Université de Montréal

Advances in scaling deep learning algorithms

par Yann N. Dauphin

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

June, 2015

Résumé

Les algorithmes d'apprentissage profond forment un nouvel ensemble de méthodes puissantes pour l'apprentissage automatique. L'idée est de combiner des couches de facteurs latents en hierarchies. Cela requiert souvent un coût computationel plus elevé et augmente aussi le nombre de paramètres du modèle. Ainsi, l'utilisation de ces méthodes sur des problèmes à plus grande échelle demande de réduire leur coût et aussi d'améliorer leur régularisation et leur optimization. Cette thèse adresse cette question sur ces trois perspectives.

Nous étudions tout d'abord le problème de réduire le coût de certains algorithmes profonds. Nous proposons deux méthodes pour entrainer des machines de Boltzmann restreintes et des auto-encodeurs débruitants sur des distributions sparses à haute dimension. Ceci est important pour l'application de ces algorithmes pour le traitement de langues naturelles. Ces deux méthodes (Dauphin et al., 2011; Dauphin and Bengio, 2013) utilisent l'échantillonage par importance pour échantilloner l'objectif de ces modèles. Nous observons que cela réduit significativement le temps d'entrainement. L'accéleration atteint 2 ordres de magnitude sur plusieurs bancs d'essai.

Deuxièmement, nous introduisont un puissant régularisateur pour les méthodes profondes. Les résultats expérimentaux démontrent qu'un bon régularisateur est crucial pour obtenir de bonnes performances avec des gros réseaux (Hinton et al., 2012). Dans Rifai et al. (2011), nous proposons un nouveau régularisateur qui combine l'apprentissage non-supervisé et la propagation de tangente (Simard et al., 1992). Cette méthode exploite des principes géometriques et permit au moment de la publication d'atteindre des résultats à l'état de l'art.

Finalement, nous considérons le problème d'optimiser des surfaces non-convexes à haute dimensionalité comme celle des réseaux de neurones. Tradionellement, l'abondance de minimum locaux était considéré comme la principale difficulté dans ces problèmes. Dans Dauphin et al. (2014a) nous argumentons à partir de résultats en statistique physique, de la théorie des matrices aléatoires, de la théorie des réseaux de neurones et à partir de résultats expérimentaux qu'une difficulté plus profonde provient de la prolifération de points-selle. Dans ce papier nous proposons aussi une nouvelle méthode pour l'optimisation non-convexe.

Keywords: apprentissage profond, réseaux de neurones, optimisation à haute dimensoin, machine de Boltzmann, auto-encodeurs.

Summary

Deep learning algorithms are a new set of powerful methods for machine learning. The general idea is to combine layers of latent factors into hierarchies. This usually leads to a higher computational cost and having more parameters to tune. Thus scaling to larger problems will require not only reducing their computational cost but also improving regularization and optimization. This thesis investigates scaling from these three perspectives.

We first study the problem of reducing the computational cost of some deep learning algorithms. We propose methods to scale restricted Boltzmann machines (RBM) and denoising auto-encoders (DAE) to very high-dimensional sparse distributions. This is important for applications of deep learning to natural language processing. Both methods (Dauphin et al., 2011; Dauphin and Bengio, 2013) rely on importance sampling to subsample the learning objective of these models. We show that this greatly reduces the training time, leading to 2 orders of magnitude speed ups on several benchmark datasets without losses in the quality of the model.

Second, we introduce a powerful regularization method for deep neural nets. Experiments have shown that proper regularization is in many cases crucial to obtaining good performance out of larger networks (Hinton et al., 2012). In Rifai et al. (2011), we propose a new regularizer that combines unsupervised learning and tangent propagation (Simard et al., 1992). The method exploits several geometrical insights and was able at the time of publication to reach state-of-the-art results on competitive benchmarks.

Finally, we consider the problem of optimizing over high-dimensional non-convex loss surfaces like those found in deep neural nets. Traditionally, the main difficulty in these problems is considered to be the abundance of local minima. In Dauphin et al. (2014a) we argue, based on results from statistical physics, random matrix theory, neural network theory, and empirical evidence, that the vast majority of critical points are saddle points, not local minima. We also propose a new optimization method for non-convex optimization.

Keywords: deep learning, neural networks, high-dimensional non-convex optimization, Boltzmann machines, auto-encoders.

