating Directional Derivatives . . . 66
4.4.2 Sampling from ν and s_x . . . 67
4.4.3 Relationship with Manifold Regularization . . . 69
4.5 Theoretical Analysis . . . 70
4.6 Experiments . . . 78
4.6.1 Synthetic Data . . . 78
4.6.2 Natural Data . . . 82
4.7 Discussion . . . 86

5 Learning with Pseudo-Ensembles . . . 88

5.1 Motivation . . . 88
5.2 What is a Pseudo-Ensemble? . . . 90
5.3 Related Work . . . 93
5.4 The Pseudo-Ensemble Agreement Regularizer . . . 97
5.4.1 The Effect of PEA Regularization on Feature Co-adaptation 99
5.4.2 Relating PEA Regularization to Standard Dropout . . . 100
5.4.3 PEA Regularization for Semi-supervised Learning . . . 102
5.5 Testing PEA Regularization . . . 103
5.5.1 Fully-supervised MNIST . . . 103
5.5.2 Semi-supervised MNIST . . . 104
5.5.3 Transfer Learning Challenge . . . 107
5.6 Improved Sentiment Analysis using Pseudo-Ensembles . . . 109

5.7 Discussion . . . 112

6 Variational Generative Stochastic Networks with Collaborative Shaping . 115

6.1 Motivation . . . 115
6.2 Background . . . 118
6.2.1 Generalized Denoising Auto-encoders . . . 118
6.2.2 Training with Walkback . . . 121
6.3 Simple Generative Stochastic Networks . . . 123
6.4 Variational Simple GSNs . . . 126
6.4.1 Explaining the Variational Free Energy . . . 127
6.4.2 Using a Modified Variational Free Energy . . . 130
6.4.3 Relating Posterior KL to Mutual Information . . . 131
6.5 Collaborative Generative Networks . . . 132
6.6 Generating Random Walks on Manifolds . . . 135
6.7 Experiments . . . 137
6.8 Discussion . . . 145

7 Data Generation as Sequential Decision Making . . . 147

7.1 Motivation . . . 147
7.2 Directed Generative Models as Sequential Decision Processes . . . 151
7.2.1 Deep AutoRegressive Networks . . . 151
7.2.2 Generalized Guided Policy Search . . . 152
7.2.3 Time-reversible Stochastic Processes . . . 154
7.2.4 A Path-wise KL Bound . . . 156
7.2.5 Learning Generative Stochastic Processes with LSTMs . . . 158
7.2.6 Extending the LSTM-based Model . . . 160
7.3 Developing Models for Sequential Imputation . . . 163
7.3.1 A Direct Representation for Sequential Imputation Policies 165
7.3.2 Representing Sequential Imputation Policies using LSTMs . 166
7.4 Experiments . . . 167
7.4.1 Problems with VAE Imputation . . . 173
7.4.2 Additional Qualitative Results for GPSI Models . . . 174
7.5 Discussion . . . 174

8 General Discussion . . . 178

Appendix 185

8.1 A unified log-likelihood bound for generative models 185
8.2 Deriving the unified log-likelihood bound 186
8.3 Bounding expected log-likelihood for $x \sim \mathcal{D}_x$ 186
8.4 Assimilating denoising auto-encoders and GSNs 187
8.5 Training RBMs using a path-wise KL bound 190
8.6 Note to Readers 194

References 195

LIST OF TABLES

Table page

3–1 Comparing classification performance of different features on BCI EEG data 56

5–1 Measuring quantitative performance of PEA regularization on semi-supervised MNIST 107

5–2 Measuring performance of perturbation regularization in a sentiment analysis task 112

7–1 Quantitative performance of sequential imputation models on MNIST and TFD 171

LIST OF FIGURES

Figure page

3–1 Principal Parameter Components – an example 35

3–2 Weight trajectories for synthetic time-varying structure estimation tests 49

3–3 Measuring basis structure recovery with sythetic data 49

3–4 Time-varying structure features – classification performance on synthetic data 51

3–5 Time-varying structure features – classification performance on BCI EEG data 54

3–6 Examining archetypal network structures learned for two experimental subjects 55

4–1 Joint input/output distribution for SynthSin1d data 78

4–2 Comparing performance of different regularizers on SynthSin2d data . 80

4–3 Comparing performance of different methods on Boston Housing data 83

4–4 Empirical results in support of SAR theory 85

5–1 How to compute perturbed network activations for PEA regularization 96

5–2 Examining qualitative effects of PEA regularization 105

5–3 Feedforward computation in the Recursive Neural Tensor Network . . 109

6–1 Computation graphs for Generalized DAEs, GSNs, and Simple GSNs 120

6–2 Properties of local and non-local corruption processes 122

6–3 Computation graph for the unrolled, self-looped VAE guided by collaborative shaping 136

6–4 Comparing Markov Chain dynamics with and without collaborative shaping 137

6–5 Illustrating quantitative and qualitative performance on MNIST . . . 140

6–6 Illustrating quantitative and qualitative performance on TFD 141

7–1 Computation graphs for DRAW model and our augmented DRAW model 161

7–2 Visualizing LSTM-based sequential generation 163

7–3 Computation graphs for "direct" and LSTM-based sequential imputation models 168

7–4 Comparing quantitative performance of sequential imputation models on MNIST 169

7–5 Visualizing imputation policies learned by our models 172

7–6 Visualizing imputation policies learned for MNIST 175

7–7 Visualizing imputation policies learned for SVHN 176

7–8 Visualizing imputation policies learned for TFD 177

CHAPTER 1
General Introduction

One could say that machine learning – as a field of study – is the art of designing, reasoning about, and practically effecting systems for converting data into decisions. The tasks that can be tackled by such systems are often sorted into three categories: supervised learning, unsupervised learning, and reinforcement learning. We provide a summary of these categories in Chapter 2. Though they are often considered separate, the boundaries between these categories can dissolve in many interesting and practically relevant settings.

Two key subproblems consistently arise when applying machine learning to any of the categories mentioned above. First, one must construct a hypothesis space broad enough to contain a reasonable solution to the problem at hand. Second, one must design a search process capable of finding one of these reasonable solutions by interacting with some available data.[1] When the hypothesis space selected for a particular problem does not contain a reasonable solution, it will be impossible to find a reasonable solution, no matter how effective the search

[1] By "hypothesis space" we mean, e.g., the particular model one will use to try and solve a problem. Individual hypotheses in the hypothesis space correspond to assigning particular values to the model's parameters. I.e. each hypothesis corresponds to a fully-specified system for turning data into decisions.

process might be. When the selected hypothesis space is a good fit to the problem at hand, the search process might still fail if it cannot convert interaction with the available data into progress towards finding a reasonable solution.

Thus, in realistic settings, one must strike a balance between the representational capacity of the hypothesis space to search through, the power of the search process, and the amount of data available to guide the search. By selecting a hypothesis space with sufficiently high representational capacity, one can guarantee that it (almost) surely contains a good solution to the problem at hand. But, increasing the size of the hypothesis space places additional burdens on the search process. When the hypothesis space is large, there may exist many hypotheses which seem reasonable in light of the limited available data. To complicate matters, the available data may contain noise that the search process can mistake for signal.[2] In either case – i.e. when dealing with underdetermination due to limited data or when being misled by noise – one must be careful to avoid *overfitting*. Overfitting happens when the search process picks a hypothesis that performs well with the available data, but whose performance degrades significantly when applied to new data. Search processes will be prone to overfitting when operating in large hypothesis spaces guided by limited amounts of noisy data. These challenging conditions are ubiquitous in practical settings.

[2] Noise can be defined in a problem-dependent way as variation which is *uninformative* w.r.t. the decisions required for solving the problem. Signal refers to variation which is *informative* w.r.t. the decisions required for solving the problem.

Attempts to mitigate overfitting are often described as regularization. In a broad sense, regularization refers to the purposeful injection of biases into the hypothesis search process. The intent of these biases is to reduce the amount of data required for the search process to distinguish between good and bad hypotheses, and to reduce the tendency for the search process to be misled by noise or other forms of uncertainty.

When designing a regularizer (i.e. a bias to inject into the search process), one might consider two approaches: driving the search *towards* good hypotheses and driving the search *away from* bad hypotheses. Though these approaches overlap, most regularization techniques can be interpreted more easily in terms of one or the other. For example, a good hypothesis for a typical image classification problem should be invariant to small translations. We can encourage this behavior by using a model architecture that combines multiple stages of convolution and pooling, e.g. a convolutional neural network. Conversely, a bad hypothesis for a typical regression problem might oscillate wildly over the input domain. We can discourage this behavior by biasing search away from hypotheses that encode stronger dependencies on the input than can be supported by the available data.

1.1 Overview of Thesis Contributions

For the most part, this thesis concerns the development of new methods for regularization. It presents five main contributions. The first four contributions consider regularization in the context of supervised learning, unsupervised learning, and semi-supervised learning – which combines supervised and unsupervised learning. The final part of the thesis makes new connections between learning in

directed generative models – which can be used for supervised or unsupervised learning – and reinforcement learning.

The thesis is structured as follows. In Chapter 2 we present general background information required throughout the thesis. Chapter 3 develops a new method for structure estimation in time varying graphical models. This problem is challenging because network structure estimation involves many parameters – i.e. $\mathcal{O}(n^2)$ parameters for n-dimensional observations – and tracking temporal variation may require reducing the number of sample observations that are used directly in each structure estimation. We collectively regularize the multiple temporally-localized structure estimations by restricting them all to a transformed, simpler parameter space. By performing estimation in the transformed space, we can represent variation among the estimated structures using only a few parameters, even though the structures themselves are defined by many parameters. By shaping the transformed parameter space, we can promote information transfer among the temporally-localized structure estimations. Our algorithm mixes locally-weighted regression, sparse coding, and regression-based sparse network structure estimation. Interestingly, the idea of controlling capacity by forcing parameter sharing among multiple instances of a base model provides a direct conceptual link between the methods in Chapter 3 and the deep models investigated in Chapters 5-7.

Chapter 4 combines Monte Carlo integration with gradient approximation via finite differences to construct regularizers that control the first and higher-order

derivatives of a broad class of linearly-parametrized functions. The general motivation for this work was to construct regularizers providing some of the desirable effects of kernel-based regularization and methods based on smoothing splines, i.e., controlling the observation-space derivatives of a function approximator. Loosely speaking, our regularizers encode a bias towards choosing hypotheses that are less wiggly and wobbly. The benefit of our regularizers, as opposed to standard Reproducing Kernel Hilbert Space (abbr. RKHS) regularization, is that they can be used with (almost) arbitrary sets of features. In contrast, RKHS regularization requires the use of a fixed, positive semi-definite kernel. Through empirical tests, we show that the added flexibility of our regularizers offers advantages over powerful RKHS regularization, and that the derivative-controlling properties of our regularizers offer advantages over widely applicable generic regularizers like classic ℓ_2 regularization.

Chapter 5 develops a general approach to controlling the robustness of a function approximator subject to uncertainty in the observations used to train it, and uncertainty in the model structure itself. This work was motivated by the wide-spread empirical success of dropout [44]. Previous efforts at interpreting dropout mainly focused on showing its equivalence with some form of re-weighted ℓ_2 regularization on the model parameters. But, dropout acts by directly modifying a model's behavior, so it should be easier and more informative to reason about dropout directly in these terms, without reference to model parameters. Secondly, if one allows arbitrarily complex re-weighting of ℓ_2 regularization terms on a per-parameter basis, then one can recapitulate the effects of any modification

of the training objective as a form of "re-weighted ℓ_2 regularization", which is not very informative.

Starting from this motivation, we interpret dropout as regularization which acts by controlling the distributional properties of a "pseudo-ensemble" derived from some base model by subjecting it to some perturbation process. We then develop a "pseudo-ensemble agreement" regularizer, which encourages the output of a model to remain self-consistent when it is subjected to a perturbation process. Under some assumptions, we show that encouraging (a certain form of) self-consistency is equivalent to dropout. Because our regularizer does not depend on supervisory information, we can apply it in the semi-supervised setting. The tests we present in Sec. 5.5 show that regularizing for pseudo-ensemble agreement can produce state-of-the-art semi-supervised learning performance on some standard image classification benchmarks. In another set of tests, in Sec. 5.6, we show that the pseudo-ensemble approach can produce state-of-the-art results on a standard sentiment analysis benchmark.

Chapter 6 was initially motivated by a desire to learn data-dependent perturbation models suited to generating perturbed data for use in our "pseudo-ensemble agreement" regularization framework. This work turned into a study of methods for regularizing the complexity of distributions learned by a generative model, and for shaping the dynamics of particular sorts of Markov chains. Specifically, we developed an approach to training the transition operator of a Markov chain whose asymptotic distribution is shaped to match some target distribution. We based our approach on previous work involving generalized denoising auto-encoders

[10], Generative Stochastic Networks [9], variational auto-encoders [51, 80], and classification-based approaches to Approximate Bayesian Computation [38, 39, 32]. While developing our approach, we encountered surprising behavior from variational auto-encoders trained for use with continuous data. In particular, we noticed that the "prior term" in the variational free-energy optimized by the auto-encoder can sometimes be overwhelmed by the "reconstruction term".[3] To ameliorate this issue, we proposed treating the "prior term" in the free-energy explicitly as a means of balancing between reconstruction fidelity and complexity of the distribution represented by the variational auto-encoder. In empirical tests, this regularization provided significantly improved results on a standard benchmark for generative models.

While working on these ideas, it became clear that mapping from a structurally trivial distribution – e.g. a diagonal Gaussian distribution – to a structurally complex distribution – e.g. a natural image distribution – is prone to failure when forced into a single step. Thus, we decided to investigate the use of "multi-stage models", in which the process of generating an output involves multiple steps of refinement. This approach would allow us to first learn a reduced-complexity approximation of the target distribution and then learn a conditional mapping from points in that approximation to points in the target distribution. By decomposing the mapping from a simple distribution to a complex distribution into multiple steps, each conditional mapping from a distribution of lesser complexity

[3] See Chapter 6 for more precise, more complete statements of these points.

to a distribution of greater complexity might remain simple. This closely echoes many of the motivations for denoising auto-encoders and Generative Stochastic Networks [10, 9].

In initial experiments we trained models which generated an observation via iterative refinement. For some fixed number of steps these models iterated between looking at the current observation and sampling a change to the observation from a stochastic policy. The stochastic policy was trained such that its terminal state distribution – i.e. the distribution over observations it constructed via iterative refinement – would match some target distribution. While we initially performed iterative refinement directly in the observation space, it would also be reasonable to perform the refinement steps in a latent space.

We eventually focused our attention on exploring connections among a menagerie of models introduced over the last couple of years, and on developing formal analogies between the methods used to train them and methods developed for reinforcement learning. In addition to these efforts, we also developed models capable of representing powerful policies for sequential data imputation, that is, models that can represent policies for constructing hypotheses about the y in $p(y|x)$ via iterative refinement. When x contains no information about y this turns into a classic generative model for $p(y)$, but estimating $p(y|x)$ also describes almost all forms of prediction.

The empirical results of our efforts, in terms of sequential generation and imputation, are strong. In Chapter 7, we also present connections between sequential

data generation – i.e. iterative hypothesis refinement – and reinforcement learning. These connections should lead to further algorithm development.

1.2 Statement of Authorship

The work presented in this thesis comes from five previously published papers. We now provide authorship information for these papers.

Chapter 3 is based on:

> Philip Bachman and Doina Precup. "Improved Estimation in Time Varying Models". In *International Conference on Machine Learning (ICML)*, 2012. – [4]

The lead author – of the paper above and the thesis in hand – developed the problem formulation, developed the algorithm, developed the empirical tests, and wrote most of the paper. The second author provided advisory feedback w.r.t. the design and development of the project, and contributed directly to the writing and editing of the paper.

Chapter 4 is based on:

> Philip Bachman, Amir-massoud Farahmand, and Doina Precup. "Sample-based Approximate Regularization". In *International Conference on Machine Learning (ICML)*, 2014. – [3]

The lead author developed the problem formulation, developed the algorithm, developed the empirical tests, and wrote $\sim \frac{1}{2}$ of the paper. The second author developed the theoretical analysis, provided feedback on the experiment design, and contributed significantly to the general writing and editing of the paper. The third author provided advisory feedback w.r.t. the design and development of the project, and contributed directly to the writing and editing of the paper.

Chapter 5 is based on:

> Philip Bachman, Ouais Alsharif, and Doina Precup. "Learning with Pseudo-Ensembles". In *Advances in Neural Information Processing Systems (NIPS)*, 2014. – [2]

The lead author developed the problem formulation, developed the algorithm, designed and implemented most of the empirical tests, and wrote $\sim \frac{3}{4}$ of the paper. The second author designed, implemented, and wrote paper content for the empirical tests described in Sec. 5.5.3 of this thesis. The third author provided advisory feedback w.r.t. the design and development of the project, and contributed directly to the writing and editing of the paper.

Chapter 6 is based on:

> Philip Bachman and Doina Precup. "Variational Generative Stochastic Networks with Collaborative Shaping". In *International Conference on Machine Learning (ICML)*, 2015. – [6]

The lead author developed the conceptual content, developed the algorithms, developed the empirical tests, and wrote most of the paper. The second author provided advisory feedback w.r.t. the design and development of the project, and contributed directly to the writing and editing of the paper.

Chapter 7 is based on:

> Philip Bachman and Doina Precup. "Data Generation as Sequential Decision Making". In *Advances in Neural Information Processing Systems (NIPS)*, 2015. – [5]

The lead author developed the conceptual content, developed the algorithms, developed the empirical tests, and wrote most of the paper. The second author provided advisory feedback w.r.t. the design and development of the project, and contributed directly to the writing and editing of the paper.

CHAPTER 2
General Background

This chapter briefly reviews material helpful for understanding work presented in the rest of this thesis. It focuses on general problem formulations and basic techniques. Later chapters provide references to more specific material.

2.1 Supervised Learning

In supervised learning problems, the objective is to learn a function $f(x) : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes an expected cost:

$$\mathcal{L}(\mathcal{D}_{xy}, f) \triangleq \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \mathcal{L}(x, y, f) \tag{2.1}$$

for (input,output) pairs sampled from a distribution $\mathcal{D}_{xy}$, using a scalar cost $\mathcal{L}(x, y, f)$ that factorizes over the $(x, y) \sim \mathcal{D}_{xy}$. Typically, one is presented with a set of samples $(X, Y) \triangleq \{(x_1, y_1), ..., (x_n, y_n)\}$, where $(x_i, y_i) \sim \mathcal{D}_{xy}$. f is only allowed to interact with a subset (X_{tr}, Y_{tr}) of the $(x_i, y_i) \in (X, Y)$, and an empirical approximation of Eq. 2.1, given by:

$$\tilde{\mathcal{L}}(\mathcal{D}_{xy}, f) \triangleq \frac{1}{|X_{te}|} \sum_{(x,y)\in(X_{te},Y_{te})} \mathcal{L}(x, y, f) \tag{2.2}$$

is measured on a disjoint subset (X_{te}, Y_{te}).[1] Researchers have studied endless variants of this basic form. See, e.g., "Elements of Statistical Learning II" by Hastie, Tibshirani, and Friedman for numerous examples [41].

2.1.1 Regression

Regression is probably the oldest form of supervised learning – dating back to Laplace, Legendre, and Gauss circa 1800.[2] For least-squares regression, the objective is given by:

$$\begin{aligned} f^* &= \arg\min_{f \in \mathcal{F}} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \left[\mathcal{L}(x, y, f)\right] \\ &= \arg\min_{f \in \mathcal{F}} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \left[(y - f(x))^2\right], \end{aligned} \tag{2.3}$$

where $\mathcal{F}$ is a hypothesis space in which to search for f^*.

The standard approach to least-squares regression is to define a linear hypothesis space $\mathcal{F}_\phi : \mathcal{X} \to \mathcal{Y}$ such that each $f_\theta \in \mathcal{F}_\phi$ is given by $f_\theta(x) \triangleq \theta^\top \phi(x)$. We use the subscript ϕ to indicate dependence of $\mathcal{F}_\phi$ on a "feature transform" $\phi : \mathcal{X} \to \mathcal{X}_\phi$, and use the subscript θ to indicate dependence of each f_θ on a parameter vector θ. With these assumptions, the least-squares objective is given

[1] One subtlety worth mentioning is that $\tilde{\mathcal{L}}(\mathcal{D}_{xy}, f)$ may be computed while f interacts with the (x, y) pairs. In this setting, f may interact with pairs used to compute $\tilde{\mathcal{L}}$, but only after they have been used by $\tilde{\mathcal{L}}$. I.e. (x, y) pairs can go $(X_{te}, Y_{te}) \to (X_{tr}, Y_{tr})$, but not $(X_{tr}, Y_{tr}) \to (X_{te}, Y_{te})$.

[2] Laplace is credited with least absolute deviation regression, while Legendre and Gauss are credited with least-squares regression. Least absolute deviation regression penalizes $|y_i - f(x_i)|$ and least-squares penalizes $(y_i - f(x_i))^2$. The former is minimized by the median of $p(y|x)$, and the latter is minimized by the mean.

by:

$$\begin{aligned} f_\theta^* &= \arg\min_{f_\theta \in \mathcal{F}_\phi} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} [\mathcal{L}(x, y, f_\theta)] \\ &= \arg\min_{f_\theta \in \mathcal{F}_\phi} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \left[(y - \theta^\top \phi(x))^2\right]. \end{aligned} \tag{2.4}$$

Next, we define a fixed "training" set (X, Y) of (x, y) pairs, and define $\Phi \triangleq \phi(X)$ as the matrix whose rows are given by $\phi(x)$ for $x \in X$. Given the fixed set (X, Y), an empirical surrogate for the expected cost in Eq. 2.4 is:

$$\begin{aligned} \mathcal{L}(X, Y, f_\theta) &\triangleq \frac{1}{|X|} \sum_{(x,y)\in(X,Y)} (y - \theta^\top \phi(x))^2 \\ &= \frac{1}{|X|} ||Y - \Phi\theta||_2^2 \\ &= \frac{1}{|X|} (Y - \Phi\theta)^\top (Y - \Phi\theta) \\ &= \frac{1}{|X|} \left(\theta^\top \Phi^\top \Phi\theta - 2\theta^\top \Phi^\top Y + Y^\top Y\right). \end{aligned} \tag{2.5}$$

Now, taking the gradient of Eq. 2.5 and setting it to zero, we have:

$$\begin{aligned} -\frac{2}{|X|}\Phi^\top Y + \frac{2}{|X|}(\Phi^\top \Phi)\theta &= 0 \\ (\Phi^\top \Phi)\theta &= \Phi^\top Y \\ \theta^* &= (\Phi^\top \Phi)^{-1} \Phi^\top Y. \end{aligned} \tag{2.6}$$

So, for linear least-squares regression with features given by $\phi(x)$, we can find the parameters θ^* which minimize Eq. 2.4 by solving Eq. 2.6.

2.1.2 Regularization

The matrix inversion in Eq. 2.6 can be problematic. In addition to potentially large computational costs – i.e. $\sim \mathcal{O}(d^3)$ when $\phi(x) \in \mathbb{R}^d$ – the inversion can be

unstable when some eigenvalues of $\Phi^\top\Phi$ are close to 0. To combat this problem, one can extend the objective in Eq. 2.4 to have the following form:

$$f_\theta^* = \arg\min_{f_\theta \in \mathcal{F}_\phi} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \left[(y - \theta^\top\phi(x))^2\right] + \lambda||\theta||_2^2, \tag{2.7}$$

where λ modulates the *regularization* term $\lambda||\theta||_2^2$. Adding this regularization changes the analytical solution in Eq. 2.6 to:

$$\theta^* = (\Phi^\top\Phi + \lambda\mathbf{I})^{-1}\Phi^\top Y, \tag{2.8}$$

where $\mathbf{I}$ denotes an appropriately-sized identity matrix. The regularization term $\lambda||\theta||_2^2$ can be interpreted in several ways:

- It adds a "preconditioner" to $\Phi^\top\Phi$ to stabilize the required inversion – by pushing (small) eigenvalues of $\Phi^\top\Phi$ away from 0.
- It shrinks the hypothesis space $\mathcal{F}_\phi$ by constraining the ℓ_2 norm of θ.
- It corresponds to assuming an isotropic Gaussian prior with mean 0 and variance $\propto \frac{1}{\lambda}$ for θ, and then finding the Maximum A Posteriori solution.

As an alternative to the regularization in Eq. 2.7, one could try optimizing:

$$f_\theta^* = \arg\min_{f_\theta \in \mathcal{F}_\phi} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_{xy}} \left[(y - \theta^\top\phi(x))^2\right] + \lambda||\theta||_1, \tag{2.9}$$

where the regularization term $\lambda||\theta||_1$ penalizes the ℓ_1 norm of θ. This regularizer is called the Lasso [104]. Two useful interpretations of the Lasso are that:

- It shrinks the hypothesis space $\mathcal{F}_\phi$ by constraining the ℓ_1 norm of θ.
- It corresponds to assuming a Laplacian prior with mean 0 and variance $\propto \frac{1}{\sqrt{\lambda}}$ for θ, and then finding the Maximum A Posteriori solution.

The key difference between the regularizers $\lambda||\theta||_2^2$ and $\lambda||\theta||_1$ is that the Lasso produces optimal solutions θ^* in which many of the parameters are exactly 0. In contrast, constraining the ℓ_2 norm produces optimal parameter vectors with lower variance but, almost certainly, no exact 0s. We make heavy use of linear regression and ℓ_1/ℓ_2 regularizers throughout Chapters 3 and 4.

2.1.3 Classification

While regression is used primarily for predicting "numerical" values, many common supervised learning problems involve predicting "categorical" values.[3] Categorical prediction is known as classification. In classification problems the ys in the observed (x, y) pairs come from an unstructured set of disjoint, discrete outcomes. For simplicity, we only discuss binary classification, in which each y indicates one of two possible outcomes.

Logistic regression is the most common approach to binary classification [41]. For logistic regression, we assume that the y_i in $(X, Y) \triangleq \{(x_1, y_1), ..., (x_n, y_n)\}$ take values in $\{-1, +1\}$. If we assume a hypothesis space $\mathcal{F}_\phi$ defined as for Eq. 2.4, and consider an empirical surrogate for the true expected loss, we can write the logistic regression objective as:

$$f_\theta^* = \arg\min_{f_\theta \in \mathcal{F}_\phi} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i f_\theta(x_i)\right)\right) + \lambda||\theta||_2^2, \tag{2.10}$$

[3] Here "numerical" is intended primarily as a contrast to "categorical".

where we've added ℓ_2 regularization just to be safe. The objective in Eq. 2.10 is also often written in the form:

$$f_\theta^* = \arg\min_{f_\theta \in \mathcal{F}_\phi} \frac{1}{n} \sum_{i=1}^{n} -\log p_\theta(y = y_i | f_\theta(x_i)) + \lambda ||\theta||_2^2, \tag{2.11}$$

with class probabilities $p_\theta(y = -1|f_\theta(x_i))$ and $p_\theta(y = 1|f_\theta(x_i))$ given by:

$$\begin{aligned} p_\theta(y = -1|f_\theta(x_i)) &\triangleq \frac{1}{1 + \exp\left(\theta^\top \phi(x_i)\right)} \\ p_\theta(y = 1|f_\theta(x_i)) &\triangleq \frac{\exp\left(\theta^\top \phi(x_i)\right)}{1 + \exp\left(\theta^\top \phi(x_i)\right)}. \end{aligned}$$

The optimization in Eqs. 2.10/2.11 is agreeably convex, and efficiently solvable by iterative methods, but does not permit a direct analytical solution. We use various forms of logistic regression throughout Chapters 3-5.

2.1.4 Sophisticated Hypotheses

Linear least-squares and logistic regression are not sufficiently powerful for many practical applications. Though "feature engineering" can extend the applicability of linear methods, it is often difficult and time consuming to define good features for each task. Thus, much effort has been put into developing more sophisticated hypothesis spaces for use in supervised learning. We briefly describe kernel methods and deep learning, with further details provided in later chapters.

Kernel-based methods [88] use hypotheses $f_\theta(x_i) \triangleq \sum_{j=1}^m \theta_j \phi(x_i)^\top \phi(x_j)$, where the required inner-products are provided by a positive semi-definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, i.e. $\phi(x_i)^\top \phi(x_j) = k(x_i, x_j)$. The power in these methods is that the feature transform $\phi(x_i)$ is not evaluated explicitly, and that we can apply an ℓ_2 regularizer $\lambda || \sum_{j=1}^m \theta_j \phi(x_j)||_2^2$ using the Tikhonov regularizer $\lambda \theta^\top K \theta$, where the

entry in row i and column j of K is given by $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$. Depending on the kernel, the regularizer $\lambda\theta^\top K\theta$ may provide a major improvement over simple ℓ_2 regularization using $\lambda||\theta||_2^2$. We develop similar regularizers for use with non-kernel features in Chapter 4.

Deep learning methods have recently become the focus of much attention due to excellent empirical results in many tasks, particularly for computer vision [55]. We provide more information about specific methods in Chapters 5-7, which present our contributions to deep learning. In supervised deep learning, the feature transform $\phi : \mathcal{X} \to \mathcal{X}_\phi$ is not assumed to be fixed, and is optimized at the same time as θ. The only requirement for ϕ to be useable as part of a deep learning solution is that it should be adjustable to reduce the cost $\mathcal{L}(X, Y, f_\theta)$ based on interactions with the observation pairs in (X, Y). This requirement is typically satisfied by choosing a parametric ϕ for which the gradient of $\mathcal{L}(X, Y, f_\theta)$ w.r.t. the parameters of ϕ is computable. Typically, to be considered legitimately deep, ϕ should be constructed by composing multiple simpler transformations.

2.2 Unsupervised Learning

In unsupervised learning problems, y is not given, so we use a scalar cost $\mathcal{L}(X, f)$ which only depends on f and X. Unsupervised learning encompasses, e.g., methods for dimension reduction, dictionary learning, and generative modelling. We use these methods throughout Chapters 3, 6, and 7.

2.2.1 Dimension Reduction

Even when data is observed in a high-dimensional space, one can often describe it using a much smaller set of "latent factors". E.g. images of an individual's

face, in a fixed setting, can vary dramatically in pixel-space based on whether the individual is looking up, down, left, or right. Yet, in some sense, the images only vary along two axes: looking up↔down and looking left↔right. For some tasks, knowing the description of an image in terms of these two axes may be all that is required, with all other information in the image acting as noise.

Dimension reduction is the task of learning compact representations of data. Dimension reduction methods can be task-agnostic or task-aware. Task-agnostic methods are designed to be generally applicable, while task-aware methods are designed to retain only information relevant to a particular task. Both types of methods are defined by three key choices: how to transform an observation into a lower-dimensional representation, how to measure the utility of the information retained by the transformation, and what distributional properties the transformed observations should have.

To illustrate task-agnostic dimension reduction, consider Principal Components Analysis (abbr. PCA). In PCA, the dimension reduction transformation is given by $f_A(x) \triangleq Ax$, where $f_A(x)$ describes the transformed point and A is a $d \times n$ matrix – assuming $f_A(x) \in \mathbb{R}^d$ and $x \in \mathbb{R}^n$. For the second choice, i.e. how to measure the utility of information retained, PCA uses simple squared reconstruction error. For the third choice, i.e. distributional properties of the transformed points, PCA requires coordinates in the transformed space to be uncorrelated. Putting

these choices together, PCA optimizes:

$$f_A^* = \arg\min_{f_A \in \mathcal{F}} \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} ||x - A^\top Ax||_2^2 \tag{2.12}$$
$$\text{subject to: } \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \left[(Ax)(Ax)^\top\right] \text{ is diagonal,}$$

where observations are sampled from a distribution $\mathcal{D}_x$, and we assume that all coordinates of x have a mean of 0. The diagonal assumption in the second line forces A to have orthogonal rows, aside from those in the left null space of X. Rows in the optimal A can only be in the left null space of X if the rank of X is less than d, otherwise some of the "information retention capacity" of A would be wasted on these rows. If X has rank $n' < d$, then the optimal A will have exactly $d - n'$ rows in the left null space of X. If we assume X has rank $n' > d$, the optimal A for Eq. 2.12 will have rows that align with the eigenvectors of $XX^\top$ having the d largest eigenvalues. Thus PCA computes the optimal rank-d approximation of X – as measured by squared error.

2.2.2 Encoder-Decoder Methods

Encoder-decoder methods generalize dimension reduction methods that, like PCA, consider mapping both to and from the transformed space.[4] Encoder-decoder methods require choices very similar to dimension reduction:

- How to encode a normal observation into an encoded one.

[4] Some dimension reduction methods, like Multi-dimensional scaling [41], Isomap [103], and t-SNE [24], only map from $\mathcal{X}$ to the transformed space. These methods do not attempt to map transformed points back into $\mathcal{X}$.

- How to decode an encoded observation back into a normal one.
- How to measure the utility of the information retained by the learned encoder-decoder pair.
- What distributional properties the encoded observations should have.

The key differences from dimension reduction methods are the requirement of a decoder, and measurement of information retention through the encode→decode process.

The basic components of an encoder-decoder method are: an encoder $q : \mathcal{X} \to \beta$, a decoder $p : \beta \to \mathcal{X}$, an "information retention" cost $\mathcal{L}^{info}(q, p, x)$, and a distributional cost $\mathcal{L}^{dist}(q, p, x)$. The spaces $\mathcal{X}$ and β can contain vectors, sequences, distributions over vectors, distributions over sequences, etc. This class of methods encompasses denoising auto-encoders [10], variational auto-encoders [51, 80], LDA [15], Gaussian mixture models, sparse coding [74], etc. Note that these methods are all easily interpreted as some form of directed generative model for $x \in \mathcal{X}$, where we assume the models use latent variables. If we extend our definition so that $\mathcal{X}$ becomes $\mathcal{X} \times \mathcal{Y}$, then we can include any directed generative model for $p(y|x)$, $p(x|y)$, $p(x)$, and $p(y)$. We could also change $\mathcal{X}$ to be $\mathcal{X} \times \mathcal{Y} \times \mathcal{Z}$, and so on.

As an example, the standard sparse coding objective is given by [62]:

$$\underset{A \in \mathcal{A}}{\text{minimize}} \; \mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \left[\min_{\beta} \left[||x - A\beta||_2^2 + ||\beta||_1 \right] \right], \tag{2.13}$$

where $\mathcal{A}$ contains only matrices whose columns have unit norm. Here, $||x - A\beta||_2^2$ gives the information retention cost $\mathcal{L}^{info}(q, p, x)$ and $||\beta||_1$ gives the distributional

cost $\mathcal{L}^{dist}(q, p, x)$. The objective above is equivalent to variational inference with Dirac-delta posterior approximations in a "euclidean topic model" with a Laplace prior over the "topic coefficients". We can easily transform this to a standard topic model, i.e. LDA [15], as follows:

$$\underset{A \in \mathcal{A}}{\text{minimize}} \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \left[\min_{\beta} \left[\text{KL}(x \,||\, A\beta) + \log \text{dir}(\beta|\alpha) \right] \right], \tag{2.14}$$

where $\mathcal{A}$ contains only column-stochastic matrices, x/β are also column-stochastic, and $\text{dir}(\beta|\alpha)$ gives the probability of multinomial distribution β under a Dirichlet with concentration parameter α. This describes variational inference for LDA using Dirac-delta posterior approximations, and with the count vectors x normalized to be multinomial distributions. Note that the encoders in Eq. 2.13 and Eq. 2.14 are described non-parametrically by the minimizations over β.

2.2.3 Generative Modelling

The encoder-decoder framework described in the previous subsection includes a wide range of conditional and unconditional generative models, but we treat them separately in this subsection, as is tradition [41, 13]. The most common goal for a generative model p is to maximize the expected log-likelihood:

$$\underset{p}{\text{maximize}} \underset{x \sim \mathcal{D}_x}{\mathbb{E}} \log p(x), \tag{2.15}$$

which is equivalent to minimizing the KL divergence $\text{KL}(\mathcal{D}_x \,||\, p)$. Aside from maximizing the expected log-likelihood, one may also want efficient sampling, training, evaluation, etc. To be useful for making decisions, one may also want p to expose useful information about each x – e.g. via posteriors over some latent

variables. We discuss three basic types of generative models: directed models without latent variables, directed models with latent variables, and undirected models with latent variables. While the overall class of models which one could reasonably describe as "generative" is quite broad, for the purposes of this thesis we will focus on methods that have been developed by deep learning researchers over the past 5-10 years. Such methods provide the basis for the work presented in Chapters 6 and 7.

The Neural Autoregressive Distribution Estimator (abbr. NADE) is a directed generative model without latent variables [57]. NADE works by using Bayes' rule to factor $p(x)$ into a sequence of conditionals:

$$p(x) \triangleq p_0(x_0) \prod_{i=1}^{n} p_i(x_i|x_{i-1}, ..., x_0). \tag{2.16}$$

This requires the assumption of an ordering $0...n$ over the $n+1$ coordinates of $\mathcal{X}$. The clever trick in NADE is to share parameters between all of the $p_i(...)$. This can be done using a neural network and some masks to occlude x_j for $j \geq i$ when evaluating the conditional $p_i(x_i|x_{i-1}, ..., x_0)$, which may only condition on x_k for $k < i$. Additional improvements have been made to NADE including using randomized orderings to train a shared-parameter ensemble [106], and alternative masking techniques for better computational efficiency [29].

Directed generative models with latent variables have a lot of nice properties. We explore these models in some depth through Chapters 6 and 7. A directed

model with latent variables represents $p(x)$ using:

$$p(x) \triangleq \sum_z p(x, z), \tag{2.17}$$

and factorizes its representation of $p(x, z)$ into a collection of conditional and unconditional distributions. The only restrictions on the factorization are that it must be acyclic, and that each observable variable $x_i \in \mathcal{X}_i$ and each latent variable $z_i \in \mathcal{Z}_i$ must be generated by exactly one distribution, which can be either conditional or unconditional. Variables can be conditioned on by any number of distributions, as long as the acyclic property is maintained. The basic directed model with latent variables lumps all x_i into a single $x \in \mathcal{X}$ and all z_i into a single $z \in \mathcal{Z}$. This model places a prior $p(z)$ over z and constructs $p(x)$ as follows:

$$p(x) \triangleq \sum_z p(x|z)p(z), \tag{2.18}$$

which also requires the conditional $p(x|z)$. A useful property of directed models with latent variables is that one can easily evaluate $\log p(x, z)$ for any (x, z) whenever one can easily evaluate the log-likelihoods of all un/conditional distributions in the factorization of $p(x, z)$.

The Restricted Boltzmann Machine (abbr. RBM) is a well-known undirected generative model with latent variables [92]. The RBM computes the log-likelihood of a pair of binary vectors (x, z) as follows:

$$\log p_W(x, z) = x^\top W z - \log Z_W, \tag{2.19}$$

where W is a matrix of parameters and $\log Z_W$ indicates the log of the "partition function" which normalizes $p_W(x, z)$ so that $\sum_{x,z} p_W(x, z) = 1$. The RBM has the convenient property that $p_W(x|z)$ and $p_W(z|x)$ are both trivial to sample and evaluate. However, the term $\log Z_W$ makes it difficult to evaluate any marginal log-likelihoods or gradients of marginal log-likelihoods. One can compute a Monte Carlo estimate of $\nabla_W \log p_W(x_0)$ using:

$$\tilde{\nabla}_W \log p_W(x_0) = x_0 z_0^\top - x_\infty z_\infty^\top, \tag{2.20}$$

where $p_W(x) = \sum_z p_W(x, z)$ and $\{z_0, x_1, z_1, ..., x_\infty, z_\infty\}$ are samples from an infinitely long Gibbs chain over (x, z). The Gibbs chain runs by repeatedly sampling $z_i \sim p_W(z_i|x_i)$ and $x_{i+1} \sim p_W(x_{i+1}|z_i)$. The gradient in Eq. 2.20 uses:

$$\nabla_W \log Z_W = \mathop{\mathbb{E}}_{x,z \sim p_W(x,z)} x z^\top, \tag{2.21}$$

and computes $\tilde{\nabla}_W \log Z_W$ based on a single sample $(x_\infty, z_\infty) \sim p_W(x, z)$.

The problem with Eq. 2.20 is that running the Gibbs chain for an infinite number of steps is impractical. In [42], G. E. Hinton introduced a method for training RBMs called Contrastive Divergence (abbr. CD). In CD-k training, one uses a Monte Carlo gradient approximation given by:

$$\tilde{\nabla}_W \log p_W(x_0) = x_0 z_0^\top - x_k z_k^\top, \tag{2.22}$$

where $\{z_0, x_1, z_1, ..., x_k, z_k\}$ are samples from a Gibbs chain of length k. We develop related methods, based on sampling from Gibbs chains, in Chapter 6.

2.3 Semi-supervised Learning

In semi-supervised learning problems, one has access to a set (X^l, Y^l) of (x, y) pairs in which each label y is known, and a set X^u of observations for which the corresponding labels are unknown. The goal of semi-supervised learning is to use the unlabelled data in X^u to reduce label prediction errors for new (x, y) pairs.[5]

A variety of methods have been developed for semi-supervised learning. One related pair of methods are self-training [89] and co-training [16]. In self-training, the classifier is trained with the labeled data in (X^l, Y^l) and then used to provide labels for the unlabeled points in X^u. Some portion of the most confident new labels are retained, and the classifier is then re-trained using (X^l, Y^l) *and* the newly labeled points from X^u. This process then repeats. Co-training involves multiple classifiers, each trained on their own "view" of the observations.[6] The classifiers are all trained with the available labels and then used to guess labels for a subset of the unlabeled points, based on their associated views of the data. The newly labeled points are added to the training set and then the process iterates.

[5] We describe semi-supervised learning in the context of classification, but its possible applications include all prediction problems. In settings where both $\mathcal{X}$ and $\mathcal{Y}$ contain non-trivial structure, one could leverage "non-paired" samples from both $\mathcal{X}$ and $\mathcal{Y}$ to improve predictions in either direction.

[6] View here could mean anything that causes their predictions to differ from each other. This includes, e.g. different subsamples of the training set, different parameter initializations (for non-convex methods), different model architectures, etc.

Two other significant approaches to semi-supervised learning are Transductive SVMs [47] and manifold regularization [8]. Transductive SVMs are based on the intuition that boundaries between classes, as viewed from $\mathcal{X}$, should fall in regions where $\mathcal{D}_{xy}$ has low density. This can be further motivated by the assumption that points in the same class tend to cluster together, and point clusters from different classes tend to be well-separated. The Transductive SVM augments the standard SVM objective – i.e. minimizing hinge loss and RKHS norm – with a loss that penalizes uncertain predictions for the unlabeled points. In principle, this forces the decision boundary away from regions of high density in $\mathcal{D}_{xy}$, subject to the restrictions imposed by RKHS norm minimization and the selected kernel. In manifold regularization, one encourages a function f to maintain local consistency by using a regularizer which penalizes the sum:

$$\sum_{x_i, x_j \in X^l \cup X^u} w_{ij}(f(x_i) - f(x_j))^2, \tag{2.23}$$

in which the weights w_{ij} are proportional to some non-negative measure of similarity between pairs of points x_i/x_j, and the sum is over all pairs of points in the joint labeled+unlabeled training set. This regularizer only allows large changes in f between points x_i/x_j for which w_{ij} is small. When the weights w_{ij} are large and non-zero only in a local neighborhood around each point x_i, f will vary slowly within any given neighborhood, but can accumulate large changes over extended chains of neighborhoods, or in gaps between neighborhoods.

CHAPTER 3
Improved Estimation in Time-Varying Models

3.1 Motivation

Locally adapted parametrizations can produce flexible representations from relatively rigid components. E.g., locally weighted regression procedurally defines a non-linear function approximator using an infinite collection of locally adapted linear function approximators.[1] Constructing a compound model by composing many instances of a simple base model can reduce bias relative to the base model, but may also increase variance. The reduced effective sample size used for training each locally adapted instance of the base model causes the increased variance, but it is also what allows the model to focus on locally salient patterns, and is thus unavoidable when training a locally adapted model.

This chapter describes a method for striking a better balance in this bias/variance trade-off by learning a transformed space in which to perform the searches for locally adapted parametrizations. By making this transformed

[1] The abstract notion of locality is extremely general. It can refer to temporal proximity, spatial proximity, task relatedness, etc. Our applications will treat temporal proximity as the relevant measure of locality, but the problem formulation and methods we describe are more broadly applicable.

space simpler, e.g. lower dimensional, than the original parameter space, we can reduce the effective sample size required for stable estimation of each locally adapted parametrization. If this reduction is large enough, we may be able to increase the locality of each model and thereby reduce model bias, but without suffering an increase in variance. By shaping the transformed space based on the performance of multiple parametrizations estimated within it, we can encourage it to permit the variation necessary for capturing changes in the optimal base model from location to location.

A common approach to improving model efficacy in machine learning is to first transform the *data* into an alternate representation prior to model estimation, ideally in a way that amplifies useful information while attenuating noise. Algorithms exemplifying this approach include: PCA, ICA [46], nonlinear-dimension reduction, e.g. [103], and dimension reduction for regression [28, 20]. Such methods can be either task-agnostic or task-aware. Task-agnostic dimension reduction methods, like PCA, are applied without knowledge of the task for which the transformed data will be used. This makes these methods more general, but perhaps less effective for any given task. Task-aware dimension reduction methods, like those used in dimension reduction for regression, are less general but can be more

aggressive about removing information not related to the target task. I.e. task-aware methods can remove information that is "noise" relative to the task-at-hand, but "signal" for some other task.[2]

Another line of work considers transformations of the *model* used to describe the data, either by reducing its degrees of freedom, or by seeking a model form amenable to more powerful estimation procedures. Examples of the first approach include DiscLDA [56] and supervised dimensionality reduction using Bayesian mixture models [84], which both seek useful linear reductions of the parameters of a generative model. The second approach includes, e.g., spectral methods for learning transformed representations of HMMs [91] and PSRs [17].