Contents

Résumé . . . . . **ii**

Summary . . . . . **iii**

Contents . . . . . **iv**

List of Figures . . . . . **viii**

List of Tables . . . . . **ix**

1 Introduction **1**
1.1 Introduction to machine learning 2
1.2 Model families 4
1.3 Optimization 5
1.4 Regularization 6
1.5 Supervised Learning 7
1.5.1 Naive Bayes 8
1.5.2 Logistic regression 8
1.5.3 Deep Neural Networks 9
1.6 Unsupervised Learning 9

2 Deep Learning **11**
2.1 Deep neural networks 11
2.1.1 Approximation power 13
2.1.2 The power of distributed representations 13
2.1.3 Practical details 14
2.2 Restricted Boltzmann machines 15
2.2.1 Conditionals 16
2.2.2 Sampling 17
2.2.3 Learning 17
2.3 Regularized auto-encoders 20
2.3.1 Denoising auto-encoders 20
2.3.2 Contractive auto-encoders 23
2.3.3 Links between auto-encoders and RBMs 25
2.4 Stacking RBMs and AEs 25

2.4.1 Deep belief nets . . . 26
2.4.2 Stacked auto-encoders . . . 26
2.5 Why does pretraining work? . . . 27
2.6 Beyond pretraining . . . 28
2.7 Challenges . . . 29

3 Prologue to first article . . . 30
3.1 Article Detail . . . 30
3.2 Context . . . 30
3.3 Contributions . . . 31

4 Scaling DAEs to high-dimensional sparse inputs with importance sampling . . . 32
4.1 Related Work . . . 33
4.2 Denoising Auto-Encoders . . . 34
4.2.1 Introduction . . . 34
4.2.2 Training . . . 36
4.2.3 Motivation . . . 36
4.3 Scaling the Denoising Auto-Encoder . . . 37
4.3.1 Challenges . . . 37
4.3.2 Scaling the Encoder: Sparse Dot Product . . . 37
4.3.3 Scaling the Decoder: Reconstruction Sampling . . . 37
4.4 Implementation . . . 40
4.4.1 Encoder . . . 40
4.4.2 Decoder . . . 40
4.5 Experiments . . . 42
4.6 Conclusion . . . 45

5 Prologue to second article . . . 47
5.1 Article Detail . . . 47
5.2 Context . . . 47
5.3 Contributions . . . 48

6 Scaling RBMs to high-dimensional sparse inputs with importance sampling . . . 49
6.1 Reconstruction Sampling . . . 50
6.2 Restricted Boltzmann Machines . . . 51
6.3 Ratio Matching . . . 52
6.4 Stochastic Ratio Matching . . . 53
6.5 Experimental Results . . . 55
6.5.1 Using SRM to train RBMs . . . 57
6.5.2 Using RBMs as feature extractors for NLP . . . 59

6.6 Conclusion . . . 60

7 Prologue to third article . . . 62
7.1 Article Detail . . . 62
7.2 Context . . . 62
7.3 Contributions . . . 63

8 Regularizing deep networks with a geometric approach . . . 64
8.1 Contractive auto-encoders (CAE) . . . 66
8.1.1 Traditional auto-encoders . . . 66
8.1.2 First order and higher order contractive auto-encoders . . . 67
8.2 Characterizing the tangent bundle captured by a CAE . . . 68
8.2.1 Conditions for the feature mapping to define an atlas on a manifold . . . 69
8.2.2 Obtaining an atlas from the learned feature mapping . . . 69
8.3 Exploiting the learned tangent directions for classification . . . 70
8.3.1 CAE-based tangent distance . . . 70
8.3.2 CAE-based tangent propagation . . . 71
8.3.3 The Manifold Tangent Classifier (MTC) . . . 71
8.4 Related prior work . . . 72
8.5 Experiments . . . 74
8.6 Conclusion . . . 77