This chapter presents a different view of model transformations. In Sec. 3.2, we present a generalized formulation of the problem of estimating useful transformations of model parameters. In Sec. 3.3, we present a sequence of examples showing how this formulation encompasses several of the previously mentioned methods for data and model transformation. The resulting problem depends on estimating a transformation of the parameter space of a model and on estimating multiple parameterizations within the transformed space. In Sec. 3.4, we present

[2] People often seem to think of "information" and "noise" as diametrically opposed, but this is wrong. Information is just distinguishability and variation, without any necessary reference to meaning or utility. The proper terminological juxtaposition is between "signal" and "noise". Signal and noise are context dependent, whereas information is not.

a novel algorithm for modelling time varying sparse network structures underlying sequential observations. In Sec. 3.5 and 3.6, we use synthetic data and data drawn from real-world BCI EEG experiments [87, 14] to empirically validate our algorithm.

3.2 Learning Transformed Representations of Model Parameters

The problems considered in this chapter arise from the following objective:

$$B^* = \arg\min_B \left[\ell(f, X, B)\right], \tag{3.1}$$

in which the loss ℓ measures the fitness of the model f for the data $X \triangleq \{(x_1, y_1), ..., (x_m, y_m)\}$, given a set $B \triangleq \{\beta_1, ..., \beta_{m'}\}$ of *multiple* parametrizations of f. We seek an optimal set of parametrizations B^*. With suitable definitions of f and ℓ, many previously-studied problems can be expressed in this framework.

To motivate the objective in Eq. 3.1, we begin by expressing standard linear regression in the appropriate form. For this purpose, we define f as the residual produced by parameter vector β for input x and output y:

$$f((x, y), \beta) \triangleq \beta^\top x - y.$$

For multiple parameter vectors β_i and observations (x_j, y_j), we define ℓ as proportional to the log-likelihood of observing the residuals $f((x_j, y_j), \beta_i)$, $\forall i, j$, assuming they are normally distributed:

$$\ell(f, X, B) \triangleq \sum_{i=1}^{m'} \sum_{j=1}^{m} f((x_j, y_j), \beta_i)^2. \tag{3.2}$$

We usually think of the loss in this case as having $m' = 1$. However, considering $m' > 1$ does not modify the solution because, for all β_i, loss is measured equally over all (x_j, y_j), which implies that $\beta_i = \beta_j, \forall \beta_i, \beta_j \in B^*$.

Starting from this point, we can transform standard linear regression into kernel weighted linear regression as follows:

$$\begin{aligned} f((x,y),\beta) &\triangleq \beta^\top x - y \\ \ell(f,X,B) &\triangleq \sum_{i=1}^{m'} \sum_{j=1}^{m} k(\mu_i, x_j) f((x_j,y_j),\beta_i)^2, \end{aligned} \tag{3.3}$$

where the kernel weighting function $k(x, x')$ measures similarity between locations in the input space, and each β_i is associated with a location μ_i. We consider the choice of location μ_i for each β_i as included in the definition of the loss ℓ.

Introducing the kernel k allows each β_i in Eq. 3.3 to have local rather than global effect, which leads to a different parametrization β_i at each location μ_i. However, in Eq. 3.3 there is no real need to optimize jointly over B, as there are no constraints linking the different $\beta_i \in B$. Allowing multiple local parametrizations of f is useful for increasing the power of simple models; estimation of time varying covariance matrices in financial modelling and estimation of time varying auto-regressions in econometrics are two well-studied examples of this idea.

To illustrate a problem in the form of Eq. 3.1 in which the elements of B^* are not independent, consider the following optimization for $\beta_i \triangleq (\beta_i^\mu, \beta_i^\Sigma, \beta_i^\pi)$:

$$\begin{aligned} f((x,y),\beta) &\triangleq \beta^\pi p(x|\beta^\mu, \beta^\Sigma) \\ \ell(f,X,B) &\triangleq -\log\left(\prod_{j=1}^{m}\left[\sum_{i=1}^{m'} f((x_j,y_j),\beta_i)\right]\right), \end{aligned} \tag{3.4}$$

in which $\beta_i^\pi \geq 0$, $\sum_i \beta_i^\pi = 1$, and $p(x|\beta^\mu, \beta^\Sigma)$ gives the probability of sampling x from a Gaussian distribution with mean β^μ and covariance β^Σ. Minimizing Eq. 3.4 corresponds to estimating a Gaussian mixture model [13] for the data $X \triangleq \{x_1, ..., x_m\}$. Eq. 3.4 entangles the $\beta_i \in B^*$ through the sum in the log-likelihood and the "distribution" constraint on the mixture weights, i.e. $\sum_i \beta_i^\pi = 1$.

Note that in the last two examples, the estimation of B^* may be subject to high variance. To mitigate this variance, and to exploit possible structure in the set of optimal parametrizations B^*, we introduce a "generating" function g. This g takes an input $\hat{\beta}$ and transforms it into an output β, such that β provides a valid parametrization of f. The function g can thus be used to express both regularities and restrictions in the way f is parametrized. By estimating each $\beta_i \in B^*$ via g, i.e. as $\beta_i \triangleq g(\hat{\beta}_i)$, we restrict the $\beta_i \in B^*$ to varying as permitted by g.

For instance, in a time varying model the optimal temporally local parametrizations of f may travel through a low-dimensional manifold embedded in the full parameter space of f. The structure of such a manifold could be of interest, and restricting temporally local parameter estimations to this manifold could significantly reduce their variance with only a small increase in bias.

We now reformulate Eq. 3.1 to use transformed parametrizations. Given a generating function g, a base model f, a loss ℓ, and observations X, we get:

$$g^* = \arg\min_g \left[\min_{\hat{B}} \left[\ell(f|g, X, \hat{B}) \right] \right], \tag{3.5}$$

in which $\hat{B} \triangleq \{\hat{\beta}_1, ..., \hat{\beta}_{m'}\}$ is a set of inputs to g and $f|g$ denotes parametrization of f via g. I.e. each $\hat{\beta}_i \in \hat{B}$ generates a valid parametrization, $\beta_i \triangleq g(\hat{\beta}_i)$, of f.

If we define $g(\hat{\beta}) \triangleq \hat{\beta}$, then Eq. 3.5 exactly reproduces Eq. 3.1. If we allow g to take an arbitrarily complex form, then we again recover the optimization in Eq. 3.1, as we can define $g(\hat{\beta}_i) \triangleq \beta_i$ for each β_i in the optimal B^*. Interesting cases of Eq. 3.5 arise when g is more carefully chosen. The next section illustrates some useful problems that arise from different definitions of g, f, and ℓ.

3.3 Examples of Transformed Multiple Parametrization

As a first example, consider performing a locally weighted regression analogous to that in Eq. 3.3, but with the local parametrizations restricted to a linear subspace of the full parameter space of f. I.e., let $g(\hat{\beta}) \triangleq A\hat{\beta}$, where A is a matrix of parameters for g. For this, we can re-write Eq. 3.5 as follows:

$$A^* = \arg\min_{A} \left[\min_{\hat{B}} \left[\sum_{i=1}^{m'} \sum_{j=1}^{m} k(\mu_i, x_j)(x_j^\top A\hat{\beta}_i - y_j)^2 \right] \right], \tag{3.6}$$

in which the loss ℓ associates each $\hat{\beta}_i$ with a location μ_i. If one views $x_j^\top A\hat{\beta}_i$ as a reduction of x_j into the space spanned by the rows of A, followed by a linear regression in that space, then the objective in Eq. 3.6 is closely related to methods developed for linear dimension reduction for regression based on non-parametric estimators [86, 114]. Relatively minor modifications, like regularizing the $\hat{\beta}_i$s via $\lambda \sum_i ||\hat{\beta}_i||_1$, weaken this link. An example application of Eq. 3.6 to time varying linear regression is presented in Figure 3–1.

As a second example, we restate the mixture of Gaussians model from Eq. 3.4 under the constraint that the means $\{\beta_1^\mu, ..., \beta_{m'}^\mu\}$ of the parametrizations $\{\beta_1, ...\beta_{m'}\}$ lie within a linear subspace of the observation space, i.e. $\beta_i \triangleq$

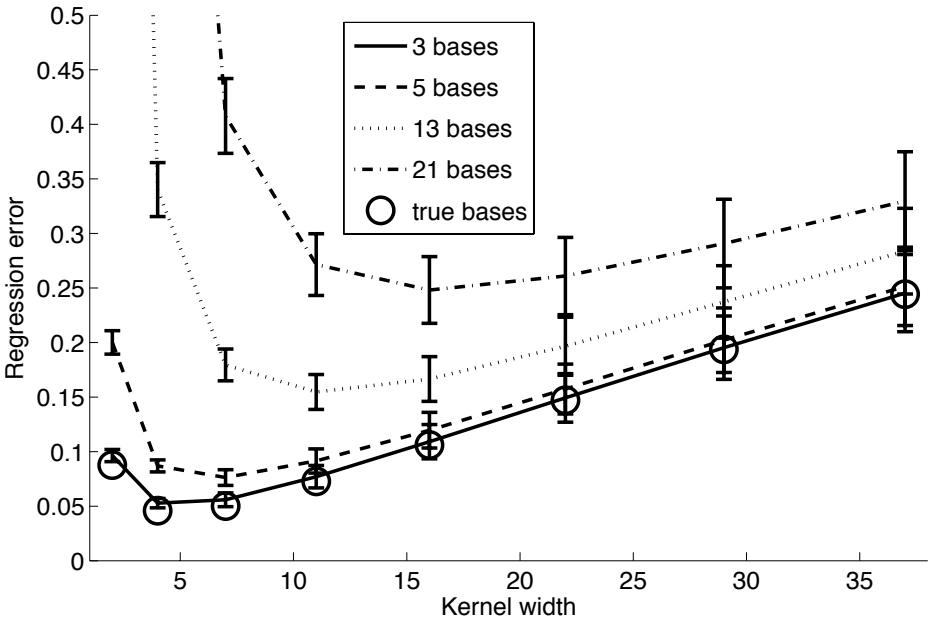


Figure 3–1: An application of the transformed locally weighted regression described in Eq. 3.6 to time varying linear regression. In this case, the target function $y_t = \beta_t^\top x_t$ at time t was defined by a linear combination of three bases, i.e. $\beta_t = \alpha_t^1\beta_t^1 + \alpha_t^2\beta_t^2 + \alpha_t^3\beta_t^3$. Change in the target function over time was controlled by changes in the basis coefficients β_t^i, while the basis functions α_t^i remained fixed. Observations x_t were 21-dimensional, noise was added to the outputs y_t, and trajectories of the αs were generated to vary smoothly over time. The plots show errors (lower is better) when different numbers of bases were learned (by optimizing Eq. 3.6 on a training set) and subsequently applied to locally weighted regression (on a held-out test set). Errors are plotted as a function of the kernel width used during test regressions. Reducing the number of parameters estimated in each local regression reduces the error, by permitting smaller kernels and minimizing estimation variance due to noise/dimensionality. The performance obtained by three learned bases nearly matches that obtained by the three true bases.

$(g(\hat{\beta}_i^{\mu}), \hat{\beta}_i^{\Sigma}, \hat{\beta}_i^{\pi})$[3] , with g defined as for Eq. 3.6. The resulting optimization is:

$$A^* = \arg\min_{A} \left[\min_{\hat{B}} -\log \left(\prod_{j=1}^{m} \left[\sum_{i=1}^{m'} \hat{\beta}_i^{\pi} p(x_j | A\hat{\beta}_i^{\mu}, \hat{\beta}_i^{\Sigma}) \right] \right) \right]. \tag{3.7}$$

The optimization in Eq. 3.7 was shown to be useful for classification tasks in [84].

3.3.1 Sparse Coding

We now formulate a problem equivalent to the well-studied form of sparse coding based on ℓ_1-regularized least-squares regression [62]. First, we define $g(\hat{\beta}) \triangleq A\hat{\beta}$ under the constraint that the columns of A have unit norm. Then, we let $f(x, g(\hat{\beta})) \triangleq ||x - g(\hat{\beta})||_2$. Finally, we define ℓ as:

$$\ell(f|g, X, \hat{B}) \triangleq \sum_{i=1}^{m} f(x_i, g(\hat{\beta}_i))^2 + \lambda \sum_{i=1}^{m} ||\hat{\beta}_i||_1, \tag{3.8}$$

where the second sum induces sparsity in the reconstruction of each x_i and λ controls the trade-off between sparsity and reconstruction error. This combination of g, f, and ℓ leads to an optimization equivalent to ℓ_1-regularized sparse coding:

$$A^* = \arg\min_{A} \left[\min_{\hat{B}} \left[\sum_{i=1}^{m} ||x_i - A\hat{\beta}_i||_2^2 + \lambda \sum_{i=1}^{m} ||\hat{\beta}_i||_1 \right] \right], \tag{3.9}$$

where the columns of A form the reconstruction bases and each $\hat{\beta}_i$ encodes an input x_i in terms of these bases. Eq. 3.9 can also be interpreted as an infinite mixture of Gaussians in which the mixture components all have the same isotropic covariance and share the parameters for their means via the columns of A. The

[3] Note that we have *not* transformed the covariances $\hat{\beta}_i^{\Sigma}$

latent space, in which the $\hat{\beta}$ reside, acts as a continuous index into the mixture. The "prior" weights of the mixture components are fixed – determined in this case by "doubly-rectified exponential" distributions aligned with the axes of $\hat{\beta}$-space. Under this interpretation, the optimization in Eq. 3.9 is equivalent to maximization of a variational bound on the log-likelihood of X, using point estimates for the parameters in A and point estimates for the posterior distributions over $\hat{\beta}$-space for each observation x_i. Constraints on the column norms of A can be interpreted as the result of a strong prior.

3.3.2 PCA

To define an optimization that produces a solution equivalent to PCA [41], we first let $g(\hat{\beta}) \triangleq A\hat{\beta}$ under the constraint that $A^\top A = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Then, we define $f(x, g(\hat{\beta})) \triangleq ||x - g(\hat{\beta})||_2$. Finally, we define the loss as:

$$\ell(f|g, X, \hat{B}) \triangleq \sum_{i=1}^{m} f(x_i, g(\hat{\beta}_i))^2 + \lambda||\Sigma_{\hat{B}}||_1, \tag{3.10}$$

where $\Sigma_{\hat{B}} \triangleq \frac{1}{m}\sum_{i=1}^{m} \hat{\beta}_i\hat{\beta}_i^\top$ is computed from the transformed parameter vectors in $\hat{B} \triangleq \{\hat{\beta}_1, ..., \hat{\beta}_m\}$, and $||\Sigma_{\hat{B}}||_1$ indicates $\sum_{i,j} |\Sigma_{\hat{B}}^{i,j}|$. With g, f, and ℓ defined this way, we can write the full optimization as follows:

$$A^* = \arg\min_{A} \left[\min_{\hat{B}} \left[\sum_{i=1}^{m} ||x_i - A\hat{\beta}_i||_2^2 + \lambda||\Sigma_{\hat{B}}||_1 \right] \right]. \tag{3.11}$$

Proposition 1. *When $\lambda > 0$ and each coordinate of X is centered (i.e. has a mean of 0), the matrix A^* that minimizes Eq. 3.11 has columns aligned with the principal component directions of $X \triangleq \{x_1, ..., x_m\}$.*

Proof. First, consider applying the diagonal-only penalty $\lambda \sum_i |\Sigma^{i,i}_{\hat{B}}|$, which corresponds to ℓ_2 regularization via $\frac{\lambda}{m} \sum_i ||\hat{\beta}_i||_2^2$. For any matrix A with orthonormal columns (i.e. any A such that $A^\top A = \mathbf{I}$), the value of:

$$\min_{\hat{B}} \left[\sum_{i=1}^{m} ||x_i - A\hat{\beta}_i||_2^2 + \frac{\lambda}{m} \sum_{i=1}^{m} ||\hat{\beta}_i||_2^2 \right] \tag{3.12}$$

is the same due to the rotational invariance of the squared ℓ_2 norm. The value of Eq. 3.12 serves as a lower bound on the objective in Eq. 3.11 for any proper A, as $\lambda ||\Sigma_{\hat{B}}||_1 \geq \frac{\lambda}{m} \sum_{i=1}^{m} ||\hat{\beta}_i||_2^2$ for all $\hat{B}$. Equality, and thus the lower bound, is achieved only when $\Sigma_{\hat{B}}$ for the optimal $\hat{B}$ for Eq. 3.12 is diagonal.

From the properties of ℓ_2 regularization, the optimal $\hat{B}$ for Eq. 3.12 is $(1 + \frac{\lambda}{m})^{-1} XA$, where row i of $\hat{B}$ is $\hat{\beta}_i$ and row i of X is x_i. From the previous reasoning, the optimal A for Eq. 3.11 diagonalizes $\hat{B}^\top \hat{B} = (1 + \frac{\lambda}{m})^{-2} A^\top X^\top X A$. Such diagonalization occurs only if the columns of A align with the eigenvectors of $X^\top X$. These eigenvectors define the principal component directions of $X \triangleq \{x_1, ..., x_m\}$ when the coordinates of X are centered. □

3.4 Learning Transformed Parametrizations of Network Structure

In this section, we use the problem formulation in Eq. 3.5 to derive a novel algorithm for estimating time varying network structures using locally adapted linear combinations of learned basis structures. Based on connections between our algorithm and sparse coding [74], we then extend our algorithm to learn task-driven basis structures, guided by the work in [69]. We begin with a brief review of prior work on network structure estimation.

3.4.1 Sparse Network Structure Estimation

Significant effort has gone into developing methods for estimating sparsely structured Gaussian graphical models. A Gaussian graphical model (abbr. GGM) explains a set of m n-dimensional observations $X \triangleq \{x_1, ..., x_m\}$, $x_i \in \mathbb{R}^n$ using a set of n vertices – one for each coordinate of the $x_i \in X$ – and a set of edges, each of which describes the coupling between its incident vertices [26]. The edge weights in a GGM encode a covariance Σ, and the GGM equates to modelling X with a normal distribution $\mathcal{N}(\vec{0}, \Sigma)$. We assume that the observations are standardized to have zero mean for each coordinate.

Many existing methods addressing GGMs focus on estimating their structure, which refers to the pattern of zero/non-zero edges. These methods typically work with the precision matrix (i.e. Σ^{-1}) implied by a GGM, as non-zero entries in Σ^{-1} correspond to non-zero edges in the GGM. Additionally, estimating the structure of Σ^{-1} is facilitated by the following relationship:

$$\rho_{ij} = \frac{\tilde{\sigma}_{ij}}{\sqrt{\tilde{\sigma}_{ii}\tilde{\sigma}_{jj}}}, \tag{3.13}$$

in which ρ_{ij} indicates the partial correlation between dimensions i and j of the $x \in X$ conditioned on the values of all other dimensions. $\tilde{\sigma}_{ij}$ indicates the entry in row i and column j of Σ^{-1}. Thus, a non-zero partial correlation ρ_{ij} between dimensions i and j of the observations in X implies that $\Sigma^{-1}_{ij} \neq 0$, which implies an edge between i and j in the GGM for X. Conversely, if ρ_{ij} is 0, it implies that $\Sigma^{-1}_{ij} = 0$, which implies that there is no edge between i and j in the GGM for X. This relationship between partial correlations and GGM structure leads to efficient

methods for GGM structure estimation, as partial correlations can be directly estimated by "self-regression". The key trick will be to induce sparsity during the self-regression, e.g. by applying ℓ_1 regularization, to make many of the estimated ρ_{ij} precisely zero.

The use of self-regression for network structure estimation is based on the following results [58]:

$$x_t^i = \sum_{j \neq i} x_t^j \tilde{\rho}_{ij} + \epsilon_t^i, \tag{3.14}$$

in which x_t^i represents the value of dimension i in observation $x_t \in X$, $\tilde{\rho}_{ij}$ is a real-valued scalar, and ϵ_t^i is uncorrelated with x_t^i if and only if:

$$\tilde{\rho}_{ij} = -\frac{\tilde{\sigma}_{ij}}{\tilde{\sigma}_{ii}} = \rho_{ij}\sqrt{\frac{\tilde{\sigma}_{jj}}{\tilde{\sigma}_{ii}}}, \text{ from which} \tag{3.15}$$

$$\rho_{ij} = \text{sign}(\tilde{\rho}_{ij})\sqrt{\tilde{\rho}_{ij}\tilde{\rho}_{ji}}. \tag{3.16}$$

Hence, $\tilde{\rho}_{ij}$ can be efficiently estimated for any given i using linear regression of the response variables $\{x_1^i, ..., x_m^i\}$ on the covariates $\{x_1^{\setminus i}, ...x_m^{\setminus i}\}$, in which $x_t^{\setminus i}$ indicates a vector including all dimensions except i; ρ_{ij} can then be computed as well.

Most existing methods for GGM structure estimation assume that Σ^{-1} is sparse. This assumption is motivated by the difficulty of estimating all $\mathcal{O}(n^2)$ parameters in Σ^{-1} in many practical settings where observations are high-dimensional and training data is limited. The sparsity assumption can (provably) make estimation more efficient, and permits us to distinguish between value estimation methods and structure estimation methods. Value estimation methods attempt to determine each entry in Σ^{-1} precisely [118], whereas structure

estimation methods only attempt to determine the pattern of zero/non-zero entries in Σ^{-1} [26, 98, 53]. The sparsity assumption can be incorporated into the self-regression process by using sparsifying regression techniques, such as the Lasso [104]. Structure estimation methods based on sparse self-regression have been shown to consistently estimate structure in Σ^{-1} under suitable conditions [70, 110, 53].

One line of work focuses on extending (sparse) GGM structure estimation methods to settings where the estimated structures vary over time [1, 52, 97, 98, 118, 53]. We focus on the KELLER algorithm from [97], as the algorithm we present in Section 3.4 can be seen as its natural extension to using transformed parametrizations following the form of Eq. 3.5. KELLER relies on two main assumptions: sparsity in the time varying network structures, and smoothness in the changes of these structures over time. This second assumption distinguishes KELLER from methods such as [1] and [52], which assume abrupt changes in structure. Though we only consider extending KELLER in this chapter, it would be straightforward to similarly extend the algorithms in [1] and [52].

To estimate the structure of a network at time t, given a sequence of T n-dimensional observations $X \triangleq \{x_1, .., x_T\}$, KELLER performs a set of n independent ℓ_1-regularized locally weighted regressions, with the i^{th} regression estimating the values $\tilde{\rho}_{ij}$ as described for Eq. 3.14-3.16. By using locally weighted regression, the estimated $\tilde{\rho}_{ij}$ are specifically adapted to the predominant network structure underlying the observations at/around time t. For time t, the corresponding

optimization objective can be written as follows:

$$A_t^* = \underset{A \in \mathbb{R}^{n \times n}}{\arg\min} \sum_{t'=1}^{T} k(t, t') \, ||x_{t'} - A x_{t'}||_2^2 + \lambda ||A||_1, \tag{3.17}$$

in which $k(t, t')$ computes a weight based on temporal proximity, diagonal entries of A are fixed at 0, $||A||_1$ indicates $\sum_i \sum_j |A_{ij}|$, and λ controls the trade-off between sparsity and self-regression error. We clamp the diagonal of A to 0 to force regressions in the form of Eq. 3.14. After estimating A^* according to Eq. 3.17, KELLER performs additional post-processing to make the estimated structure consistent with the assumption of an undirected GGM (i.e. A^* should be symmetric). Roughly speaking, this means inferring an edge between any pair of vertices (i, j) such that $A_{ij} \neq 0$ or $A_{ji} \neq 0$. An estimation similar to Eq. 3.17, but without the additional symmetrization, is used in [98] for directed networks.

Under typical conditions, the weighting kernel $k(t, t')$ in Eq. 3.17 renders the estimate of A_t^* approximately independent from observations at times remote from t. Thus, the information available when estimating each A_t^* is significantly reduced from that present in the full dataset X. Such informational profligacy seems undesirable when one considers that an observation sequence may come from a process which produces regularities in the structure of an optimal time varying GGM for the sequence. In the next subsection, we develop an algorithm that alleviates these deficiencies.

3.4.2 Using Basis Structures to Estimate Network Structures

We now transform the optimization in Eq. 3.17 similarly to the way in which we transformed locally weighted regression from Eq. 3.3 to Eq. 3.6. Succinctly:

we represent each locally adapted A_t^* using a linear combination of a set of k basis matrices $\hat{A} \triangleq \{A^1, ..., A^k\}$, with the diagonal of each A^i clamped to 0. The resulting approach revolves around the following optimization:

$$\hat{\beta}_t = \underset{\hat{\beta} \in \mathbb{R}^k}{\arg\min} \sum_{t'=1}^{T} k(t, t') \, ||x_{t'} - \sum_{i=1}^{k} \hat{\beta}^i A^i x_{t'}||_2^2 + \lambda \, ||\hat{\beta}||_1, \tag{3.18}$$

in which $\hat{\beta}^i$ is the i^{th} element of $\hat{\beta}$, $||\hat{\beta}||_1$ is an ℓ_1 regularization term, and λ controls the strength of regularization. Given $\hat{\beta}_t$, we define $A_t^* \triangleq \sum_{i=1}^{k} \hat{\beta}_t^i A^i$. The optimization in Eq. 3.18 involves a fixed set of basis matrices $\hat{A}$. But, what we really want is to jointly optimize the loss in Eq. 3.18 over all times $1 \leq t \leq T$, with respect to both the $\hat{\beta}_t \in \hat{B} \triangleq \{\hat{\beta}_1, ...\hat{\beta}_T\}$ and the $A^i \in \hat{A}$. By estimating $\hat{B}$ and $\hat{A}$ jointly, information extracted from the entire sequence can be used in the estimation of each A_t^*. This approach reduces variance by performing each temporally local estimation in k-dimensional $\hat{\beta}$-space instead of $\mathcal{O}(n^2)$-dimensional A-space. This approach reduces informational profligacy by allowing global information sharing through the learned basis structures $A^i \in \hat{A}$.

The desired joint optimization over $\hat{B}$ and $\hat{A}$ is easy to express in the form of Eq. 3.5. Let $g(\hat{\beta}) \triangleq \sum_{i=1}^{k} \hat{\beta}^i A^i$ for $A^i \in \hat{A}$. Next, we define $f(x, g(\hat{\beta})) \triangleq ||x - g(\hat{\beta})x||_2$ and we define $\ell(f|g, X, \hat{B})$ as:

$$\ell(f|g, X, \hat{B}) \triangleq \sum_{t=1}^{T} \sum_{t'=1}^{T} k(t, t') \, f(x_{t'}, g(\hat{\beta}_t))^2 + \lambda \sum_{t=1}^{T} r(\hat{\beta}_t), \tag{3.19}$$

in which $r(\hat{\beta}_t)$ represents an arbitrary sparsifying regularization penalty on $\hat{\beta}_t$. Finally, we express the full joint optimization as follows:

$$\hat{A}^* = \arg\min_{\hat{A}} \min_{\hat{B}} \sum_{t=1}^{T} \sum_{t'=1}^{T} \Bigg[k(t,t') \, ||x_{t'} - \sum_{i=1}^{k} \hat{\beta}_t^i A^i x_{t'}||_2^2 + \lambda_\beta \sum_{t=1}^{T} r(\hat{\beta}_t) + \lambda_A \sum_{i=1}^{k} ||A^i||_1 \Bigg], \tag{3.20}$$

in which we introduce an entry-wise ℓ_1 regularization term for each A^i, which enforces the structural sparsity assumption. Our method thus produces a set $\hat{A}^*$ of sparse network structures A^i, with which the true time varying network structure can be effectively approximated using only a sparse subset of the $A^i \in \hat{A}^*$.

The joint optimization in Eq. 3.20 is similar to the sparse coding objective:

$$A^* = \arg\min_{A \in \mathbb{R}^{n \times k}} \left[\min_{B} \sum_{i=1}^{m} \left(||x_i - A\beta_i||_2^2 + \lambda ||\beta_i||_1 \right) \right], \tag{3.21}$$

in which $B \triangleq \{\beta_1, ...\beta_m \, | \beta_i \in \mathbb{R}^k\}$, λ controls the trade-off between reconstruction accuracy and code sparsity, and the columns of A are constrained to unit norm. We can emphasize this similarity by introducing the concept of time varying *pseudo-dictionaries* $D_t \in \mathbb{R}^{n \times k}$, in which column i of D_t is defined as $A^i x_t$. Using the pseudo-dictionaries D_t, we can rewrite Eq. 3.20 as follows:

$$\hat{A}^* = \arg\min_{\hat{A}} \left[\min_{\hat{B}} \sum_{t=1}^{T} \left[\left(\sum_{t'=1}^{T} k(t,t') \, ||x_{t'} - D_{t'} \hat{\beta}_t||_2^2 \right) + \lambda_\beta r(\hat{\beta}_t) \right] \right], \tag{3.22}$$

in which we elide the sparsifying penalty on $A^i \in \hat{A}^*$ for notational brevity. From Eq. 3.22, it can be seen that the inner optimization over $\hat{B}$ in Eq. 3.20 can be treated as a collection of sparse coding problems. For our purposes, we set the

regularization term $\lambda_\beta r(\hat{\beta}_t)$ to:

$$\frac{\alpha\lambda_\beta}{2}||\hat{\beta}_t||_2^2 + (1-\alpha)\lambda_\beta||\hat{\beta}_t||_1, \text{ where } 0 \leq \alpha \leq 1, \tag{3.23}$$

which corresponds to elastic-net regularization [121]. We choose this regularizer to meet the assumptions required for the task-driven dictionary learning described in [69], which we will incorporate into our algorithm in the next subsection.

The analogy between our method and sparse coding leads naturally to a method for performing the joint optimization in Eq. 3.20. As in sparse coding, we can jointly optimize over $\hat{A}$ and $\hat{B}$ using an EM-like block coordinate descent procedure that alternates between optimizing $\hat{B}$ while holding $\hat{A}$ fixed and optimizing $\hat{A}$ while holding $\hat{B}$ fixed. These sub-problems are both convex and efficiently solvable. When optimizing $\hat{B}$ with $\hat{A}$ held fixed, we compute the optimal $\hat{\beta}_t$s via elastic-net regressions solved with the publicly available, highly optimized glmnet package [27]. When optimizing $\hat{A}$ with $\hat{B}$ held fixed, we compute the partial gradients of the objective in Eq. 3.22 w.r.t. the entries of each pseudo-dictionary D_t, and then backpropagate these partial gradients through the pseudo-dictionary formation process to get partial gradients w.r.t. each entry of each basis structure A^i. We symmetrize the partial gradient of Eq. 3.22 w.r.t. each A^i by setting $\partial A^i_{uv} \triangleq \frac{1}{2}(\partial A^i_{uv} + \partial A^i_{vu})$. We also set $\partial A^i_{uu} \triangleq 0, \forall u$ to keep the diagonal of each A^i clamped to 0. In the next subsection we refer to these (unsupervised) partial gradients as $\nabla_{A^i}\ell_u$. Using the computed gradients, we then take a single gradient descent step to update each A^i.

The full joint optimization process iterates between updating the $\hat{\beta}_i \in \hat{B}$ via the regression in Eq. 3.18 and performing a single gradient descent update of the entries in each $A^i \in \hat{A}$. We dynamically select the step size for gradient descent updates in each iteration by line search. We iterate between optimizing $\hat{A}$ and $\hat{B}$ until approximate convergence. We perform the iterative optimization using subsampled batches of the available observations, which yields a stochastic gradient descent approach to jointly optimizing Eqns. 3.20/3.22. The cost of optimizing $\hat{B}$ is linear in the cost of solving a low-dimensional sparse coding problem with glmnet, which is fast. The cost of optimizing $\hat{A}$ is linear in the number of basis matrices and quadratic in the dimension of x. Empirically, the optimization over $\hat{A}$ dominates computation time.

3.4.3 Supervised Basis Structure Learning

By adapting the work of [69], we now extend our algorithm to incorporate task-driven dictionaries of network structures. We consider the task of minimizing differentiable supervised loss functions that can be written as follows:

$$L_s(X, \hat{B}, \omega) \triangleq \sum_{t=1}^{T} \ell_s(\omega^\top \hat{\beta}_t, y_t) + \frac{\nu}{2}||\omega||_2^2, \tag{3.24}$$

where $\omega \in \mathbb{R}^k$, y_t is the target output at time t, and the $\hat{\beta}_t \in \hat{B}$ were produced to minimize Eq. 3.22. This includes any differentiable linear function of the sparse

structure codes $\hat{\beta}_t \in \hat{B}$.[4] For application to classification tasks, we choose the binomial deviance loss of logistic regression, i.e. for $y_t \in \{-1, +1\}$:

$$\ell_s(\omega^\top \hat{\beta}_t, y_t) \triangleq \log(1 + e^{-y_t \omega^\top \hat{\beta}_t}). \tag{3.25}$$

The crux of task-driven dictionary learning is to convert the readily available gradients of ℓ_s w.r.t. the structure codes $\hat{\beta}_t$ into gradients w.r.t. the pseudo-dictionaries D_t with which they were computed according to Eq. 3.22. These gradients w.r.t. D_t can then be easily converted into gradients w.r.t. the $A_i \in \hat{A}$, which is all we need for task-driven adaptation of the dictionary $\hat{A}$. Unfortunately, the optimization which produces the structure codes $\hat{\beta}_t$ makes backpropagating $\nabla_{\hat{\beta}_t} \rightarrow \nabla_{D_t}$ non-trivial. The authors of [69] show that if sparsifying regularization in the form of Eq. 3.23 is used when estimating each $\hat{\beta}_t$, then the gradient of the supervised loss ℓ_s w.r.t. the pseudo-dictionary D_t can be computed as follows:

$$\nabla_{D_t} \ell_s(\omega^\top \hat{\beta}_t, y_t) = -D_t \phi_t \hat{\beta}_t^\top + (x_t - D_t \hat{\beta}_t) \phi_t^\top, \tag{3.26}$$

in which $\phi_t \in \mathbb{R}^k$ is defined as follows:

$$\phi_{t:\Lambda} \triangleq (D_{t:\Lambda}^\top D_{t:\Lambda} + \alpha \lambda_\beta I)^{-1} \nabla_{\hat{\beta}_{t:\Lambda}} \ell_s(\omega^\top \hat{\beta}_t, y_t), \quad \phi_{t:V} \triangleq 0, \tag{3.27}$$

[4] More generally, we could use any loss that is differentiable w.r.t. the sparse structure codes $\hat{\beta}_t \in \hat{B}$. I.e. the methods in [69] can backpropagate arbitrary gradients through the code estimation process, assuming $\hat{\beta}_t$ is regularized by Eq. 3.23.

where Λ denotes the indices of non-zero entries in the sparse structure code $\hat{\beta}_t$, V indicates the complementary set of indices, and $\alpha\lambda_\beta$ is the ℓ_2 regularization weight from Eq. 3.23. $\phi_{t:\Lambda}$ indicates the entries of ϕ_t with indices in Λ and $\phi_{t:V}$ indicates the complementary set of entries. We use similar notation for selecting subsets of D_t and $\hat{\beta}_t$. Once we have gradients of ℓ_s w.r.t. each D_t (denoted by $\nabla_{D_t}\ell_s$), we can backpropagate them through the pseudo-dictionary formation process and sum them across times $1 \leq t \leq T$ to get gradients w.r.t. each $A^i \in \hat{A}$ (denoted by $\nabla_{A^i}\ell_s$).

Given unsupervised gradients $\nabla_{A^i}\ell_u$, computed as described at the end of Sec. 3.4.2, and supervised gradients $\nabla_{A^i}\ell_s$ computed as described above, we define the final gradients for stochastic gradient descent on the combined unsupervised/supervised objective as follows:

$$\partial A^i \triangleq \gamma\nabla_{A^i}\ell_u + (1-\gamma)\nabla_{A^i}\ell_s, \quad \text{with } 0 \leq \gamma \leq 1, \tag{3.28}$$

where γ is a mixing parameter controlling the relative influence of supervised and unsupervised learning. As before, we enforce symmetry and clamp diagonals to zero prior to using these gradients for basis updates.

3.5 Synthetic Network Analysis

We now presents tests based on simulated observation sequences that illustrate how our algorithm can recover recurring patterns in the structure of a time varying network. We generated observation sequences for these tests by drawing the observation x_t at time t from a normal distribution $\mathcal{N}(0, \Sigma_t)$, in which Σ_t was a convex combination of four basis covariance matrices. I.e. $\Sigma_t \triangleq \sum_{i=1}^{4} \alpha_t^i \Sigma^i$, with

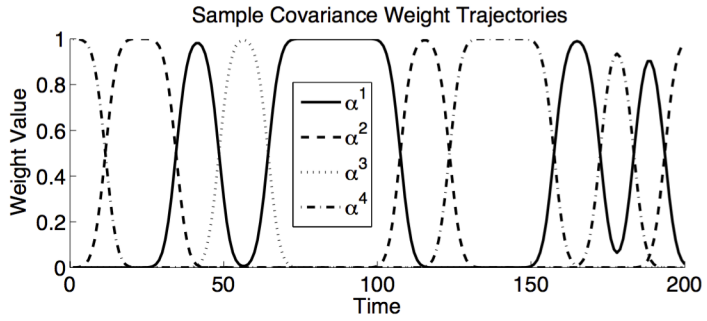


Figure 3–2: Trajectories for the weights α_t^i used in the tests described in Sec. 3.5. Note the smooth transitions between "structural regimes", which cause problems for methods expecting abrupt structural changes.

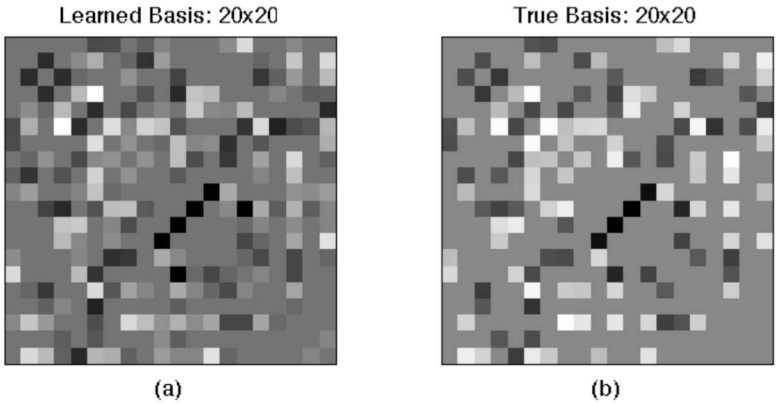


Figure 3–3: The left panel is the best-match learned basis for the true basis on the right. We took this example directly from one of the sequences used in our tests. This pairing represents a match quality within one standard deviation of the mean for this network size. Gross structural similarities are readily apparent. The diagonal of the true basis has been removed to facilitate comparison.

$\sum_{i=1}^{4} \alpha_t^i = 1$ and $0 \leq \alpha_t^i \leq 1$. We generated smooth trajectories for the α_t^i. We show an example set of trajectories in Fig. 3–2. We generated each basis matrix Σ^i by symmetrically removing two thirds of the off-diagonal entries – we preferentially removed entries with small magnitude – from a random covariance matrix with eigenvalues uniformly distributed between 0 and 1. We then rescaled the diagonal entries to ensure positive definiteness. The right panel of Fig. 3–3 shows an example of the sparse basis structures used for generating our observation sequences. We generated observations ranging from 10-dimensional to 40-dimensional. For each tested dimensionality, we generated 25 sequences of 5000 observations, with the first 3000 reserved for training and the last 2000 reserved for testing. We generated each sequence using different basis matrices Σ^i and different α_t^i trajectories. We computed average results using all 25 sequences.

Methods based on Eq. 3.17 are much better suited for these tests than methods expecting abrupt "change point" structure (such as those in [1] and [52]). Hence, we compared three methods for estimating time varying network structure: the locally weighted ℓ_1-regularized self-regression described in Eq. 3.17, the same self-regression followed by projection of each estimated structure onto the leading principal components of all structures estimated for the training set, and our full method for jointly learning basis structures and structure codes.

The basic self-regression approach used in our tests can be considered equivalent to KELLER [97]. Projection of each estimated structure A_t^* onto the leading principal components of all $\{A_1^*, ..., A_{3000}^*\}$ is itself novel, and can be seen as non-adaptive approximation to our method. When executing our method, we initialized

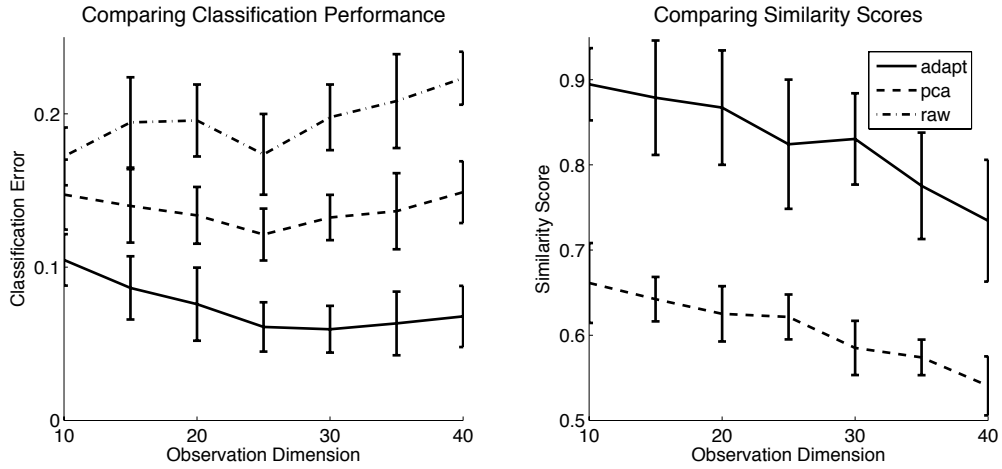


Figure 3–4: Left plot: mean classification error for a binary classification task as a function of input dimension. Our adaptive basis learning method (label: adapt) generates better features than the self-regression-based (label: raw) and PCA-based (label: pca) approaches. Right: the fidelity of the principal structures produced by using PCA on the A_t^* computed according to Eq. 3.17 and the basis structures produced by adapting these principal structures to minimize Eq. 3.20. The bases produced by our method more accurately capture the generative structure underlying the test sequences.

the structure dictionary $\hat{A}$ using the "principal structures" computed from the $\{A_1^*, ..., A_{3000}^*\}$ produced by basic self-regression. For our tests, we used six principal structures with the PCA-based method and learned six basis structures when applying our full method.

We measured test performance for a classification task in which the class of each x_t was determined as follows: $y_t \triangleq 1$ if $\alpha_t^1 + \alpha_t^2 \geq \alpha_t^3 + \alpha_t^4$ and $y_t \triangleq -1$ otherwise. In addition to measuring performance on this classification task, we also estimated a similarity score between the sets of estimated structures and the true precision matrices underlying each sequence.

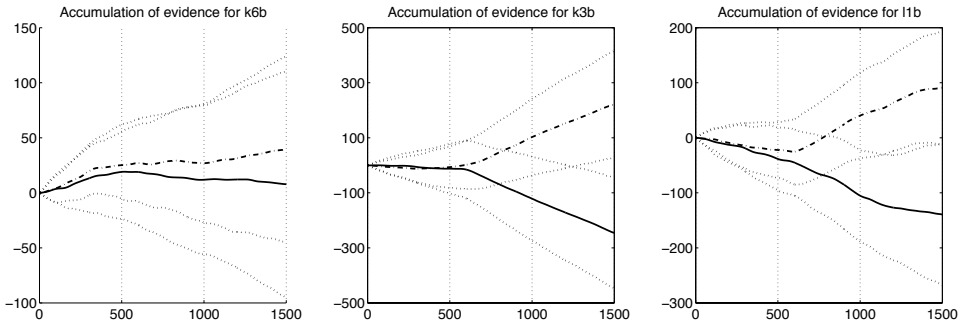
Classification was performed based on the locally adapted parametrizations produced by each method for each time t. For basic self-regression this was a matrix A_t^*. For the PCA-based method, this was the same matrix A_t^* projected onto the 6 leading principal structures found by running PCA on $\{A_1^*, ..., A_{3000}^*\}$. For our full method, this was the sparse structure code $\hat{\beta}_t$ produced by optimizing the objective in Eq. 3.22. We used these locally adapted parametrizations as features for an ℓ_2-regularized logistic regression classifier. The target class was determined by y_t. Fig. 3–4 presents the results. From these tests, we conclude that our method can learn basis structures and structure codes that capture useful information about regularities in time varying sparse network structure.

We measured similarity between the learned bases and the true bases using a form of pairwise matrix correlation. First we set the diagonal entries of each matrix to zero, then we set their off-diagonal entries to zero mean and unit norm. After this normalization, we "vectorized" each matrix and computed the

dot product between the resulting vectors. Roughly speaking, this computes something like a cosine distance between the matrices. This measure ranges from -1 to 1, with values closer to 1 indicating greater similarity. For each sequence and each method, we used the similarity score to find the best match to each $(\Sigma^i)^{-1}$ from among the set of basis structures produced by that method. We then averaged best match scores for each method over both bases and sequences, which produced an overall score for each dimensionality. Fig. 3–4 shows the similarity scores achieved by the PCA-based method and by our method, with the bases produced by our method consistently displaying greater similarity to the true bases than those produced by PCA alone. Fig. 3–3 shows a typical example of a best match produced by our method during these tests. The learned basis is clearly qualitatively similar to the true basis.

3.6 BCI EEG Analysis

To test the real-world performance of our algorithm, we applied it to the analysis of EEG data from a Brain Computer Interface (abbr. BCI) motor imagery experiment available as task 3a from BCI competition III [87, 14]. In this task, the objective is to infer the motor action visualized, but not actually performed, by a subject during a set of test trials. A number of trials, each comprising a sequence of EEG readings and a label indicating the visualized motor action, is available for training each subject-specific model prior to testing. In each trial, a cue indicating a motor action is given to the subject, after which the subject visualizes that action for several seconds. Cortical activity during each trial was measured by a

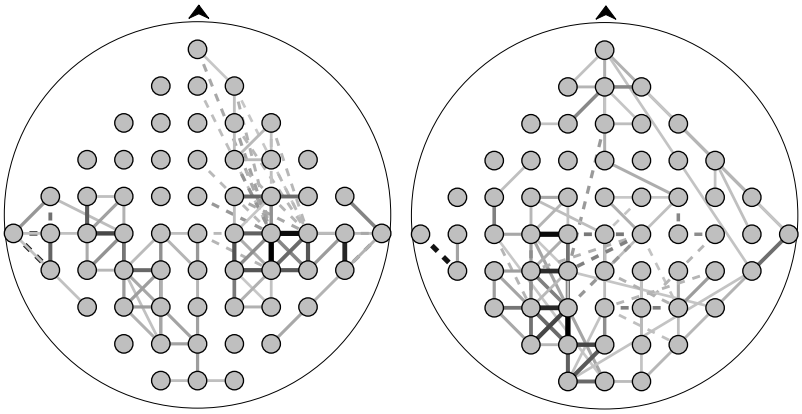


(a) Sub: k6b (b) Sub: k3b (c) Sub: l1b

Figure 3–5: Behavior of the evidence accumulation classifier learned using features produced by our algorithm, averaged over all trials for each subject. As indicated by Table 3–1, subject k6b proved difficult for both our method and RCSP. With the other two subjects, the discriminative capacity of our bases can be clearly seen in the rapid bifurcation of their induced classifier response after the cue time at $t = 500$. Lines trending upwards indicate left hand trials and lines trending downwards represent right hand trials.

set of 60 electrodes placed on the scalp, taking measurements at 250Hz. Readings from these electrodes provided the x_t for our algorithm.

We used left hand and right hand trials from this dataset for the subjects l1b, k3b, and k6b. Several trials from each subject were discarded due to significant artifacts, as measured by deviation from a Gaussian model of the mean behavior of the joint set of trials for a subject. We also applied a whitening transform $VD^{-\frac{1}{2}}V^\top$ to each subject's data prior to analysis, where D was a diagonal matrix containing the eigenvalues of the data and the columns of V were the corresponding eigenvectors. Roughly speaking, this whitening removed the "mean network structure", which might otherwise have dominated the estimation process. We set kernel widths and regularization weights for the optimization in Eq. 3.20 uniformly for all subjects and trials, following a brief manual search.



(a) k3b-b3 (b) k3b-b9

Figure 3–6: This figure illustrates bases three (on left) and nine (on right) learned for subject k3b. Basis three had the largest weight in the instant-scale classifier for this subject, while basis nine had the fourth largest. The line thickness in this figure indicates the magnitude of edge weights in the sparse network structure represented by each basis. Dashed lines correspond to reduced covariance between points relative to the mean covariance, while solid lines indicate increased covariance. Both bases show some degree of lateral localization in the structural variations they represent. The decoupling between right motor cortex and right frontal cortex in basis three is particularly striking.

	L1b	K6b	K3b
RCSP	0.100 (0.056)	0.363 (0.118)	0.103 (0.048)
ADAPT	**0.041** (0.053)	0.330 (0.098)	**0.056** (0.033)
JOINT	0.052 (0.052)	**0.253** (0.080)	0.063 (0.039)

Table 3–1: Classification error means and standard deviations for each subject in a set of BCI motor imagery experiments for Regularized Common Spatial Patterns (RCSP), our algorithm (ADAPT), and a classifier based on the combined set of features from RCSP/ADAPT (JOINT). Best results for each subject are in bold. Our algorithm outperforms RCSP, which is currently used in practice. The classifier built on the joint feature set produces the best performance when averaged across all subjects. Differences between JOINT and RCSP are significant in all cases. Differences between JOINT and ADAPT are only significant in the case of subject K6b.

We learned a dictionary of 20 sparse basis structures for each subject using our algorithm in an unsupervised fashion (i.e. $\gamma = 1$). We then performed 20 rounds of randomized cross-validation in which we split the trials for each subject 4/1 into training/test sets. We trained three classifiers in each round of cross-validation: a classifier based on the $\hat{\beta}_t$ inferred by our algorithm on the test set after a period of supervised basis updates on the training set with $\gamma = 0.75$, a classifier based on the output of a set of 20 RCSP filters [67], and a classifier based on concatenating our features with the RCSP features. We selected parameters for RCSP to maximize expected performance across all subjects.

For our algorithm, we built a trial-scale classifier by considering the structure codes $\hat{\beta}_t$ and class labels y_t for individual times t in all training trials as feature/label pairs for training an ℓ_2-regularized logistic regression classifier. This instant-scale classifier was thus trained to classify individual times t, rather than entire trials. Given the encoding of a particular trial in terms of a set of $\hat{\beta}_t$, we

computed a trial-scale classification by accumulating (i.e. summing) the output of the instant-scale classifier over the first three post-cue seconds of the trial. After this evidence accumulation phase, we determined the final trial-scale classification by the sign of the accumulated instant-scale outputs. We trained the RCSP filters as described in [67], and then used the squared responses of these filters to the observations x_t as input features to an instant-scale classifier, which we trained as described for our algorithm. We also trained analogous instant-scale and trial-scale classifiers using the combined features provided by our algorithm and the RCSP filters. Classification results for each subject are shown in Table 3–1, and a visual representation of the evidence accumulation process based on our features is shown in Fig. 3–5. Trial-scale classifiers constructed as we've just described have the advantage of being amenable to "early exit", in the spirit of decision making using drift-diffusion processes.

These tests show that our approach can produce informative features in a real-world scenario, with the results suggesting that our features may be used to supplement, rather than replace, the widely-used RCSP features. This observation is supported by the results in Table 3–1, which show that combining our features with those produced by RCSP led to minor reductions in performance for subjects L1b and K3b, but significantly improved performance for subject K6b.

3.7 Discussion

This chapter introduced a problem formulation based on learning transformed parametrizations of multiply parameterized models. Starting from this formulation, we developed a novel algorithm for learning representations of sparse

structure in time varying networks with recurring structural motifs. We used tests on synthetic data to show that our algorithm behaves as desired under suitable conditions, while an application to BCI EEG data showed the potential value of our algorithm in more realistic conditions.

Our approach should extend readily to other types of tasks, such as analysis of time varying weather and traffic patterns. It is also worth investigating alternative parameter transformation methods. Our algorithm is readily extensible to the estimation of time varying structure in Dynamic Bayesian Networks – for which an algorithm without parameter sharing through time was previously presented in [98]. Extending our approach to work with non-linear parameter transformations may significantly increase its power.

CHAPTER 4
Sample-based Approximate Regularization

4.1 Motivation

The previous chapter investigated methods for controlling variation among multiple instances of a simple base model. The way we gained this control was by placing several constraints on the parametrization of each instance of the base model. Specifically, we restricted all of the parametrizations to a shared linear subspace of the base model's full parameter space. We also regularized the axes of the shared subspace to be sparse in the full space, and regularized each base model instance's parametrization to reside in a sparse subspace of the shared subspace. These methods for controlling variation are all most easily interpreted, and justified, in terms of their effects on the estimated parameters.

In this chapter, motivated by the intuition that a model's *behavior* is often of more interest than its particular parameters, we develop a method for regularizing a common class of models based on how their response to an input changes as the input changes. Fortunately, for this model class, the way we want to regularize model behavior can be translated into direct regularization of model parameters.

Many approaches to supervised learning can be interpreted as a process of first transforming an input x into a set of features $\phi(x)$ and then computing an output $\phi(x)^\top \mathbf{w}$ as a linear function of those features. For example, SVMs first transform an example $x \in \mathcal{X}$ into a set of features $\phi(x) \in \mathbb{R}^p$ by computing

$\phi(x) \triangleq [k(x, x_1), \cdots, k(x, x_p)]$, where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a reproducing kernel and each x_i is a point in $\mathcal{X}$. Typically, the points $\{x_1, ..., x_p\}$ used to compute $\phi(x)$ are taken from the training set. The SVM then computes an output $f(x) \triangleq \phi(x)^\top \mathbf{w}$ as a linear function of $\phi(x)$ [88]. Similarly, the hidden layers of a feedforward neural network compute a non-linear feature transform $x \to \phi(x)$, after which a final layer computes a linear function $f(x) \triangleq \phi(x)^\top \mathbf{w}$.

A major reason for the success of SVMs is arguably their use of regularization that is "natural" in the RKHS tied to their kernel. For some choices of kernels, this regularization has a smoothness inducing interpretation [88].[1] For example, the RKHS norm induced by the popular Gaussian RBF kernel penalizes all orders of derivatives of the learned function [117]. Spline-based methods, which are ubiquitous in statistics but less common in machine learning, also rely on smoothness inducing, derivative-based penalties. In particular, for univariate inputs and additive models, a second-order derivative penalty can be applied exactly in the nonparametric setting, leading to cubic smoothing splines [109]. But, this exact penalty quickly becomes intractable as the training set grows or as one attempts to model interactions between multiple variables. While attempts have been made to produce computationally efficient approximations of spline-like penalties [25, 113], full spline-based methods generally scale unfavorably.

[1] Reasonable definitions for smoothness abound. Roughly speaking, for our purposes, we consider regularization smoothness inducing if it directly acts to control the derivatives of a function.

In this chapter, we present a method for efficiently approximating a general class of derivative-based penalties. We call this method Sample-based Approximate Regularization (SAR). Our approach applies to any hypothesis space that is linearly parameterized, i.e., in which the input x is transformed into a feature space $\phi(x)$ and the output is then computed as $\phi(x)^\top \mathbf{w}$. Hypothesis spaces of this type include SVM-style approximators, feedforward neural networks, and various other types of function approximators that use features suited to particular application domains, e.g. SIFT, MFCC, etc [68, 21]. Based on the success of derivative-based penalties in the RKHS and spline settings [76], and on the empirical success of domain-specific features, it seems desirable to obtain derivative-based regularizers that can work with a wide range of features.

Our SAR method can be used to augment or replace the standard ℓ_1 and ℓ_2 regularizers that are commonly used with linear function approximators that do not, or cannot, use the "kernel trick". Conveniently, the regularizers produced by SAR are of the Tikhonov-type (i.e., $J(f_{\mathbf{w}}) = \mathbf{w}^\top \Sigma \mathbf{w}$ for some Σ), and can thus be applied efficiently with standard software. The sample complexity of SAR depends only loosely on the complexity of the features ϕ, and not at all on the size of the training set. SAR thus improves on the cost of spline-based approaches to regularizing derivatives.

We begin the rest of this chapter by presenting a general formulation for smoothness inducing, derivative-based regularizers (Section 4.3). We then present our approach for approximating these regularizers efficiently (Section 4.4), and analyze its theoretical properties (Section 4.5). We prove that the regularizers

produced by SAR converge efficiently to the exact penalties they're designed to approximate, and we bound the loss of a SAR-regularized regression estimator compared to the loss of an estimator shaped by the exact regularizer approximated by SAR.[2] Finally, we present empirical results illustrating the power of the proposed approach in both synthetic and real-world domains (Section 4.6).

4.2 The SAR Approach

We begin this section by describing two functionals, based on first and second-order directional-derivatives, that provide measures of function "smoothness" suited for use as regularizers. We show how to approximate these functionals using a Monte Carlo approach based on finite-difference estimates of directional derivatives. We then extend our approach to produce regularizers for controlling higher-order derivatives. Section 4.5 provides rigorous upper bounds on the error of our approximation process and its rate of convergence.[3]

[2] These analytical results are the work of Amir-massoud Farahmand, who was a co-author of the paper in which this work was originally presented [3].

[3] The theoretical results in Section 4.5 are for first-order regularization, but similar results should hold for higher orders. Our theoretical results address the correctness of our approximation method, which is orthogonal to the correctness of the bias induced by the regularizers we approximate. So, while concerns based on, e.g., the degeneracy of solutions derived from first-order Laplacians in the non-parametric setting, are worth considering, their context is disjoint from that of our theory.

4.3 Smoothness-inducing Regularizers

Consider a function $f : \mathcal{X} \to \mathbb{R}$ and a measure $\nu \in \mathcal{M}(\mathcal{X})$. If $f'(x)$ exists and is $L_2(\nu)$-integrable, then for $\mathcal{X} \triangleq \mathbb{R}$ a natural measure of the smoothness of f is:

$$\int_{\mathcal{X}} |f'(x)|^2 \mathrm{d}\nu. \tag{4.1}$$

One typically extends Eq. 4.1 to multi-dimensional domains $\mathcal{X} \triangleq \mathbb{R}^d$ by integrating the squared norm of the gradient. We propose instead a more general form, written as follows:

$$J_1(f) \triangleq \int_{\mathcal{X}} \oint_{S_x} (s^\top \nabla f)^2 \mathrm{d}s_x \mathrm{d}\nu. \tag{4.2}$$

The inner integral is over the surface S_x of a hyper-sphere centered at x, according to a location-dependent measure over directions $s_x \in \mathcal{M}(S_x)$. Each $s \in S_x$ corresponds to a unit-length vector pointing away from x in some direction. If s_x is set to a uniform distribution over the unit hyper-sphere for all $x \in \mathcal{X}$, then $J_1(f)$ is proportional to the integrated squared norm of $\nabla_x f$, due to the linearity of the dot-product. If s_x is the set of delta functions on the coordinate vectors of $\mathcal{X}$, $J_1(f)$ penalizes the integrated squared norm of $\nabla_x f$ exactly, as in the typical multi-dimensional extension of Eq. 4.1.

The generalized derivative penalty $J_1(f)$ allows flexibility in assigning location-dependent importance to the variation of f along particular directions. It is also highly amenable to sample-based approximation.

Another reasonable measure of the smoothness of f uses its second-order derivatives. In one dimension, if $f''(x)$ exists and is $L_2(\nu)$-integrable, then we can

write:

$$\int_{\mathcal{X}} |f''(x)|^2 \mathrm{d}\nu, \tag{4.3}$$

which gives the standard penalty used in, e.g., cubic smoothing splines [109]. We extend the penalty in Eq. 4.3 to multiple dimensions as follows:

$$J_2(f) \triangleq \int_{\mathcal{X}} \oint_{S_x} \left(s^\top (H_x f) s\right)^2 \mathrm{d}s_x \mathrm{d}\nu, \tag{4.4}$$

where $s \in S_x$ are again unit-length vectors jointly covering all directions pointing away from x, and $H_x f$ is the Hessian of f evaluated at x. When s_x is uniform over S_x, $J_2(f)$ penalizes the squared Frobenius norm $||H_x f||_F^2$ (integrated w.r.t. ν), which provides regularization that has proven useful in previous work [82, 49]. While the integral over quadratic forms in Eq. 4.4 might seem odd at first, as with Eq. 4.2, it encompasses a broad range of regularizers – due to flexibility in the choice of ν and s_x – and is highly amenable to sample-based approximation.

4.4 The SAR Method

The goal of our approach is to efficiently approximate regularizers in the form of Eqns. 4.2 and 4.4. The functionals J_1 and J_2 both involve integrating some quantity w.r.t. measures ν and s_x. SAR efficiently approximates the integrands in J_1 and J_2 using finite-difference estimates of directional derivatives and approximates the integrals using a Monte Carlo approach based on samples from ν and s_x. We call methods to sample from ν *point samplers* and methods to sample from s_x *direction samplers*.

We focus on linearly-parameterized functions $f_{\mathbf{w}}(x) \triangleq \phi(x)^\top \mathbf{w}$, where $\phi : \mathcal{X} \to \mathbb{R}^p$ is a fixed feature transform whose components are one-dimensional

Algorithm 1 SAR($\tilde{p}_x$, $\tilde{p}_{s_x}$, N, ϕ, i, ϵ)

1: $\tilde{\Sigma}_i :=$ zero matrix of size $p \times p$.
2: **for** $j = 1$ to N **do**
3: Sample X_j from $\tilde{p}_x$.
4: Sample S_j from $\tilde{p}_{s_{X_j}}$.
5: Compute $\delta_i^\epsilon(X_j, S_j, \phi)$ (see: Eq. 4.11 for defn.)
6: $\tilde{\Sigma}_i := \tilde{\Sigma}_i + \delta_i^\epsilon(X_j, S_j, \phi)\delta_i^\epsilon(X_j, S_j, \phi)^\top$.
7: **end for**
8: **return** $\frac{1}{N}\tilde{\Sigma}_i$.

measurable functions $\{\varphi_i\}_{i=1}^p$. We denote the vector of trainable parameters as $\mathbf{w} \in \mathbb{R}^p$. We denote the function space defined by the span of ϕ as $\mathcal{F}$.

Given a point sampler $\tilde{p}_x$, a direction sampler $\tilde{p}_{s_x}$, a sample size N, and a derivative order i, Algorithm 1 produces a matrix $\tilde{\Sigma}_i$. Using this matrix, we define the i^{th}-order SAR cost functional as $\tilde{J}_i(f_\mathbf{w}) \triangleq \mathbf{w}^\top \tilde{\Sigma}_i \mathbf{w}$, for functions $f_\mathbf{w} \in \mathcal{F}$. To simultaneously regularize multiple derivative orders, their corresponding $\tilde{\Sigma}_i$ can be combined via element-wise summation.

Once an approximate regularizer $\tilde{\Sigma}_i$ has been produced according to Algorithm 1, any method for estimating Tikhonov-regularized linear models can be applied. The computational cost of SAR comes from lines 3-6 of Algorithm 1. Assuming reasonably efficient point/direction samplers $\tilde{p}_x/\tilde{p}_{s_x}$, the feature extraction in line 5 and the outer products in line 6 dominate the cost of SAR. If the expected cost of computing $\phi(x)$ is c_ϕ, the target derivative order is i, and $\phi(x) \in \mathbb{R}^p$, then line 5 costs $(i+1)c_\phi$ per sample and line 6 costs p^2 per sample. Lines 3-6 each execute N times when using N samples to compute $\tilde{\Sigma}_i$. Depending on c_ϕ and p, either line 5 or 6 may dominate the overall cost.

We now describe approaches for approximating the relevant directional derivatives and for constructing samplers $\tilde{p}_x/\tilde{p}_{s_x}$.

4.4.1 Approximating Directional Derivatives

For functions $f_{\mathbf{w}} \in \mathcal{F}$, the first-order forward finite difference in direction s is given by $\frac{1}{\epsilon}\mathbf{w}^\top(\phi(x+\epsilon s) - \phi(x))$. Thus, we can write:

$$\langle \nabla_x f_{\mathbf{w}}, s\rangle^2 \approx \mathbf{w}^\top \delta_1^\epsilon(x,s,\phi)\delta_1^\epsilon(x,s,\phi)^\top \mathbf{w}, \tag{4.5}$$

for which we define the first-order *finite difference vector*:

$$\delta_1^\epsilon(x,s,\phi) \triangleq \frac{1}{\epsilon}(\phi(x+\epsilon s) - \phi(x)). \tag{4.6}$$

For a point x and direction s, the term $s^\top(H_x f)s$ in Eq. 4.4 is equivalent to the second-order directional derivative of f, at x, in direction s. This has a finite difference approximation:

$$s^\top(H_x f)s \approx \frac{f(x) - 2f(x+\epsilon s) + f(x+2\epsilon s)}{\epsilon^2}. \tag{4.7}$$

For $f_{\mathbf{w}} \in \mathcal{F}$, the square of this term is given by:

$$(s^\top(H_x f_{\mathbf{w}})s)^2 \approx \mathbf{w}^\top \delta_2^\epsilon(x,s,\phi)\delta_2^\epsilon(x,s,\phi)^\top \mathbf{w}, \tag{4.8}$$

for which we define the second-order finite difference vector:

$$\delta_2^\epsilon(x,s,\phi) \triangleq \frac{1}{\epsilon^2}(\phi(x) - 2\phi(x+\epsilon s) + \phi(x+2\epsilon s)). \tag{4.9}$$

Based on Eqns. 4.5 and 4.8, the key to SAR is that the integrals in the regularization functionals J_1 and J_2 can be approximated by:

$$\mathbf{w}^\top \left(\int_{\mathcal{X}} \oint_S \delta_i^\epsilon(x, s, \phi) \delta_i^\epsilon(x, s, \phi)^\top \mathrm{d}s_x \, \mathrm{d}\nu \right) \mathbf{w}, \tag{4.10}$$

which uses finite difference vectors to regularize derivatives of order $i \in \{1, 2\}$. The double integral in Eq. 4.10 is straightforward to approximate via Monte Carlo.

To regularize higher-order derivatives with SAR, only the finite difference vectors used in Eq. 4.10 need to be modified. We can regularize derivatives of arbitrary order i using the finite difference vectors defined recursively as:

$$\delta_i^\epsilon(x, s, \phi) = \sum_{j=0}^{i} (-1)^j \binom{i}{j} \frac{\phi\left(x + (i - j)\epsilon s\right)}{\epsilon^i}. \tag{4.11}$$

When regularizing a single order i with fixed ϵ, the denominator ϵ^i in Eq. 4.11 can be ignored, as it is constant for all δ_i^ϵ. In this case, numerical precision (for $\epsilon^i \to 0$) is not an issue. A similar idea can be applied when regularizing across multiple orders. We note that we did not encounter any numerical problems in the experiments.

4.4.2 Sampling from ν and s_x

We now describe concrete methods to sample points and directions from ν and s_x. Suppose we are given a set $\mathcal{D}_n \triangleq \{X_1, X_2, \ldots, X_n\}$ of "training" observations

Algorithm 2 FuzzyPointSampler($\mathcal{D}_n$, N, $\mathcal{L}$).

1: **for** $j = 1$ to N **do**
2: Sample X_j from $\mathcal{D}_n$ uniformly at random.
3: Sample a direction S_j uniformly at random.
4: Sample a perturbation length ϵ_j from $\mathcal{L}$.
5: Add $\tilde{X}_j = X_j + \epsilon_j S_j$ to $\mathcal{D}'_N$.
6: **end for**
7: **return** $\mathcal{D}'_N$.

$X_i \in \mathbb{R}^d$, drawn from the source distribution $p(x)$. We will approximate ν using N samples, contained in a set $\mathcal{D}'_N$.[4]

A natural way to construct the sampler is to draw values from an approximation to $p(x)$ obtained by applying small stochastic perturbations to the existing points $\mathcal{D}_n$. This approach is (loosely) motivated by the manifold/cluster assumption underlying most work on semi-supervised learning. To define this sampler, first define $\mathcal{L}$ to be a distribution over lengths, which will determine the degree of "smoothing" to apply during sampling. Given $\mathcal{L}$, Algorithm 2 samples from an empirical approximation to $p(x)$ convolved with the isotropic distribution with length distribution $\mathcal{L}$. We only require a length distribution because the distribution we convolve with $p(x)$ is isotropic.

We also consider a second point sampler which samples approximately from the uniform distribution over $\mathcal{X}$, which mimics the distribution implicit in the RKHS regularization accompanying Gaussian RBFs. As defined in Algorithm 3,

[4] In the supervised learning setting, $\mathcal{D}_n$ contains label information as well, but we ignore it in the process of generating $\mathcal{D}'_N$. In a semi-supervised setting, we can also use unlabelled data.

Algorithm 3 BlurryBoxSampler($\mathcal{D}_n$, N, $\mathcal{L}$)

1: Compute the minimal bounding box for the $\mathcal{D}_n$.
2: **for** $j = 1$ to N **do**
3: Sample X_j uniformly from within the box.
4: Sample a direction S_j uniformly at random.
5: Sample a step length ϵ_j from $\mathcal{L}$.
6: Add $\tilde{X}_j = X_j + \epsilon_j S_j$ to $\mathcal{D}'_N$.
7: **end for**
8: **return** $\mathcal{D}'_N$.

this sampler samples from a uniform distribution over the smallest axis-aligned box enclosing D_n convolved with the isotropic distribution with length distribution $\mathcal{L}$. We add the smoothing (via convolution with $\mathcal{L}$) to mitigate potential "boundary effects" arising from the choice of axis-aligned box.

We consider two methods to sample directions from s_x. The first is to sample a unit direction uniformly at random. The second is to sample a unit direction uniformly at random, transform it by some matrix, and then rescale it to unit length. The first method produces a regularizer that penalizes derivatives in all directions equally, and the second biases the penalty based on the eigenvectors and eigenvalues of the transform matrix. Developing direction samplers with location-dependent biases, e.g., to emphasize invariance w.r.t. small translations/rotations in an object recognition task, may also prove useful.

4.4.3 Relationship with Manifold Regularization

Some work related to SAR, found in the manifold learning literature, uses the data distribution to define data-dependent regularizers. For instance, [18] defines a density-based regularizer in the form of Eq. 4.2 with a uniform measure over directions. However, their practical implementation was restricted to a first-order

derivative-based regularizer using Gaussian basis functions. This restricted form was due to their demand for an analytically tractable regularizer. SAR can be used to regularize higher-order derivatives, and can work with a far broader class of features. These benefits come from removing the requirement for an analytically tractable integral, and using Monte Carlo integration and approximate derivatives instead.

When data belongs to a low-dimensional manifold, a common choice of regularizer is to use the norm of the Laplace-Beltrami operator on the manifold. However, this norm and the operator on which it is based cannot be computed analytically in most cases, so sample-based approximations are used, e.g. the graph Laplacian operator [120, 8].[5] SAR is more general and is not designed with the goal of approximating the Laplace-Beltrami-based regularizer. Though, if one knew the tangent space of the data manifold for each point in the input domain $\mathcal{X}$, then one could approximate manifold regularization by using SAR with a direction sampler that was biased to sample directions tangent to the data manifold.

4.5 Theoretical Analysis

The goal of this section is two-fold. First, we study the behaviour of a SAR-based regularized least-squares regression estimator (Theorem 2). Second, we focus on the convergence behaviour of the sample-based approximate regularizer $\tilde{J}_N(f)$

[5] As discussed by [72], using the first-order derivative is not appropriate for high-dimensional input spaces, so one might also need to use higher-order (manifold) derivatives [49, 119].

to the desired regularizer $J(f)$. We provide two results, one in the form of the supremum of the empirical process (Theorem 3) and the other in the form of the supremum of the modulus of continuity of the empirical process (Theorem 4). For simplicity, we only study the 1^{st}-order derivative-based regularizer, when using central finite differences for SAR. The results in this section were developed by Amir-massoud Farahmand, and further details can be found in the supplemental material of the original paper [3].

Let us first define some notations. The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ is denoted by $\nabla f(x)$. We denote the central difference approximation of the gradient by $(\triangle_\varepsilon f)(x) \triangleq [(\triangle_\varepsilon f)_1(x) \cdots (\triangle_\varepsilon f)_d(x)]$ with $(\triangle_\varepsilon f)_i(x) \triangleq \frac{f(x+\varepsilon e_i)-f(x-\varepsilon e_i)}{2\varepsilon}$, where e_i are the unit coordinate vectors and ε is the finite difference step size.

Given a probability distribution $\nu \in \mathcal{M}(\mathcal{X})$, the desired 1^{st}-order derivative-based regularizer[6] is $J(f) \triangleq \int \|\nabla f(x)\|^2 \, \mathrm{d}\nu(x)$. Given N samples $\mathcal{D}'_N \triangleq \{X'_1, \ldots, X'_N\}$ with $X'_i \overset{\text{i.i.d.}}{\sim} \nu$, we define the sample-based approximate regularizer as: $\tilde{J}_N(f) \triangleq \frac{1}{N} \sum_{i=1}^N \|\triangle_\varepsilon f(X'_i)\|^2$. We also define $J_N(f) \triangleq \frac{1}{N} \sum_{i=1}^N \|\nabla f(X'_i)\|^2$. Note that for $f_{\mathbf{w}} \in \mathcal{F}$, we have $J(f_{\mathbf{w}}) = \mathbf{w}^\top \Sigma \mathbf{w}$ using the true Grammian

[6] If $\mathcal{X}$ is a proper open subset of $\mathbb{R}^d$, $(\triangle_\varepsilon f)_i(X')$ may not be defined for some samples X' close to the boundary of $\mathcal{X}$, (because one side can be outside the domain). If we ensure that $\mathrm{supp}(\nu)$ is at least ε away from the boundary in the $\ell\infty$-norm, all the results hold with $\mathcal{X} \subset \mathbb{R}^d$ instead of $\mathcal{X} = \mathbb{R}^d$.

$\Sigma \triangleq \int \sum_{i=1}^{d} \frac{\partial \phi(x)}{\partial x_i} \frac{\partial \phi(x)}{\partial x_i}^\top \mathrm{d}\nu(x)$. Similarly, we have $\tilde{J}_N(f_{\mathbf{w}}) = \mathbf{w}^\top \tilde{\Sigma}_N \mathbf{w}$ using the approximate empirical Grammian $\tilde{\Sigma}_N \triangleq \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{d} (\triangle_\varepsilon \phi)_j^\top (X_i')(\triangle_\varepsilon \phi)_j (X_i')$.[7] For a fixed $L > 0$, the truncation operator $\beta_L : \mathcal{F} \to \mathcal{F}$ is defined as $(\beta_L f)(x) \triangleq f(x)$ when $|f(x)| \leq L$ and $(\beta_L f)(x) \triangleq \operatorname{sgn}\big(f(x)\big) L$ otherwise. I.e., the truncation operator bounds the maximum magnitude of $f(x)$.

The regression setup is as follows. Let $\mathcal{D}_n \triangleq \{(X_i, Y_i)\}_{i=1}^n$ be a dataset with $X_i \overset{\text{i.i.d.}}{\sim} \mu$. Assume that the probability distribution generating the data is such that $|Y| \leq L$ (almost surely) with $L > 0$. Denote by $f^*(x) \triangleq \mathbb{E}[Y|X = x]$ the regression function, which in general may not belong to $\mathcal{F}$. Given a "regression" dataset $\mathcal{D}_n$ and an independent "regularizer" dataset $\mathcal{D}_N'$, the SAR-based regression estimator $\hat{f}_n$ is defined as the L-truncated estimator $\hat{f}_n \triangleq \beta_L \bar{f}_n$, with

$$\bar{f}_n \leftarrow \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \left(f(X_i) - Y\right)^2 + \lambda \tilde{J}_N(f). \tag{4.12}$$

We now provide an upper bound on the performance of this estimator. To state our result, for $k \geq 1$, we define

$$D_k(\phi) \triangleq \max_{i=1,\ldots,d} \sup_{x \in \mathcal{X}} \left\| \frac{\partial^k \phi(x)}{\partial x_i^k} \right\|_2.$$

[7] Note that the meaning of subscripts of J and $\tilde{J}$ is different from Section 4.4. Here J_N and $\tilde{J}_N$ refer to the use of N samples to estimate the 1^{st}-order derivative (using the true derivative or its finite difference approximation, respectively). In the previous section, we used J_i and $\tilde{J}_i$ to refer to the i^{th}-order derivative-based regularizer and its SAR version. We always use N to refer to the number of samples, so no confusion should arise.

If the k^{th} partial derivatives are not defined, we set $D_k(\phi) \triangleq \infty$. For our results, we only require the existence of $D_1(\phi)$ and $D_3(\phi)$. Full proofs of Theorems 2, 3, and 4 are available in the supplementary material for [3].

Theorem 2. *Assume that all* $\{\varphi_i\}_{i=1}^p$ *are thrice differentiable and* $\sup_{x\in\mathcal{X}} \|\phi(x)\|_2 \leq R$. *Moreover, suppose that* $\lambda_{\min}(\tilde{\Sigma}_N)$, *i.e. the smallest eigenvalue of* $\tilde{\Sigma}_N$, *is bounded away from zero. Under these assumptions, there exist constants* $c_1, c_2 > 0$ *such that for any fixed* $\delta > 0$, *with probability at least* $1-\delta$, *we have:*

$$\int \left|\hat{f}_n(x) - f^*(x)\right|^2 \mathrm{d}\mu(x) \leq$$
$$2 \min_{\mathbf{w}\in\mathbb{R}^p, \|f_\mathbf{w}\|_\infty \leq L} \Bigg\{ 2\int |f_\mathbf{w} - f^*(x)|^2 \,\mathrm{d}\mu(x) + 2\lambda J(f_\mathbf{w}) +$$
$$\lambda \|\mathbf{w}\|_2^2 \, d\Big(\frac{8D_1^2(\phi)\log(3/\delta)}{3N} +$$
$$\frac{\varepsilon^2}{6} D_3(\phi)[2D_1(\phi) + \frac{\varepsilon^2}{6} D_3(\phi)] \Big) \Bigg\}$$
$$+ \frac{c_1 L^6 R^2}{\lambda_{\min}(\tilde{\Sigma}_N)} \frac{\log(nL)}{n\lambda} + \frac{c_2 L^4 \log(1/\delta)}{n}.$$

This result simultaneously shows the effects of function approximation and estimation errors, the way regularization coefficient λ and $J(f_\mathbf{w})$ determine their trade-off, and the error caused by SAR. The term $\min_{\mathbf{w}\in\mathbb{R}^p} \int |f_\mathbf{w} - f^*(x)|^2 \mathrm{d}\mu(x) + \lambda J(f_\mathbf{w})$ is the (regularized) approximation error and indicates how well the target function f can be approximated in a subset of $\mathcal{F}$. The subset is determined by λ and the true regularization functional $J(f_\mathbf{w}) \triangleq \mathbf{w}^\top \Sigma \mathbf{w}$. As usual in regularized estimators, increasing λ might increase the approximation error, but in exchange it decreases the estimation error $O(\frac{\log(n)}{n\lambda})$, and vice versa. If $\mathcal{F}$ as defined by the

basis functions "matches" the target function (i.e., if f^* can be well-approximated by a function in $\mathcal{F}$ with a small $J(f)$), then we can learn the target function fast. This is where feature engineering or data-dependent feature generation would show their benefits. It is noticeable that this result does not depend on the dimension of the feature space p.

Results similar to this part of the theorem are known in the supervised learning literature, cf. Theorem 21.1 of [40] for regularized regression in $C^k(\mathbb{R})$ (re. splines), Theorem 7.23 of [100] for regularized loss in an RKHS, and [99] for strongly convex objectives – which is satisfied for a convex loss and the ℓ_2 regularizer – and linear function spaces.

The effect of using $\tilde{J}_N(f)$ instead of the true regularizer $J(f)$ in Eq. 4.12 appears in the $O(\|\mathbf{w}\|_2^2\,[\frac{1}{N}+\varepsilon^2])$ term. The interesting observation here is that the effect depends on the norm of $\mathbf{w}$, so if the true function can be well-approximated by a "simple" function (measured according to $\|\mathbf{w}\|_2$), we would not suffer much from the error caused by SAR.

To better understand the behaviour of the bound, consider the case that $J(f_{\mathbf{w}}) \triangleq \|\mathbf{w}\|_2^2$ and the target function f^* belongs to $\mathcal{F}$, i.e. $f^* = f_{\mathbf{w}^*}$ for some $\mathbf{w}^*$. Ignoring the constants and the logarithmic terms, by choosing $\lambda = \frac{1}{\|\mathbf{w}^*\|\sqrt{n}}$ to optimize the tradeoff between $\lambda J(f^*_{\mathbf{w}})$ and $\frac{1}{n\lambda}$, we get the upper bound of $O(\frac{\|\mathbf{w}^*\|_2}{\sqrt{n}}[1+\frac{1}{N}+\varepsilon^2])$.

Remark 1. One could get $\int |f_{\mathbf{w}} - f^*(x)|^2 \mathrm{d}\mu(x) + \lambda J(f_{\mathbf{w}})$ inside the minimizer instead of the current one, which has a multiplicative constant of 2, at the price

of having $O(\frac{\|\mathbf{w}\|_2^2}{\sqrt{N}} + \frac{1}{\sqrt{n}})$ instead of $O(\frac{\|\mathbf{w}\|_2^2}{N} + \frac{1}{n})$. This depends on whether we use Bernstein's inequality or Hoeffding's inequality in the proofs.

Remark 2. The quantity $\lambda_{\min}(\tilde{\Sigma}_N)$ in the theorem is a random function of $\mathcal{D}'_N$ and can be calculated given $\mathcal{D}'_N$.

We now depart from the context of regression and focus on the SAR procedure itself. The first result is a uniform upper bound on the difference between the desired regularizer $J(f)$ and its approximation $\tilde{J}_N(f)$, for any function $f \in \mathcal{F}_B \triangleq \left\{ \phi^\top \mathbf{w} \,:\, \mathbf{w} \in \mathbb{R}^p, \|\mathbf{w}\|_2 \leq B \right\}$. I.e., functions in the ball with radius B w.r.t. the ℓ_2-norm of $\mathbf{w}$.

Theorem 3 (Supremum of the Empirical Process $|\tilde{J}_N(f) - J(f)|$)**.** *Assume that all $\{\varphi_i\}_{i=1}^p$ are thrice differentiable. Under this assumption, for any fixed $\delta > 0$ and $B > 0$, we have:*

$$\sup_{f \in \mathcal{F}_B} \left| \tilde{J}_N(f) - J(f) \right| \leq \frac{B^2 \varepsilon^2}{6} d D_3(\phi) \left(2D_1(\phi) + \frac{\varepsilon^2}{6} D_3(\phi) \right) + 32 B^2 d D_1^2(\phi) \sqrt{\frac{2p \log\left(\frac{128 B^2 d D_1^2(\phi) N}{\delta} \right)}{N}} + \frac{1}{N},$$

with probability at least $1 - \delta$.

This theorem shows the effects of the estimation error and the finite difference approximation error. Eliding minor terms, the simplified behaviour of the estimation error is $O(B^2 \sqrt{\frac{p}{N}})$. The dependence on N and p is common to the usual uniform deviation bounds in statistical learning for functions from a p-dimensional linear vector space. The effect of the size of the function space also manifests itself through B^2, which bounds the squared ℓ_2 norm of $\mathbf{w}$.

The effect of the finite difference approximation error is $O(B^2\varepsilon^2)$ – neglecting terms depending on the smoothness of the basis functions. The ε^2 dependence is the usual residual error from the central difference approximation of a derivative. If instead we used a forward (or backward) estimate of the derivative, we would get ε behaviour. The dependence on B is because functions $\phi^\top \mathbf{w}$ with larger $\|\mathbf{w}\|_2$ might have a larger derivatives, so their finite difference approximation could have a larger residual error.

Theorem 3 provides an upper bound for the supremum of the empirical process only over a subset $\mathcal{F}_B$ of $\mathcal{F}$, but it does not provide a non-trivial result for the supremum of $|\tilde{J}_N(f) - J(f)|$ over all of $\mathcal{F}$. This is expected as, for large $\mathbf{w}$, the true regularizer $J(f_{\mathbf{w}})$ would be large too, and the deviation of $\tilde{J}_N(f_{\mathbf{w}})$ around it can also be large. Nonetheless, we can still study the behaviour of the empirical process as a function of $J(f)$. This is known as the modulus of continuity result in the empirical process theory (or relative deviation of error). The following theorem provides such a result. Here we denote $a \vee b \triangleq \max\{a, b\}$.

Theorem 4 (Modulus of Continuity for the Empirical Process $|\tilde{J}_N(f) - J(f)|$)**.** *Assume that all* $\{\varphi_i\}_{i=1}^p$ *are thrice differentiable. Suppose that* $\lambda_{\min}(\Sigma)$*, i.e. the smallest eigenvalue of* Σ*, is bounded away from zero. W.l.o.g., assume that* $256dD_1^2(\phi) \geq 1$*. Let* $\alpha > 0$*. Under these assumptions, there exists* $c_1(\alpha)$ *such that*

for any fixed $\delta > 0$, we have

$$\sup_{f\in\mathcal{F}} \frac{\left|\tilde{J}_N(f) - J(f)\right|}{\left[J(f) \vee \lambda_{\min}(\Sigma)\right]^{1+\alpha}} \leq \frac{1}{\lambda_{\min}^{1+\alpha}(\Sigma)} \left[2^5 d D_1^2(\phi) \sqrt{\frac{2p \log\left(\frac{512 d D_1^2(\phi) c_1(\alpha) N}{\delta}\right)}{N}} + \frac{d\varepsilon^2}{3!} D_3(\phi) \left[2D_1(\phi) + \frac{\varepsilon^2}{3!} D_3(\phi) \right] \right],$$

with probability at least $1-\delta$. Here $c_1(\alpha)$ can be chosen as follows: For $0 < \alpha \leq \frac{1}{4e\log(2)} \approx 0.1327$, $c_1(\alpha) = 8[2 - \frac{W_{-1}(-4\alpha\log(2))}{4\alpha\log(2)}]$ (in which W_{-1} is the lower branch of Lambert W-function), and $c_1(\beta) = 16$ otherwise.

We can elucidate this result by seeing how it works in the context of Theorem 3, by restricting $\mathcal{F}$ to $\mathcal{F}_B$. In this case, $J(f) \leq O(B^2)$, so we get $\sup_{f\in\mathcal{F}_B} |\tilde{J}_N(f) - J(f)| \leq c_2(d, D_1, D_3, \Sigma) B^{2(1+\alpha)} [\sqrt{\frac{p\log(c_1(\alpha)N/\delta)}{N}} + \varepsilon^2]$ instead of the $c_3(d, D_1, D_3) B^2 [\sqrt{\frac{p\log(B^2N/\delta)}{N}} + \varepsilon^2]$ in Theorem 3. The major difference is in the exponent of B. When α goes to zero, $B^{2(1+\alpha)}$ decreases, but the term $c_1(\alpha)$ inside the logarithm increases. As can be seen from the definition of $c_1(\alpha)$, when $\alpha \to 0$, $c_1(\alpha)$ blows up. Overall, even though Theorem 3 provides a slightly tighter upper bound on the error for $\mathcal{F}_B$, Theorem 4 can be considered a stronger result as it holds for all functions in $\mathcal{F}$.

Remark 3. The effect of the input space dimension d on SAR's statistical properties, as can be seen in all results, is quite mild, and only appears in constants. SAR's sampling is a typical Monte Carlo integration, for which convergence rate is

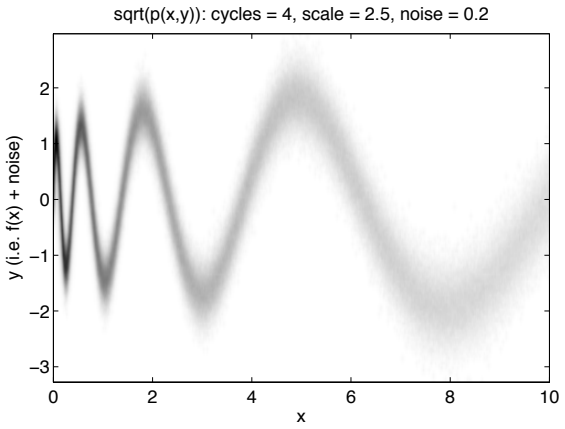


Figure 4–1: The SynthSin1d joint distribution. The sampling density over each oscillation is constant and the noise is proportional to the wave magnitude.

dimension-independent. The minor effect of d is due to using finite differences and the way we have defined D_k.

4.6 Experiments

4.6.1 Synthetic Data

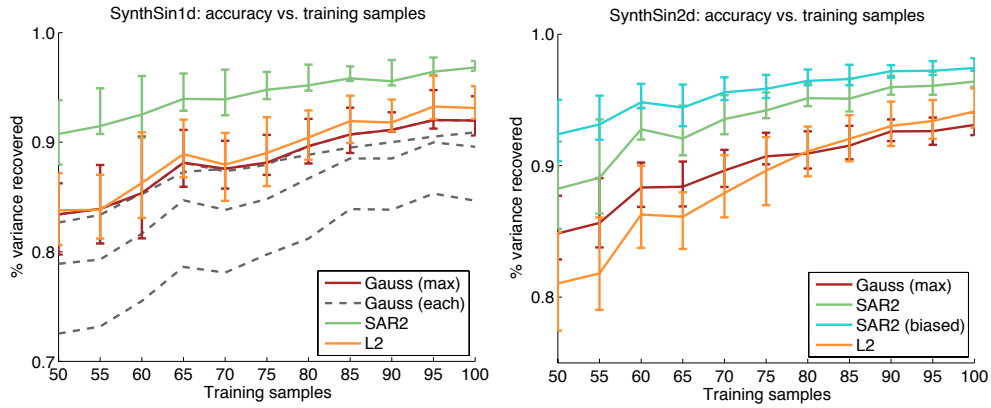
We begin our empirical examination of SAR with tests illustrating some of its strengths and validating the theory presented in Section 4.5. All RBFs used in these tests were Gaussian.

Our first tests involved least-squares regression with inputs $x \in \mathbb{R}$ and outputs $y \in \mathbb{R}$. The data distribution was designed to emphasize SAR's ability to regularize heterogenous basis functions. This contrasts with standard RKHS regularization, which uses more restricted collections of bases. The joint distribution over (x, y) was set so four cycles of a sine wave occurred over the input domain, each with a wavelength 2.5 times longer than the previous one. The wave amplitude was scaled linearly from 1 to 2 over the input domain. The density of x

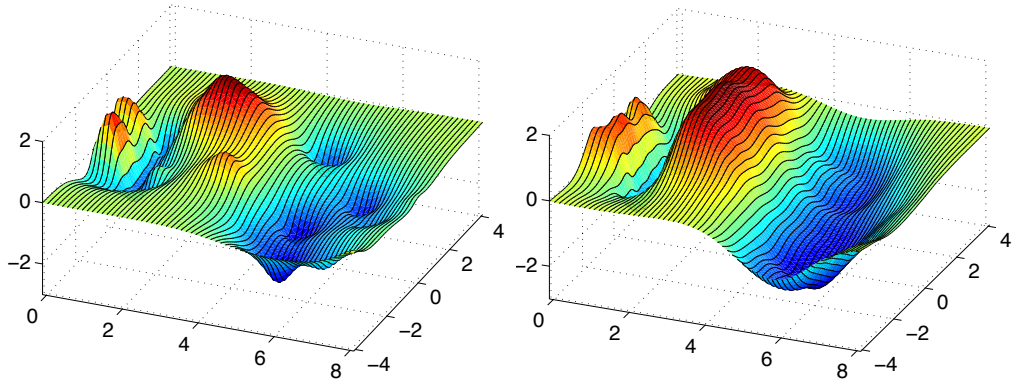
was set so the expected number of observations seen for each cycle was the same. The training y values were corrupted by zero-mean Gaussian noise with standard deviation scaling linearly from 0.2 to 0.4 over the input domain. Performance was measured using uncorrupted y values. We call this distribution SynthSin1d and illustrate it in Fig. 4–1.