9 Prologue to Fourth article . . . 78
9.1 Article Detail . . . 78
9.2 Context . . . 78
9.3 Contributions . . . 79

10 Identifying the challenges in high-dimensional non-convex optimization . . . 80
10.1 The prevalence of saddle points in high dimensions . . . 81
10.2 Experimental validation of the prevalence of saddle points . . . 83
10.3 Dynamics of optimization algorithms near saddle points . . . 85
10.4 Generalized trust region methods . . . 87
10.5 Attacking the saddle point problem . . . 88
10.6 Experimental validation of the saddle-free Newton method . . . 91
10.6.1 Existence of Saddle Points in Neural Networks . . . 91
10.6.2 Effectiveness of saddle-free Newton Method in Deep Feedforward Neural Networks . . . 93
10.6.3 Recurrent Neural Networks: Hard Optimization Problem . . . 94
10.7 Conclusion . . . 95
10.8 Appendix . . . 96

10.8.1 Description of the different types of saddle-points 96
10.8.2 Reparametrization of the space around saddle-points 97
10.8.3 Empirical exploration of properties of critical points 97
10.8.4 Proof of Lemma 1 99
10.8.5 Implementation details for approximate saddle-free Newton . 100
10.8.6 Experiments 100

11 Conclusion **102**

References **104**

List of Figures

2.1 Graphical depiction of a one layer neural network (DNN) 12
2.2 Graphical model of the restricted Boltzmann machine (RBM) . . . 15
2.3 Schematic of the Denoising Auto-Encoder 21
2.4 Graphical model of the deep belief network (DBN). Image reproduced from Bengio (2009b). 26

4.1 Schematic of the Denoising Auto-Encoder 35
4.2 Experimental Results of reconstruction sampling on Amazon (small set) 43
4.3 Experimental Results of reconstruction sampling on RCV1 44
4.4 Learning curve with reconstruction sampling on the Full Amazon set 46
4.5 Embedding learned by the sampled DAE on Amazon 46

6.1 Average speed-ups achieved with SRM 58
6.2 Average norm of the gradients for $x_i = 1$ and $x_i = 0$ 59

8.1 Visualization of the tangents learned by CAEs 75
8.2 Visualization of the tangents learned by local PCA 75

10.1 Empirical validation of the prevalence of saddle points 84
10.2 Behavior of optimizers near saddle points 86
10.3 Evaluation of optimization methods for small MLPs 90
10.4 Empirical results using SFN for deep auto-encoders and recurrent networks 93
10.5 Different saddle point structures 98

List of Tables

6.1 Generative performance of the RBMs trained with SRM. 57
6.2 Classification results on RCV1 for SRM pretrained DNNs 59
6.3 Classification results on 20 Newgroups for SRM pretrained DNNs . 60

8.1 Classification accuracy with the tangent distance given by the CAE 76
8.2 Classification results with the MTC in a semi-supervised setting . . 76
8.3 Classification results with the MTC in a fully supervised setting . . 77
8.4 Classification results with the MTC for Covertype 77

List of Abbreviations

AE	Auto-Encoder
AIS	Annealed Importance Sampling
CAE	Contractive Auto-Encoder
CD	Contrastive Divergence
CNN	Convolutional Neural Network
DAE	Denoising Auto-Encoder
DBN	Deep Belief Network
DNN	Deep Neural Network
I.I.D	Independent and Identically-Distributed
KL	Kullback-Leibler
MLP	Multi-Layer Perceptron
MTC	Manifold Tangent Classifier
PCD	Persistent Contrastive Divergence
RBM	Restricted Boltzmann Machine
RM	Ratio Matching
SDAE	Stacked Denoising Auto-Encoder
SFN	Saddle-Free Newton
SGD	Stochastic Gradient Descent
SML	Stochastic Maximum Likelihood
SRM	Stochastic Ratio Matching
SVM	Support Vector Machine

Acknowledgments

I feel lucky to have been part of the LISA lab (predecessor of the MILA). Yoshua Bengio instilled in his lab a deep passion for creativity and a degree of openness that allows collaboration to flourish. This has had a profound impact on me. I am also grateful to Yoshua for the experience and advice he has shared with me.

I am thankful to have had amazing people to discuss and argue about ideas with. There have been too many to count, but I am particularly in the debt of my collaborators: Salah Rifai (AKA the idea factory), Xavier Glorot, Gregoire Mesnil, Xavier Muller, Harm de Vries, Pascal Vincent, Surya Ganguli, Kyun-hyun Cho, Caglar Gulcere. I would also like to thank Ian Goodfellow, Aaron Courville, Guillaume Alain, Laurent Dinh and Mehdi Mirza. Frederic Bastien too for his work on Theano and giving us clusters that work. These are some of the people that have made my days at the lab pleasant.