The smooth sinusoid underlying SynthSin1d seems amenable to RKHS-regularized RBF regression, but actually causes serious problems due to large changes in the length scale of useful correlations over the input domain. When restricted to fixed bandwidth RBFs, the RKHS approach will always underperform on some part of the function not suited to the chosen bandwidth, as shown by results in Figure 4–2a.

Using SynthSin1d, we compared the performance of SAR2 regularization with L2 regularization and RKHS regularization of Gaussian RBFs. SAR2 and L2 regularization were applied to four RBFs anchored at each training point, with bandwidths $\gamma \in \{2, 4, 8, 16\}$. RKHS regularization was applied independently at each bandwidth using the same RBFs. I.e., four RKHS-regularized solutions were learned for each train/test set. We compared the performance of the three methods on training sizes $\in$ [50...100]. For each training size, 100 training sets were sampled from SynthSin1d (with output noise). For each set, the function learned with each regularizer was tested on 5000 points sampled from SynthSin1d (without output noise). Regularization weights for each method were set independently for each training size, to maximize measured performance.



(a) (b)



(c) SynthSin2d: L2 (d) SynthSin2d: SAR2+bias

Figure 4–2: Full descriptions of the tests underlying these plots are given in the main text. (a) shows the performance of SAR2 on SynthSin1d when learning with Gaussian RBFs with multiple bandwidths, w.r.t. the number of training samples. We also plot the performance of L2 regularization applied to the same RBFs and the performance of RKHS-regularized regressions for each fixed-bandwidth subset of the RBFs. The best performance of RKHS regularization over the considered bandwidths is highlighted in red and per-bandwidth performances are plotted separately in gray. (b) is analogous to (a), but for tests based on SynthSin2d. (b) also plots the performance of SAR2 with biased directional sampling, which penalizes non-linearity in the learned function more along the axis which, for SynthSin2d, was uninformative. (c)/(d) compare the qualitative behavior of L2 and biased SAR2 on SynthSin2d.

We measured performance as the percentage of variance in the true function which was recovered by the learned approximation:

$$\% \text{ variance recovered} = 1 - \frac{\sum_i (\hat{y}_i - y_i)^2}{\sum_j (y_j - \bar{y})^2}, \tag{4.13}$$

in which $\hat{y}_i$ gives the value of the learned approximation at test point x_i, y_i gives the value of the true function at x_i, and $\bar{y}$ gives the mean of the true function. The value of Eq. 4.13 approaches 1 as the approximation approaches the true function, and is 0 for an approximation that always predicts the mean $\bar{y}$. So, larger values are better.

Figure 4–2a plots the mean performance of each regularization method for each considered training set size, with error bars indicating the upper and lower quartiles over the 100 tests at each size. The performance of RKHS regularization at each bandwidth is plotted in gray and the maximum performance is in red. In these tests, SAR2 significantly outperformed both L2 regularization using the same basis functions and RKHS regularization using any of the fixed-bandwidth subsets of the basis functions.

Our second tests extended the form of SynthSin1d to inputs $(x_1, x_2) \in \mathbb{R}^2$ and outputs $y \in \mathbb{R}$. We call this distribution SynthSin2d. Importantly, the value of y depended most strongly on x_1, making x_2 relatively uninformative. We performed 100 tests at each of the same training sizes as for SynthSin1d. SAR2 and L2 regularization were applied to collections of three Gaussian RBFs anchored at each training point, with bandwidths $\gamma \in \{0.5, 2, 8\}$. RKHS regularization was applied independently for each fixed-bandwidth RBF subset. Regularization weights for

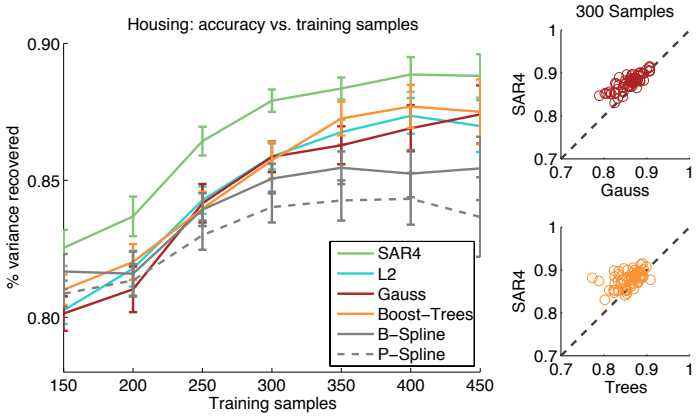
each method were set separately at each training size, to maximize measured performance. We also measured the performance of SAR2 regularization with direction sampling biased as follows: we selected a direction (x_1, x_2) uniformly, multiplied its x_2 by 10, and then rescaled (x_1, x_2) to the desired length. A SAR regularizer computed subject to this bias more severely penalizes change in the estimated function along the x_2 axis, which was known to be less informative.

Figure 4–2b shows that SAR2 significantly improves on the performance of strong RKHS regularization applied to a more restricted set of basis functions and simple L2 regularization applied to an equally diverse set of basis functions. Adding a "correct" bias during regularizer construction further improves the advantage of SAR2, particularly for small training sets. Figure 4–2c/d qualitatively compares the behavior of L2 and biased SAR2 regularization. Biased SAR2 "interpolates" noticeably better than L2.

4.6.2 Natural Data

We write "full RBF" for RBFs based on the values of *all features* of an observation, and we write "univariate RBF" for RBFs based on the value of *a single feature* of an observation. RBFs were Gaussian and RKHS regularization was applied during estimation, unless noted otherwise.

We tested SAR with the "Boston housing" dataset from UCI/StatLib, which comprises 506 observations $x \in \mathbb{R}^{13}$ describing features of neighborhoods in the Boston area (circa 1978), with the prediction target being the median value of homes in each neighborhood. We preprocessed the observations by setting features to zero mean and unit variance, and setting the targets to zero mean.



(a) (b)

Figure 4–3: (a) compares performance on the "Boston housing" data. Error bars give 95% confidence intervals. (b) shows per-round outcomes of each train/test round at training set size 300. y axes give accuracy of SAR4 and x axes give accuracy of RKHS-regularized Gaussian RBFs (top) or boosted trees (bottom).

We compared six methods: L2, SAR4, Gaussian RBFs, 4th-order B-spline RBFs, additive P-splines, and boosted trees. We measured performance with respect to Eq. 4.13.

We performed tests with training sets of size 150-450. For each size, 100 rounds of randomized cross validation were performed, with non-training examples used for evaluating performance. When boosting trees, we set the maximum depth to 3 and performed 250 rounds of boosting with a shrinkage factor of 0.1, which maximized measured performance. For other methods, we set regularization weights separately for each training size to maximize measured performance. We selected kernel bandwidths to maximize performance for 300 training samples.

L2, SAR4, and Gauss all used full Gaussian RBFs centered on each training point with bandwidth $\gamma = 0.05$ fixed across all tests. B-spline used 4th-order

B-spline RBFs centered on each training point with bandwidth $\gamma = 0.2$. P-spline applied 4th-order regularization to 2nd-order additive B-spline bases with 30 knots per dimension. In addition to full RBFs, L2 and SAR4 used a collection of univariate RBFs, with the RBFs on each axis centered on the empirical deciles of the corresponding features. The bandwidth of each univariate RBF was set to the maximum of the distances to its upper and lower "neighbors". The single binary feature in this dataset was represented by just two univariate RBFs, centered on its min/max values. Univariate RBF structure was not optimized.

SAR4 estimated approximate regularizers for first through fourth-order derivatives and combined the resulting matrices naively, by an unweighted sum. SAR4 used a compound point sampler which drew 75% of its samples from the fuzzy point sampler in Algorithm 2 and 25% of its samples from the blurry box sampler in Algorithm 3. Both samplers were constructed strictly from the training set during each round of CV. An unbiased direction sampler with stochastic lengths was used. The length distributions $\mathcal{L}$ in point/direction sampling were set to the half of a normal distribution ≥ 0, with standard deviation set to 0.5/0.2 times the median nearest-neighbor distance in the training set. A lower bound of 0.05 was set on the effective step length ϵ.[8] The sampler parameters were not optimized.

Figure 4–3 presents these tests. SAR4 consistently outperformed the other methods, as seen in 4–3a. Figure 4–3b examines relative performance more closely,

[8] This was always much less than the standard deviation of $\mathcal{L}$.

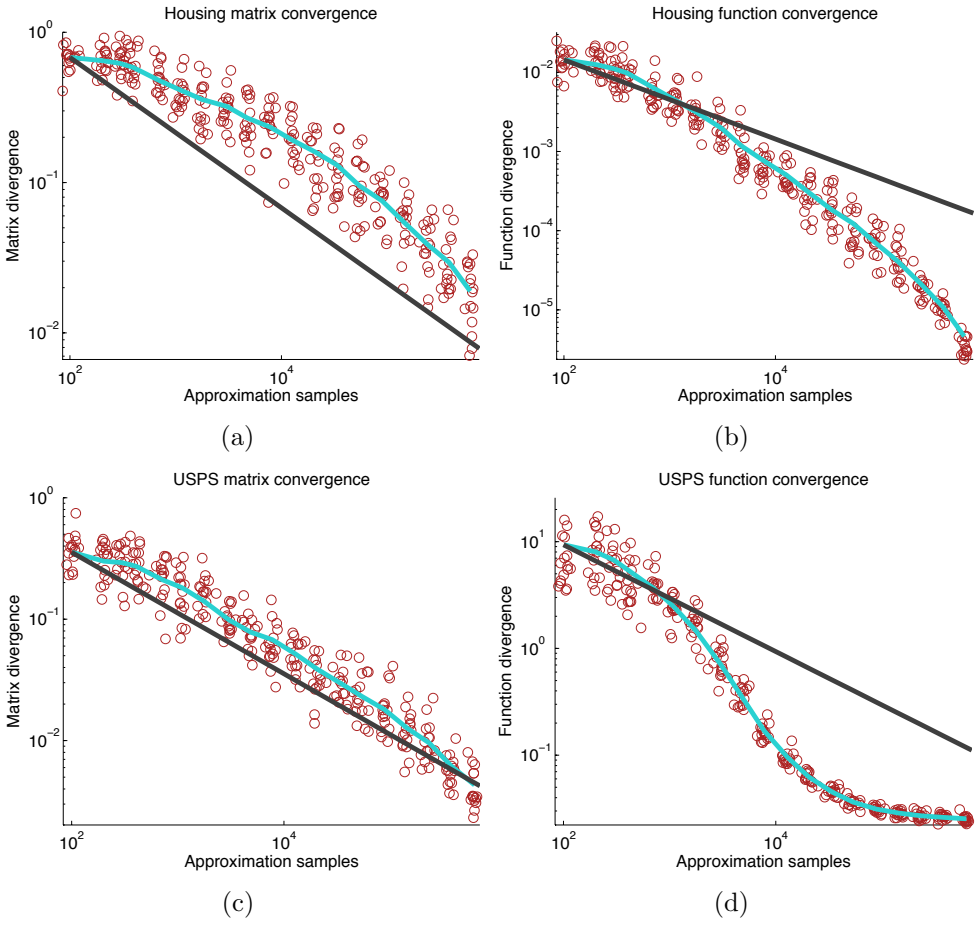


Figure 4–4: SAR4 convergence on housing and USPS data. Left column: convergence of the regularizer matrix. Right column: convergence of the learned function. Parameters for (a)/(b) were as before, using 50 training sets of size 300. USPS results in (c)/(d) used 50 random training sets of size 500. y axes in (a)/(c) give $\sup_{\mathbf{w}:||\mathbf{w}||_2 \leq 1} |\tilde{J}_N(f_{\mathbf{w}}) - \tilde{J}_\infty(f_{\mathbf{w}})|$, which tracks convergence of the SAR-induced penalty. The lighter line plots empirical mean at each sample size and the darker line plots the theoretical rate $1/\sqrt{N}$. y axes in (b)/(d) measure difference between the function induced by a regularizer based on x samples and the function induced by a converged regularizer. The difference between two functions was computed as the expectation of the square of their difference at points sampled from the data distribution (approximating the quantity bounded in Theorem 2).

by plotting results on individual train/test splits for training sets of size 300. SAR4 outperformed boosted trees and Gauss-RBF on most splits. Figure 4–4 examines SAR4's convergence in this setting.

Our final tests used the standard USPS/MNIST digit recognition datasets. We tested on 100 randomly sampled training sets of size 500/2500 and tested on points not seen in training. We compared standard L2, RKHS, and SAR4 regularization using sampler parameters matching those used for tests on the housing data. Each method used full Gaussian RBFs at each training point (as for an SVM), with bandwidth $\gamma = 0.015/0.025$, which were selected to maximize performance of RKHS regularization. We optimized the 1-vs-all squared hinge loss. Regularization weights were set to maximize measured performance. For L2/RKHS/SAR4 the mean and standard deviation of classification accuracy in these tests was 92.7(0.5)/94.0(0.4)/94.1(0.4) for USPS and 94.5(0.02)/95.2(0.01)/95.4(0.02) for MNIST. Both RKHS and SAR4 significantly outperformed L2 on USPS. All pairwise comparisons were significant on MNIST. Figure 4–4 illustrates convergence of SAR on the USPS data. Note that MNIST tests used $\phi(x) \in \mathbb{R}^{2500}$ for $x \in \mathbb{R}^{784}$, which is reasonably high-dimensional.

4.7 Discussion

SAR provides a general approach to controlling complexity in a broad class of functions, i.e., those representable by linear combinations of a fixed set of basis functions, by minimizing the n^{th}-order derivative. For $n = 1$, we provided bounds on the error in the regularizer produced by SAR and showed that the

approximation process is reasonably sample-efficient. The main benefit of SAR is its flexibility, as can be seen from the empirical examination.

SAR raises a number of other interesting questions. On the theoretical side, it would be interesting to analyze SAR for higher-order derivatives, establish the influence of structure in the point measure ν and direction measure s_x, or make precise the relation between SAR and Laplacian-based regularization. On the practical side, developing heuristic approaches to reduce the effective sample complexity, as well as point and direction samplers that better leverage prior knowledge is desirable. Reducing the per-sample cost of SAR by leveraging techniques for reduced-rank kernel approximation in SVMs and implementing SAR so as to take advantage of sparsity in $\phi(x)$ both seem worthwhile, as they could significantly reduce the cost of the outer-products in line 6 of Algorithm 1. When regularizing simultaneously over multiple derivative orders, in practice one needs to select relative regularization strengths for each order. We used an unweighted sum of the regularizers. Choosing relative strengths in a more principled fashion may lead to improved performance.

CHAPTER 5
Learning with Pseudo-Ensembles

5.1 Motivation

The previous chapter developed an approach to controlling the derivatives of a learned function, under the assumption that the function approximator was linear in terms of a fixed set of features. In this setting, the "true" regularizer that we sought was representable by a single fixed matrix. We were thus able to apply the regularizer approximately by precomputing a sample-based approximation of this matrix. But, our precomputed matrix would no longer apply the desired penalty if the features used by the function approximator were to change. Yet, for many tasks, it can be hard to find or construct a fixed set of features that perform better than features learned adaptively. With this motivation, we now investigate the use of regularizers for more powerful models that incorporate feature learning. Regularizing the derivatives of a function can be interpreted as a way of minimizing variation in the function with respect to uncertainty in its input. In this chapter, we develop regularizers which control variation in a function approximator with respect to a more general sort of uncertainty.

Combining multiple models into an ensemble provides a simple, effective way of obtaining robust performance in the presence of various types of uncertainty. Some approaches to ensembling (i.e. combining models into an ensemble) work by training multiple instances of the same model on perturbed versions of the

available training distribution. E.g., bagging uses "bootstrap" samples of the distribution to train each ensemble member and boosting re-weights the distribution before training each ensemble member. Other approaches to ensembling introduce uncertainty in the model type or structure. E.g., when training a new tree for a random forest one often restricts each decision node to considering a randomly sampled subset of the available features.

Dropout and its conceptual cousins have achieved great empirical success in training deep models while imposing uncertainty throughout the decision making system, i.e. in both the data and the model. While some effort has been spent on formally interpreting dropout as a regularizer on the parameters of a model, little effort has been spent leveraging relations between dropout and traditional ensemble methods or other approaches to learning robust models. Yet, analogies between dropout and ensemble methods were prominent in its original presentation. Additionally, we feel it is most worthwhile to consider how biases introduced into a hypothesis search process – i.e. the regularizers used when training a model – affect the *general behavior* of the joint model/data system produced by the search process. In contrast, the effect of these biases on the *particular parameters* found by the search process is rather inconsequential.

In this chapter we take a closer look at how methods like dropout, which force a decision making system to operate under uncertainty, can be interpreted as ways of shaping the behavioral properties of a *pseudo-ensemble*. We begin by formalizing the notion of a pseudo-ensemble, which is an ensemble of child models spawned from a parent model by perturbing it with some noise process.

Sec. 5.2 defines pseudo-ensembles, after which Sec. 5.3 discusses the relationships between pseudo-ensembles and standard ensemble methods, as well as existing notions of robustness. After defining the pseudo-ensemble framework, we can leverage it to create new algorithms. In Sec. 5.4, we develop a novel regularizer that minimizes (some measure of) variation in the output of a model when it is subject to noise on its inputs and its internal state (or structure). We also discuss the relationship between this regularizer and standard dropout methods. In Sec. 5.5 we show that our regularizer can reproduce the performance of dropout in a fully-supervised setting, while also naturally extending to the semi-supervised setting. In the semi-supervised setting, our approach produces state-of-the-art performance on some real-world datasets.[1] Sec. 5.6 presents a case study in which we extend the Recursive Neural Tensor Network from [93] by converting it into a pseudo-ensemble. We generate the pseudo-ensemble using a noise process based on Gaussian parameter fuzzing and latent subspace sampling, and empirically show that both types of perturbation contribute to significant performance improvements beyond that of the original model. Sec. 5.7 provides discussion.

5.2 What is a Pseudo-Ensemble?

In formal terms, a pseudo-ensemble comprises a potentially infinite ensemble of *child models* spawned from some *parent model* via perturbations generated by

[1] Better results have been obtained following the original publication (in [2]) of the work described in this chapter.

some *noise process*. Consider a data distribution p_{xy} which we want to approximate using a parametric *parent model* f_θ. A pseudo-ensemble is a collection of ξ-perturbed *child models* $f_\theta(x;\xi)$, where ξ comes from a *noise process* p_ξ. Dropout [44] provides the clearest existing example of a pseudo-ensemble. Dropout samples subnetworks from a source network by randomly masking the activity of subsets of its input/hidden layer nodes. The parameters shared by the subnetworks, through their common source network, are trained to minimize the expected loss of the individual subnetworks. In pseudo-ensemble terms, the source network is the *parent model*, each sampled subnetwork is a *child model*, and the *noise process* consists of sampling a node mask and using it to extract a subnetwork.

The noise process used to generate a pseudo-ensemble can take fairly arbitrary forms. The only requirement is that sampling a noise realization ξ, and then imposing it on the parent model f_θ, be computationally tractable. This generality allows deriving a variety of pseudo-ensemble methods from existing models. For example, for a Gaussian Mixture Model, one could perturb the means of the mixture components with, e.g., Gaussian noise and their covariances with, e.g., Wishart noise. Noise processes that can be sampled from and imposed on the parent model efficiently, rather than just tractably, are preferable. Noise processes for which the expected pseudo-ensemble output can be computed without sampling are most desirable of all, as this permits efficient evaluation of the full ensemble prediction.

The goal of learning with pseudo-ensembles is to produce models robust to the perturbations imposed by the noise process. In formal terms, the general

pseudo-ensemble objective for supervised learning can be written as follows[2] :

$$\underset{\theta}{\text{minimize}} \underset{(x,y)\sim p_{xy}}{\mathbb{E}} \underset{\xi\sim p_\xi}{\mathbb{E}} \mathcal{L}(f_\theta(x;\xi), y), \tag{5.1}$$

where $(x, y) \sim p_{xy}$ is an (input, output) pair drawn from the data distribution, $\xi \sim p_\xi$ is a *noise realization*, $f_\theta(x;\xi)$ represents the output of a child model spawned from the parent model f_θ via ξ-perturbation, and $\mathcal{L}(\hat{y}, y)$ is the loss suffered when predicting $\hat{y}$ for input x with true output y.

For example, in dropout a noise realization ξ is drawn from the noise process p_ξ by sampling a random drop mask ξ^i for each layer of the parent network. Computing the ξ-perturbed output $f_\theta(x;\xi)$ for input x then proceeds by computing the ξ-perturbed output of the i^{th} layer as follows:

$$f^i_\theta(x;\xi) \triangleq F^i_\theta(f^{i-1}_\theta(x;\xi) \odot \xi^{i-1}), \text{ for } 2 \le i \le d, \text{ with } f_\theta(x;\xi) \triangleq f^d_\theta(x;\xi), \tag{5.2}$$

in which $f^1_\theta(x;\xi) \triangleq x$, $\odot$ indicates element-wise multiplication, and $F^i_\theta(\cdot)$ indicates the input→output transformation applied by layer i. Each drop mask ξ^i is a vector of random variables that are zero with probability $1 - \rho_i$ and take value $\frac{1}{(1-\rho_i)}$ with probability ρ_i. Note that the ξ-perturbed output of layer i refers to the output of layer i when layers $< i$ are subject to perturbation but layer i is not.

The generality of the pseudo-ensemble approach comes from broad freedom in describing the noise process p_ξ and the mechanisms by which ξ perturbs the

[2] It is easy to formulate analogous objectives for unsupervised learning, maximum likelihood, etc.

parent model f_θ. Many useful methods could be developed by exploring novel noise processes for generating perturbations beyond the simple masking noise that has been considered for neural networks and the feature noise that has been considered in the context of linear models.[3] For example, [83] develops a method for learning "ordered representations" by applying dropout/masking noise in a deep auto-encoder while enforcing a particular "nested" structure among the random masking variables in ξ.

5.3 Related Work

Pseudo-ensembles are closely related to traditional ensemble methods and to methods for learning models robust to input uncertainty. By optimizing the expected loss of individual ensemble members' outputs, pseudo-ensembles differ from boosting, which iteratively augments an ensemble to minimize the loss of the joint output [41]. Meanwhile, the child models in a pseudo-ensemble share parameters and structure through their parent model, which will tend to correlate their behavior. This distinguishes pseudo-ensembles from traditional "independent member" ensemble methods, like bagging and random forests, which typically prefer diversity in the behavior of their members. This diversity provides variance reduction when the member outputs are averaged to produce the ensemble output [41]. Interestingly, the regularizers we introduce in Sec. 5.4 explicitly *minimize* diversity in the behavior of their pseudo-ensemble members.

[3] Since publication of [2], various groups have produced work along these lines.

To understand the differences between pseudo-ensembles and boosting/bagging more clearly, consider these objectives derived from Eq. 5.1:

$$\text{Boosting PE:} \qquad \mathop{\mathbb{E}}_{(x,y)\sim p_{xy}} \mathcal{L}\left(\mathop{\mathbb{E}}_{\xi\sim p_\xi}\left[f_\theta(x;\xi)\right], y\right) \tag{5.3}$$

$$\text{Bagging PE:} \qquad \mathop{\mathbb{E}}_{\xi^i\sim p_\xi} \mathop{\mathbb{E}}_{(x,y)\sim p^i_{xy}} \mathcal{L}(f_{\theta^i}(x;\xi^i), y). \tag{5.4}$$

The objective in Eq. 5.3 minimizes the expected loss suffered by the joint/mean output of a pseudo-ensemble, which makes it more like boosting. The objective in Eq. 5.4 optimizes the expected loss suffered by individual ensemble members drawn at random from the pseudo-ensemble, but without any parameter sharing, which makes it more like bagging. The key points are that Eq. 5.3 moves the loss $\mathcal{L}$ outside the expectation over noise and that Eq. 5.4 associates the child model generated by ξ^i with its own input distribution p^i_{xy} and its own set of parameters θ^i.[4] The boosting pseudo-ensemble prediction would be given by $\mathbb{E}_{p_\xi} f_\theta(x;\xi)$, and the bagging pseudo-ensemble prediction would be given by $\frac{1}{n}\sum_{i=1}^n f_{\theta^i}(x)$, assuming n noise realizations ξ^i sampled during optimization.

The definition and use of pseudo-ensembles are both strongly motivated by the intuition that models trained to be robust to noise should generalize better than models that are sensitive to small perturbations. Previous work on robust learning has overwhelmingly concentrated on perturbations affecting the inputs

[4] Transposing the (log-likelihood) loss with the expectation, and how it significantly affects ensemble behavior, seems linked to some known deficiencies of variational inference. Variational inference in Bayesian models involves a similar (Jensen's) transposition of log and expectation.

to a model. For example, the optimization community has produced a large body of theoretical and empirical work addressing "stochastic programming" [90] and "robust optimization" [12]. Stochastic programming seeks to produce solutions to, e.g., linear programs that performs well *on average*, with respect to known distributions over perturbations of parameters in the problem definition[5] . Robust optimization generally seeks to produce solutions to, e.g., linear programs with optimal *worst case* performance over given sets of possible perturbations of parameters in the problem definition.

While minimizing expected loss makes the pseudo-ensemble objective in Eq. 5.1 more similar to stochastic programming, it may be worth investigating objectives for use with pseudo-ensembles based on controlling worst-case behavior, as one does in robust optimization.[6]

Several well-known machine learning methods have been shown equivalent to certain robust optimization problems. For example, [115] shows that using the

[5] Note that "parameters" in a linear program are analogous to inputs in standard machine learning terminology, as they are observed quantities (rather than quantities optimized over).

[6] Since the publication of [2], training models to be robust in the worst case has been successfully applied in the fully-supervised and semi-supervised settings. Generally speaking, the methods introduced for controlling worst-case behavior are more computationally costly than methods for controlling expected behavior, but have produced stronger results. Moderating worst case behavior grows naturally from the ideas presented in this chapter, and also from the various recent investigations of "adversarial" examples in the deep learning literature. One advantage of working with expected case behavior is that it more naturally leads one to consider the full distributional properties of a computational system subject to noise.

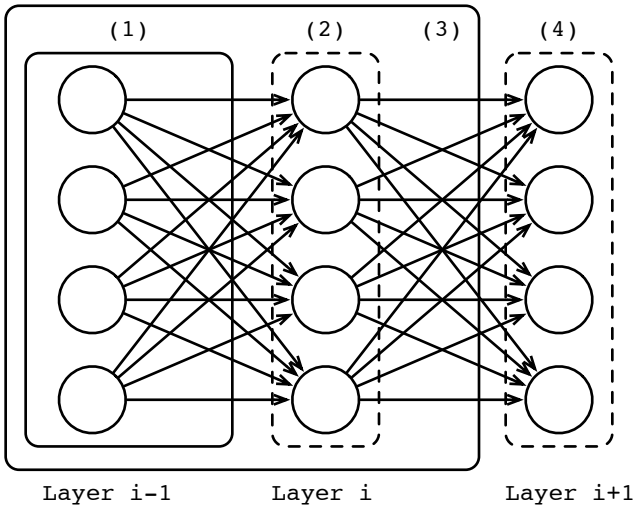


Figure 5–1: How to compute a partial noisy output f_θ^i: (1) compute the ξ-perturbed output $\tilde{f}_\theta^j$ for layers $j < i$, (2) compute the noise-free f_θ^i using the noisy $\tilde{f}_\theta^{i-1}$, (3) ξ-perturb f_θ^i to get $\tilde{f}_\theta^i$, (4) repeat up through the layers $> i$. The f_θ^i computed for each layer are random variables whose stochasticity stems from the noise process p_ξ which produces the perturbations ξ. Distributional properties of these random variables provide a useful glimpse into how the parent model f_θ interacts with the data. Controlling various distributional properties of these random variables provides a very general approach to regularizing the behavior of the joint data/model system, rather than simply regularizing parameters without regard to their role in the data/model system.

Lasso (i.e. ℓ_1 regularization) in a linear regression model is equivalent to a robust optimization problem. [116] shows that learning a standard SVM (i.e. hinge loss with ℓ_2 regularization in the corresponding RKHS) is also equivalent to a robust optimization problem. Supporting the notion that noise-robustness improves generalization, [116] prove many of the statistical guarantees that make SVMs so appealing directly from properties of their robust optimization equivalents, rather than using more complicated proofs involving, e.g., VC-dimension.

Work that considers approaches to learning linear models with inputs perturbed by different sorts of noise is more closely related to pseudo-ensembles. [23] shows how to efficiently learn a linear model that (globally) optimizes expected performance w.r.t. certain types of noise (e.g. Gaussian, zero-masking, Poisson) on its inputs, by marginalizing over the noise. Particularly relevant to our work is [108], which studies dropout (applied to linear models) closely, and shows how its effects are well-approximated by a Tikhonov (i.e. quadratic/ridge) regularization term that can be estimated from both labeled and unlabeled data. The authors of [108] leveraged this label-agnosticism to achieve state-of-the-art performance on several sentiment analysis tasks.

While all the work described above considers noise on the input space, pseudo-ensembles involve noise in the model space. This can actually be seen as a superset of input space noise, as a model can always be extended with an initial "identity layer" that copies the noise-free input. Noise on the input space can then be reproduced by noise on the initial layer, which is now part of the model space.

5.4 The Pseudo-Ensemble Agreement Regularizer

We now present Pseudo-Ensemble Agreement (PEA) regularization, which can be applied easily in a fairly general class of computation graphs. For concreteness, we present it in the context of deep, layered neural networks. PEA regularization operates by controlling distributional properties of the random vectors $\{f_\theta^2(x;\xi), ..., f_\theta^d(x;\xi)\}$, where $f_\theta^i(x;\xi)$ gives the activities of the i^{th} layer of f_θ in response to x when layers $< i$ are perturbed by ξ while layer i is left unperturbed. Fig. 5–1 illustrates the construction of these random vectors. We will assume that

layer d is the output layer, i.e. $f_\theta^d(x)$ gives the output of the unperturbed parent model in response to x and $f_\theta^d(x;\xi) \triangleq f_\theta(x;\xi)$ gives the response of the child model generated by perturbing parent model f_θ with noise ξ. The distribution of each random vector $f_\theta^i(x;\xi)$ emerges from interactions between the network structure f, the network parameters θ, the input x, and the noise distribution p_ξ.

Given the random vectors $f_\theta^i(x;\xi)$, PEA regularization is defined as follows:

$$\mathcal{R}(f_\theta, p_x, p_\xi) \triangleq \mathop{\mathbb{E}}_{x\sim p_x} \mathop{\mathbb{E}}_{\xi\sim p_\xi} \left[\sum_{i=2}^{d} \lambda_i \mathcal{V}_i(f_\theta^i(x), f_\theta^i(x;\xi)) \right], \tag{5.5}$$

where f_θ is the parent model to regularize, $x \sim p_x$ is an unlabeled observation, $\mathcal{V}_i(\cdot,\cdot)$ is the "variability" penalty imposed on the distribution of activities in the i^{th} layer of the pseudo-ensemble spawned from f_θ, and λ_i controls the relative importance of $\mathcal{V}_i$. Note that for Eq. 5.5 to act on the "variability" of the $f_\theta^i(x;\xi)$, we should have $f_\theta^i(x) \approx \mathbb{E}_\xi\, f_\theta^i(x;\xi)$. This approximation holds reasonably well for many useful neural network architectures [7, 111]. One can also regularize:

$$\mathcal{R}(f_\theta, p_x, p_\xi) \triangleq \mathop{\mathbb{E}}_{x\sim p_x} \mathop{\mathbb{E}}_{(\xi_1,\xi_2)\sim p_\xi} \left[\sum_{i=2}^{d} \lambda_i \mathcal{V}_i(f_\theta^i(x;\xi_1), f_\theta^i(x;\xi_2)) \right], \tag{5.6}$$

which compares the outputs of two randomly sampled child models rather than comparing a single child model against the ensemble expectation. In most of our experiments we compute penalties of this second form, which remains tractable even if the ensemble expectation is not efficiently computable. We also consider penalties based on several different measures of variability $\mathcal{V}_i(\cdot,\cdot)$, which we will introduce as needed.

5.4.1 The Effect of PEA Regularization on Feature Co-adaptation

One of the original motivations for dropout was that it helps prevent "feature co-adaptation" [44]. That is, dropout encourages individual features (i.e. hidden unit activities) to remain helpful, or at least not become harmful, when other features are removed from their local context. We provide some support for that claim by examining the following optimization objective:

$$\underset{\theta}{\text{minimize}} \underset{(x,y)\sim p_{xy}}{\mathbb{E}} [\mathcal{L}(f_\theta(x), y)] + \underset{x\sim p_x}{\mathbb{E}} \underset{\xi\sim p_\xi}{\mathbb{E}} \left[\sum_{i=2}^{d} \lambda_i \mathcal{V}_i(f_\theta^i(x), f_\theta^i(x;\xi)) \right], \tag{5.7}$$

in which the supervised loss $\mathcal{L}$ depends only on the parent model f_θ and the pseudo-ensemble only appears in the PEA regularization term. For simplicity, let $\lambda_i = 0$ for $i < d$, $\lambda_d = 1$, and $\mathcal{V}_d(v_1, v_2) \triangleq \text{KL}(\text{softmax}(v_1)||\,\text{softmax}(v_2))$, where softmax is the standard softmax and $\text{KL}(p_1||p_2)$ is the KL-divergence between p_1 and p_2 (we denote this penalty by $\mathcal{V}^k$). We use $\text{xent}(\text{softmax}(f_\theta(x)), y)$ for the loss $\mathcal{L}(f_\theta(x), y)$, where $\text{xent}(\hat{y}, y)$ is the cross-entropy between the predicted distribution $\hat{y}$ and the true distribution y. Eq. 5.7 never explicitly passes label information through a ξ-perturbed network, so ξ only acts through its effects on the distribution of the parent model's predictions when subjected to ξ-perturbation. In this case, Eq. 5.7 trades off accuracy against feature co-adaptation, as measured by the degree to which the feature activity distribution at layer i is affected by perturbation of the feature activity distributions for layers $< i$.

We test this regularizer empirically in Sec. 5.5.1. The observed ability of this regularizer to reproduce the performance benefits of standard dropout supports the notion that discouraging "co-adaptation" plays an important role in dropout's

empirical success. Also, by acting strictly to make the output of the parent model more robust to ξ-perturbation, the performance of this regularizer rebuts the claim in [111] that noise-robustness plays only a minor role in the success of dropout.

5.4.2 Relating PEA Regularization to Standard Dropout

The authors of [108] showed that, assuming a noise process ξ such that $\mathbb{E}_\xi[f_\theta(x;\xi)] = f_\theta(x)$, linear logistic regression under the influence of dropout optimizes the following objective:

$$\sum_{i=1}^{n} \mathop{\mathbb{E}}_{\xi} \left[\ell(f_\theta(x_i;\xi), y_i)\right] = \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)) + R(f_\theta), \tag{5.8}$$

where $f_\theta(x_i) \triangleq \theta^\top x_i$, $\ell(f_\theta(x_i), y_i)$ is the logistic regression loss, and the regularization term is:

$$R(f_\theta) \triangleq \sum_{i=1}^{n} \mathop{\mathbb{E}}_{\xi} \left[A(f_\theta(x_i;\xi)) - A(f_\theta(x_i))\right], \tag{5.9}$$

where $A(\cdot)$ indicates the log partition function for logistic regression.

Using only a KL-d penalty $\mathcal{V}^k(\cdot,\cdot)$ at the output layer, logistic regression with PEA regularization in the form of Eq. 5.5 minimizes:

$$\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \mathop{\mathbb{E}}_{\xi} \left[\mathrm{KL}\left(\mathrm{softmax}(f_\theta(x_i)) \,||\, \mathrm{softmax}(f_\theta(x_i;\xi))\right)\right]. \tag{5.10}$$

Defining distribution $p_\theta(x)$ as $\text{softmax}(f_\theta(x))$, and C as the classes considered by $p_\theta(x)$, we can re-write the PEA regularization part of Eq. 5.10 to get:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\xi}\left[\text{KL}\left(p_\theta(x)\,||\,p_\theta(x;\xi)\right)\right] = \mathop{\mathbb{E}}_{\xi}\left[\sum_{c\in C} p_\theta^c(x)\log\frac{p_\theta^c(x)}{p_\theta^c(x;\xi)}\right] \qquad (5.11)\\
=\;& \sum_{c\in C}\mathop{\mathbb{E}}_{\xi}\left[p_\theta^c(x)\log\frac{\exp f_\theta^c(x)\sum_{c'\in C}\exp f_\theta^{c'}(x;\xi)}{\exp f_\theta^c(x;\xi)\sum_{c'\in C}\exp f_\theta^{c'}(x)}\right]\\
=\;& \sum_{c\in C}\mathop{\mathbb{E}}_{\xi}\left[p_\theta^c(x)\left((f_\theta^c(x)-f_\theta^c(x;\xi)) + (\log\sum_{c'\in C}\exp f_\theta^{c'}(x;\xi) - \log\sum_{c'\in C}\exp f_\theta^{c'}(x))\right)\right]\\
=\;& \sum_{c\in C}\mathop{\mathbb{E}}_{\xi}\left[p_\theta^c(x)(f_\theta^c(x)-f_\theta^c(x;\xi)) + p_\theta^c(x)(A(f_\theta(x;\xi))-A(f_\theta(x)))\right]\\
=\;& \mathop{\mathbb{E}}_{\xi}\left[\sum_{c\in C} p_\theta^c(x)(A(f_\theta(x;\xi))-A(f_\theta(x)))\right]\\
=\;& \mathop{\mathbb{E}}_{\xi}\left[A(f_\theta(x;\xi))-A(f_\theta(x))\right],
\end{aligned}$$

which brings us to the regularizer in Eq. 5.9. Thus, if we assume a logistic regression "primary objective", PEA regularization based on KL-divergence of the pseudo-ensemble members is equivalent to dropout whenever the noise process p_ξ generates the pseudo-ensemble such that $\mathbb{E}_{p_\xi}[f_\theta(x;\xi)] = f_\theta(x)$. We now briefly explain each step in the derivation:

1. By definition of KL-divergence.
2. By definition of the p_θs and by swapping an expectation and a sum.
3. By well-known properties of log and exp.
4. By distributivity and by definition of logistic regression's log-partition function.
5. By assumption that $\mathbb{E}_\xi[f_\theta(x;\xi)] = f_\theta(x)$ and by expectation/sum swapping.

6. By independence of $A(f_\theta(x;\xi)) - A(f_\theta(x))$ from c and $\sum_{c\in C} p_\theta^c(x) = 1$.

5.4.3 PEA Regularization for Semi-supervised Learning

PEA regularization is directly applicable to the semi-supervised setting, as the variability penalties $\mathcal{V}_i$ do not require label information. We train networks for semi-supervised learning in two ways, both of which apply the objective in Eq. 5.1 on labeled examples and PEA regularization on the unlabeled examples. The first way applies a tanh-based penalty $\mathcal{V}^t$ and the second way applies a cross-entropy-based penalty $\mathcal{V}^x$, which we define as follows:

$$\mathcal{V}^t(\bar{y},\tilde{y}) \triangleq ||\tanh(\bar{y}) - \tanh(\tilde{y})||_2^2 \quad \mathcal{V}^x(\bar{y},\tilde{y}) \triangleq \text{xent}(\text{softmax}(\bar{y}), \text{softmax}(\tilde{y})), \quad (5.12)$$

where $\bar{y}$ and $\tilde{y}$ represent the outputs of a pair of independently sampled child models, and tanh is applied element-wise. The xent-variance penalty can be further expanded as:

$$\mathcal{V}^x(\bar{y},\tilde{y}) = \text{KL}(\text{softmax}(\bar{y})||\,\text{softmax}(\tilde{y})) + \text{ent}(\text{softmax}(\bar{y})), \quad (5.13)$$

where $\text{ent}(\cdot)$ denotes the entropy. Thus, $\mathcal{V}^x$ combines the KL-divergence penalty with an entropy penalty, which has been shown to perform well in a semi-supervised setting [34, 61]. We use a noise process which applies the following perturbations to each layer's inputs:

1. Add Gaussian noise with mean 0 and variance σ^2.
2. Apply dropout masking noise, with drop probability ρ.

For our semi-supervised learning experiments, we set ρ to 0.2 for the input layer and to 0.5 for all other layers. We selected σ^2 based on performance on a

validation set, and used the same value for all layers. We chose between the two output-layer penalties $\mathcal{V}^t/\mathcal{V}^x$ based on observed performance.

5.5 Testing PEA Regularization

We tested PEA regularization in three scenarios: supervised learning on MNIST digits, semi-supervised learning on MNIST digits, and semi-supervised transfer learning on a dataset from the NIPS 2011 Workshop on Challenges in Learning Hierarchical Models [60]. Full implementations of our methods, written with THEANO [11], and scripts/instructions for reproducing all of the results in this section are available online at `http://github.com/Philip-Bachman/Pseudo-Ensembles`.

5.5.1 Fully-supervised MNIST

The MNIST dataset comprises 60k 28x28 grayscale hand-written digit images for training and 10k images for testing. For the supervised tests we used SGD hyperparameters roughly following those in [44]. We trained networks with two hidden layers of 800 units each, using rectified-linear activations and an ℓ_2-norm constraint of 3.5 on incoming weights for each unit. For both standard dropout (abbr. SDE) and PEA, we used logistic regression loss at the output layer. We initialized hidden layer biases to 0.1, output layer biases to 0, and inter-layer weights to zero-mean Gaussian noise with $\sigma = 0.01$. We trained all networks for 1000 epochs with no early stopping (i.e. performance was measured for the final network state). We trained for a fixed number of updates because the validation set was merged with the training set, leaving no data for monitoring performance.

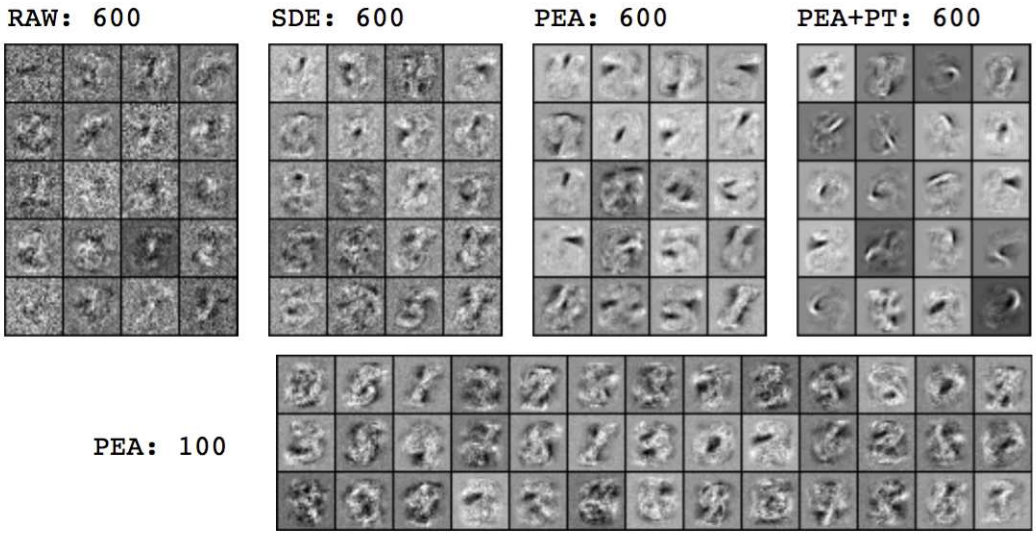
Preliminary monitoring with a validation set suggested that early stopping would not improve performance significantly.

SDE obtained 1.05% error averaged over five random initializations. Using PEA penalty $\mathcal{V}^k$ at the output layer and computing classification loss/gradient only for the unperturbed parent network, we obtained 1.08% averaged error. For these tests the noise process applied unit masking but not bias noise. Thus, we matched the performance of dropout by training the same network while ignoring the effects of masking noise on the classification loss, but encouraging the network to be robust to masking noise (as measured by $\mathcal{V}^k$). This result supports the equivalence between dropout and this particular form of PEA regularization, which we also derived formally in Section 5.4.2.

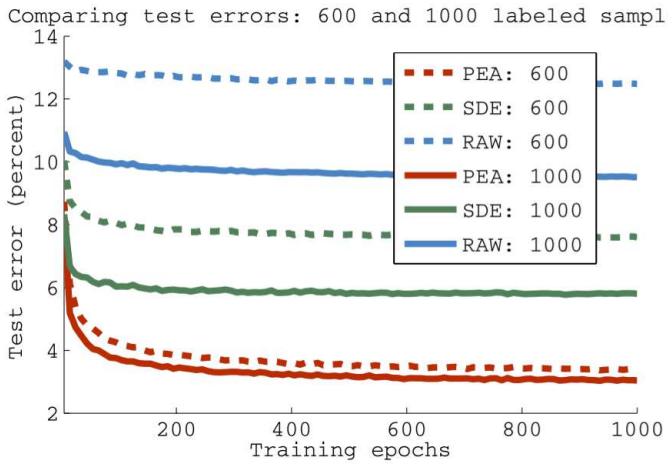
5.5.2 Semi-supervised MNIST

We tested semi-supervised learning on MNIST following the protocol described in [112]. These tests split MNIST's 60k training samples into labeled/unlabeled subsets, with the labeled sets containing $n_l \in \{100, 600, 1000, 3000\}$ samples. For labeled sets of size 600, 1000, and 3000, the full training data was randomly split 10 times into labeled/unlabeled sets and results were averaged over the splits. For labeled sets of size 100, we averaged over 50 random splits. The labeled sets had the same number of examples for each class. We tested PEA regularization with and without denoising auto-encoder pre-training [107][7] . Pre-trained networks were always PEA-regularized with the cross-entropy penalty $\mathcal{V}^x$

[7] See our code for a perfectly complete description of our pre-training.



(a)



(b)

Figure 5–2: Performance of PEA regularization for semi-supervised learning using the MNIST dataset. The top row of filter blocks in (a) were the result of training a fixed network architecture on 600 labeled samples using: weight norm constraints only (RAW), standard dropout (SDE), standard dropout with PEA regularization on unlabeled data (PEA), and PEA preceded by pre-training as a denoising auto-encoder [107] (PEA+PT). The bottom filter block in (a) was the result of training with PEA on 100 labeled samples. (b) shows test error over the course of training for RAW/SDE/PEA, averaged over 10 random training sets of size 600/1000.

on the output layer. Non-pre-trained networks used the tanh penalty $\mathcal{V}^t$ on the output layer, except when the labeled set was of size 100, in which case $\mathcal{V}^x$ was used. In this final case, we gradually increased the λ_i over the course of training, as suggested by [34]. We generated the pseudo-ensembles for these tests using masking noise and Gaussian input+bias noise with $\sigma = 0.1$. Each network had two hidden layers with 800 rectified-linear units. Weight norm constraints and SGD hyperparameters were set as for supervised learning.

Table 5–1 compares the performance of PEA regularization with previous results. Aside from CNN, all methods in the table are "general", i.e. do not use convolutions or other image-specific techniques to improve performance. The main comparisons of interest are between PEA(+) and other methods for semi-supervised learning with neural networks, i.e. E-NN, MTC+, and PL+. E-NN (EmbedNN from [112]) uses a nearest-neighbors-based graph Laplacian regularizer to make predictions "smooth" with respect to the manifold underlying the data distribution p_x. MTC+ (the Manifold Tangent Classifier from [81]) regularizes predictions to be smooth with respect to the data manifold by penalizing gradients in a learned approximation of the tangent space of the data manifold. PL+ (the Pseudo-Label method from [61]) uses the joint-ensemble predictions on unlabeled data as "pseudo-labels", and treats them like "true" labels. The classification losses on true labels and pseudo-labels are balanced by a scaling factor which is carefully modulated over the course of training. PEA regularization (without pre-training) outperforms all previous methods in every setting except 100 labeled samples, where PL+ performs better, but with the benefit of pre-training. By

adding pretraining (i.e. PEA+), we achieve a two-fold reduction in error when using only 100 labeled samples.

	TSVM	NN	CNN	E-NN	MTC+	PL+	SDE	SDE+	PEA	PEA+
100	16.81	25.81	22.98	16.86	12.03	10.49	22.89	13.54	10.79	**5.21**
600	6.16	11.44	7.68	5.97	5.13	4.01	7.59	5.68	**2.44**	2.87
1000	5.38	10.70	6.45	5.73	3.64	3.46	5.80	4.71	**2.23**	2.64
3000	3.45	6.04	3.35	3.59	2.57	2.69	3.60	3.00	**1.91**	2.30

Table 5–1: Performance of semi-supervised learning methods on MNIST with varying numbers of labeled samples. From left-to-right the methods are Transductive SVM , neural net, convolutional neural net, EmbedNN [112], Manifold Tangent Classifier [81], Pseudo-Label [61], standard dropout plus fuzzing [44], dropout plus fuzzing with pre-training, PEA, and PEA with pre-training. Methods with a "+" used contractive or denoising autoencoder pre-training [107]. The testing protocol and the results left of MTC+ were presented in [112]. The MTC+ and PL+ results are from their respective papers and the remaining results are our own. We trained SDE(+) using the same network/SGD hyperparameters as for PEA. The only difference was that the former did not regularize for pseudo-ensemble agreement on the unlabeled examples. We measured performance on the standard 10k test samples for MNIST. All of the 60k training samples not included in a given labeled training set were made available as unlabeled data. The best result for each training size is in **bold**.

5.5.3 Transfer Learning Challenge

The organizers of the NIPS 2011 Workshop on Challenges in Learning Hierarchical Models [60] proposed a challenge to improve performance on a target domain by using labeled and unlabeled data from two related source domains. The labeled data source was CIFAR-100 [54], which contains 50k 32x32 color images in 100 classes. The unlabeled data source was a collection of 100k 32x32 color images taken from Tiny Images [54]. The target domain comprised 120 32x32 color images divided unevenly among 10 classes. Neither the classes nor the images

in the target domain appeared in either of the source domains. The winner of this challenge used convolutional Spike and Slab Sparse Coding, followed by max pooling and a linear SVM on the pooled features [30]. Labels on the source data were ignored and the source data was used to pre-train a large set of convolutional features. After applying the pre-trained feature extractor to the 120 training images, this method achieved an accuracy of 48.6% on the target domain, which represents the best previously published result on this dataset.

We applied semi-supervised PEA regularization by first using the CIFAR-100 data to train a deep network comprising three max-pooled convolutional layers followed by a fully-connected hidden layer which fed into a logistic regression output layer. After this supervised pre-training, we removed the hidden and output layers, replaced them with a pair of fully-connected hidden layers feeding into an ℓ_2-hinge-loss output layer[8] , and then fine-tuned the fresh part of the network using the 120 labeled images from the target domain. For this final training phase, which involved three layers, we tried standard dropout and dropout with PEA regularization using the source data as unlabeled data. Standard dropout achieved 55.5% accuracy, which improved to 57.4% when we added PEA regularization on the source data. In this setting, PEA regularization benefits vs. standard dropout by being able to enforce robustness of the learned classifier

[8] We found that ℓ_2-hinge-loss performed better than logistic regression in this setting. Switching to logistic regression degrades the dropout and PEA results but does not change their ranking.

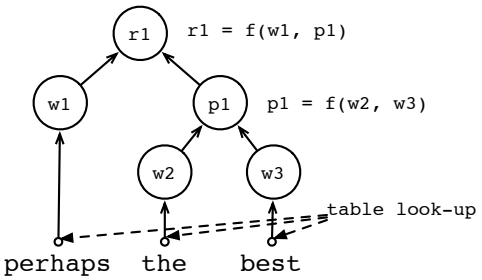


Figure 5–3: How to feedforward through the Recursive Neural Tensor Network. First, the tree structure is generated by parsing the input sentence. Then, the vector for each node is computed by look-up at the leaves (i.e. words/tokens) and by a tensor-based transform of the node's children's vectors otherwise.

using a large number of in-domain, unlabeled samples. Standard dropout can only enforce robustness of the classifier on the small set of labeled samples. While most of the improvement over the previous state-of-the-art (i.e. 48.6%) was due to dropout and an improved training strategy (i.e. supervised pre-training vs. unsupervised pre-training), controlling the feature activity and output distributions of the pseudo-ensemble on unlabeled data allowed significant further improvement.

5.6 Improved Sentiment Analysis using Pseudo-Ensembles

We now show how the Recursive Neural Tensor Network (RNTN) from [93] can be augmented using pseudo-ensembles, and evaluate the resulting model on the Stanford Sentiment Treebank (STB) task. The STB task involves predicting the sentiment of short phrases extracted from movie reviews on RottenTomatoes.com. Ground-truth labels for the phrases, and the "sub-phrases" produced by processing them with a standard parser, were generated using Amazon Mechanical Turk. In addition to pseudo-ensembling, we also used a more "compact" bilinear

form in the function $f : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ that the RNTN applies recursively as shown in Figure 5–3. The computation for the i^{th} dimension of the original f (for $v_i \in \mathbb{R}^{n \times 1}$) was defined as:

$$f_i(v_1, v_2) = \tanh([v_1; v_2]^\top T_i [v_1; v_2] + M_i [v_1; v_2; 1]), \tag{5.14}$$

in which T_i indicates a matrix slice of tensor T, M_i indicates a vector slice of matrix M, and ";" indicates vector concatenation. We replace this with:

$$f_i(v_1, v_2) = \tanh(v_1^\top T_i v_2 + M_i [v_1; v_2; 1]). \tag{5.15}$$

In the original RNTN, T is $2n \times 2n \times n$ and in our "compact" RNTN it is $n \times n \times n$. The other parameters in the RNTNs are a transform matrix $M \in \mathbb{R}^{n \times 2n+1}$ and a classification matrix $C \in \mathbb{R}^{c \times n+1}$. Each RNTN node outputs c class probabilities for its vector v using $\text{softmax}(C[v; 1])$.

We initialized the model with pre-trained word vectors. The pre-training used `word2vec` on the training and dev set, with three modifications: dropout/fuzzing was applied during pre-training (to match the conditions in the full model), the word vector norms were constrained to produce pre-trained vectors with standard deviation 0.5, and tanh was applied during `word2vec` (to match conditions in the full model). All code required for these experiments is publicly available online at: `http://github.com/Philip-Bachman/Pseudo-Ensembles`.

We generated pseudo-ensembles from a parent RNTN using two types of perturbation: subspace sampling and weight fuzzing. We performed subspace sampling by keeping only $\frac{n}{2}$ randomly sampled latent dimensions out of the n in

the parent model when processing a given *phrase tree*. Using the same sampled dimensions for a full phrase tree reduced computation time significantly, as the parameter matrices/tensor could be "sliced" to include only the relevant dimensions.[9] During training we sampled a new subspace each time a phrase tree was processed and computed test-time outputs for each phrase tree by averaging over 50 randomly sampled subspaces. We performed weight fuzzing during training by perturbing parameters with zero-mean Gaussian noise before processing each phrase tree and then applying gradients w.r.t. the perturbed parameters to the unperturbed parameters. We did not fuzz during testing. Weight fuzzing has an interesting interpretation as an implicit convolution of the objective function (defined w.r.t. the model parameters) with an isotropic Gaussian distribution. In the case of recursive/recurrent neural networks this may prove quite useful, as convolving the objective with a Gaussian reduces its curvature, thereby mitigating some problems stemming from ill-conditioned Hessians [75]. For further description of the model and training/testing process, see the supplementary material and the code from `http://github.com/Philip-Bachman/Pseudo-Ensembles`.

Following the protocol suggested by [93], we measured root-level (i.e. whole-phrase) prediction accuracy on two tasks: fine-grained sentiment prediction and

[9] This allowed us to train significantly larger models before over-fitting offset increased model capacity. Training these larger models would have been tedious without the parameter slicing permitted by subspace sampling, as feedforward for the RNTN is $\mathcal{O}(n^3)$. But, cutting n in half for each model update granted an 8x speed-up, which made these larger models practical.

	RNTN	PV	DCNN	CTN	CTN+F	CTN+S	CTN+F+S
Fine-grained	45.7	**48.7**	48.5	43.1	46.1	47.5	48.4
Binary	85.4	87.8	86.8	83.4	85.3	87.8	**88.9**

Table 5–2: Fine-grained and binary root-level prediction performance for the Stanford Sentiment Treebank task. RNTN is the original "full" model presented in [93]. CTN is our "compact" tensor network model. +F/S indicates augmenting our base model with weight fuzzing/subspace sampling. PV is the Paragraph Vector model in [59] and DCNN is the Dynamic Convolutional Neural Network model in [48].

binary sentiment prediction. The fine-grained task involves predicting classes from 1-5, with 1 indicating strongly negative sentiment and 5 indicating strongly positive sentiment. The binary task is similar, but ignores "neutral" phrases (those in class 3) and considers only whether a phrase is generally negative (classes 1/2) or positive (classes 4/5). Table 5–2 shows the performance of our compact RNTN in four forms that include none, one, or both of subspace sampling and weight fuzzing. Using only ℓ_2 regularization on its parameters, our compact RNTN approached the performance of the full RNTN, roughly matching the performance of the second best method tested in [93]. Adding weight fuzzing improved performance past that of the full RNTN. Adding subspace sampling improved performance further and adding both noise types pushed our compact RNTN well past the full RNTN, resulting in state-of-the-art performance on the binary task.

5.7 Discussion

In this chapter, we proposed the formal notion of a pseudo-ensemble, which captures methods such as dropout [44] and feature noising in linear models [23, 108] that have received significant attention. Using the conceptual framework

provided by pseudo-ensembles, we showed how to construct and apply a regularizer which exhibited strong empirical performance and provided insight into the mechanisms behind dropout's success. We also showed how pseudo-ensembles can be used to improve the performance of an already powerful model on a real-world sentiment analysis benchmark. We anticipate that this idea, which unifies several rapidly evolving lines of research, will prove useful for developing new algorithms, especially for semi-supervised learning.

An important topic for future investiagation is how to define, or better yet learn, the best perturbation processes for any given problem. In the case of, e.g., image data one might imagine that small content-preserving non-linear transformations such as skews, rotations, and translations might be useful. In fact, this sort of data augmentation is required for achieving state-of-the-art performance in many computer vision tasks. While a useful set of domain-specific *input* perturbations is easy to construct for many vision problems, it is not at all clear how to do so for other data modalities. Even in the case of vision, it is not clear how to usefully perturb the *internal behavior* of a model, e.g. feature maps in deep convolutional layers, in order to maximize generalization. Ideally, one might be able to define a trainable perturbation process and a useful measure of invariance, and then train the perturbation process to maximize invariance in the learned predictive model. One first idea for how to do this would be to apply noise to a model's activations, interpret them as latent variables, and then minimize mutual information between the model's input and output.

Our primary intent in this chapter was to provide a framework that helps unify several rapidly evolving lines of research, rather than to focus on any particular methods operating within this framework (e.g. the methods presented in this chapter). Though many pseudo-ensemble methods have been published recently, the potential for developing new models within this framework still seems largely untapped. An interesting analogy supporting the power of encouraging robustness by imposing noise can be found in a system called "Chaos Monkey", developed by Netflix, that randomly removes components from a distributed system [45]. By running the Chaos Monkey, Netflix forced their software developers and system architects to confront the sorts of problems that arise when careful engineering meets an uncertain reality, thereby producing a more resilient system.

CHAPTER 6
Variational Generative Stochastic Networks with Collaborative Shaping

6.1 Motivation

The previous chapter presented a method for regularizing a model by perturbing its inputs and structure, while asking the model to maintain a consistent output. Ideally, the perturbations imposed on the model would be representative of invariances relevant to the current task. For example, in an image recognition task one might encourage a classifier to be robust to small translations and rotations of its input by perturbing the available training data with translations and rotations, while penalizing the model for changes in its output. In a speech recognition task, one might add natural background noise or apply some sort of time dilation to the available data. On the other hand, for many tasks it is not obvious what type of perturbations might represent meaningful invariances. For this reason, we wanted to develop models capable of learning to generate perturbations exhibiting structure pertinent to some task. This lead naturally to working with generative models.

Significant effort has gone into developing models capable of effectively synthesizing samples from complicated distributions. We approach this task with multiple objectives. We want to learn a model distribution $\mathcal{G}$ which is practically indistinguishable from a target distribution $\mathcal{D}$, we want to generate locally-coherent sequences of samples from $\mathcal{G}$, we want to generate independent samples

from $\mathcal{G}$, and we want our algorithms for training/inference/sampling to be efficient. No existing generative models simultaneously meet all of our objectives.

While our desire to sample "locally-coherent"[1] random walks may seem odd at first, we motivate it by considering the "manifold hypothesis" in the context of semi-supervised learning. Specifically, one might assume that a classifier should produce decisions which are more or less constant over regions of high data density, with changes restricted to regions of low data density – this idea has been called the "manifold hypothesis". If only a small number of labeled observations are available for training the classifier, but a much larger number of unlabeled observations are also available from the same data distribution, then a method for synthesizing locally-coherent random walks through the data distribution could be helpful in two ways. First, we could generate additional samples in the vicinity of any labeled observation by running many short locally-coherent random walks starting from it and then "transfer" its label to the generated samples. Second, we could generate additional samples in the vicinity of any unlabeled observation by running many short locally-coherent random walks starting from it and then regularize the classifier to produce a relatively constant prediction for the generated samples, e.g. using the techniques described in the previous chapter.

[1] We use this term to describe random walks with strong short-term autocorrelation, which generally make transitions in accordance with the target distribution $\mathcal{D}$. I.e. the chain steps are like $\frac{T(x'|x)}{T(x''|x)} \approx \frac{\mathcal{D}(x')}{\mathcal{D}(x'')}$ and $\mathbb{E}_{T(x'|x)} ||x' - x||_2 < c$ for some small c. Though, given these constraints, we prefer chains with minimal medium/long-term autocorrelation. Intuitively, our preferences select for random walks which move slowly, but do not get stuck for too long in the same place.

The general method we develop for achieving our desiderata can be viewed as a class of Generative Stochastic Networks (GSNs) [9]. We show that any model trained with the *walkback* procedure [10] is encompassed by our method. Rather than starting from denoising auto-encoders as in [10, 9], we leverage recent methods for training deep, directed generative models, e.g. [51, 80, 71], and base our method on variational auto-encoders. By feeding the output of such a variational auto-encoder back into itself, we construct a Markov chain whose stationary distribution provably (in the non-parametric, infinite-data limit) converges to the target distribution.

As an alternative to the walkback procedure for training GSNs, we propose an approach based on recent work in Approximate Bayesian Computation. We partner a generative model with a function approximator that estimates the log-density ratio between the model-generated distribution $\mathcal{G}$ and the target distribution $\mathcal{D}$, in what can be seen as a *collaborative* alternative to the *adversarial* approaches in [38, 39, 32]. We show that the global minimizer of the resulting objective (in the non-parametric, infinite-data limit) is achievable only when the model distribution matches the target distribution. Our collaborative approach also permits a simpler proof of correctness than the adversarial approach.

To control the complexity of our models, we introduce a regularization term close in spirit to reinforcement learning methods such as relative entropy policy search [77] and other approaches which depend on a notion of "natural system dynamics", e.g. [105]. Specifically, we re-weight the standard KL(posterior $||$ prior) term that appears in the variational free-energy. For a generative model $p(x) =$

$\sum_z p(x|z)p(z)$, with latent variables $z \in \mathcal{Z}$, this approach provides a direct mechanism for trading complexity in $p(x)$ against the ability to exactly reproduce the training distribution whenever most of the complexity in $p(x)$ is captured by the latent variables. E.g. if $p(x|z)$ is an isotropic Gaussian whose mean varies with z, but whose variance is the same for all z, then restricting $\mathrm{KL}(p(z|x) \,||\, p(z))$ to be small for all x forces $p(x)$ to be approximately an isotropic Gaussian.[2]

Our models permit efficient generation of independent samples, efficient generation of sequences of samples representing locally-coherent random walks along the data manifold, and efficient evaluation of a variational lower-bound on the log-likelihood assigned to arbitrary inputs. We show that our approach produces models which significantly outperform the GSNs in [9] and the adversarial networks in [32] in terms of test-set log-likelihood and qualitative behavior.

6.2 Background

This section provides a brief summary of prior work on denoising auto-encoders and Generative Stochastic Networks. These methods provide the theoretical and practical grounding for how we will construct Markov chains.

6.2.1 Generalized Denoising Auto-encoders

In the Generalized Denoising Auto-encoder (DAE) framework [10], one trains a *reconstruction distribution* $p_\theta(x|\tilde{x})$ to match the conditional distribution $\mathcal{P}(x|\tilde{x})$ implicit in an infinite set of pairs $\{(x_1, \tilde{x}_1), ..., (x_n, \tilde{x}_n)\}$ generated by first drawing each $x_i \in \mathcal{X}$ from the *target distribution* $\mathcal{D}$ and then generating each $\tilde{x}_i \in \mathcal{X}$ by

[2] We provide more formal discussion of related points later in this chapter.

applying some stochastic *corruption process* $q_\phi(\tilde{x}|x)$ to x_i. The corruption process should be constructed so that, given a corrupted value $\tilde{x}$, there is some uncertainty about which x was fed into $q_\phi(\tilde{x}|x)$ to produce $\tilde{x}$.[3]

Given the corruption process q_ϕ and the reconstruction distribution p_θ, where θ and ϕ denote trainable parameters, one can construct a Markov chain over $x \in \mathcal{X}$ by iteratively sampling a point x_t from $p_\theta(x_t|\tilde{x}_{t-1})$ and then sampling a point $\tilde{x}_t$ from $q_\phi(\tilde{x}_t|x_t)$. The chain is initialized at $t = 0$ by sampling x_0 directly from $\mathcal{D}$ and its transition operator $T_\theta(x_t|x_{t-1})$ can be computed by marginalizing over $\tilde{x}_{t-1}$.

Given a few small assumptions on the forms of q_ϕ and p_θ, and the larger assumption that $p_\theta(x|\tilde{x})$ provides a consistent estimator of $\mathcal{P}(x|\tilde{x})$ as the number of training samples $x \sim \mathcal{D}$ goes to infinity, it was shown in [10] that the Markov chain constructed from the iterative process described above will be ergodic and have a stationary distribution π_θ which matches $\mathcal{D}$ (i.e. $\forall x, \pi_\theta(x) = \mathcal{D}(x)$).

[3] Here, one might be tempted to say something like "applying q_ϕ to the base distribution $\mathcal{D}$ adds entropy" or "the corruption process injects entropy into the base distribution". But, these statements, and many other information-theoretic analogies, fall apart in continuous spaces. The quantity of interest is actually the mutual information between $\tilde{x}$ and x in the joint distribution $(x, \tilde{x})$ generated by sampling $x \sim \mathcal{D}$ and $\tilde{x} \sim q_\phi(\tilde{x}|x)$ (though, this too has some problems). This mutual information does not break in continuous spaces and is not sensitive to, e.g., scale and dimension in the same way that entropy is. Symmetry of the mutual information also reflects the symmetric roles played by x and $\tilde{x}$ in the model – and this symmetry is not entirely obvious. This mutual information also directly relates to the $\mathrm{KL}(q\,||\,p)$ term in Eq. 6.1, the significance of which we touch on in Section 6.4. We take a formal look at some of these issues in Section 6.4.3.

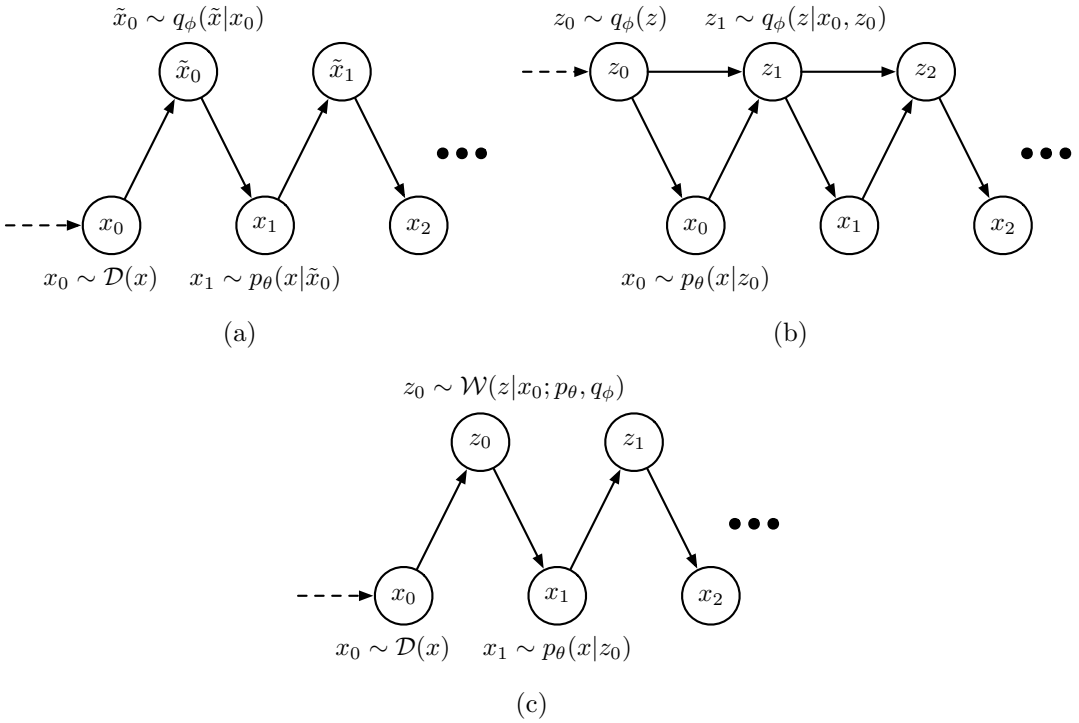


Figure 6–1: (a) shows how to construct the Markov chain associated with a Generalized DAE with reconstruction distribution p_θ, corruption process q_ϕ, and target distribution $\mathcal{D}$. (b) shows how to construct the Markov chain associated with a GSN with reconstruction distribution p_θ and corruption process q_ϕ. We overload notation and define $q_\phi(z)$ to be the distribution matching the stationary distribution of the GSN chain over z. (c) shows how to construct the Markov chain associated with a Simple GSN that uses a reconstruction distribution p_θ and a walkback process $\mathcal{W}(z|x; q_\phi, p_\theta)$ formed by wrapping p_θ and q_ϕ through the walkback procedure (see Algorithm 4 for details).

Algorithm 4 Walkback for a General GSN

Input: data sample x, corruptor q_ϕ, reconstructor p_θ
Initialize an empty training pair list $\mathcal{P}_{xz} = \{\,\}$
Set $\hat{z}$ to some initial vector in $\mathcal{Z}$ (the zero vector suffices).
for $i = 1$ **to** $k_{burn-in}$ **do**
  Sample $\check{z}$ from $q_\phi(z|x, \hat{z})$ then set $\hat{z}$ to $\check{z}$.
end for
Set $\hat{x}$ to x.
for $i = 1$ **to** $k_{roll-out}$ **do**
  Sample $\check{z}$ from $q_\phi(z|\hat{x}, \hat{z})$ then set $\hat{z}$ to $\check{z}$.
  Sample $\check{x}$ from $p_\theta(x|\hat{z})$ then set $\hat{x}$ to $\check{x}$.
  Add pair $(x, \hat{z})$ to $\mathcal{P}_{xz}$.
end for
Return: $\mathcal{P}_{xz}$.

All of the discussion in [10] assumed that both x_i and $\tilde{x}_i$ for each training pair $(x_i, \tilde{x}_i)$ inhabited the same space $\mathcal{X}$, although this was not required for their proofs. The Generative Stochastic Network (GSN) framework [9] thus made the jump of assuming a corruption process $q_\phi(z_t|x_{t-1}, z_{t-1})$. This extends the Generalized DAE framework by introducing a *latent* space $\mathcal{Z} \neq \mathcal{X}$, and by allowing the current latent state z_t to depend on the previous latent state z_{t-1}. In contrast, $\tilde{x}_{t-1}$ in the Generalized DAE framework – its analog of z_t – depends only on the previous observable state x_{t-1}. Figures 2 (a) and (b) illustrate the graphical models corresponding to Generalized DAEs and GSNs.

6.2.2 Training with Walkback

A method called *walkback* training was proposed for Generalized DAEs in [10] and used again for GSNs in [9]. The motivation for walkback training was to mitigate difficulties encountered in practical, finite-data settings, where many values for the latent variables $z \in \mathcal{Z}$ that were rarely (if ever) visited during

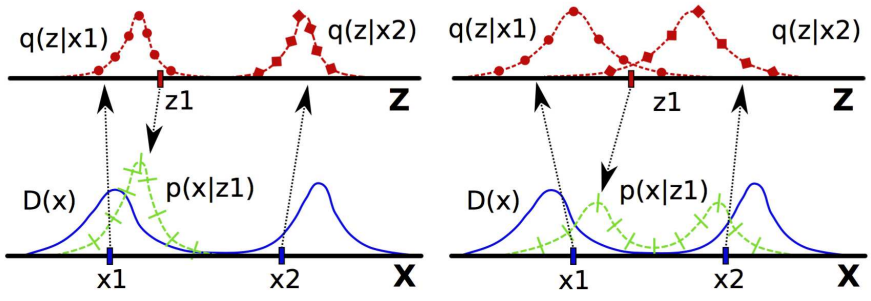


Figure 6–2: (a) The corruption process $q(z|x)$ is more local, making $p(x|z)$ simple but causing slow mixing between modes of $\mathcal{D}$ in the q/p chain. (b) The corruption process $q(z|x)$ is less local, making $p(x|z)$ tricky to model but causing fast mixing between modes of $\mathcal{D}$ in the q/p chain.

training would appear when sampling from the resulting Markov chain. For such "untrained" values of z, the reconstruction distribution $p_\theta(x|z)$ may perform poorly, by not having sufficiently small $\mathrm{KL}(\mathcal{P}(x|z) \,||\, p_\theta(x|z))$. This breaks the assumption that $p_\theta(x|z)$ provides a consistent estimator for $\mathcal{P}(x|z)$ and may lead to degenerate behavior in the Markov chain constructed from p_θ and q_ϕ.

The consistency assumption can also break when the model used for the reconstruction distribution $p_\theta(x|z)$ is simply incapable of representing $\mathcal{P}(x|z)$. E.g. if the corruption applied by q_ϕ is too non-local, then $\mathcal{P}(x|z)$ may be multi-modal or contain other sophisticated structure that cannot be captured by a simple, unimodal $p_\theta(x|z)$. The desire to keep q_ϕ local, and thus keep $\mathcal{P}(x|z)$ simple, must be weighed against the desire to produce a Markov chain that mixes well. If $\mathcal{D}$ contains multiple well-separated modes, q_ϕ *must* be somewhat non-local for the q_ϕ/p_θ chain to mix well. See Fig. 6–2 for an illustration of this balancing act between local and non-local corruption.

The walkback procedure can be interpreted as a "wrapper" function which takes the corruption process q_ϕ and the reconstruction distribution p_θ, and then samples from a process $\mathcal{W}(z|x; q_\phi, p_\theta)$ which procedurally generates a distribution over $z \in \mathcal{Z}$ given any $x \in \mathcal{X}$, as shown in Alg. 4. The samples (x, z) generated by Algorithm 4 are used to train the reconstruction model $p_\theta(x|z)$, and the reconstruction loss is also backpropagated through the sampling steps, which provides an additional training signal for p_θ and q_ϕ. For example, in the original GSN paper [9], the reconstruction distribution $p_\theta(x|z)$ for a GSN which emulates Gibbs sampling in a Deep Boltzmann Machine was trained on pairs (x, z) sampled from the walkback process described in Alg 4. The returned set of pairs $\mathcal{P}_{xz}$ can be viewed as containing data $(x, z) \sim \mathcal{W}(z|x; q_\phi, p_\theta)$, where $\mathcal{W}$ is specified procedurally rather than directly. The reconstruction distribution $p_\theta(x|z)$ is then trained to approximate the conditional $\mathcal{P}_\mathcal{W}(x|z)$ in the $(x, \hat{z})$ pairs from Alg. 4.

6.3 Simple Generative Stochastic Networks

We define a "Simple GSN" as any GSN in which the corruption process renders z_t independent of z_{t-1} given x_{t-1}. Simple GSNs thus represent the minimal, direct extension of Generalized DAEs to corruption processes that may produce outputs in a different space from their inputs. Fig. 1(c) shows the structure of a simple GSN based on iteratively sampling from a walkback process $\mathcal{W}(z|x; q_\phi, p_\theta)$ and a reconstruction distribution $p_\theta(x|z)$.

The class of Simple GSNs is in fact quite general, and covers all GSNs trained with the walkback procedure. We now give versions of the theorems from [10] modified for Simple GSNs, which show that training with enough data and with

sufficiently powerful function approximators p_θ/q_ϕ produces a Markov chain whose asymptotic distribution exists and matches the target distribution.

Theorem 5. *If $p_\theta(x|z)$ is a consistent estimator of the true conditional distribution $\mathcal{P}(x|z)$ and the transition operator $T_\theta(x_{t+1}|x_t)$ that samples z_t from $q_\phi(z_t|x_t)$ and x_{t+1} from $p_\theta(x_{t+1}|z_t)$ defines an ergodic Markov chain, then as the number of examples used to train $p_\theta(x|z)$ goes to infinity (i.e. as $p_\theta(x|z)$ converges to $\mathcal{P}(x|z)$), the asymptotic distribution of the Markov chain defined by T_θ converges to the target distribution $\mathcal{D}$.*

The proof is a direct translation of the proof for Theorem 1 in [10], but with zs replacing $\tilde{x}$s. Briefly, drawing an initial x from $\mathcal{D}$ and then sampling alternately from $q_\phi(z|x)$ and $\mathcal{P}(x|z)$ is equivalent to sampling from a Gibbs chain for the joint distribution over (x_i, z_i) generated by repeatedly sampling $x_i \sim \mathcal{D}$ and $z_i \sim q_\phi(z|x_i)$. By assuming ergodicity, we guarantee the existence of an asymptotic distribution for the Markov chain. Since $p_\theta(x|z)$ converges to the true $\mathcal{P}(x|z)$, the asymptotic distribution of the chain converges to the marginal distribution of x in the Gibbs chain, which is just $\mathcal{D}(x)$. We now consider the ergodicity assumption.

Corollary 6. *Let $\mathcal{X}$ be a set in which every pair of points is connected by a finite-length path contained in $\mathcal{X}$. Suppose that for each $x \in \mathcal{X}$ there exists a "shell" set $\mathcal{S}_x \subseteq \mathcal{X}$ such that all paths between x and any point in $\mathcal{X} \setminus \mathcal{S}_x$ pass through some point in $\mathcal{S}_x$ whose shortest path to x has length > 0. Suppose that $\forall x' \in \mathcal{S}_x \cup \{x\}, \exists z_{xx'}$ such that $q_\phi(z_{xx'}|x) > 0$ and $p_\theta(x'|z_{xx'}) > 0$. Then, the Markov chain with transition operator $T_\theta(x_{t+1}|x_t) = \sum_z p_\theta(x_{t+1}|z) q_\phi(z|x_t)$ is ergodic.*

Proof. The chain is aperiodic because the assumptions imply that $\forall x, \exists z_{xx}$ such that $q_\phi(z_{xx}|x) > 0$ and $p_\theta(x|z_{xx}) > 0$, so $T_\theta(x_{t+1} = x|x_t = x) > 0$. To show that the chain is irreducible, note that by assumption, $\forall x' \in \mathcal{S}_x, T(x_{t+1} = x'|x_t = x) > 0$. For any $x' \notin \mathcal{S}_x$, consider the shortest path from x to x' and note that $\exists y \in \mathcal{S}_x$ on this path such that the shortest path between x and y is > 0 and $T_\theta(x_{t+1} = y|x_t = x) > 0$. Hence, $T(x_{t+1} = x'|x_t = x) > 0$, as the path $x \to x'$ can be decomposed into a finite sequence of finite-length segments, each with non-zero transition probability. Because the chain is over a finite state space it is also positive recurrent. As the chain is aperiodic, irreducible, and positive recurrent, it is also ergodic. □

The restricted dependency structure of Simple GSNs lets us avoid some complications faced by the proofs in [9]. Our proof of Corollary 2 also avoids reliance on an ϵ ball. The use of an ϵ ball in [9] breaks a bit in discrete or discontinuous spaces, in which paths starting at x with length $> \epsilon$ may contain no "segments" overlapping with the set of all paths starting at x with length $\leq \epsilon$. I.e. if we naïvely generalize the notion of an ϵ ball to discrete/discontinuous spaces it inadequately captures the critical concept of a "shell" of points around x with non-zero "minimum radius" that fully separates x from all points outside the shell, where x can transition directly to any point in the shell with non-zero probability.

Training $p_\theta(x|z)$ for any GSN using samples generated by walkback as described in Algorithm 4 corresponds to training a Simple GSN built around the reconstruction distribution p_θ and corruption process $\mathcal{W}(z|x; q_\phi, p_\theta)$. The key observation here is that the samples in $\mathcal{P}_{xz}$ generated by Algorithm 4 are obtained

by relating *multiple* sampled latent variables $\hat{z}$ back to the *single* observable variable x given as input. In Sec. 6.5 we present a mechanism based on Approximate Bayesian Computation that allows us to more directly shape the multi-step dynamics of a GSN's Markov chain. Our approach allows the chain to run "freely" during training, rather than always forcing it to return to its starting point, as walkback does.

One can interpret walkback as an effective way to construct a more dispersed distribution over the latent space $\mathcal{Z}$ than would be provided by the original corruption process q_ϕ, while shaping that dispersion to fit the model capacity of p_θ. Though not explicitly stated in the existing work on GSNs, it seems that balancing between maximizing dispersion of the corruption process and the ease of modeling the reconstruction distribution plays a role for GSNs analogous to balancing between minimizing the KL divergence $\mathrm{KL}(q_\phi(z|x)||p(z))$ and maximizing the expected conditional log-likelihood $\mathbb{E}_{z\sim q_\phi(z|x)} \log p_\theta(x|z)$ when training a generative model $p_\theta(x)$ with variational methods, or balancing between following the "natural dynamics" of the system and optimizing reward in policy search [77, 105]. The next section expands on this relation to variational inference, and Sec. 6.4.3 takes a more formal look at dispersion in the p_θ/q_ϕ system.

6.4 Variational Simple GSNs

We now develop a Simple GSN which can efficiently generate locally-coherent random walks along the manifold described by the target distribution $\mathcal{D}$, efficiently generate independent (approximate) samples from the target distribution $\mathcal{D}$, and efficiently compute a lower-bound on the log-likelihood assigned by the model to

arbitrary inputs. We do this by replacing the denoising auto-encoders in existing examples of Generalized DAEs and GSNs with variational auto-encoders [51, 80], while interpreting the two competing terms in the variational free-energy $\mathcal{F}$ (see Eq. 6.1) as representing an explicit trade-off between the dispersion of $q_\phi(z|x)$ and the ease of modeling $p_\theta(x|z)$.

Suppose that, in addition to $p_\theta(x|z)$ and $q_\phi(z|x)$, we also have access to a distribution $p_*(z)$ over $\mathcal{Z}$ (which could be learned or fixed a priori). Given these three distributions, we can define the *derived distribution* $p_\theta(x; p_*)$ such that $p_\theta(x; p_*) = \sum_z p_\theta(x|z)p_*(z)$, and the variational free-energy $\mathcal{F}(x; q_\phi, p_\theta, p_*)$, which provides an upper-bound on the negative log-likelihood of $x \in \mathcal{X}$ under $p_\theta(x; p_*)$:

$$\mathcal{F}(x; q_\phi, p_\theta, p_*) = -\sum_z [q_\phi(z|x) \log p_\theta(x|z)] + \mathrm{KL}(q_\phi(z|x)||p_*(z)) \tag{6.1}$$

$$\geq -\log p_\theta(x; p_*). \tag{6.2}$$

We now derive this bound step-by-step.

6.4.1 Explaining the Variational Free Energy

Given distributions $p_\theta(x|z)$, $q_\phi(z|x)$, and $p_*(z)$, we can define several *derived distributions*:

$$p_\theta(x; p_*) \triangleq \sum_z p_\theta(x|z)p_*(z) \tag{6.3}$$

$$p_\theta(z|x; p_*) \triangleq \frac{p_\theta(x|z)p_*(z)}{p_\theta(x; p_*)} \tag{6.4}$$

$$p_\theta(x, z; p_*) \triangleq p_\theta(x|z)p_*(z) = p_\theta(z|x; p_*)p_\theta(x; p_*). \tag{6.5}$$

Given these distributions, we now work "backwards" from $\log p_\theta(x; p_*)$:

$$\begin{aligned}
\log \quad p_\theta(x; p_*) &= \sum_z q_\phi(z|x) \log p_\theta(x; p_*) && (6.6)\\
&= \sum_z q_\phi(z|x) \log \frac{p_\theta(z|x; p_*) p_\theta(x; p_*)}{p_\theta(z|x; p_*)} \\
&= \sum_z q_\phi(z|x) \log \frac{p_\theta(x, z; p_*)}{p_\theta(z|x; p_*)} \\
&= \sum_z q_\phi(z|x) (\log p_\theta(x, z; p_*) - \log q_\phi(z|x) + \log q_\phi(z|x) - \log p_\theta(z|x; p_*)) \\
&= \sum_z q_\phi(z|x) \log \frac{p_\theta(x, z; p_*)}{q_\phi(z|x)} + \sum_z q_\phi(z|x) \log \frac{q_\phi(z|x)}{p_\theta(z|x; p_*)} \\
&= \sum_z q_\phi(z|x) \left(\log p_\theta(x|z) + \log \frac{p_*(z)}{q_\phi(z|x)} \right) + \mathrm{KL}(q_\phi(z|x) || p_\theta(z|x; p_*)) && (6.7)\\
&\geq \sum_z q_\phi(z|x) \log p_\theta(x|z) - \mathrm{KL}(q_\phi(z|x) || p_*(z)) && (6.8)\\
&\geq -\mathcal{F}(x; q_\phi, p_\theta, p_*), && (6.9)
\end{aligned}$$

where Eqns. 6.8-6.9 define the variational free-energy:

$$\begin{aligned}
\mathcal{F}(x; q_\phi, p_\theta, p_*) &= -\sum_z q_\phi(z|x) \log p_\theta(x|z) && (6.10)\\
&\qquad + \mathrm{KL}(q_\phi(z|x) || p_*(z)) \\
&\geq -\log p_\theta(x; p_*). && (6.11)
\end{aligned}$$

These equations follow from simple algebraic manipulation and Eqn. 6.7-6.8 comes from non-negativity and definition of KL.

In this derivation of the variational free-energy, we treated p_θ, q_ϕ, and p_* simply as computational mechanisms for producing valid distributions over the

appropriate spaces. This emphasizes the fact that, for any triplet of distributions (p_θ, q_ϕ, p_*), $\mathcal{F}(x; q_\phi, p_\theta, p_*)$ can be computed and gives a lower-bound on $\log p_\theta(x; p_*)$ for the *derived distribution* $p_\theta(x; p_*)$. I.e. we need not interpret any of these distributions as approximate posteriors, likelihoods, etc. Letting go of such interpretations may make it easier to find interesting, effective, new applications of these sorts of bounds.

Note that the derived distributions we used result strictly from interactions between $p_\theta(x|z)$ and $p_*(z)$, and are independent of $q_\phi(z|x)$. Therefore, we could change the domain of $q_\phi(z|x)$ to some alternate, arbitrary space $\mathcal{Y}$ such that $q_\phi(z|y)$ produces distributions over $\mathcal{Z}$ given inputs from $\mathcal{Y}$. Plugging such a q_ϕ into Eqn. 6.10 (and substituting some ys for some xs appropriately) still produces a valid free-energy $\mathcal{F}(x, y; q_\phi, p_\theta, p_*)$, which still upper-bounds the negative log-likelihood of x under $p_\theta(x; p_*)$. We refer to the resulting system comprising $q_\phi(z|y)$, $p_\theta(x|z)$, and $p_*(z)$ as a variational transcoder.

Variational transcoding is a very general mechanism which encompasses standard variational auto-encoders, where $\mathcal{Y} \equiv \mathcal{X}$, and methods for sequence-to-sequence learning, image-to-text generation, Bayesian classification, etc. E.g., the sequence-to-sequence learning method in [102] can be interpreted as a variational transcoder in which $p_\theta(x|z)/q_\theta(z|y)$ are constructed from LSTMs, $p_*(z)$ is, e.g., an isotropic Gaussian distribution over $\mathcal{Z}$, and the distributions output by $q_\phi(z|y)$ are fixed to be Dirac deltas over $\mathcal{Z}$. Moving to a more comprehensively Bayesian approach in such systems, rather than pure MAP, may prove quite beneficial.

6.4.2 Using a Modified Variational Free Energy

Given $\mathcal{F}(x; q_\phi, p_\theta, p_*)$, we can maximize a lower-bound on the expected log-likelihood of samples $x \sim \mathcal{D}$ under the model $p_\theta(x; p_*)$ by minimizing:

$$\mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\mathcal{F}(x; q_\phi, p_\theta, p_*)\right] = \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{z\sim q_\phi(z|x)}[-\log p_\theta(x|z)] + \mathrm{KL}(q_\phi(z|x)||p_*(z))\right] \tag{6.12}$$

$$= \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{z\sim q_\phi(z|x)}\left[-\log p_\theta(x|z) + \log\frac{q_\phi(z|x)}{p_*(z)}\right]\right]. \tag{6.13}$$

It is useful to compare this to the objective for Generalized DAEs, i.e. Eq. 4 in [10]:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{\tilde{x}\sim q_\phi(\tilde{z}|z)}[-\log p_\theta(x|\tilde{x}) + \lambda\Omega(\theta, x, \tilde{x})]\right], \tag{6.14}$$

in which $\Omega(\theta, x, \tilde{x})$ is a regularization term for limiting the capacity of p_θ when the number of available samples $x \sim \mathcal{D}$ is finite, and for preventing p_θ from becoming a trivial "identity mapping". The basic training process for both Eq. 6.12 and 6.14 can be described as follows: draw a sample $x \sim \mathcal{D}$, apply a random corruption to get $z/\tilde{x} \sim q_\phi(\cdot|x)$, then adjust the parameters to reduce $-\log p_\theta(x|\cdot)$ and an added regularization term. Training with walkback simply changes the $\tilde{x} \sim q_\phi(\tilde{z}|z)$ in Eq. 6.14 to $\tilde{x} \sim \mathcal{W}(\tilde{x}|x; p_\theta, q_\phi)$.