I am also thankful to my parents for sharing with me their passion for knowledge and my aunts, cousins and friends for supporting me through these 5 years. To my roommates Fedo, Manu and Marina sorry for forgetting the milk so many times but it was for a good cause. And thanks to Robert Panet-Raymond for taking the time to do a college course that motivated me to do a Ph.D.

1 Introduction

Deep learning algorithms are a new development in machine learning. They are generating a lot of interest because they have achieved state-of-the-art results in significant benchmarks for artificial intelligence. These tasks include computer vision (Krizhevsky et al., 2012), language modelling (Mikolov et al., 2011), and automatic speech recognition (Seide et al., 2011). These advances and theoretical considerations have led some to hypothesize that deep learning may be a key component in learning *AI-hard* tasks (Bengio, 2009a). AI-hard tasks are those whose difficulty is equivalent to solving general purpose artificial intelligence. In particular, deep learning is a powerful solution to the problem of perception in intelligent systems. Whereas traditional machine learning requires humans to craft task-specific features, deep learning automatically learns features from raw data. It does so by learning a hierarchy of non-linear features from the input data. One significant challenge that remains to be solved on the road to solving AI-hard tasks is that of scale. Deep learning algorithms must be scaled both in terms of the sizes of the datasets they can handle and the size of the models themselves. For reference, one of the biggest deep learning (Krizhevsky et al., 2012) system has 10^9 connections, while the human brain has 10^{15} connections. This thesis studies the issue of scaling deep learning algorithms.

Chapter 1 is an introduction to machine learning. It is followed by a review of the new developments brought forth by deep learning in Chapter 2. The subsequent chapters describe work that has been done in the context of this thesis. Chapters 4 and 6 describe new methods to scale unsupervised deep learning algorithms to high-dimensional sparse distributions. In chapter 8, we propose a new regularization method for large deep neural networks. Finally, we investigate the problem of optimizing high-dimensional deep neural networks in chapter 10 and propose practical solutions.

1.1 Introduction to machine learning

Machine learning is the study of algorithms that can learn from examples. It is a form of artificial intelligence that combines notions from both statistics and optimization. The goal of a learning algorithm is to automatically learn a function $\hat{f}$ that can perform some task of interest. In comparison, most other AI approaches rely on human labor to specify program behaviour (Hayes-Roth et al., 1984). The distinctive quality of learning algorithms is that they are not explicitly programmed for the task of interest. For example, such an algorithm could learn to differentiate between cats and dogs using only a set of examples $\mathcal{D} = \{(\mathbf{x}_0, \mathbf{y}_0), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$. On the one hand, the algorithm must discover the hidden statistical link between the images $\mathbf{x}_i$ and the corresponding labels $\mathbf{y}_i$. This involves a search process over many different functions or hypotheses. Based on the dataset $\mathcal{D}$ it might reach the hypothesis for instance that cats have upright ears while dogs do not. However, it is not as simple as finding an hypothesis that fits the dataset. While all the dogs in $\mathcal{D}$ may have droopy ears it does not guarantee that all dogs do. Machine learning requires statistical induction: how to make good inferences from past data. Thus, the learning algorithm must find a hypothesis that fits the available data and even the unseen future data. This section explains how machine learning solves this problem.

In machine learning, learning consists in searching for the best function f within a family of functions $\mathcal{F}$ that performs a task. This function f is found through optimization. The ability of a function f to perform the task is measured by a so-called loss function $\mathcal{L}$. For example, the error function might measure the number of misclassified objects within $\mathcal{D}$. Mathematically, we can express learning as approximating the following operation

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \mathcal{L}(f(\mathbf{x}), \mathbf{y}) \tag{1.1}$$

In other words, we are trying to find a function $\hat{f}$ that most accurately predicts the labels $\mathbf{y}$ from the inputs $\mathbf{x}$. While there are other formulations, this one reveals some of the key aspects of machine learning. First, it is key to find an efficient approximation of the arg min operation. The naive solution of exhaustive search on the set $\mathcal{F}$ will not scale. There are several models families where $\mathcal{F}$ is infinite.