We consider the objective in Eq. 6.12 from a GSN perspective and treat it as comprising two terms: a reconstruction error $-\log p_\theta(x|z)$ and a dispersion maximization term $\mathrm{KL}(q_\phi(z|x)\,||\,p_*(z))$. Based on a desire to keep q_ϕ well-dispersed for all $x \in \mathcal{X}$, and to keep the degree of the dispersion relatively consistent across $x \in \mathcal{X}$, we replace the basic KL term in Eq. 6.12 with the following:

$$\lambda\left(\mathrm{KL}\left(q_\phi(z|x)||p_*(z)\right) + \gamma\left([\mathrm{KL}(q_\phi(z|x)||p_*(z)) - \bar{K}]_+\right)^2\right), \tag{6.15}$$

in which $\bar{K} = \mathbb{E}_{x\sim\mathcal{X}}\,\mathrm{KL}(q_\phi(z|x)||p_*(z))$ and $[\cdot]_+$ indicates positive rectification, i.e. truncating negative values to 0.

Eq. 6.15 penalizes both the magnitude and variance of the per-example KL, while avoiding any pressure to increase KL. In effect, we are treating the per-example term $\mathrm{KL}(q_\phi(z|x)||p_*(z))$ as a a generic random variable, and shaping its distribution to have small mean and low weight in its "right tail". Under an information-theoretic interpretation of our cost, in which $\mathrm{KL}(q_\phi(z|x)||p_*(z))$ describes the length of the code our model assigns to x, the penalty in Eq. 6.15 enacts the notion that none of our observations should receive a code significantly longer than the average code. We make this modification to allow tighter control over the trade-off between reconstruction fidelity and dispersion of the corruption process. Although re-weighting the KL term in Eq. 6.12 may seem odd to those already familiar with variational methods, we emphasize that the free-energy $\mathcal{F}(x; q_\phi, p_\theta, p_*)$ from Eq. 6.1 still provides a valid upper-bound on $-\log p_\theta(x; p_*)$. The next subsection further discusses this modified variational free-energy.

6.4.3 Relating Posterior KL to Mutual Information

If we assume that $q_\phi(z|x)$ is close enough to $p(z|x; p_*)$, then the term $\mathrm{KL}(q_\phi(z|x)||p_*(z))$ in Eqn. 6.10 approximates the mutual information between z and x when they are sampled from the joint distribution $p_\theta(x, z; p_*) = p_\theta(x|z)p_*(z)$.

This can be seen with the following bit of algebra:

$$
\begin{aligned}
I(x;z) &= \sum_x \sum_z p_\theta(x,z;p_*) \log \frac{p_\theta(x,z;p_*)}{p_\theta(x;p_*)p_*(z)} && (6.16)\\
&= \sum_x \sum_z p_\theta(z|x;p_*)p_\theta(x;p_*) \log \frac{p_\theta(z|x;p_*)p_\theta(x;p_*)}{p_\theta(x;p_*)p_*(z)} && (6.17)\\
&= \sum_x p_\theta(x;p_*) \sum_z p_\theta(z|x;p_*) \log \frac{p_\theta(z|x;p_*)}{p_*(z)} && (6.18)\\
&= \sum_x p_\theta(x;p_*) \, \mathrm{KL}(p_\theta(z|x;p_*) \,||\, p_*(z)) && (6.19)\\
&= \mathop{\mathbb{E}}_{x\sim p_\theta(x;p_*)} \left[\mathrm{KL}(p_\theta(z|x;p_*) \,||\, p_*(z))\right]. && (6.20)
\end{aligned}
$$

In other words, for the distributions $p_\theta(x;p_*)$, $p_\theta(x,z;p_*)$, and $p_\theta(z|x;p_*)$ derived from $p_\theta(x|z)$ and $p_*(z)$, the mutual information between x and z in the joint $p_\theta(x,z;p_*)$ is equal to the expected KL divergence of the derived posteriors $p_\theta(z|x;p_*)$ from the prior $p_*(z)$ when x is sampled from the derived marginal $p_\theta(x;p_*)$. Consequently, re-weighting the relevant KL term in the free-energy allows us to approximately control the mutual information between x and z in our directed generative models. When the prior $p_*(z)$ and generator $p_\theta(x|z)$ are both simple, e.g. isotropic Gaussians, a small mutual information between x and z implies a simply-structured $p_\theta(x;p_*)$.

6.5 Collaborative Generative Networks

In this section we present a general method for shaping the distribution $\mathcal{G}$ produced by a generative model g_θ to be *practically* indistinguishable from a target distribution $\mathcal{D}$. We use the term practically literally, not colloquially. I.e. we shape $\mathcal{G}$ so that a reasonably strong function approximator says it matches $\mathcal{D}$, up to the practical limits of the approximator. We develop this method so that, as described

in Section 6.6, we can use it to directly shape the dynamics of the Markov chain constructed from a variational Simple GSN.

The general approach of estimating the parameters of some generative model g_θ to minimize some computable measure of dissimilarity between $\mathcal{G}$ and $\mathcal{D}$ is called Approximate Bayesian Computation. Examples of Approximate Bayesian Computation include spectral methods based on the "method of moments", which train the parameters of g_θ to match some of the statistical moments of $\mathcal{G}$ to the corresponding moments observed in an empirical sample from $\mathcal{D}$, and more recent approaches based on minimizing the ability of some classifier to distinguish between $\mathcal{G}$ and $\mathcal{D}$ [38, 39, 32]. Motivated by the recent empirical success of this latter approach in training deep generative models [32], we develop a related approach which offers improved stability and a simpler proof of correctness.

We define an objective for jointly optimizing a *generator* function g_θ and a *guide* function f_ψ which shapes the distribution $\mathcal{G}$ produced by g_θ to match a target distribution $\mathcal{D}$. Our method can be interpreted as a collaboration between g_θ and f_ψ, in contrast with the adversarial approach presented in [32]. It is based on optimizing a term which encourages f_ψ to approximate the log-density-ratio $\log \frac{\mathcal{D}(x)}{\mathcal{G}(x)}$ for $x \in \mathcal{X}$ while also using f_ψ as a guidance signal for redistributing the mass emitted by g_θ. Our objective comprises two parts: one optimized by f_ψ and the other optimized by g_θ. We show that g_θ and f_ψ can simultaneously minimize their respective objectives if and only if $\forall x$, $\mathcal{G}(x) = \mathcal{D}(x)$ and $f_\psi(x) = \log \frac{\mathcal{D}(x)}{\mathcal{G}(x)} = 0$.

The objective $\mathcal{L}_f$ for f_ψ is the basic logistic regression loss for a binary classifier, under the assumption of equal prior probability for the positive class $\mathcal{D}$

and the negative class $\mathcal{G}$, i.e.:

$$\mathcal{L}_f = \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[\ell(f_\psi(x))\right] + \mathop{\mathbb{E}}_{x\sim\mathcal{G}}\left[\ell(-f_\psi(x))\right], \tag{6.21}$$

where $\ell(f) = \log(\exp(-f)+1)$ is the binomial deviance loss. The objective $\mathcal{L}_g$ for g_θ is based on, e.g., a one-sided absolute value loss:

$$\mathcal{L}_g = \mathop{\mathbb{E}}_{x\sim\mathcal{G}}\left[|f_\psi(x)| \cdot \mathbb{I}[f_\psi(x) < 0]\right], \tag{6.22}$$

in which $\mathbb{I}[\cdot]$ is a binary indicator function. Really, the loss for each $x \sim g_\theta$ can be any function of f_ψ which takes a constant value c (e.g., w.l.o.g., $c = 0$) for $f_\psi > 0$, and values greater than c for all $f_\psi < 0$. For practical purposes, one should probably stick with monotonically decreasing functions.

Theorem 7. *The objectives $\mathcal{L}_f$ and $\mathcal{L}_g$ are simultaneously optimized with respect to f_ψ and g_θ if and only if $\forall x \in \mathcal{X}, \mathcal{G}(x) = \mathcal{D}(x)$ and $f_\psi(x) = \log\frac{\mathcal{D}(x)}{\mathcal{G}(x)} = 0$.*

Proof. By definition of $\mathcal{L}_f$, it is minimized w.r.t. f_ψ if and only if $\forall x,\ f_\psi(x) = \log\frac{\mathcal{D}(x)}{\mathcal{G}(x)}$ [41]. If $\mathcal{L}_f$ is minimized w.r.t. f_ψ then we know furthermore that either $\forall x,\ f_\psi(x) = \log\frac{\mathcal{D}(x)}{\mathcal{G}(x)} = 0$ or $\exists x$ s.t. $f_\psi(x) = \log\frac{\mathcal{D}(x)}{\mathcal{G}(x)} > 0$ and $\exists x'$ s.t. $f_\psi(x') = \log\frac{\mathcal{D}(x')}{\mathcal{G}(x')} < 0$. The former situation results in $\mathcal{L}_g = 0$. The latter situation results in $\mathcal{L}_g > 0$, because $|f_\psi(x)| \cdot \mathbb{I}[f_\psi(x) < 0] \geq 0$ with equality only when $f_\psi(x) \geq 0$. Thus, whenever $\mathcal{L}_f$ is optimized w.r.t. f_ψ, $\mathcal{L}_g$ can obtain its minimum possible value of 0 if and only if $\forall x,\ \mathcal{D}(x) = \mathcal{G}(x)$. □

Note that this proof works for any $\mathcal{L}_g$ which involves an expectation (w.r.t. $x \sim \mathcal{G}$) over a quantity which is everywhere non-negative and equal to

0 if and only if $f_\psi(x) \geq 0$. We leave a detailed investigation of the relative merits of the various possible $\mathcal{L}_g$ for future work.

The key characteristic that distinguishes our objective from [32] is that, given a fixed guide function f_ψ, our objective for the generator g_θ pushes the mass in over-dense regions of $\mathcal{G}$ towards zero-contours of f_θ while leaving the mass in under-dense regions unmoved. In contrast, the adversarial objective in [32] drives all mass emitted by g_θ towards local maxima of f_ψ. These local maxima will typically correspond to points in $\mathcal{X}$, while the zero-contours sought by our objective will correspond to regions in $\mathcal{X}$. We can also incorporate additional terms in $\mathcal{L}_g$, under the restriction that the added terms are minimized when $\forall x,\ \mathcal{D}(x) = \mathcal{G}(x)$. E.g. moment matching terms for matching the mean and covariance of $x \sim \mathcal{G}$ with those of $x \sim \mathcal{D}$ can be included to help avoid the occasional empirical "collapses" of $\mathcal{G}$ that were described in [32].

6.6 Generating Random Walks on Manifolds

We now combine the variational Simple GSNs from Sec. 6.4 with the collaborative distribution shaping mechanism from Sec. 6.5. Our goal is to directly train the Markov chain constructed by unrolling a variational Simple GSN to produce locally-contiguous walks along the manifold of the target distribution $\mathcal{D}$, and to have the asymptotic distribution of the chain approximate $\mathcal{D}$.

The collaborative mechanism described in Sec. 6.5 pairs a generator g_θ with a guide function f_ψ. For the generator, we propose using a variational Simple GSN that has been unrolled into a Markov chain. We initialize the chain with a sample $x_0 \sim \mathcal{D}$ and then repeatedly generate $\{x_1, ..., x_t, x_{t+1}, ..., x_n\}$ by sampling

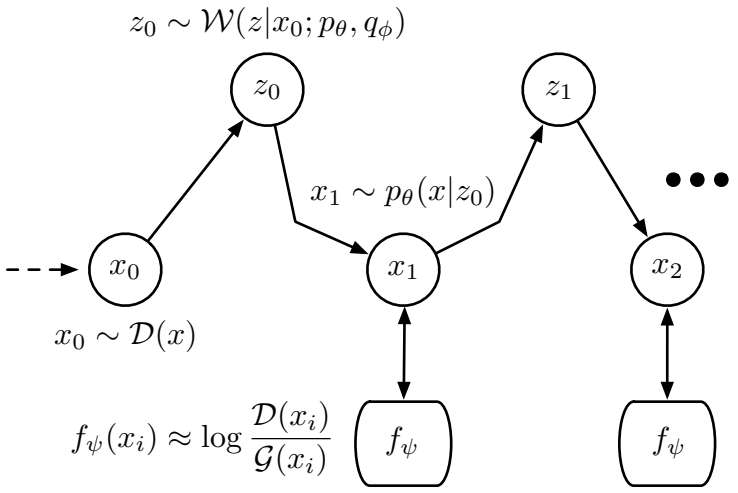
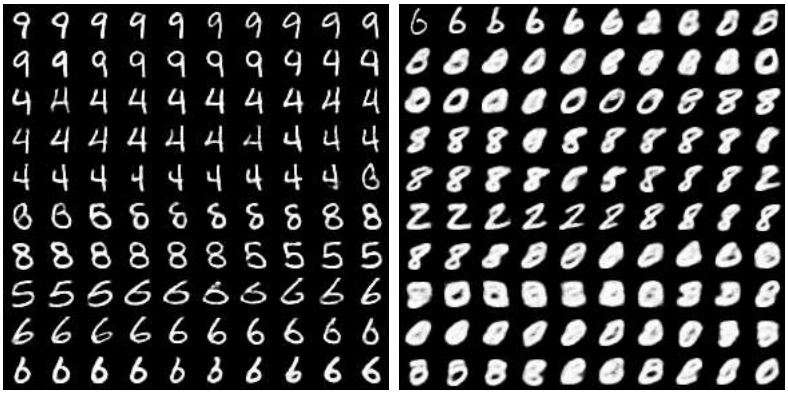


Figure 6–3: The VCG Loop: a self-looped variational auto-encoder whose asymptotic distribution is shaped by a guide function f_ψ using the collaborative mechanism described in Sec. 6.5. Loss for the p_θ/q_ϕ loop is measured using a combination of the modified free-energy in Eq. 6.12 and the collaborative shaping term in Eq. 6.22. We use stochastic backpropation through time to push this loss back through multiple steps of the p_θ/q_ϕ loop. Loss for f_ψ (according to Eq. 6.21) is based on an equal number of samples generated by the p_θ/q_ϕ and samples directly from $\mathcal{D}$. We backpropagate this loss through the computations performed by f_ψ as for a standard feedforward network.



(a) (b)

Figure 6–4: Comparing the chains generated by models learned with and without collaboratively-guided unrolling. The samples in (a) were generated by a corrupt-reconstruct pair q_ϕ/p_θ trained for 100k updates as a variational auto-encoder (VAE), and then 100k updates as a 6-step unrolled chain guided by a function f_ψ as described in Sec. 6.5. The samples in (b) were generated by a model with the same architecture and hyperparameters as in (a), but with 200k updates of VAE training. We show every fifth sample emitted by each chain.

$z_t \sim q_\phi(z|x_t)$ followed by $x_{t+1} \sim p_\theta(x|z_t)$. In other words, we self-loop a variational auto-encoder by piping its output back into its input. The unrolled joint system is depicted in Fig. 2. In practice, we implement p_θ, q_ϕ and the guide function f_ψ using neural networks, whose specific architectures are described further in Sec. 6.7. Intuitively, whenever the distribution generated by the p_θ/q_ϕ chain strays away from $\mathcal{D}$, as measured by f_ψ, the offending samples are guided back towards $\mathcal{D}$ by the gradient of f_ψ. In a sense, f_ψ builds an embankment around $\mathcal{D}$ which guides the chain back towards $\mathcal{D}$ whenever it starts to wander away.

6.7 Experiments

We now present tests examining the behavior of our models on the MNIST and TFD datasets. We chose these datasets to allow direct comparison with [9]

and [32]. We quantitatively measure the performance of our models, qualitatively illustrate interesting properties of the samples they generate, and illustrate the effect of controlling the posterior KL divergence $\mathrm{KL}(q_\phi(z|x)||p_*(z))$ more aggressively than in the standard variational free energy of Eq. 6.1.

Our first tests with MNIST data examined the benefits of training with explicit chain unrolling and collaborative shaping, as described in Sec. 6.6 and illustrated in Fig. 6–3. We represented q_ϕ and p_θ using neural networks with two hidden layers of 1500 rectified-linear units and we set the latent space $\mathcal{Z}$ to $\mathbb{R}^{64}$. We used a Gaussian with identity covariance for the prior distribution $p_*(x)$. The output of q_ϕ was a pair of vectors in $\mathbb{R}^{64}$, one representing the mean of a Gaussian distribution over $\mathcal{Z}$ and the other representing the element-wise log-variances of the distribution. Given this q_ϕ/p_*, $\mathrm{KL}(q_\phi(z|x)||p_*)$ was easy to compute analytically, and its gradients with respect to ϕ were readily available. For the reconstruction distribution, we defined $p_\theta(x|z)$ as a factored Bernoulli distribution, with Rao-Blackwellisation over the possible binarizations of each x. I.e., given a configuration z of the latent variables, the output of p_θ was a vector in $\mathbb{R}^{784}$ – the number of pixels in each MNIST digit – which was then passed through a sigmoid to get a "mean image" $\hat{x}$. The loss for $\hat{x}$ given a target image x was $-\log p_\theta(x|z) = -\mathrm{sum}(x \odot \log \hat{x} + (1-x) \odot \log(1-\hat{x}))$, where $\odot$ denotes element-wise multiplication and we sum the vector entries. The gradients of $-\log p_\theta(x|z)$ w.r.t. θ were directly available, and we backpropagated through sampling $z \sim q_\phi(z|x)$ to get gradients w.r.t. ϕ using the techniques in [51, 80].

We used a Maxout network [33] with two hidden layers of 1200 units in 300 groups of 4 for the guide function f_ψ. We used dropout and Gaussian noise regularization at the input and hidden layers of f_ψ. For $\mathcal{L}_g$ in Eq. 6.22, we used a half-rectified elastic-net [121], with the linear and quadratic parts weighted equally. We unrolled our chains for 6 steps. The distributions $\mathcal{D}$ and $\mathcal{G}$ for training f_ψ according to Eq. 6.21 were given by the MNIST training set and the x_i emitted by the unrolled chains. We passed gradients through the unrolled computation graph via (stochastic) backpropagation through time.

We performed model updates using gradients computed from mini-batches of 100 distinct examples from the training set. We passed each examples in each mini-batch through the unrolled model 5 times, thus generating 5 independent chain runs per example. We trained using basic SGD with momentum. We pre-trained p_θ and q_ϕ as a variational auto-encoder (VAE) for 100k updates by running the model in Fig. 6–1(c) for only a single step. After 100k VAE updates we "forked" the model into a "multi-step guided" model and a "one-step VAE" model and performed a further 100k updates to each of the now-independent models. We implemented our models in Python using the THEANO library [11]. Code implementing the models described in this paper is available online at: `github.com/Philip-Bachman/ICML-2015`. The code provides full details on learning rates and other hyper-parameters.

Fig. 6–4 illustrates the significantly improved dynamics of chains trained with unrolling and collaborative shaping. The chains in Fig. 6–4 start at the top left and run from left-to-right and top-to-bottom. The true samples from $\mathcal{D}$ used to

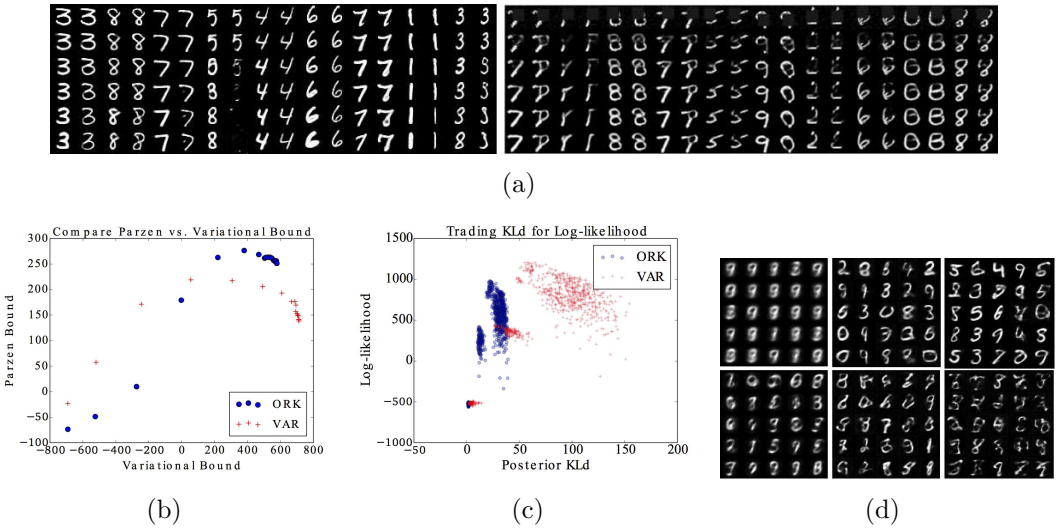
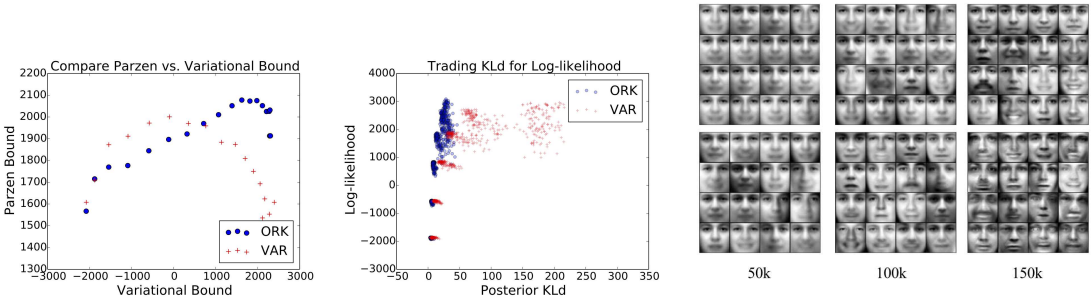


Figure 6–5: MNIST results. (a) compares the behavior of chains generated by models trained with over-regularized posterior KL divergence (ORK) and chains generated by models trained to minimize the standard variational free energy (VAR). The top row gives the image with which each chain was initialized, and the left/right chain in each pair sharing the same initialization represent chains generated by the ORK/VAR models. For the left block of examples the chains were allowed to run "freely" after initialization, and we show every 5th sample generated by the chains. For the right block the chains were run under "partial control", wherein some subset of the pixels were held fixed at their initial values while the remaining pixels were occluded at initialization and "painted in" by the model over multiple steps. We show every 2nd generated sample. (b) shows how the ORK and VAR models perform with respect to bounds on the validation set log-likelihood provided by the Gaussian Parzen density estimator described in [19] and the variational free energy described in Eq. 6.1. We computed these values after every 10k parameter updates. (c) shows the joint distribution over the reconstruction and posterior KL terms (i.e. $\log p_\theta(x|z)$ and $\mathrm{KL}(q_\phi(z|x)||p_*(z))$) in Eq. 6.1 measured at several points during the training of the ORK and VAR models. We computed these values after every 30k parameter updates by averaging over 10 passes through the underlying p_θ/q_ϕ VAE for each of 150 examples selected at random from a validation set. (d) shows independent samples from the ORK (top row) and VAR (bottom row) models after 30k, 60k, and 120k parameter updates.

(a)



(b) (c) (d)

Figure 6–6: TFD results. (a) compares the behavior of chains generated by models trained with over-regularized posterior KL divergence (ORK) and chains generated by models trained to minimize the standard variational free energy (VAR). The top row gives the image with which each chain was initialized, and the left/right chain in each pair sharing the same initialization represent chains generated by the ORK/VAR models. For the left block of examples the chains were allowed to run "freely" after initialization, and we show every 5th sample generated by the chains. For the right block the chains were run under "partial control", wherein some subset of the pixels were held fixed at their initial values while the remaining pixels were occluded at initialization and "painted in" by the model over multiple steps. We show every 2nd generated sample. (b) shows how the ORK and VAR models perform with respect to bounds on the validation set log-likelihood provided by the Gaussian Parzen density estimator described in [19] and the variational free energy described in Eq. 6.1. We computed these values after every 10k parameter updates. (c) shows the joint distribution over the reconstruction and posterior KL terms (i.e. $\log p_\theta(x|z)$ and $\mathrm{KL}(q_\phi(z|x)||p_*(z))$) in Eq. 6.1 measured at several points during the training of the ORK and VAR models. We computed these values after every 30k parameter updates by averaging over 10 passes through the underlying p_θ/q_ϕ VAE for each of 150 examples selected at random from a validation set. (d) shows independent samples from the ORK (top row) and VAR (bottom row) models after 50k, 100k, and 150k parameter updates.

initialize the chains are in the top left corners of (a)/(b). We downsampled the samples emitted by these chains 5x for this figure. Training with unrolling and collaborative shaping allows the chain to continually generate clean samples while exhibiting rapid mixing between the modes of the target distribution. Without explicit unrolling during training, the chain can only perform a few steps before degrading into samples that are well-separated from the target distribution. Qualitatively, the samples generated by the model trained with collaboratively-guided unrolling compare favourably with those presented for GSNs in [9].

For our second tests with the MNIST images, we used networks with roughly the same structure as in our first tests but with $\mathcal{Z}$ set to $\mathbb{R}^{50}$, 1000 units in each of the hidden layers in q_ϕ/p_θ, and a Gaussian reconstruction distribution $p_\theta(x|z)$. To effect this final change, we interpreted the vector produced by the output layer of p_θ as the mean of a Gaussian distribution over $\mathcal{X}$ and we shared a single "bias" parameter across all z to model the element-wise log-variance of $p_\theta(x|z)$. We thus modeled the target distribution using an infinite mixture of isotropic Gaussians, with the mixture weights of each Gaussian fixed a priori and with their individual locations and shared scale adjusted to match the training data.

For Fig. 6–5(a)-(d), we trained a pair of models. We trained the first model with $\lambda = 4$ and $\gamma = 0.1$ in Eq. 6.15. We refer to this model as ORK, for over-regularized KL. We trained the second model to minimize the basic free energy in Eq. 6.1. We refer to this model as VAR. As in our first MNIST experiments, we initialized each model with pre-training by running the model in Fig. 6–1(c) a single step. We performed 80k pre-training updates using mini-batches and SGD

updates as described for our first experiments. We then performed another 120k updates by unrolling the chains for 6 steps following the graph in Fig. 6–3. We defined and trained the guide function f_ψ as in our first experiments.

Fig. 6–5 illustrates interesting behaviors of the ORK and VAR models. We found that the ORK model with strong regularization on $\mathrm{KL}(q_\phi(z|x)||p_*(z))$ was able to significantly out-perform the VAR model according to the Gaussian Parzen density estimator (GPDE) test described in [19]. The GPDE test approximates the distribution of a generative model by drawing 10k samples from the model and then using those samples as the mixture means for a uniformly-weighted mixture of 10k Gaussians, with a shared isotropic variance selected for the mixture components based on a validation set. On the test set, the best ORK model scored 265, which compares favorably to the 214 in [9] and the 225 in [32]. The best VAR model scored 220, which is somewhat surprising given the visually-poor quality of its generated samples compared to those in [9, 32]. The peak performance by this metric occurred much earlier in training for the VAR model than for the ORK model. Using the same network architecture, but with λ in Eq. 6.15 increased to 24, we were able to achieve a score of 330 on the GPDE test.

Interestingly, the GPDE log-likelihood bound began to decrease rapidly beyond a certain point in training, even while the variational bound continued to increase. Qualitatively, this behavior is clearly reflected in the samples shown in Fig. 6–5(d), which we drew directly from the models by sampling from their priors $p_*(x)$. Samples generated from the ORK model remain reasonable throughout training, but eventually suffer on the GPDE bound due to excess "sharpness".

Samples drawn from the VAE model after 150k updates, at which point the VAE model significantly outperforms the ORK model in terms of the variational bound, are hardly recognizable as handwritten digits. In effect, the model is concentrating its posterior mass, as given by $q_\phi(z|x)$, on increasingly small regions of $\mathcal{Z}$, in exchange for significant reductions in the reconstruction cost $-\log p_\theta(x|z)$. This seems to lead to most of the mass of $p_*(z)$ falling on zs which have little or no mass under any of the $q_\phi(z|x)$ estimated for the xs in our finite training set. By forcing the $q_\phi(z|x)$ to be more dispersed over $\mathcal{Z}$, our added KL terms in Eq. 6.15 help mitigate this issue, albeit at the cost of less precise reconstruction of any particular digit. The scatter plots in Fig. 6–5 illustrate the evolution of the GPDE/variational bounds over the course of training and the particular trade-off between reconstruction cost and posterior KL that is obtained by the ORK and VAR models.

We performed analogous tests with the TFD dataset, which comprises 48x48 grayscale images of frontal faces in various expressions. We made a few changes from the similar MNIST tests. We expanded the hidden layers of the networks representing q_ϕ/p_θ to 2000 rectified-linear units each and we expanded the latent space $\mathcal{Z}$ to $\mathbb{R}^{100}$. We preprocessed the images to have pixel intensities in the range [0...1], as in [9, 32]. We extended the pre-training phase to 150k updates and the unrolled, collaboratively-guided phase was reduced to 60k updates. Interestingly, in these tests the ORK model did not suffer significantly in terms of the variational bound, while achieving dramatically improved performance on the GPDE bound. For comparison, best previous results on the GPDE bound for this

dataset are 2050 [32] and 2110 [10]. Our ORK model scored 2060 on the test set using a GPDE variance selected on the validation set. When we increased λ for the ORK model from 5 to 30, the GPDE score increased to 2130.

6.8 Discussion

In this chapter, we developed an approach for learning generative models belonging to a simple, restricted subset of Generative Stochastic Networks. We used variational auto-encoders as building blocks. We generated Markov chains by looping the output of these auto-encoders back into the input, and trained them to generate random walks along a target manifold by shaping the Markov chain dynamics using feedback from a guide function trained to discriminate between samples emitted by the chain and samples drawn from the data manifold. A key conceptual contribution of our approach is that we run the generative process as an unrolled Markov chain according to its natural dynamics, i.e. the same way we want to run it "in the wild", and then correct differences between exhibited and desired behavior by providing direct feedback about their magnitude and location – as opposed to forcing the behavior in some way during training and/or generation. Our experimental evaluations demonstrate that this direct approach to shaping behavior is beneficial in practice.

In the long run, we believe it will be interesting to focus on interpreting our method from a reinforcement learning point of view. In this case, the "ease of modelling" the reconstruction distribution $p(x|z)$ can be viewed as a reward to be maximized, and may be easily replaced or augmented with other sources of reward. The current approach of using backpropagation through time for the training could

also be replaced by more efficient methods based on eligibility traces. While we only considered generating random walks over manifolds in this paper, in future work we would like to apply our approach to modelling distributions over observed trajectories on manifolds, as seen in, e.g., speech, video, motion capture, and other sequential data.

CHAPTER 7
Data Generation as Sequential Decision Making

7.1 Motivation

The previous chapter developed models for generating random walks through some data distribution. One difficulty we encountered during this work was that mapping from a complicated observation distribution to a significantly simpler posterior distribution was difficult to accomplish in a single step. We wanted simple posterior distributions when generating random walks because this would allow the walks to mix faster (due to more overlap in the posteriors of similar observations), but this was hard to achieve while maintaining precise reconstruction of the target distribution. Intuitively, it may be easier to construct a complicated distribution by sequentially composing multiple simpler distributions, rather than transforming a simple distribution into a complicated distribution in a single step. This is another form of the intuition which motivates feature learning and feature composition in deep learning. During our work in the previous chapter, we also found it difficult to directly train Markov chains to perform conditional inference of missing values. Thus, in this chapter we develop methods for training generative models which construct complex distributions by sequentially composing simpler distributions, and which are capable of directly conditioning on exogenous information and on feed-back from their own behavior. The sequential nature of these models also leads to strong connections with reinforcement learning.

Directed generative models are naturally interpreted as specifying sequential procedures for generating data. Given the directed (acyclic) graph for such a model, one can generate a sample observation by sampling each node conditioned on its parents, as soon as all of its parents have been sampled. Nodes without parents can be sampled from any distribution without latent variables – the latent variables would otherwise be their parents in the directed model – including distributions with trainable parameters.

Traditionally, we think of this process as sampling, but one could also view this process as a sequence of *decisions* for how to set the value of each subsequent node, conditioned on the values of its parents, which procedurally generates data from the distribution.[1] Under this interpretation, one can view the conditional distribution of any particular node, given its parents, as a stochastic policy for setting that node's value. Hence, setting the value of each node, while generating data from a directed generative model, can be likened to taking an action in a Markov Decision Process (abbr. MDP). The large body of existing work on reinforcement learning provides a powerful set of conceptual and algorithmic tools

[1] We permit a broad interpretation of each node in a directed generative model. I.e., each node can be discrete or continuous, while representing a scalar, vector, structured object, etc. Also note that most (finite-time) procedures for sampling from undirected graphs or cyclic directed graphs can be straightforwardly represented by acyclic directed graphs, possibly with some parameters shared between multiple nodes in the graph. E.g., n steps of Gibbs sampling in an RBM can be represented by n clones of the underlying undirected model, arranged into a directed Bayesian network.

with which to address such sequential decision making problems. With the work in this chapter, we hope to encourage a more complete transfer of these tools to the extended processes driving advances in generative modelling.

We begin by reinterpreting several recently introduced generative models as sequential decision making processes, and show how changes inspired by our point of view can improve the performance of the LSTM-based model introduced in [36]. We then explore the connections between directed generative models and reinforcement learning more fully by developing an approach to training policies for sequential data imputation. We base our approach on formulating data imputation as a finite-horizon MDP which can also be interpreted as a deep, directed graphical model. As the data to impute grows to cover the full observation, we show that our approach successfully transitions to classical (i.e. unconditional) generative modelling.

We propose two policy representations for the sequential imputation MDP. One extends the model in [36] by adding an explicit feedback loop into the generative process, and the other implements the MDP more literally. We train our models/policies using techniques motivated by Guided Policy Search [64, 65, 66, 63].

The objective functions and algorithms we use to train our policies can also be interpreted as maximizing a variational bound on the log-likelihood of the correct imputations. Under this interpretation, we train generative models with structural similarities to NADE-k [78], Multi-prediction Boltzmann Machines [31], or the multi-modal conditional RBM from [95]. Our models are designed so

that we can use the effective methods developed in [51, 80] for training directed generative models via parametric variational inference. Perhaps the largest benefit of our approach is that including latent variables in the model, rather than having a deterministic path from input to a collection of independent predictions for each output variable, allows us to capture complicated relationships among the output variables without resorting to post-hoc application of an MRF/CRF. Contemporary with our work, similar motivations lead to a variational (non-recurrent) conditional generative model in [96]. Our approach can be seen as a generalization of [96], in which the prediction is refined over multiple stochastic passes through a model like that described in [96]. I.e., our approach can incorporate multiple stochastic refinement steps and each refinement step can involve multiple steps of sampling.

We examine the performance of our models across imputation problems covering a range of difficulties (i.e. different amounts of data to impute and different "missingness mechanisms"), and across multiple datasets. We compare our models to each other and to two simple baselines. Our models significantly outperform the baselines. We also present preliminary tests that exhibit the value of temporal depth in the sequential imputation process. Overall, we propose that developing better models and better benchmarks for imputation is a worthwhile endeavor, as imputation includes both classification and generative modelling as special cases.

7.2 Directed Generative Models as Sequential Decision Processes

Directed generative models have grown rapidly in popularity relative to their undirected counter-parts [51, 80, 71, 37, 50, 94, 79] (etc.). Reasons include: the development of efficient methods for training them, the ease of sampling from them, and the frequent availability of efficient bounds on their log-likelihoods. The growth in available computing power compounds these benefits. While data generation is typically viewed as a process of sampling from a sequence of latent variables in these models, one can also think of *setting* the latent variables to appropriate values, in a sequence of decisions conditioned on preceding decisions. In a sense, these models encode stochastic procedures for constructing plausible observations. The rest of this section develops this point of view.

7.2.1 Deep AutoRegressive Networks

The deep autoregressive networks investigated in [37] define distributions of the following form:

$$p(x) = \sum_z p(x|z)p(z), \quad \text{with} \quad p(z) = p_0(z_0) \prod_{t=1}^{T} p_t(z_t|z_0, ..., z_{t-1}), \tag{7.1}$$

in which x indicates a generated observation and $z_0, ..., z_T$ represent the $T + 1$ latent variables in the model. The distribution $p(x|z)$ may also be factored similarly to $p(z)$. The factored form of $p(z)$ in Eqn. 7.1 can represent arbitrary distributions over the latent variables, and work in [37] mainly concerned approaches to parameterizing the conditionals $p_t(z_t|z_0, ..., z_{t-1})$ that restricted representational power in exchange for computational tractability. To appreciate the generality of Eqn. 7.1, one might consider using z_t that are univariate, multivariate, structured, etc.

Constructing a model via this sequential factorization permits a natural approach to training it.

7.2.2 Generalized Guided Policy Search

We adopt a broader interpretation of Guided Policy Search than one might initially take from, e.g., [64, 65, 66, 63]. Our definition of guided policy search includes any optimization of the general form:

$$\underset{p,q}{\text{minimize}} \underset{i_q \sim \mathcal{I}_q}{\mathbb{E}} \underset{i_p \sim \mathcal{I}_p(\cdot|i_q)}{\mathbb{E}} \left[\underset{\tau \sim q(\tau|i_q,i_p)}{\mathbb{E}} \left[\ell(\tau, i_q, i_p) \right] + \lambda \operatorname{div}\left(q(\tau|i_q, i_p), p(\tau|i_p) \right) \right], \quad (7.2)$$

in which p indicates the *primary policy*, q indicates the *guide policy*, $\mathcal{I}_q$ indicates a distribution over information available only to q, $\mathcal{I}_p$ indicates a distribution over information available to both p and q, $\ell(\tau, i_q, i_p)$ computes the cost of trajectory τ in the context of i_q/i_p, and $\operatorname{div}(q(\tau|i_q, i_p), p(\tau|i_p))$ computes some measure of dissimilarity between the trajectory distributions generated by p/q. As $\lambda \in \mathbb{R}^+$ goes to infinity, Eqn. 7.2 enforces the constraint $p(\tau|i_p) = q(\tau|i_q, i_p)$, $\forall \tau, i_p, i_q$. Additional terms for controlling, e.g., the entropy of p/q can also be added. E.g., one might encourage $q(\tau|i_q, i_p)$ to vary smoothly with changes in i_q/i_p. The power of the objective in Eq. 7.2 stems from two main points: the guide policy q can use information i_q that is not available to the primary policy p, and the primary policy need only be trained to minimize the dissimilarity term $\operatorname{div}(q(\tau|i_q, i_p), p(\tau|i_p))$.

For example, directed models structured as in Eqn. 7.1 can be interpreted as specifying policies for a finite-horizon MDP whose terminal state distribution encodes $p(x)$. In this MDP, the state at time $1 \leq t \leq T+1$ is defined by $\{z_0, ..., z_{t-1}\}$. The policy picks an action $z_t \in \mathcal{Z}_t$ at time $1 \leq t \leq T$, and an action

$x \in \mathcal{X}$ at time $t = T + 1$. I.e., the policy can be written as $p_t(z_t|z_0, ..., z_{t-1})$ for $1 \leq t \leq T$, and as $p(x|z_0, ..., z_T)$ for $t = T + 1$. The initial state $z_0 \in \mathcal{Z}_0$ is drawn from $p_0(z_0)$. Executing the policy for a single trial produces a trajectory $\tau \triangleq \{z_0, ..., z_T, x\}$, and the distribution over xs from these trajectories is just $p(x)$ in the corresponding directed generative model.

The authors of [37] train deep autoregressive networks by maximizing a variational lower bound on the training set log-likelihood. To do this, they introduce a variational distribution q which provides $q_0(z_0|x^*)$ and $q_t(z_t|z_0, ..., z_{t-1}, x^*)$ for $1 \leq t \leq T$, with the final step $q(x|z_0, ..., z_T, x^*)$ given by a Dirac-delta at x^*. Given these definitions, the training in [37] can be interpreted as guided policy search for the MDP described in the previous paragraph. Specifically, the variational distribution q provides a guide policy $q(\tau|x^*)$ over trajectories $\tau \triangleq \{z_0, ..., z_T, x^*\}$:

$$q(\tau|x^*) \triangleq q(x|z_0, ..., z_T, x^*) q_0(z_0|x^*) \prod_{t=1}^{T} q_t(z_t|z_0, ..., z_{t-1}, x^*). \tag{7.3}$$

The primary policy p generates trajectories distributed according to:

$$p(\tau) \triangleq p(x|z_0, ..., z_T) p_0(z_0) \prod_{t=1}^{T} p_t(z_t|z_0, ..., z_{t-1}), \tag{7.4}$$

which does not depend on x^*. In this case, x^* corresponds to the guide-only information $i_q \sim \mathcal{I}_q$ in Eqn. 7.2. We now rewrite the variational optimization as:

$$\underset{p,q}{\text{minimize}} \underset{x^* \sim \mathcal{D}_x}{\mathbb{E}} \left[\underset{\tau \sim q(\tau|x^*)}{\mathbb{E}} [\ell(\tau, x^*)] + \text{KL}(q(\tau|x^*) \,||\, p(\tau)) \right], \tag{7.5}$$

where $\ell(\tau, x^*) \triangleq 0$ and $\mathcal{D}_x$ indicates the target distribution for the terminal state of the primary policy p.[2] When expanded, the KL term in Eqn. 7.5 becomes:

$$\mathrm{KL}(q(\tau|x^*) \,||\, p(\tau)) = \tag{7.6}$$
$$\mathop{\mathbb{E}}_{\tau \sim q(\tau|x^*)} \left[\log \frac{q_0(z_0|x^*)}{p_0(z_0)} + \sum_{t=1}^{T} \log \frac{q_t(z_t|z_0, ..., z_{t-1}, x^*)}{p_t(z_t|z_0, ..., z_{t-1})} - \log p(x^*|z_0, ..., z_T) \right].$$

Thus, the variational approach used in [37] for training directed generative models can be interpreted as a form of generalized guided policy search. As the form in Eqn. 7.1 is broad enough to represent any finite directed generative model, the preceding derivation extends to all models we consider in this chapter.

7.2.3 Time-reversible Stochastic Processes

One can simplify Eqn. 7.1 by assuming suitable relationships among $\mathcal{X}$ and $\mathcal{Z}_0, ..., \mathcal{Z}_T$. E.g., the authors of [94] proposed a model in which $\mathcal{Z}_t \triangleq \mathcal{X}$ for all t and $p_0(x_0)$ is a Gaussian. Under these assumptions we write the model as:

$$p(x_T) = \sum_{x_0, ..., x_{T-1}} p_T(x_T|x_{T-1}) p_0(x_0) \prod_{t=1}^{T-1} p_t(x_t|x_{t-1}), \tag{7.7}$$

where $p(x_T)$ indicates the terminal state distribution of the non-stationary, finite-horizon Markov process determined by $\{p_0(x_0), p_1(x_1|x_0), ..., p_T(x_T|x_{T-1})\}$. Note

[2] As defined, this MDP involves no reward, which is peculiar. We could pull the $-\log p(x^*|z_0, ..., z_T)$ term from the KL and put it in the cost $\ell(\tau, x^*)$, but we prefer the "path KL" formulation for its elegance. We further discuss the significance of the path KL objective in Sec. **??**. We abuse notation by defining $\mathrm{KL}(\delta(x = x^*) \,||\, p(x))$ as $-\log p(x^*)$.

that, throughout this paper, we (ab)use sums over latent variables and trajectories which could/should be written as integrals.

The authors of [94] observed that, for any reasonably smooth target distribution $\mathcal{D}_x$ and sufficiently large T, one can define a (reverse-time) stochastic process $q_t(x_{t-1}|x_t)$ with simple, time-invariant dynamics that transforms the distribution $q(x_T) \triangleq \mathcal{D}_x$ into the Gaussian distribution $p_0(x_0)$. We can write this q as:

$$q_0(x_0) = \sum_{x_1,...,x_T} q_1(x_0|x_1)\mathcal{D}_x(x_T) \prod_{t=2}^{T} q_t(x_{t-1}|x_t) \approx p_0(x_0). \tag{7.8}$$

Next, we define $q(\tau)$ as the distribution over trajectories $\tau \triangleq \{x_0, ..., x_T\}$ generated by the reverse-time process determined by $\{q_1(x_0|x_1), ..., q_T(x_{T-1}|x_T), \mathcal{D}_x(x_T)\}$. We define $p(\tau)$ as the distribution over trajectories generated by the forward-time process determined by $\{p_0(x_0), p_1(x_1|x_0), ...p_T(x_T|x_{T-1})\}$. The training in [94] performs guided policy search using guide trajectories sampled from q, i.e.:

$$\underset{p,q}{\text{minimize}} \; \mathop{\mathbb{E}}_{\tau\sim q(\tau)} \left[\log \frac{q_1(x_0|x_1)}{p_0(x_0)} + \sum_{t=1}^{T-1} \log \frac{q_{t+1}(x_t|x_{t+1})}{p_t(x_t|x_{t-1})} + \log \frac{\mathcal{D}_x(x_T)}{p_T(x_T|x_{T-1})} \right], \tag{7.9}$$

which just minimizes $\mathrm{KL}(q||p)$. If the log-densities in Eqn. 7.9 are tractable, which can be readily obtained by construction, then this minimization can be done using basic Monte-Carlo. If, as in [94], the reverse-time process q is not trained, then Eqn. 7.9 simplifies to: $\text{minimize}_p \, \mathbb{E}_{q(\tau)} \left[-\log p_0(x_0) - \sum_{t=1}^{T} \log p_t(x_t|x_{t-1}) \right]$.

This trick for generating guide trajectories exhibiting a particular distribution over terminal states x_T – i.e. running dynamics backwards in time starting from $x_T \sim \mathcal{D}_x$ – may prove useful in settings other than those considered in [94]. We

provide further discussion of this material in the next subsection. In particular, we derive Eqn. 7.9 as an upper bound on $\mathbb{E}_{x\sim\mathcal{D}_x}[-\log p(x)]$.

7.2.4 A Path-wise KL Bound

We now show that the objective in Eqn. 7.9 describes the KL divergence $\mathrm{KL}(q_\tau \,||\, p_\tau)$, and that it provides an upper bound on $\mathbb{E}_{\mathcal{D}_x}[-\log p(x_T)]$. First, for $\tau \triangleq \{x_0, ..., x_T\}$, we define:

- $p(\tau_{>0}|x_0) \triangleq p(x_1, ..., x_T|x_0) \triangleq \prod_{t=1}^{T} p_t(x_t|x_{t-1})$.
- $p(\tau) \triangleq p(x_1, ..., x_T|x_0)p_0(x_0) \triangleq p_0(x_0)\prod_{t=1}^{T} p_t(x_t|x_{t-1})$.
- $q(\tau_{<T}|x_T) \triangleq q(x_0, ..., x_{T-1}|x_T) \triangleq \prod_{t=1}^{T} q_t(x_{t-1}|x_t)$.
- $q(\tau) \triangleq q(x_0, ..., x_{T-1}|x_T)\mathcal{D}_x(x_T) \triangleq \mathcal{D}_x(x_T)\prod_{t=1}^{T} q_t(x_{t-1}|x_t)$.

Next, we derive:

$$
\begin{aligned}
p(x_T) &= \sum_{x_0,\ldots,x_{T-1}} p_0(x_0) p(x_1, ..., x_T|x_0) \frac{q(\tau_{<T}|x_T)}{q(\tau_{<T}|x_T)} && (7.10)\\
&= \sum_{x_0,\ldots,x_{T-1}} p_0(x_0) p(\tau_{>0}|x_0) \frac{q(\tau_{<T}|x_T)}{q(\tau_{<T}|x_T)} && (7.11)\\
&= \sum_{x_0,\ldots,x_{T-1}} q(\tau_{<T}|x_T) \frac{p_0(x_0) p(\tau_{>0}|x_0)}{q(\tau_{<T}|x_T)} && (7.12)\\
&= \sum_{x_0,\ldots,x_{T-1}} q(x_0, ..., x_{T-1}|x_T) \cdot \left(p_0(x_0) \prod_{t=1}^{T} \frac{p_t(x_t|x_{t-1})}{q_t(x_{t-1}|x_t)} \right) && (7.13)\\
\log p(x_T) &\geq \sum_{x_0,\ldots,x_{T-1}} q(x_0, ..., x_{T-1}|x_T) \cdot \log \left(p_0(x_0) \prod_{t=1}^{T} \frac{p_t(x_t|x_{t-1})}{q_t(x_{t-1}|x_t)} \right) && (7.14)\\
&\geq \mathop{\mathbb{E}}_{q(\tau_{<T}|x_T)} \left[\log p_0(x_0) - \log \prod_{t=1}^{T} \frac{q_t(x_{t-1}|x_t)}{p_t(x_t|x_{t-1})} \right] && (7.15)\\
&\geq \mathop{\mathbb{E}}_{q(\tau_{<T}|x_T)} \left[\log p_0(x_0) - \log \frac{q(\tau_{<T}|x_T)}{p(\tau_{>0}|x_0)} \right] && (7.16)\\
&\geq \mathop{\mathbb{E}}_{q(\tau_{<T}|x_T)} [\log p_0(x_0)] - \mathrm{KL}(q(\tau_{<T}|x_T) \,||\, p(\tau_{>0}|x_0)), && (7.17)
\end{aligned}
$$

which provides a lower bound on $\log p(x_T)$ based on sample trajectories produced by the reverse-time process q when it is started at x_T. The transition from equality to inequality is due to Jensen's inequality. Though $q(\tau_{<T}|x_T)$ and $p(\tau_{>0}|x_0)$ may at first seem incommensurable via KL, they both represent distributions over T-step trajectories through $\mathcal{X}$ space, and thus the required KL divergence is well-defined. Next, by adding an expectation with respect to $x_T \sim \mathcal{D}_x$, we derive a lower bound

on the expected log-likelihood $\mathbb{E}_{\mathcal{D}_x}[\log p(x_T)]$:

$$\begin{aligned}
\log p(x_T) &\geq \mathop{\mathbb{E}}_{q(\tau_{<T}|x_T)}\left[\log p_0(x_0) - \log \frac{q(\tau_{<T}|x_T)}{p(\tau_{>0}|x_0)}\right] && (7.18)\\
\mathop{\mathbb{E}}_{x_T\sim\mathcal{D}_x}[\log p(x_T)] &\geq \mathop{\mathbb{E}}_{x_T\sim\mathcal{D}_x}\left[\mathop{\mathbb{E}}_{q(\tau_{<T}|x_T)}\left[\log p_0(x_0) - \log \frac{q(\tau_{<T}|x_T)}{p(\tau_{>0}|x_0)}\right]\right] && (7.19)\\
&\geq \mathop{\mathbb{E}}_{q(\tau)}\left[\log p_0(x_0) - \log \frac{q(\tau_{<T}|x_T)}{p(\tau_{>0}|x_0)}\right] && (7.20)\\
&\geq \mathop{\mathbb{E}}_{q(\tau)}\left[-\log \frac{\mathcal{D}(x_T)q(\tau_{<T}|x_T)}{p_0(x_0)p(\tau_{>0}|x_0)}\right] - H_{\mathcal{D}_x} && (7.21)\\
&\geq -\mathrm{KL}(q(\tau)\,||\,p(\tau)) - H_{\mathcal{D}_x}. && (7.22)
\end{aligned}$$

These steps follow directly from the definitions of $q(\tau_{<T}|x_T)$ and $q(\tau)$. In the last two equations, we define $H_{\mathcal{D}_x} \triangleq \mathbb{E}_{x\sim\mathcal{D}_x}[-\log \mathcal{D}_x(x)]$, which gives the entropy of $\mathcal{D}_x$. Thus, when $\mathcal{D}_x$ is constant with respect to the trainable parameters, the training objective in [94] is equivalent to minimizing the path-based $\mathrm{KL}(q(\tau)\,||\,p(\tau))$.

7.2.5 Learning Generative Stochastic Processes with LSTMs

The authors of [36] introduced a model capable of learning sequentially-deep generative processes. We interpret their model as a primary policy p which generates trajectories $\tau \triangleq \{z_0, ..., z_T, x\}$ according to the distribution:

$$p(\tau) \triangleq p(x|s_\theta(\tau_{<x}))p_0(z_0)\prod_{t=1}^{T} p_t(z_t), \quad \text{with } \tau_{<x} \triangleq \{z_0, ..., z_T\}, \tag{7.23}$$

in which $\tau_{<x}$ indicates a *latent trajectory* and $s_\theta(\tau_{<x})$ indicates a *state trajectory* $\{s_0, ..., s_T\}$ computed recursively from $\tau_{<x}$ using the update $s_t \leftarrow f_\theta(s_{t-1}, z_t)$ for $t \geq 1$, with s_0 given by a constant. Each state $s_t \triangleq [h_t; v_t]$ represents the joint hidden/visible state h_t/v_t of an LSTM and $f_\theta(\text{state}, \text{input})$ computes a standard

LSTM update.[3] The authors of [36] used $\mathcal{N}(0, \mathbf{I})$ for all $p_t(z_t)$. The output distribution $p(x|s_\theta(\tau_{<x}))$ was defined as $p(x|c_T)$ with $c_T \triangleq c_0 + \sum_{t=1}^{T} \omega_\theta(v_t)$, where c_0 indicates a constant and $\omega_\theta(v_t)$ indicates, e.g., an affine transform of the LSTM visible state v_t. $\omega_\theta(v_t)$ encapsulates the "write" operation of the decoder network in [36]. p is governed by parameters θ which affect f_θ, ω_θ, s_0, and c_0.

To train p, [36] introduced a guide policy q with trajectory distribution:

$$q(\tau|x^*) \triangleq q(x|s_\phi(\tau_{<x}), x^*) q_0(z_0|x^*) \prod_{t=1}^{T} q_t(z_t|\tilde{s}_t, x^*), \quad \text{with } \tau_{<x} \triangleq \{z_0, ..., z_T\}, \quad (7.24)$$

in which $s_\phi(\tau_{<x})$ indicates a state trajectory $\{\tilde{s}_0, ..., \tilde{s}_T\}$ computed recursively from $\tau_{<x}$ using the guide policy's state update $\tilde{s}_t \leftarrow f_\phi(\tilde{s}_{t-1}, g_\phi(s_\theta(\tau_{<t}), x^*))$. In this update $\tilde{s}_{t-1}$ is the previous guide state and $g_\phi(s_\theta(\tau_{<t}), x^*)$ is a deterministic function of the target sample x^* and the partial primary state trajectory $s_\theta(\tau_{<t}) \triangleq \{s_0, ..., s_{t-1}\}$, which can be computed from $\tau_{<t} \triangleq \{z_0, ..., z_{t-1}\}$ using the primary policy's state update $s_t \leftarrow f_\theta(s_{t-1}, z_t)$. The output distribution $q(x|s_\phi(\tau_{<x}), x^*)$ is defined as a Dirac-delta at x^*.[4] Each step distribution $q_t(z_t|\tilde{s}_t, x^*)$ is a diagonal Gaussian distribution with means and log-variances given by an affine function $L_\phi(\tilde{v}_t)$ of the guide LSTM visible state $\tilde{v}_t$, and $q_0(z_0)$ is defined as identical to $p_0(z_0)$. q is governed by parameters ϕ which affect the

[3] For those unfamiliar with LSTMs, a good introduction can be found in [35]. We use LSTMs including input gates, forget gates, output gates, and peephole connections for all tests presented in this chapter.

[4] It may be useful to relax this assumption.

state updates $f_\phi(\tilde{s}_{t-1}, g_\phi(s_\theta(\tau_{<t}), x^*))$ and the step distributions $q_t(z_t|\tilde{s}_t, x^*)$. $g_\phi(s_\theta(\tau_{<t}), x^*)$ encapsulates the "read" operation of the encoder network in [36].

Using our definitions for p/q, the training objective in [36] is given by:

$$\underset{p,q}{\text{minimize}} \underset{x^*\sim\mathcal{D}_x}{\mathbb{E}} \underset{\tau\sim q(\tau|x^*)}{\mathbb{E}} \left[\log \frac{q_0(z_0|x^*)}{p_0(z_0)} + \sum_{t=1}^{T} \log \frac{q_t(z_t|\tilde{s}_t, x^*)}{p_t(z_t)} - \log p(x^*|s(\tau_{<x}))\right], \tag{7.25}$$

which can be written more succinctly as $\mathbb{E}_{x^*\sim\mathcal{D}_x} \mathrm{KL}(q(\tau|x^*) \,||\, p(\tau))$. This objective upper-bounds the log-likelihood $\mathbb{E}_{x^*\sim\mathcal{D}_x}[-\log p(x^*)]$, where we define:

$$p(x) \triangleq \sum_{\tau_{<x}} p(x|s_\theta(\tau_{<x}))p(\tau_{<x}) \tag{7.26}$$

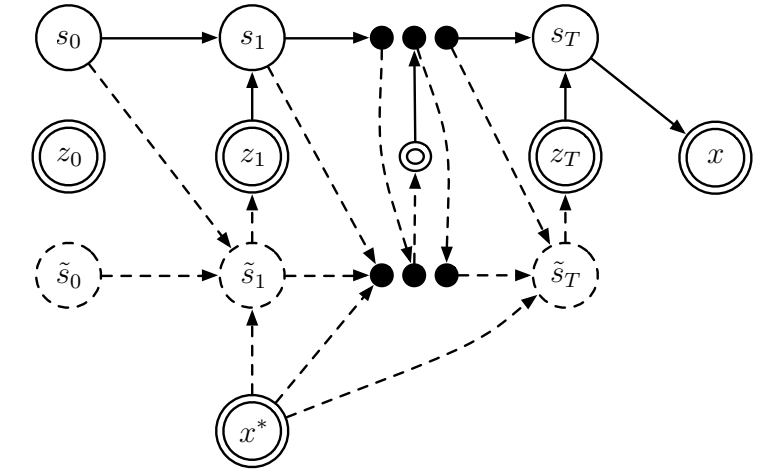
using $p(x|s_\theta(\tau_{<x}))$ and $p(\tau_{<x})$ from Eqn. 7.23. Fig. 7.2.5 illustrates this model.

7.2.6 Extending the LSTM-based Model

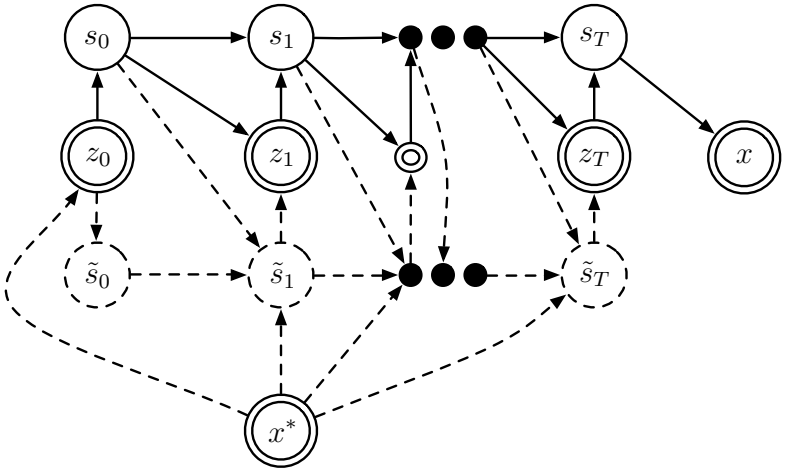
We propose changing the primary policy from the previous model to:

$$p(\tau) \triangleq p(x|s_\theta(\tau_{<x}))p_0(z_0) \prod_{t=1}^{T} p_t(z_t|s_{t-1}), \tag{7.27}$$

in which we compute $s_{t-1} \in s_\theta(\tau_{<x})$ recursively as described for Eqn. 7.23. We define $p_t(z_t|s_{t-1})$ as a diagonal Gaussian distribution with means and log-variances given by an affine function $L_\theta(v_{t-1})$ of LSTM visible state v_{t-1}, and we define $p_0(z_0)$ as $\mathcal{N}(0, \mathbf{I})$. We set s_0 using $s_0 \leftarrow f_\theta(z_0)$, where θ indicates parameters governing p. Intuitively, our changes make the "decisions" of the model more explicit by conditioning z_t on s_{t-1}, and upgrade it to an infinite mixture by adding a distribution over its initial state s_0. We represent $f_\theta(z_0)$ using a feedforward network with one hidden layer of tanh units. We also consider an alternate



(a)



(b)

Figure 7–1: (a) shows the structure of the LSTM-based model from [36], as described in Sec. 7.2.5. (b) shows the structure of our extended LSTM-based model, as described in Sec. 7.2.6. Single-edged nodes are deterministic and double-edged nodes are stochastic. Dashed nodes and edges are present only during training. Critical differences are the added edges from z_0 to $s_0/\tilde{s}_0$ in (b), and the added edges from s_t to z_{t+1} in (b). Conditioning $s_0/\tilde{s}_0$ on z_0 allows a cleaner separation of "operational regimes" for generating, e.g. 2s vs. 5s. Conditioning z_t on the state of the generative process has only a minor effect in this setting, but becomes crucial for the imputation task described in Sec. 7.3.

form for $p(x|s_\theta(\tau_{<x}))$, given by $p(x|c_T)$, where $c_T \triangleq L_\theta(h_T)$ indicates an affine transformation of the hidden part of the final LSTM state $s_T \in s_\theta(\tau_{<x})$. This uses h_t as a working memory for constructing an observation via iterative refinement.

We train our model by optimizing the objective:

$$\underset{p,q}{\text{minimize}} \underset{x^* \sim \mathcal{D}_x}{\mathbb{E}} \underset{\tau \sim q(\tau|x^*)}{\mathbb{E}} \left[\log \frac{q_0(z_0|x^*)}{p_0(z_0)} + \sum_{t=1}^{T} \log \frac{q_t(z_t|\tilde{s}_t, x^*)}{p_t(z_t|s_{t-1})} - \log p(x^*|s(\tau_{<x})) \right], \tag{7.28}$$

where we now have to care about $p_t(z_t|s_{t-1})$, $p_0(z_0)$, and $q_0(z_0|x^*)$, which could be treated as constants in the model from [36]. We define $q_0(z_0|x^*)$ as a diagonal Gaussian distribution whose means and log-variances are given by a function $g_\phi(x^*)$, which we represent using a feedforward network with one hidden layer of tanh units. Fig. 7.2.5 illustrates this model.

When trained for the binarized MNIST benchmark examined in [36], our extended model scores a negative log-likelihood of 85.3 on the test set.[5] For comparison, the score reported in [36] was 87.4. After fine-tuning the variational distribution (i.e. q) on the test set, our model's score improved to 84.6, which is quite strong considering it is an upper bound. For comparison, see the best upper bound reported for this benchmark in [79], which was 85.1. When the model is modified to use the alternate output distribution $p(x|L_\theta(h_T))$, the raw/fine-tuned

[5] Using train/validate/test splits from: `http://www.cs.toronto.edu/~larocheh/public/datasets/binarized_mnist`

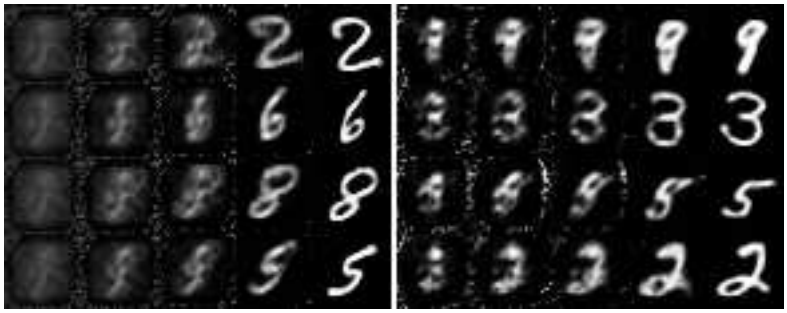


Figure 7–2: The left block shows $\sigma(c_t)$ for $t \in \{1, 3, 5, 9, 16\}$, for a policy p with $c_t \triangleq c_0 + \sum_{t'=1}^{t} L_\theta(v_{t'})$. The right block is analogous, for a model using $c_t \triangleq L_\theta(h_t)$, as described in Sec. 7.2.6.

test scores were 85.7/85.1. Fig. 7–2 shows samples from our model. Model/test code is available at `http://github.com/Philip-Bachman/Sequential-Generation`.

7.3 Developing Models for Sequential Imputation

Imputation concerns the density $p(x^u|x^k)$, where $x \triangleq [x^u; x^k]$ indicates a *complete observation* with *known values* x^k and *missing values* x^u. By expanding x^u to cover all of x, one recovers standard generative modelling. By shrinking x^u to cover a single element of x, one recovers standard classification/regression. We define a *mask* $m \in \mathcal{M}$ as a partition of x into x^u/x^k.[6] Given distribution $\mathcal{D}_m$ over $m \in \mathcal{M}$ and distribution $\mathcal{D}_x$ over $x \in \mathcal{X}$, the objective for imputation is:

$$\underset{p}{\text{minimize}} \underset{x\sim\mathcal{D}_x}{\mathbb{E}} \underset{m\sim\mathcal{D}_m}{\mathbb{E}} \left[-\log p(x^u|x^k) \right]. \tag{7.29}$$

[6] In our setting, there will always be only one mask m in the scope of a statement, and the u/k superscripts refer to partitioning x using that m.

We describe an MDP with finite-horizon T for which guided policy search minimizes a bound on Eqn. 7.29. The MDP is defined by mask distribution $\mathcal{D}_m$, complete observation distribution $\mathcal{D}_x$, and the state spaces $\{\mathcal{Z}_0, ..., \mathcal{Z}_T\}$ associated with each step. Together, $\mathcal{D}_m$ and $\mathcal{D}_x$ define a joint distribution over initial states and rewards in the MDP. For the trial determined by $x \sim \mathcal{D}_x$ and $m \sim \mathcal{D}_m$, the initial state $z_0 \sim p(z_0|x^k)$ is selected by the policy p based on the known values x^k. The cost $\ell(\tau, x^u, x^k)$ suffered by trajectory $\tau \triangleq \{z_0, ..., z_T\}$ in the context (x, m) is defined as $-\log p(x^u|\tau, x^k)$, i.e. the negative log-likelihood of p guessing the missing values x^u after following trajectory τ, while knowing the values x^k.

We consider a primary policy p with a trajectory distribution:

$$p(\tau|x^k) \triangleq p(z_0|x^k) \prod_{t=1}^{T} p(z_t|z_0, ..., z_{t-1}, x^k). \tag{7.30}$$

Here, x^k is determined by the x and m sampled for the current trial, and p cannot observe the missing values x^u. With these definitions, an optimal policy for the imputation MDP minimizes the expected cost:

$$\underset{p}{\text{minimize}} \underset{x\sim\mathcal{D}_x}{\mathbb{E}} \underset{m\sim\mathcal{D}_m}{\mathbb{E}} \underset{\tau\sim p(\tau|x^k)}{\mathbb{E}} \left[-\log p(x^u|\tau, x^k)\right]. \tag{7.31}$$

I.e. the expected negative log-likelihood of making a correct imputation on any given trial. This is a valid, but loose, (Jensen's) upper bound on the imputation objective in Eq. 7.29. We can tighten the bound by introducing a guide policy.

As in Sec. 7.2, we train p using a guide policy with access to additional information, e.g. x^u. The guide policy q has a trajectory distribution:

$$q(\tau|x^u, x^k) \triangleq q(z_0|x^u, x^k) \prod_{t=1}^{T} q(z_t|z_0, ..., z_{t-1}, x^u, x^k). \tag{7.32}$$

Given these models for p and q, guided policy search optimizes:

$$\underset{p,q}{\text{minimize}} \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x} \mathop{\mathbb{E}}_{m\sim\mathcal{D}_m} \left[\mathop{\mathbb{E}}_{\tau\sim q(\tau|i_q,i_p)} [-\log q(x^u|\tau, i_q, i_p)] + \mathrm{KL}(q(\tau|i_q, i_p)||p(\tau|i_p)) \right], \tag{7.33}$$

where we define $i_q \triangleq x^u$, $i_p \triangleq x^k$, and $q(x^u|\tau, i_q, i_p) \triangleq p(x^u|\tau, i_p)$.

7.3.1 A Direct Representation for Sequential Imputation Policies

We define the imputation trajectory $c_\tau \triangleq \{c_0, ..., c_T, c_{T+1}\}$, where each partial imputation $c_t \in \mathcal{X}$ is computed from a partial trajectory $\tau_{<t} \triangleq \{z_0, ..., z_{t-1}\}$. The partial imputation c_t encodes a policy's guess for the complete observation $x \triangleq [x^u; x^k]$ immediately prior to selecting step z_t, and c_{T+1} gives the final guess produced by the policy. We consider two approaches to computing c_t, mirroring those described in Sec. 7.2 for the LSTM-based generative model. In the first approach, for $t > 0$, $c_t \triangleq c_0 + \sum_{t'=0}^{t-1} \omega_\theta(z_{t'})$, where $\omega_\theta(z_{t'})$ is a trainable function. For $t = 0$, we define $c_0 \triangleq [c_0^u; c_0^k]$ as a trainable bias. In the second approach, $c_t \triangleq \omega_\theta(z_{t-1})$. We indicate models using the first approach with the suffix $-add$, and models using the second approach with $-jump$. We use a feedforward network with two ReLU layers to represent ω_θ.

For imputation, our primary policy p_θ computes $p_\theta(z_t|z_0, ..., z_{t-1}, x^k)$ at each step $0 \leq t \leq T$ using $p_\theta(z_t|c_t, x^k)$, which provides the means and log-variances for a diagonal Gaussian distribution over $\mathcal{Z}$. We use a feedforward network with

two ReLU layers to represent $p_\theta(z_t|c_t, x^k)$. The step model $p_\theta(z_t|c_t, x^k)$ and the imputation constructor $\omega_\theta(z_t)$ determine the behaviour of the primary policy.

We construct a guide policy q similarly to p. The guide policy shares the imputation constructor $w_\theta(z_t)$ with the primary policy. The guide policy incorporates the additional information $\hat{c}_t \triangleq [x^u; x^k] - [\sigma(c_t^u); \sigma(c_t^k)]$, i.e. the gradient of the full reconstruction log-likelihood with respect to the partial imputation produced by steps prior to step t. The guide policy chooses steps using $q_\phi(z_t|c_t, \hat{c}_t)$, which gives the means and log-variances of a diagonal Gaussian over $\mathcal{Z}$. We represent $q_\phi(z_t|c_t, \hat{c}_t)$ using a feedforward network with two ReLU layers.

We train the networks ω_θ, p_θ, and q_ϕ simultaneously on the objective:

$$\underset{\theta,\phi}{\text{minimize}} \underset{x\sim\mathcal{D}_x}{\mathbb{E}} \underset{m\sim\mathcal{D}_m}{\mathbb{E}} \left[\underset{\tau\sim q_\phi(\tau|x^u,x^k)}{\mathbb{E}} \left[-\log q(x^u|c_{T+1}^u) \right] + \text{KL}(q(\tau|x^u, x^k) \,||\, p(\tau|x^k)) \right], \tag{7.34}$$

where $q(x^u|c_{T+1}^u) \triangleq p(x^u|c_{T+1}^u)$. We train our models using Monte-Carlo roll-outs of q, and stochastic backpropagation as in [51, 80]. Fig. 7.3.2 illustrates this model. Full implementations and test code are available from `http://github.com/Philip-Bachman/Sequential-Generation`.

7.3.2 Representing Sequential Imputation Policies using LSTMs

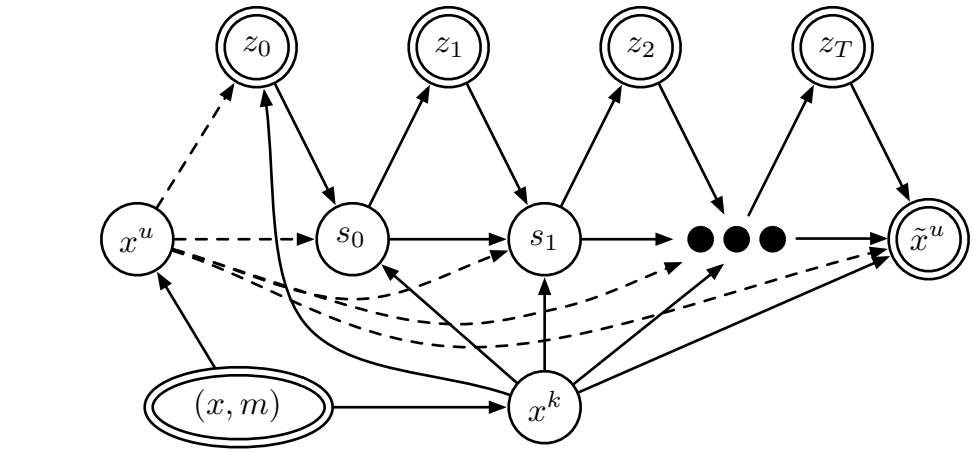
To make it useful for imputation, which requires conditioning on the exogenous information x^k, we modify the LSTM-based model from Sec. 7.2.6 to include a "read" operation in its primary policy p. We incorporate a read operation by spreading p over two LSTMs, p^r and p^w, which respectively "read" and "write" an imputation trajectory $c_\tau \triangleq \{c_0, ..., c_T, c_{T+1}\}$. Conveniently, the guide policy q for this model has the same form as the reader p^r.

Procedurally, a single full step of execution for p involves the following substeps: first p updates the reader state using $s_t^r \leftarrow f_\theta^r(s_{t-1}^r, \omega_\theta^r(c_t, s_{t-1}^w, x^k))$, then p samples a step $z_t \sim p_\theta(z_t|v_t^r)$, then p updates the writer state using $s_t^w \leftarrow f_\theta^w(s_{t-1}^w, z_t)$, and finally p updates the imputation state using $c_{t+1} \leftarrow c_t + \omega_\theta^w(v_t^w)$ (or $c_{t+1} \leftarrow \omega_\theta^w(h_t^w)$). In these updates, $s_t^{r,w} \triangleq [h_t^{r,w}; v_t^{r,w}]$ refers to the states of the ($r$eader,$w$riter) LSTMs. The LSTM updates $f_\theta^{r,w}$ are governed by disjoint subsets of the policy parameters θ. For our tests the read/write operations $\omega_\theta^{r,w}$ were both affine functions governed by disjoint subsets of θ.

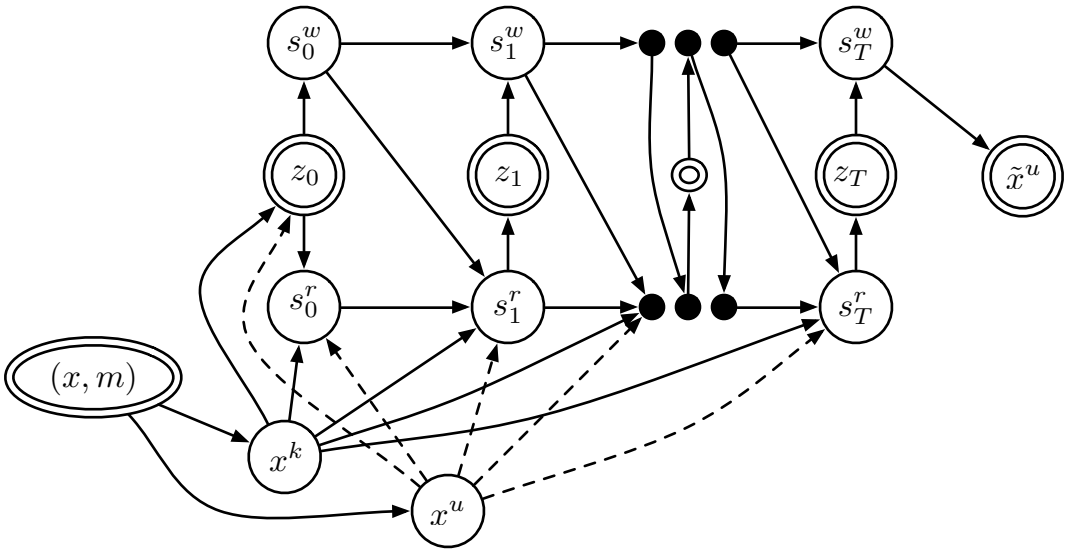
To train p, we generate sample trajectories using a guide policy q in place of p^r. This changes the first two steps in the preceding paragraph to: update the guide state $s_t^q \leftarrow f_\phi(s_{t-1}^q, \omega_\phi^q(c_t, s_{t-1}^w, x))$ and then sample $z_t \sim q_\phi(z_t|v_t^q)$. The subsequent updates for s_t^w and c_{t+1} remain the same. The guide policy's "read" function ω_ϕ^q gets to see the *complete observation* x. We use this to compute the full reconstruction log-likelihood gradient $\hat{c}_t$, as described in Sec. 7.3.1. The step distributions p_θ/q_ϕ are diagonal Gaussians whose means and log-variances are affine functions of v_t^r/v_t^q. The resulting training objective has the same form as Eq. 7.34. Fig. 7.3.2 illustrates this model.

7.4 Experiments

We tested the performance of our sequential imputation models on three datasets: MNIST (28x28), SVHN (cropped, 32x32) [73], and TFD (48x48) [101]. We converted all images to grayscale and shift/scaled them to be in the range [0...1] prior to training/testing. We measured the imputation log-likelihood $\log q(x^u|c_{T+1}^u)$ using an "independent Bernoulli" interpretation of the missing

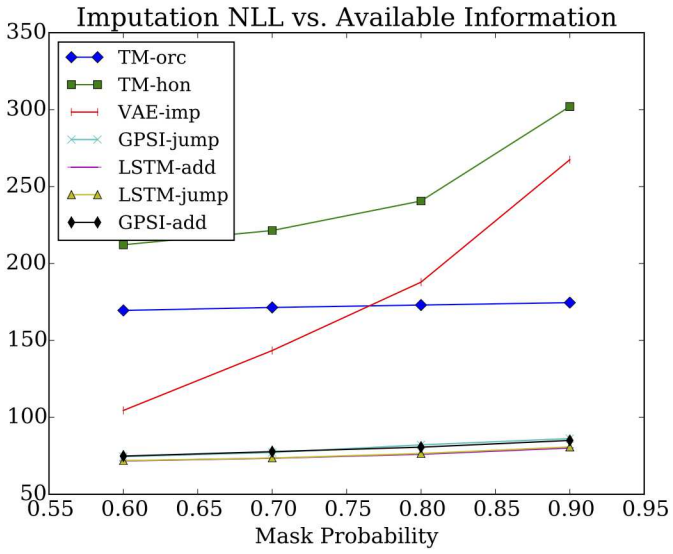


(a)

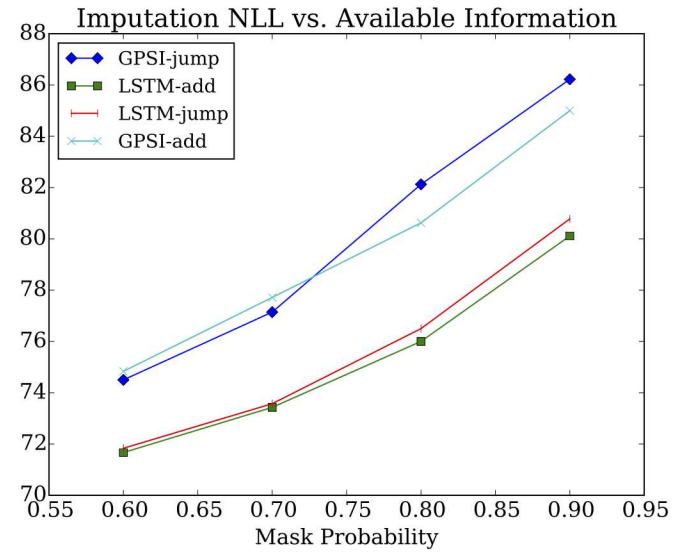


(b)

Figure 7–3: (a) shows the basic structure of the "direct" imputation model in Sec. 7.3.1. (b) shows the basic structure of the LSTM-based imputation model in Sec. 7.3.2. Single-edged nodes are deterministic and double-edged nodes are stochastic. All solid lines affect the primary and guide policies. All dashed lines affect only the guide policy. The primary policy generates imputations for test trials, and the guide policy generates imputations for training trials.



(a)



(b)

Figure 7–4: This plot compares the performance of our sequential imputation models relative to several baselines, using MNIST digits. The x-axis indicates the % of pixels which were dropped at random, and the scores are normalized by the number of imputed pixels. Reducing the available information reduced imputation accuracy.

values x^u and the imputations given by $\sigma(c^u_{T+1})$. We refer to our "direct" model variants as GPSI-add and GPSI-jump, and to our "LSTM-based" model variants as LSTM-add and LSTM-jump. GPSI models performed 6 steps of imputation, and LSTM models performed 16.[7]

We tested imputation under two types of data masking: missing completely at random (MCAR) and missing at random (MAR). In MCAR, we masked pixels uniformly at random from the source images, and indicate removal of d% of the pixels by MCAR-d. In MAR, we masked square regions, with the occlusions located uniformly at random within the borders of the source image. We indicate occlusion of a $d \times d$ square by MAR-d.

On MNIST, we tested MCAR-d for $d \in \{50, 60, 70, 80, 90\}$, where MCAR-100 corresponds to unconditional generation. On TFD and SVHN we only tested MCAR-80. On MNIST, we tested MAR-d for $d \in \{14, 16\}$. On TFD we tested MAR-25 and on SVHN we tested MAR-17. We tested our four models against three baselines. For test trials we sampled masks from the same distribution used in training, and we sampled complete observations from a held-out test set. Fig. 7–4 and Tab. 7–1 present results from these tests.

The baselines were "variational auto-encoder imputation", honest template matching, and oracular template matching. VAE imputation ran multiple steps of

[7] GPSI stands for "Guided Policy Search Imputer". The tag "-add" refers to the purely additive imputation constructor, and "-jump" refers to the constructor that can fully replace the imputation at each step.

	MNIST		TFD		SVHN	
	MAR-14	MAR-16	MCAR-80	MAR-25	MCAR-80	MAR-17
LSTM-add	170	167	1381	1377	525	568
LSTM-jump	172	169	–	–	–	–
GPSI-add	177	175	1390	1380	531	569
GPSI-jump	183	177	1394	1384	540	572
VAE-imp	374	394	1416	1399	567	624

Table 7–1: Imputation performance across several conditions. Details of the tests are provided in the main text. VAE-imp indicates the "variational auto-encoder imputation" baseline. Due to time constraints, we did not test LSTM-jump on TFD or SVHN. These scores are normalized for the number of imputed pixels.

VAE reconstruction on the partially-occluded input, with the known values fixed to their true values and the missing values re-estimated with each reconstruction step.[8] We did this for 16 steps and let the VAE freely choose its best imputation. Honest template matching imputed the missing values using values from the training image which most closely matched the test image's known values. Oracular template matching was like honest template matching, but performed matching directly on the missing values.

Our models significantly outperformed the baselines. In general, the LSTM-based models outperformed the more direct GPSI models. We evaluated the log-likelihood of imputations produced by our models using the lower bounds naturally provided by the variational objective with respect to which they were trained. Evaluating the template-based imputations is straightforward. For VAE imputation, we computed the expected log-likelihood of the imputations sampled

[8] We discuss some deficiencies of VAE imputation in Sec. 7.4.1.

(a) (b) (c)

Figure 7–5: This figure illustrates the policies learned by our models. (a): models trained for (MNIST, MAR-16). From top→bottom the models are: GPSI-add, GPSI-jump, LSTM-add, LSTM-jump. (b): models trained for (TFD, MAR-25), with models in the same order as (a) – but without LSTM-jump. (c): models trained for (SVHN, MAR-17), with models arranged as for (b).

from multiple runs of the 16-step imputation process. This provides a valid lower bound on their log-likelihood.

As illustrated in Fig. 7.4, the imputations produced by our models appear quite promising. The imputations are generally of high quality, and the models are capable of capturing strongly multi-modal reconstruction distributions (see subfigure (a)). The behavior of GPSI models changes intriguingly when we swap the imputation constructor. Using the "-jump" imputation constructor, the imputation policy learned by the direct model is rather inscrutable. We provide further qualitative results in Sec. 7.4.2, and animations of the various policies are provided in the supplemental material. When trained on the binarized MNIST benchmark discussed in Sec. 7.2.6, i.e. with binarized images and subject to

MCAR-100, the LSTM-add model produced raw/fine-tuned scores of 86.2/85.7. The LSTM-jump model scored 87.1/86.3. Anecdotally, these "closed-loop" models seemed more prone to overfitting than the "open-loop" models in Sec. 7.2.6.

7.4.1 Problems with VAE Imputation

Variational auto-encoder imputation proceeds by running multiple steps of iterative sampling from the approximate posterior $q(z|x)$ and then from the reconstruction distribution $p(x|z)$, with the known values replaced by their true values at each step. I.e. the missing values are repeatedly guessed based on the previous guessed values, combined with the true known values.

Consider an extreme case in which the mutual information between z and x in the joint distribution $p(x, z) = p(x|z)p(z)$ is 0. In this case, even if the marginal over x, i.e. $p(x) = \sum_z p(x|z)p(z)$, matches the target distribution $\mathcal{D}_x$, each sample of new guesses for the missing values will be independent samples from the marginal over those values in $\mathcal{D}_x$. Thus, the new guesses will be informed by neither the previous guesses nor the known values in the observation for which imputation is being performed.

The VAE approach to imputation also suffers due to the posterior inference model $q(z|x)$ lacking any prior experience with heavily perturbed observations. I.e., if all training is performed on unperturbed observations, then $q(z|x)$ may respond erratically when presented with perturbed observations.

While one could train a VAE for imputation by sampling imputation trajectories and then backpropagating the imputation log-likelihood through those trajectories, this approach performed poorly in preliminary experiments. In a

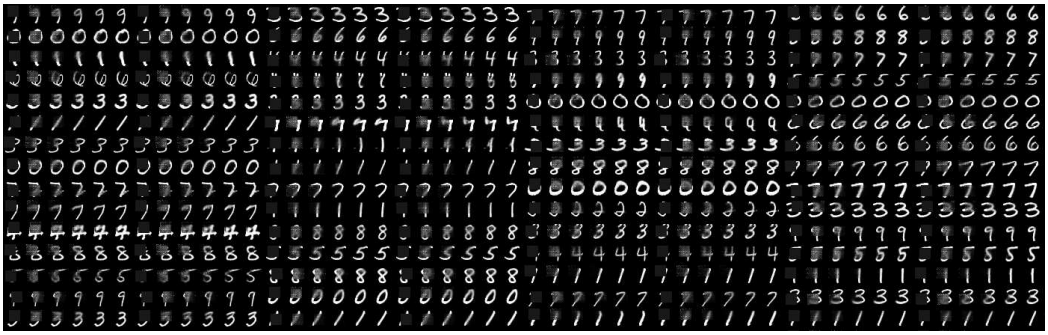
strong sense, the problem with this approach is analogous to that solved (in certain situations) by Guided Policy Search [64, 65]. I.e., the primary policy is initially so poor that training it with, e.g., policy gradient will be ineffective. By incorporating privileged information in the guide policy, one can slowly shepherd the initially poor primary policy toward gradually improving behavior.

7.4.2 Additional Qualitative Results for GPSI Models

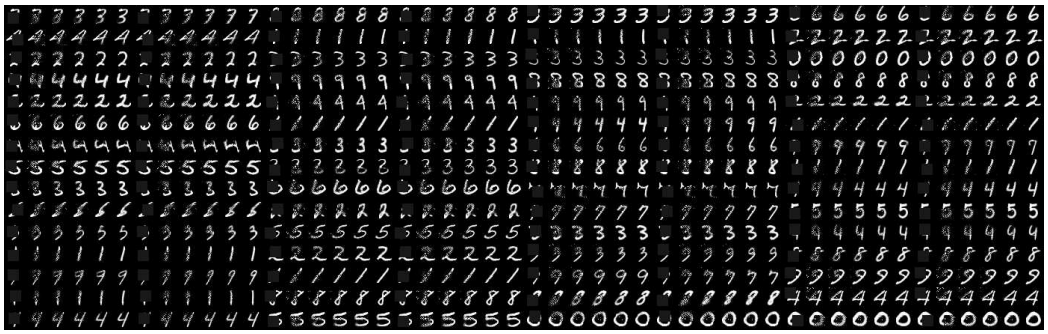
7.5 Discussion

We presented a point of view which links methods for training directed generative models with policy search in reinforcement learning. We showed how our perspective can guide improvements to existing models. The importance of these connections will only grow as generative models rapidly increase in structural complexity and effective decision depth (i.e. the maximum path length in their topologically-sorted graphs).

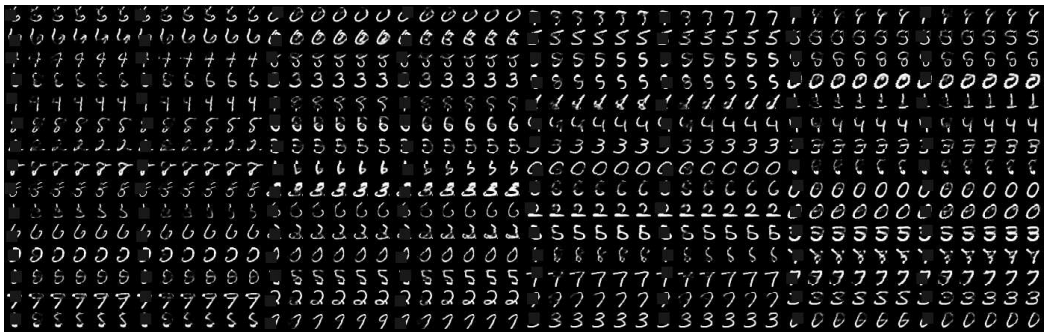
We introduced the notion of imputation as a natural generalization of standard, unconditional generative modelling. Depending on the relation between the data-to-generate and the available information, imputation spans from full unconditional generative modelling to classification/regression. We showed how to successfully train sequential imputation policies comprising millions of parameters using an approach based on Guided Policy Search [64]. Our approach outperforms the baselines quantitatively and appears qualitatively promising. Incorporating, e.g., the local read/write mechanisms from [36] should provide further improvements.



(a)

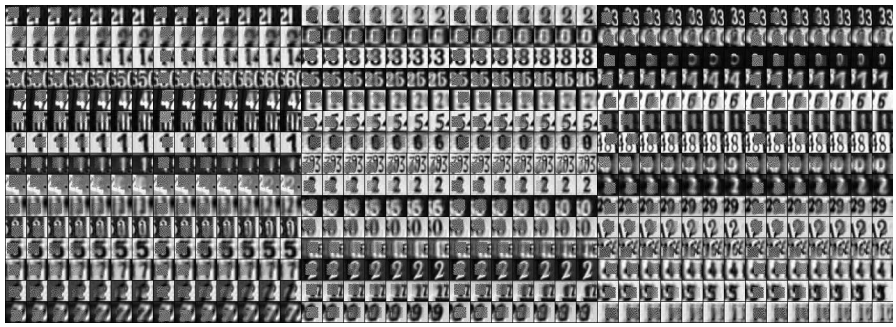


(b)

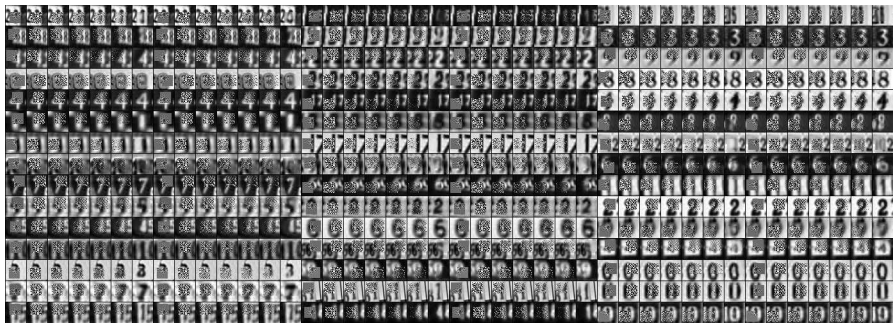


(c)

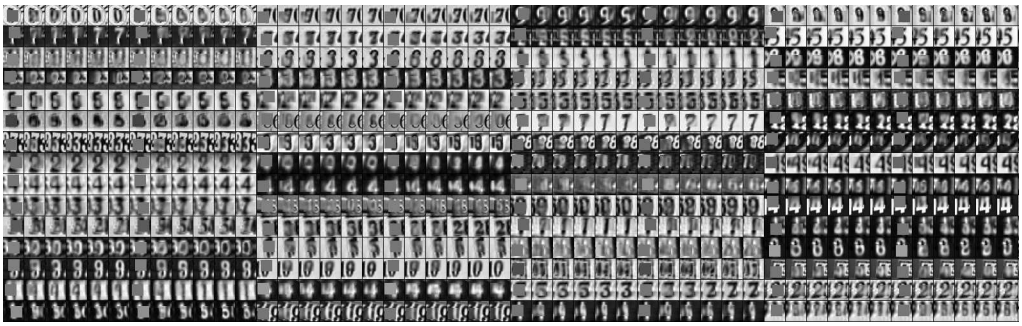
Figure 7–6: This figure illustrates roll-outs of (a) additive (b) jump, and (c) variational auto-encoder policies trained on MNIST as described in the main text. The ways in which the additive and jump policies proceed towards their final imputations are visually distinct. We ran two independent roll-outs of each policy type for each initial state, to exhibit the ability of our models to produce multimodal imputation densities. All initial states were generated by randomly occluding a 16x16 block of pixels in images taken from the validation set. I.e. these initial conditions were never experienced during training. Zoom in for best viewing.



(a)

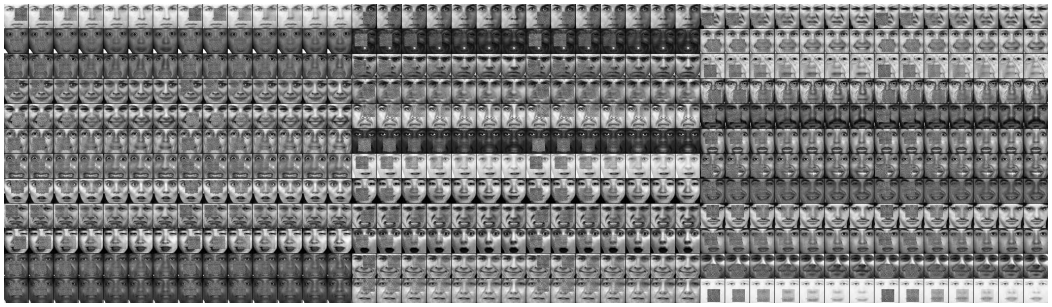


(b)



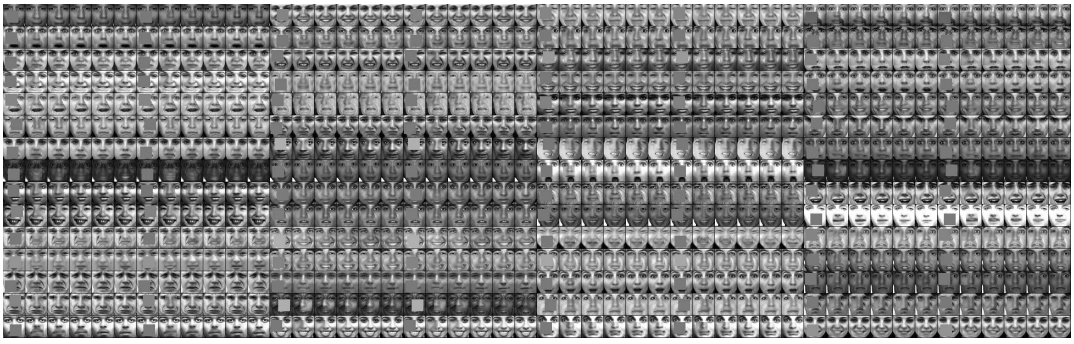
(c)

Figure 7–7: This figure illustrates roll-outs of (a) additive (b) jump, and (c) variational auto-encoder policies trained on (grayscale) SVHN as described in the main text. The ways in which the additive and jump policies proceed towards their final imputations are visually distinct. We ran two independent roll-outs of each policy type for each initial state, to exhibit the multimodality of our models' imputation policies. All initial states were generated by randomly occluding a 17x17 block of pixels in images taken from the validation set. I.e. these initial conditions were never experienced during training. Zoom in for best viewing.



(a)

(b)



(c)

Figure 7–8: This figure illustrates roll-outs of (a) additive (b) jump, and (c) variational auto-encoder policies trained on TFD as described in the main text. The ways in which the additive and jump policies proceed towards their final imputations are visually distinct. In particular, the "strategy" pursued by the jump policy is not intuitively clear. We ran two independent roll-outs of each policy type for each initial state, to exhibit the multimodality of our models' imputation policies. All initial states were generated by randomly occluding a 25x25 block of pixels in images taken from the validation set. I.e. these initial conditions were never experienced during training. Zoom in for best viewing.

CHAPTER 8
General Discussion

This thesis makes a number of contributions, focusing primarily on how to shape the way that a computational system interacts with data.

We first showed how to construct a powerful model by spreading many instances of a simpler base model around the input domain, and adapting each instance to just the data observed in its vicinity. This approach simultaneously enjoys the easy estimation of linear regression, and the representational capacity of a complex non-linear function approximator. However, the ensuing reduction in model bias is quickly overwhelmed by an increase in estimation variance as one shrinks the vicinity from which data is drawn while training each instance of the base model. We used techniques developed for sparse coding and structure estimation in sparse graphical models to reduce variance when estimating each instance of the base model. We gained this improvement by encouraging information transfer among the multiple base model instances – by restricting their estimations to a shared, simpler space. With synthetic data, our approach recovered a good description of the underlying system. Features extracted by our model were also useful in a real-word setting.

Though we developed and presented our approach in the context of structure estimation in time varying graphical models, it should be useful for other problems

involving multi-task and/or locally-adapted high-dimensional regressions. The underlying idea, i.e. granting flexibility by using many locally-adapted models while restricting complexity by forcing overlap in the model parametrizations, is directly relevant to the deep, directed models from Chapters 6 and 7. These models can all be interpreted as infinite mixtures, or hierarchical infinite mixtures, in which the parametrizations of the mixtures are entangled through the shared transformation from latent variable configurations to distributions over observations.

After working on regularization via parameter sharing, we developed an approach for controlling the derivatives of a common class of function approximators, i.e. those which compute a linear function of some features extracted from each input. The algorithm for constructing our derivative-controlling regularizers is straightforward – easy to understand and easy to code. We showed that our regularizers are theoretically sound, and offer some benefits over competing approaches. Using matching sets of features our approach improves on ℓ_2 regularization, which is often the default choice for use with heterogenous collections of non-kernel features. While the regularization permitted by kernel-based features[1] is often

[1] Referring to features for kernel methods can be a bit confusing, as there are three distinct feature spaces that play a significant role. The first, call it $\mathcal{X}$, is the natural space used to represent each input $x \in \mathcal{X}$. The second, call it Φ, is the space associated with the kernel $k(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$. An occasionally neglected, but still important, feature space is $\mathcal{X}_k$, in which each input $x_i \in \mathcal{X}$ is represented as a vector $k(x_i; \tilde{X}) \triangleq [k(x_i, \tilde{x}_1), ..., k(x_i, \tilde{x}_n)]$, where $\tilde{X} = \{\tilde{x}_1, ..., \tilde{x}_n\}$ is a set of n "kernel anchors" in $\mathcal{X}$. The space $\mathcal{X}_k$ is where the function is explicitly trained, regularized, and evaluated.

highly-effective, we showed that its requirement of a kernel could disadvantage it relative to our approach.

Learning features, as opposed to engineering them by hand, has proven overwhelmingly successful in many application domains. Though our derivative-controlling regularizers were effective when used with fixed sets of features, it was not obvious how to efficiently construct analogous regularizers for use with trainable features. We addressed this issue by encouraging models to be robust with respect to general uncertainty in their inputs and structure. We accomplished this by measuring differences in the behavior of pairs of models generated by perturbing a base model, while training the parameters of the base model to keep those differences small. This sort of "behavioral regularization" directly moderates the way that a model interacts with data. As a model's behavior, rather than the values of its parameters, is the key to success in most applications of machine learning, we feel that direct methods for shaping behavior should be used more often. We showed that our approach to controlling variability in a model's behavior can significantly improve performance in the semi-supervised and fully-supervised settings.

Motivated by the value of task-agnostic perturbations for regularization, we sought a model capable of generating task-aware perturbations. The goal was to generate random walks through the input distributions for (semi-)supervised learning tasks. By regularizing a classifier to change slowly with respect to sequences of steps from these walks, one could force the classifier to change slowly with respect to movement along the input manifold. We found it challenging to

build a model that could produce walks with the right properties. In particular, it was difficult to generate walks that moved quickly enough to be useful while staying close to the target distribution. We introduced a technique that explicitly encourages a model to generate fast-mixing walks. The technique works, in a sense, by minimizing the mutual information between subsequent steps of the walk. To keep walks near the target distribution, we introduced a technique for shaping them using Approximate Bayesian Computation. The technique works by watching to see if the walks wander away from the target distribution, and then penalizing them if they do. In response, the walks are reshaped to avoid these penalties. From this perspective, the walk-generating system can be interpreted as a reinforcement learning agent. This agent – a self-looped variational auto-encoder – learns a policy which balances three sources of reward: positive reward for staying in place, negative reward for specifying the next step too precisely, and negative reward for wandering away from the target distribution. The first two rewards come from our re-weighted variational free energy, and the last reward is provided by a learned guide function.

Our work with random walks pushed us towards three new objectives:

1. Developing generative models that construct complicated distributions by sequentially composing simpler distributions.
2. Developing techniques for incorporating feed-back loops and exogenous conditioning information into sequential generative models.[2]

[2] One could think of feed-back loops as endogenous conditioning information.

3. Developing formal and informal connections between reinforcement learning and current approaches to working with deep generative models.

The first objective requires constructing generative models[3] with decision depth greater than 1, where decision depth describes the number of sampling steps prior to producing an output. Increasing the decision depth of a model allows each decision to remain simple, while retaining the ability to form complex output distributions. This notion underlies, e.g., denoising auto-encoders [10], Generative Stochastic Networks [9], and more recent work [94, 22]. These methods, which are all based on stochastically inverting compressive processes, are only a beginning.

Incorporating feed-back loops and exogenous conditioning information into a model greatly expands the range of problems to which it can be applied. To support this point, we developed a model that performed well as a classical generative model and as an imputation model. The model architecture and training algorithm required no changes between these tasks. We only had to provide more/less exogenous conditioning information at training and test time. Our model worked by iteratively refining an output prediction, based on the available exogenous information and on feed-back from its own decisions. By constructing a prediction over multiple stages of stochastic decisions, our model easily captured structure in the true output distribution.

[3] We apply this term more broadly than usual. By "generative model", we mean any model whose output is a distribution. This includes classical generative models, classifiers, models for structured learning, models for machine translation, etc.

Though we presented a fairly basic application of feed-back loops and exogenous conditioning, these capabilitie grants our general approach real power and flexibility. For example, consider the task of per-pixel image labelling – a well-studied structured prediction problem. The structure emerges from strong local and non-local correlations created by the presence of objects in natural images. Many methods have been developed that attempt to capture this structure by post-processing "unstructured" per-pixel predictions using various grid-like graphical models over the pixel labels. It may be easier to capture structure in this problem using a variational recurrent convnet/deconvnet loop that iteratively reads the image and the current pixel labels, and then proposes and applies an update to the labels. This approach would allow the system to gradually bring its proposed labels into concordance with both each other and the image. Structure in the labels could be coordinated locally and non-locally over multiple steps of refinement. A similar process of iterative hypothesis rendering and refinement could be applied to a variety of vision tasks. The basic methods that we used for imputation could be extended to these problems with only minor changes.

Our final contribution was the initial development of connections between reinforcement learning and the computational processes involved in deep, directed generative models. In particular, we presented a point-of-view from which a diverse set of model architectures and training algorithms for deep generative models can all be seen as a generalized form of guided policy search [65]. Though the methods we used in this thesis can all be described in the framework of stochastic variational inference, the connections we made between reinforcement learning

and sequential generative models suggest some interesting research directions. We suspect that training sequential generative models will present problems beyond those faced when training deep architectures with shallow decision depth. For example, the current parameters will determine the model's training set, due to the use of a learned inference model for generating training samples. Problems inherent in this sort of feed-back loop are well-known in reinforcement learning, and the subject of intense study. As another example, one of the central problems in reinforcement learning is the exploration vs exploitation dilemma, which has not been seriously considered in the context of sequential generative models. As a final example, we note that existing techniques for training sequential generative models are all some form of policy search. It may be worth investigating alternative techniques based on actor-critic and value-based methods.

Appendix

8.1 A unified log-likelihood bound for generative models

We now derive an upper bound on the expected log-likelihood of any variable x in any generative model p for which we can compute the joint log-density $\log p(x, \text{etc})$. The expectation can be taken w.r.t. any distribution $\mathcal{D}_x$. I.e., we derive an upper bound on:

$$\mathop{\mathbb{E}}_{x \sim \mathcal{D}_x} \left[-\log \left(\mathop{\mathbb{E}}_{\text{etc} \sim p(\text{etc})} \left[p(x|\text{etc}) \right] \right) \right] \tag{8.1}$$

The bound depends on an auxiliary distribution $q(x, \text{etc})$ and requires that we can draw independent samples of (x, etc) from q, and that we can compute $\log q(x, \text{etc})$. For the bound to be practically useful, we should also be able to compute gradients of $\log p(x, \text{etc})$ and $\log q(x, \text{etc})$ with respect to their respective parameters. These are the only requirements for p and q.

Our bound ties together the standard variational free energy bound and the path-wise KL bound from the previous subsection. Our derivation follows a somewhat different form from standard (as known to me) derivations of the variational free energy. While standard interpretations of the variational free energy readily suggest methods like the variational auto-encoders in [51, 80], they are less helpful in understanding some newer methods, such as those in [94] and [85]. In contrast, our derivation clearly brings these methods – and many others, both existing and yet-to-be-conceived – into a unified framework. The form of our

derivation also makes it easy to interpret denoising auto-encoders, GSNs, etc. as members of the same family of methods.

8.2 Deriving the unified log-likelihood bound

Assume a pair of distributions $q(x_0, ..., x_n)$ and $p(x_0, ..., x_n)$ over the arbitrary collection of variables $\{x_0, ..., x_n\}$. Now, consider the following maths:

$$\begin{aligned}
\mathrm{KL}(q(x_0,..., x_n) \,||\, p(x_0, ..., x_n)) = \; & ... && (8.2)\\
= & \mathop{\mathbb{E}}_{x_0,...,x_n \sim q(x_0,...,x_n)} \left[\log \frac{q(x_0, ..., x_n)}{p(x_0, ..., x_n)} \right] \\
= & \mathop{\mathbb{E}}_{x_0 \sim q(x_0)} \left[\mathop{\mathbb{E}}_{x_1,...,x_n \sim q(x_1,...,x_n|x_0)} \left[\log \frac{q(x_0)q(x_1, ..., x_n|x_0)}{p(x_0)p(x_1, ..., x_n|x_0)} \right] \right] \\
= & \mathop{\mathbb{E}}_{x_0 \sim q(x_0)} \left[\log \frac{q(x_0)}{p(x_0)} + \mathop{\mathbb{E}}_{x_1,...,x_n \sim q(x_1,...,x_n|x_0)} \left[\log \frac{q(x_1, ..., x_n|x_0)}{p(x_1, ..., x_n|x_0)} \right] \right] \\
= & \, \mathrm{KL}(q(x_0) \,||\, p(x_0)) + \mathop{\mathbb{E}}_{x_0 \sim q(x_0)} \left[\mathop{\mathbb{E}}_{x_1,...,x_n \sim q(x_1,...,x_n|x_0)} \left[\log \frac{q(x_1, ..., x_n|x_0)}{p(x_1, ..., x_n|x_0)} \right] \right] \\
= & \, \mathrm{KL}(q(x_0) \,||\, p(x_0)) + \mathop{\mathbb{E}}_{x_0 \sim q(x_0)} \Big[\mathrm{KL}(q(x_1, ..., x_n|x_0) \,||\, p(x_1, ..., x_n|x_0)) \Big] && (8.3)\\
\geq & \, \mathrm{KL}(q(x_0) \,||\, p(x_0)) && (8.4)
\end{aligned}$$

Note that we chose our subscripts arbitrarily, so we can also say:

$$\mathrm{KL}(q(x_0, ..., x_n) \,||\, p(x_0, ..., x_n)) \geq \arg\max_i \Big[\mathrm{KL}(q(x_i) \,||\, p(x_i)) \Big] \tag{8.5}$$

8.3 Bounding expected log-likelihood for $x \sim \mathcal{D}_x$

Now, consider relabelling the distributions q and p to look like: $q(x, z_0, ..., z_n)$ and $p(x, z_0, ..., z_n)$. If the marginal $q(x)$ matches the target distribution $\mathcal{D}_x$,

i.e. $\forall x : q(x) = \mathcal{D}_x(x)$, then Eq. 8.3 and Eq. 8.5 let us say:

$$\mathrm{KL}(q(x, z_0,..., z_n) \,||\, p(x, z_0, ..., z_n)) = \ ... \tag{8.6}$$

$$= \mathrm{KL}(\mathcal{D}_x(x) \,||\, p(x)) + \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x} \Big[\mathrm{KL}(q(z_0, ..., z_n|x) \,||\, p(z_0, ..., z_n|x)) \Big] \tag{8.7}$$

$$\geq \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x} \log \frac{\mathcal{D}_x(x)}{p(x)} \tag{8.8}$$

$$\geq \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x} \big[\log \mathcal{D}_x(x) \big] + \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x} \big[-\log p(x) \big] \tag{8.9}$$

Thus, if we can draw independent samples from $q(x, z_0, ..., z_n)$, and if the densities $q(z_0, ..., z_n|x)$ and $p(x, z_0, ..., z_n)$ are tractable (we will not attempt to optimize $q(x) \triangleq \mathcal{D}_x$), then we can directly minimize a bound on $\mathbb{E}_{x\sim\mathcal{D}_x} \big[-\log p(x) \big]$.

Monte Carlo minimization of the joint KL bound on $\mathbb{E}_{x\sim\mathcal{D}_x} \big[-\log p(x) \big]$ simply repeats the following two steps:

1. Sample $(x, z_0, ..., z_n)$ from $q(x, z_0, ..., z_n) \triangleq \mathcal{D}_x(x) q(z_0, ..., z_n|x)$.
2. Do a descent step to reduce: $\log q(z_0, ..., z_n|x) - \log p(x, z_0, ..., z_n)$.

This minimization, which encompasses all existing approaches to training deep, directed generative models, is simple and widely-applicable.

8.4 Assimilating denoising auto-encoders and GSNs

To bring denoising auto-encoders [10] and GSNs [9] into our framework, we only need to modify the p and q considered in the previous subsection by allowing that q might incorporate auxiliary latent variables $\bar{z}$, and by defining:

$$p(\bar{z}) \triangleq \mathop{\mathbb{E}}_{(x,z_1,...,z_n)\sim q(x,z_1,...,z_n)} \big[q(\bar{z}|x, z_0, ..., z_n) \big] \tag{8.10}$$

We only need to make three assumptions about p and q:

- We can draw independent samples from $q(x, z_1, ..., z_n, \bar{z})$.
- We can compute $\log q(x, z_1, ..., z_n, \bar{z})$.
  - Or, if $q(x)$ is fixed a priori, we can compute $\log q(z_1, ..., z_n, \bar{z}|x)$.
- We can compute $\log p(x, z_1, ..., z_n|\bar{z})$.

These assumptions are trivial to meet when working with directed models.

We now consider minimizing $\mathrm{KL}(q(x, z_0, ..., z_n, \bar{z}) \,||\, p(x, z_0, ..., z_n, \bar{z}))$. Performing this minimization via Monte Carlo simply repeats the following two steps:

1. Sample $(x, z_0, ..., z_n, \bar{z})$ from $q(x, z_0, ..., z_n, \bar{z}) \triangleq \mathcal{D}_x(x) q(z_0, ..., z_n, \bar{z}|x)$.
2. Take a step to reduce: $\log\left(q(z_0, ..., z_n, \bar{z}|x)\mathcal{D}_x(x)\right) - \log\left(p(x, z_0, ..., z_n|\bar{z})p(\bar{z})\right)$.

Putting this all together, we have:

$$\begin{aligned}
\mathrm{KL}(q(x, z_0,..., z_n, \bar{z}) \,||\, p(x, z_0, ..., z_n, \bar{z})) = \ & ... && (8.11)\\
= \ & \mathop{\mathbb{E}}_{(x,z_0,...,z_n,\bar{z})\sim q(x,z_0,...,z_n,\bar{z})} \left[\log \frac{q(z_0, ..., z_n, \bar{z}|x)\mathcal{D}_x(x)}{p(x, z_0, ..., z_n|\bar{z})p(\bar{z})}\right] && (8.12)\\
= \ & \mathop{\mathbb{E}}_{(x,z_0,...,z_n,\bar{z})\sim q(x,z_0,...,z_n,\bar{z})} \left[\log \frac{q(x, z_0, ..., z_n|\bar{z})q(\bar{z})}{p(x, z_0, ..., z_n|\bar{z})p(\bar{z})}\right] && (8.13)\\
\geq \ & \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x}\left[\log \mathcal{D}_x(x)\right] + \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x}\left[-\log p(x)\right] && (8.14)
\end{aligned}$$

where the last step is justified by Eq. 8.5. From Eq. 8.13, it is clear that our definition for $p(\bar{z})$ in Eq. 8.10 lets us ignore $p(\bar{z})$ when minimizing $\mathrm{KL}(q \,||\, p)$. The trick here is in acknowledging that *all* valid factorizations of $q(x, z_0, ..., z_n, \bar{z})$ or $p(x, z_0, ..., z_n, \bar{z})$ – found via Bayes' rule – have the same density everywhere. In other words, minimizing Eq. 8.12 is equivalent to minimizing Eq. 8.13.

Due to the "non-parametric" definition of $p(\bar{z})$ in Eq. 8.10, the bound may become degenerate.[4] However, we can still generate samples from $\mathcal{D}_x$ by starting with a single $x \sim \mathcal{D}_x$ and then alternating between sampling $\bar{z} \sim q(z_0, ..., z_n, \bar{z}|x)$ and $x \sim p(x, z_0, ..., z_n|\bar{z})$. If $\mathrm{KL}(q \,||\, p) = 0$, this process will sample from $p(x) = \mathcal{D}_x(x)$ because the marginals over x and $\bar{z}$, and the "marginal conditionals" for $x|\bar{z}$ and $\bar{z}|x$, must be the same in p and q.[5] If any of these did not match in p and q, we would not have $\mathrm{KL}(q \,||\, p) = 0$. So, this alternating process samples from the Gibbs chain over $(x, \bar{z}) \sim p(x, \bar{z}) \triangleq p(x|\bar{z})p(\bar{z}) = p(\bar{z}|x)p(x)$, where:

$$p(x|\bar{z}) \triangleq \mathop{\mathbb{E}}_{z_0,...,z_n \sim p(z_0,...,z_n|\bar{z})} p(x|z_0, ..., z_n, \bar{z}) \tag{8.15}$$

This chain has marginal $p(x) = q(x) = \mathcal{D}_x(x)$. Thus, denoising auto-encoders and GSNs can be seen as methods of training a directed generative model p whose prior (over $\bar{z}$) is defined non-parametrically by convolving the encoder q with the data distribution $\mathcal{D}_x$. I.e., when training a DAE/GSN, one is minimizing the divergence $\mathrm{KL}(q||p)$ between the encoder distribution q and decoder distribution p. The distribution q is constructed to have marginal $q(x) \triangleq \mathcal{D}_x(x)$, and the "prior" $p(\bar{z})$ for p is defined non-parametrically as $q(\bar{z})$.

---

[4] Various remedies for this degeneracy have been investigated in the DAE/GSN framework. In Chapter 6 we avoided this issue by using a non-degenerate prior, and by controlling KL divergence between prior and posterior.

[5] Several other assumptions – they're minor compared to assuming $\mathrm{KL}(q \,||\, p) = 0$ – are required for this statement to be entirely correct. See Sec. 6.3 for details.

8.5 Training RBMs using a path-wise KL bound

Given an RBM with parameters w, define q as the RBM's Gibbs chain initialized from distribution $\mathcal{D}_x(x_0)$ over x_0 and run for k steps, and define p as the RBM's Gibbs chain initialized from distribution $p(x_k)$ over x_k and run for k steps. Also define the k-step trajectory $\tau \triangleq \{x_0, z_0, ..., x_k\}$ generated by running q starting from $x_0 \sim \mathcal{D}_x(x_0)$. With these definitions we write:

$$\begin{aligned}
\mathrm{KL}(q \,||\, p) &= \mathop{\mathbb{E}}_{\tau\sim q} \log \frac{q(\tau)}{p(\tau)} && (8.16)\\
&= \sum_{\tau} q(\tau) \log \frac{q(\tau)}{p(\tau)} && (8.17)\\
\nabla_w \mathrm{KL}(q \,||\, p) &= \sum_{\tau} \left(\nabla_w q(\tau) \log \frac{q(\tau)}{p(\tau)} + q(\tau) \nabla_w \log \frac{q(\tau)}{p(\tau)} \right) && (8.18)\\
&= \sum_{\tau} \left(q(\tau) \nabla_w \log q(\tau) \log \frac{q(\tau)}{p(\tau)} + q(\tau) \nabla_w \log \frac{q(\tau)}{p(\tau)} \right) && (8.19)\\
&= \sum_{\tau} q(\tau) \left(\nabla_w \log q(\tau) \log \frac{q(\tau)}{p(\tau)} + \nabla_w \log \frac{q(\tau)}{p(\tau)} \right) && (8.20)\\
&= \mathop{\mathbb{E}}_{\tau\sim q(\tau)} \left(\nabla_w \log q(\tau) \log \frac{q(\tau)}{p(\tau)} + \nabla_w \log \frac{q(\tau)}{p(\tau)} \right) && (8.21)\\
&= \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x(x)} \left[\mathop{\mathbb{E}}_{\tau_x\sim q(\tau|x)} \left[\nabla_w \log q(\tau_x|x) \left(\log \frac{q(\tau_x|x)}{p(\tau_x|x)} + \log \frac{\mathcal{D}_x(x)}{p(x)} \right) \right] \right] + \\
&\qquad \mathop{\mathbb{E}}_{\tau\sim q(\tau)} \left[\nabla_w \log \frac{q(\tau)}{p(\tau)} \right] && (8.22)\\
&= \mathop{\mathbb{E}}_{x\sim\mathcal{D}_x(x)} \left[\mathop{\mathbb{E}}_{\tau_x\sim q(\tau|x)} \left[\nabla_w \log q(\tau_x|x) \log \frac{q(\tau_x|x)}{p(\tau_x|x)} \right] \right] + && (8.23)\\
&\qquad \mathop{\mathbb{E}}_{\tau\sim q(\tau)} \left[\nabla_w \log \frac{q(\tau)}{p(\tau)} \right] \quad ** \log \frac{\mathcal{D}_x(x)}{p(x)} \text{ is like a baseline in RL } **
\end{aligned}$$

We go from Eq. 8.21 to Eq. 8.22 by the definition of $q(\tau) \triangleq \mathcal{D}_x(x) q(\tau_x|x)$. We note that the first line of Eq. 8.23 goes to 0 as q runs for steps $k \to \infty$. I.e. for

an infinitely deep encoder-decoder pair q/p with shared weights between models and layers, the encoder samples from the exact posterior of the decoder, when the decoder's prior $p(x_k)$ is defined as the RBM's marginal over x. A proof of this statement is given in [43], by Hinton et al. We use w to indicate the RBM's parameter matrix, and we elide bias terms for ease of presentation.

Analogous to our treatment of GSNs, we consider a prior given by:[6]

$$p(x_k) \triangleq \sum_{\tau_{<k}} \mathcal{D}_x(x_0) q(z_0|x_0) q(x_1|z_0) q(z_1|x_1) \dots q(x_k|z_{k-1}) \tag{8.24}$$

where $\tau_{<k}$ indicates the truncated trajectory $\tau_{<k} \triangleq \{x_0, z_0, ..., x_{k-1}, z_{k-1}\}$. This is just the marginal $q(x_k)$ in the trajectory distribution $q(\tau)$. With this definition of $p(x_k)$, and in light of the alternative factorization of q which gives:

$$\log \frac{q(\tau)}{p(\tau)} = \log \frac{q(x_k) q(x_0, z_0, ..., x_{k-1}, z_{k-1}|x_k)}{p(x_k) p(z_{k-1}|x_k) p(x_{k-1}|z_{k-1}) \dots p(x_0|z_0)} \tag{8.25}$$

it is clear that $p(x_k)$ contributes to neither $\log \frac{q(\tau)}{p(\tau)}$ nor $\nabla_w \log \frac{q(\tau)}{p(\tau)}$. Thus, $p(x_k)$ can be ignored completely when optimizing the path-wise KL bound in Eq. 8.21. This observation is crucial for interpreting, and perhaps improving on, RBM training using contrastive divergence.

When q runs for a finite number of steps – i.e. as in contrastive divergence – it is easiest to reason about training using Eq. 8.21. The gradients required for

[6] This distribution converges to the RBM's marginal over x_k as $k \to \infty$, which bypasses the issues with "complementary priors" faced in [43]. We will assume the RBM Gibbs chain is ergodic, which is trivial to enforce when x and z are both binary vectors.

minimizing the KL bound in Eq. 8.21 are $\nabla_w \log q(\tau)$ and $\nabla_w \log \frac{q(\tau)}{p(\tau)}$. Given these gradients, the only other required quantity is $\log \frac{q(\tau)}{p(\tau)}$. Using trajectory generating distributions q and p defined as:

$$q(\tau) \triangleq \mathcal{D}_x(x_0)q(z_0|x_0)q(x_1|z_0)q(z_1|x_1)\,...\,q(x_k|z_{k-1}) \tag{8.26}$$

$$p(\tau) \triangleq p(x_k)p(z_{k-1}|x_k)p(x_{k-1}|z_{k-1})p(z_{k-2}|x_{k-1})\,...\,p(x_0|z_0) \tag{8.27}$$

we reformulate RBM training in terms of the KL bound in Eq. 8.21.

Though the term:

$$\nabla_w \log q(\tau) \log \frac{q(\tau)}{p(\tau)} \tag{8.28}$$

in Eq. 8.21 disappears as q runs for an increasing number of steps, it may be non-negligible for finite step counts. Contrastive divergence ignores this term, which measures the effect of changing the parameters w on the KL divergence between the encoder q and the decoder p. But, this term is actually simple to compute. For the k-step trajectory $\tau \triangleq \{x_0, z_0, x_1, z_1, ..., x_k\}$ generated by q, we can write:

$$\log \frac{q(\tau)}{p(\tau)} = \log Z_{x_k} - \log Z_{x_0} \tag{8.29}$$

where $\log Z_{x_i}$ is the RBM *marginal* log-partition function for sample x_i. I.e.:

$$\log Z_{x_i} \triangleq \log \left(\sum_z \exp\left(-E(x_i, z)\right) \right) \tag{8.30}$$

where $E(x_i, z)$ is the RBM's energy function (parametrized by w). This quantity is minus the "free energy" for x_i in the RBM.[7] We define the RBM energy $E(x, z)$ as:

$$E(x, z) \triangleq -x^\top w z \tag{8.31}$$

where x and z are binary vectors and w is a suitable matrix. We can now write:

$$\begin{aligned} \log Z_{x_k} - \log Z_{x_0} &= F(x_0) - F(x_k) && (8.32)\\ &= -\sum_{j=1}^{|z|} \log\left(1 + \exp(\tilde{z}_0^j)\right) + \sum_{j=1}^{|z|} \log\left(1 + \exp(\tilde{z}_k^j)\right) && (8.33)\\ \text{where} \quad \tilde{z}_i &\triangleq x_i^\top w && (8.34) \end{aligned}$$

and we define $\tilde{z}_i^j$ as the j^{th} element of the "partial energy" vector $\tilde{z}_i$ in Eq. 8.34. Notice that Eq. 8.33 can be computed directly from τ, without further sampling or approximation. We will need the gradient of Eq. 8.32, which can be written:

$$\begin{aligned} \nabla_w \left[\log Z_{x_k} - \log Z_{x_0}\right] = &-\sum_{j=1}^{|z|} \nabla_w \log\left(1 + \exp(\tilde{z}_0^j)\right) + && (8.35)\\ &\sum_{j=1}^{|z|} \nabla_w \log\left(1 + \exp(\tilde{z}_k^j)\right)\\ = &x_k \sigma(\tilde{z}_k)^\top - x_0 \sigma(\tilde{z}_0)^\top && (8.36) \end{aligned}$$

[7] Statistical physicists really like free energies. Marginal log-partition function is a more immediately descriptive term, which also will not be confused with the variational free energy.

where $\sigma(\cdot)$ indicates element-wise sigmoid and $\tilde{z}_i$ are defined as in Eq. 8.34. Note that we assume all vectors are columns which become rows when transposed. Eq. 8.36 gives a single Monte Carlo sample of the gradient used in CD-k, i.e. contrastive divergence with k-step roll-outs. Practical applications of CD-k typically perform partial marginalization of the expectation which gets written like:

$$\nabla_w \log p(x) = \langle v_0 h_0^\top \rangle_0 - \langle v_k h_k^\top \rangle_k \tag{8.37}$$

in the contrastive divergence literature. This partial marginalization, which corresponds to the gradients in Eq. 8.36, arises naturally from our formulation.

The equality in Eq. 8.29 can be found by brute force algebra on the forwards and backwards trajectory distributions in Eq. 8.26 and Eq. 8.27.

8.6 Note to Readers

The work presented in this appendix is not complete. It was included in the thesis because it seems interesting, and it illustrates some benefits of adopting a sequential view of generative models.

References

[1] Amr Ahmed and Eric P. Xing. Recovering time-varying networks of dependencies in social and biological studies. *Proceedings of the National Academy of Science (PNAS)*, 106(29):11878–83, 2009.

[2] Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[3] Philip Bachman, Amir massoud Farahmand, and Doina Precup. Sample-based approximate regularization. In *International Conference on Machine Learning (ICML)*, 2014.

[4] Philip Bachman and Doina Precup. Improved estimation in time varying models. In *International Conference on Machine Learning (ICML)*, 2012.

[5] Philip Bachman and Doina Precup. Data generation as sequential decision making. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[6] Philip Bachman and Doina Precup. Variational generative stochastic networks with collaborative shaping. In *International Conference on Machine Learning (ICML)*, 2015.

[7] Pierre Baldi and Peter Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[8] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research (JMLR)*, 7:2399–2434, 2006.

[9] Yoshua Bengio, Éric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. Deep generative stochastic networks trainable by backprop. *arXiv:1306.1091v5 [cs.LG]*, 2014.

[10] Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[11] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math expression compiler. In *Python for Scientific Computing Conference (SciPy)*, 2010.

[12] Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3), 2011.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[14] B. Blankertz, K.-R. Müller, D. J. Krusienski, G. Schalk, J.-R. Wolpaw, A. Schlögl, G. Pfurtscheller, J. D. R. Míllan, M. Schroder, and N. Birbaumer. The bci competition iii: Validating alternative approaches to actual bci problems. *IEEE Transactions on Neural Systems Rehabilitation*, 14(2):153–159, 2006.

[15] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, 2003.

[16] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on Computational Learning Theory (COLT)*, 1998.

[17] Byron Boots and Geoff Gordon. An online spectral learning algorithm for partially observable nonlinear dynamical systems. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2011.

[18] Olivier Bousquet, Olivier Chapelle, and Matthias Hein. Measure based regularization. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS - 16)*. MIT Press, 2004.

[19] Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Quickly generating representative samples from an rbm-derived process. *Neural Computation*, 23(8):2053–2073, 2011.

[20] R. Dennis Cook and Liliana Forzani. Likelihood-based sufficient dimension reduction. *Journal of the American Statistical Association*, 104(485):197–208, 2009.

[21] Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4), 1980.

[22] Emily L. Denton, Soumith Chintala, Arthur Szlam, and Robert Fergus. Deep generative models using a laplacian pyramid of adversarial networks. *arXiv:1506.05751 [cs.CV]*, 2015.

[23] Laurens Van der Maaten, Minmin Chen, Stephen Tyree, and Killian Q. Weinberger. Learning with marginalized corrupted features. In *International Conference on Machine Learning (ICML)*, 2013.

[24] Laurens Van der Maaten and Geoffrey E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research (JMLR)*, 2008.

[25] Paul H. C. Eilers and Brian D. Marx. Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2), 1996.

[26] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9:431–441, 2008.

[27] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. Technical report, Stanford University, 2009.

[28] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. Dimensionality reduction for supervised learning with reproducing kernel hilbert spaces. *Journal of Machine Learning Research (JMLR)*, 5:73–99, 2004.

[29] Mathieu Gemain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning (ICML)*, 2015.

[30] Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. Large-scale feature learning with spike-and-slab sparse coding. In *International Conference on Machine Learning (ICML)*, 2012.

[31] Ian J. Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Multi-prediction deep boltzmann machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[32] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[33] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *International Conference on Machine Learning (ICML)*, 2013.

[34] Yves Grandvalet and Yoshua Bengio. *Semi-Supervised Learning*, chapter Entropy Regularization. MIT Press, 2006.

[35] Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850 [cs.NE]*, 2013.

[36] Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning (ICML)*, 2015.

[37] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. In *International Conference on Machine Learning (ICML)*, 2014.

[38] Michael U. Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Classifier abc. In *MCMSki IV (posters)*, 2014.

[39] Michael U. Gutmann, Ritabrata Dutta, Samuel Kaski, and Jukka Corander. Likelihood-free inference via classification. In *arXiv:1407.4981v1 [stat.CO]*, 2014.

[40] László Györfi, Michael Kohler, Adam Krzyżak, and Harro Walk. *A Distribution-Free Theory of Nonparametric Regression*. Springer Verlag, New York, 2002.

[41] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *Elements of Statistical Learning II*. 2008.

[42] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[43] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[44] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580v1 [cs.NE]*, 2012.

[45] http://github.com/Netflix/SimianArmy/wiki/Chaos Monkey.

[46] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13:411–430, 2000.

[47] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, 1999.

[48] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Association for Computational Linguistics (ACL)*, 2014.

[49] Kwang In Kim, Florian Steinke, and Matthias Hein. Semi-supervised regression using Hessian energy with an application to semi-supervised dimensionality reduction. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS - 22)*, pages 979–987. 2009.

[50] Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[51] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

[52] Mladen Kolar, Le Song, and Eric P. Xing. Sparsistent learning of varying-coefficient models with structural changes. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[53] Mladen Kolar and Eric P. Xing. On time varying undirected graphs. In *Artificial Intelligence and Statistics (AISTATS)*, 2011.

[54] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[56] Simon Lacoste-Julien, Fei Sha, and Michael I. Jordan. Disclda: Discriminative learning for dimensionality reduction and classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[57] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *International Conference on Machine Learning (ICML)*, 2011.

[58] Steffen L. Lauritzen. *Graphical Models*. Oxford Statistical Science. Oxford University Press, 1996.

[59] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning (ICML)*, 2014.

[60] Quoc V. Le, Marc Aurelio Ranzato, Ruslan R. Salakhutdinov, Andrew Y. Ng, and Joshua Tenenbaum. Workshop on challenges in learning hierarchical models: Transfer learning and optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[61] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *International Conference on Machine Learning (ICML – workshops)*, 2013.

[62] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[63] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[64] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013.

[65] Sergey Levine and Vladlen Koltun. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[66] Sergey Levine and Vladlen Koltun. Learning complex neural network policies with trajectory optimization. In *International Conference on Machine Learning (ICML)*, 2014.

[67] Fabien Lotte and Cuntai Guan. Regularizing common spatial patterns to improve bci designs: Unified theory and new algorithms. *IEEE Transactions on Biomedical Engineering*, 58(2):355–362, 2011.

[68] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE International Conference on Computer Vision (ICCV)*, 1999.

[69] Julien Mairal, Francis Bach, and Jean Ponce. Task-driven dictionary learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[70] Nicolai Meinshausen and Peter Bühlmann. High-dimensional graphs and variable selection with the lasso. *Annals of Statistics*, 34(3):1436–1462, 2006.

[71] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning (ICML)*, 2014.

[72] Boaz Nadler, Nathan Srebro, and Xueyuan Zhou. Semi-supervised learning with the graph laplacian: The limit of infinite unlabelled data. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS - 22)*, pages 1330–1338, 2009.

[73] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[74] Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[75] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulties of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2013.

[76] N. D. Pearce and M. P. Wand. Penalized splines and reproducing kernel methods. *The American Statistician*, 60(3), 2006.

[77] Jan Peters, Karen Mulling, and Yasemin Altun. Relative entropy policy search. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2010.

[78] Tapani Raiko, Li Yao, Kyunghyun Cho, and Yoshua Bengio. Iterative neural autoregressive distribution estimator (nade-k). In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[79] Danilo J. Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, 2015.

[80] Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning (ICML)*, 2014.

[81] Salah Rifai, Yann Dauphin, Pascal Vincent, Yoshua Bengio, and Xavier Muller. The manifold tangent classifier. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[82] Salah Rifai, Gregoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher-order contractive auto-encoder. In *European Conference on Machine Learning (ECML) and Principles and Practice of Knowledge Discovery in Databases (PKDD)*, 2011.

[83] Oren Rippel, Michael A. Gelbart, and Ryan P. Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning (ICML)*, 2014.

[84] Sajama and Alon Orlitsky. Supervised dimensionality reduction using mixture models. In *International Conference on Machine Learning (ICML)*, 2005.

[85] Tim Salimans, Diederik P Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning (ICML)*, 2015.

[86] Alexander M. Samarov. Exploring regression structure using nonparametric functional estimation. *Journal of the American Statistical Association*, 88(423):836–847, 1993.

[87] Alois Schlögl, Felix Lee, Horst Bischof, and Gert Pfurtscheller. Characterization of four-class motor-imagery eeg data for the bci-competition 2005. *Journal of Neural Engineering*, 2:14–22, 2005.

[88] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[89] H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11:363–371, 1965.

[90] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. *Lectures on Stochastic Programming: Modeling and Theory*. Society for Industrial and Applied Mathematics (SIAM), 2009.

[91] Sajid M. Siddiqi, Byron Boots, and Geoff Gordon. Reduced-rank hidden markov models. In *Artificial Intelligence and Statistics (AISTATS)*, 2010.

[92] Paul Smolensky. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Department of Computer Science, University of Colorado, Boulder, 1986.

[93] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[94] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015.

[95] Kihyuk Sohn, Wenling Shang, and Honglak Lee. Improved multimodal deep learning with variation of information. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[96] Kihyuk Sohn, Xinchen Yan, and Honglak Lee. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[97] Le Song, Mladen Kolar, and Eric P. Xing. Keller: estimating time-varying interactions between genes. *Bioinformatics*, 25:i128–i138, 2009.

[98] Le Song, Mladen Kolar, and Eric P. Xing. Time-varying dynamic bayesian networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[99] Karthik Sridharan, Nathan Srebro, and Shie Shalev-Shwartz. Fast rates for regularized objectives. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems (NIPS - 21)*, 2009.

[100] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer, 2008.

[101] Joshua Susskind, Adam Anderson, and Geoffrey E Hinton. The toronto face database. 2010.

[102] Ilya Sutskever and Quoc V. Le. Sequence-to-sequence learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[103] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

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

[105] Emanuel Todorov. Efficient computation of optimal action. *Proceedings of the National Academy of Science (PNAS)*, 106(18):11478–11483, 2009.

[106] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning (ICML)*, 2014.

[107] Pascal Vincent, Hugo Larochelle, and Yoshua Bengio. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008.

[108] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[109] Grace Wahba. *Spline Models for Observational Data*. SIAM [Society for Industrial and Applied Mathematics], 1990.

[110] Matrin J. Wainwright, Pradeep Ravikumar, and John D. Lafferty. High-dimensional graphical model selection using l1-regularized logistic regression. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[111] David Warde-Farley, Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. An empirical analysis of dropout in piecewise linear networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[112] Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *International Conference on Machine Learning (ICML)*, 2008.

[113] Simon N. Wood. Thin plate regression splines. *Journal of the Royal Statistical Society, Series B*, 65, 2003.

[114] Yingcun Xia, Howell Tong, W. K. Li, and Li-Xing Zhu. An adaptive estimation of dimension reduction space. *Journal of the Royal Statistical Society B*, 64(3):363–410, 2002.

[115] Huan Xu, Constantine Caramanis, and Shie Mannor. Robust regression and lasso. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[116] Huan Xu, Constantine Caramanis, and Shie Mannor. Robustness and regularization of support vector machines. *Journal of Machine Learning Research (JMLR)*, 10, 2009.

[117] Alan L. Yuille and Norberto M. Grzywacz. The motion coherence theory. In *Proceedings of the International Conference on Computer Vision*, 1988.

[118] Shuheng Zhou, John Lafferty, and Larry Wasserman. Time varying undirected graphs. *Machine Learning*, 80:295–319, 2010.

[119] Xueyuan Zhou and Mikhail Belkin. Semi-supervised learning by higher order regularization. In *International Conference on Artificial Intelligence and Statistics*, pages 892–900, 2011.

[120] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *Proceedings of the 12th International Conference on Machine LearningProceedings of the 27th International Conference on Machine Learning (ICML)*, volume 3, pages 912–919, 2003.

[121] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67(2):301–320, 2005.