In practice, optimization methods are used to navigate the space. We will discuss some of these methods in Section 1.3. Another important aspect is the choice of the model family $\mathcal{F}$. There are many different families like neural networks and support vector machines. We will cover several in Section 1.2. The choice of the model family will influence the ease of optimizing the model. More importantly, it determines how well we will be able to mimic the true function $f^*(\mathbf{x}) = \mathbf{y}$ with $\hat{f}$. If the model family is too small, it may not include a function that matches f^*. This can be one of the causes of *underfitting*. Underfitting occurs when the model does not properly match the data. However, choosing a really large model family is not a silver bullet. If it is too large, we may find a function $\hat{f}$ that behaves exactly like f^* on the examples in $\mathcal{D}$ but behaves wildly differently on unseen examples. This is a common cause of *overfitting*. Models that suffer from overfitting have confused noises in the data with actual statistical relationships. Intuitively, we can see this can be mitigated by having more examples in $\mathcal{D}$. Another solution is the use of *regularizers*. We will cover model families in Section 1.2 and regularizers in Section 1.4.

Overfitting separates machine learning from optimization. We want $\hat{f}$ to mimic f^* on examples that may not be in $\mathcal{D}$. This is known as *generalization*. Generalization is important because $\mathcal{D}$ usually contains a relatively small number of examples. Collecting a dataset that would perfectly represent f^* is extremely hard. In fact, there are several tasks where the number of examples is infinite. Consider for examples problems where the input is an image. There are an infinite number of possible images with continuous-valued pixel intensities. The generalization error tells us the average error on all possible examples. Assuming the examples are sampled from a distribution $(\mathbf{x}, \mathbf{y}) \sim P(\mathbf{x}, \mathbf{y})$, the generalization error also known as the risk is

$$R(f) = \int_{\mathbf{x},\mathbf{y}} \mathcal{L}(f(\mathbf{x}), \mathbf{y}) P(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} \tag{1.2}$$

Notice the relation between Equation 1.1 and 1.2. In Equation 1.1 the integral is replaced by an empirical average on $\mathcal{D}$. Thus, Equation 1.1 minimizes the empirical risk $\hat{R}$, giving it the name empirical risk minimization (Vapnik, 1999). In general, it is not possible to compute $R(f)$. The integral may involve an infinite number of elements and $P(\mathbf{x}, \mathbf{y})$ is usually not known. We can minimize the generalization

error by the proxy of the empirical risk $\hat{R}$, though the resulting generalization error is not perfectly matched by the minimized empirical risk. Thankfully, the generalization error can be probabilistically upper-bounded by a function of the empirical risk (Vapnik, 1999; Valiant, 1984). In the next sections, we explore in more depth the issues raised in this introduction.

1.2 Model families

A model family is the set of functions $\mathcal{F}$ that are explored. The choice of the model family is very important. It determines the functions that can be learned and how efficiently they can be learned. There are two big branches of of model families.

In the *parametric* families, the different models are obtained by modifying a finite parameter vector. For example, consider the problem of predicting the flip of a coin. We could model this with a family of the form *the coin will fall on heads with probability* $\mathbf{d}$. The variable $\mathbf{d}$ is the parameter that will differentiate between different models. We can see here for example that the number of models is infinite if the parameter $\mathbf{d}$ is floating point with infinite precision. In general, you can write the model families as

$$\mathcal{F} = \{f(\theta)|\theta \in \Theta\} \tag{1.3}$$

where $\theta = (\theta_1, \ldots, \theta_n)$ is the parameter vector and Θ is the set of possible parameters (often $\Theta = \mathbb{R}^n$). The function f uses the parameter vector to determine its operation in some way. Another example of a parametric model family is a Turing machine with different program tapes to read. An advantage of parametric models is that they have a compact representation θ. In many cases, this makes optimization of these models straightforward. Popular parametric models include the logistic regression and deep neural networks with a fixed size. Without considering cross-validation, deep learning algorithms fall mainly in this category and thus this document will focus on parametric models. Several parametric models will be discussed in later sections.

On the other hand we have *non-parametric* families. Contrary to *parametric* families, they cannot be represented by a finite vector of parameters. Typically, the size of the model will needs to grow with the number of training examples. Non-parametric models encompass a wide array of very different models because there are no limitations on their form. This makes sense because they are defined as the complement of the more restricted parametric set of families. In some cases, it is an advantage that $\mathcal{F}$ can be very large. This can allow non-parametric models to fit the data very well. The downside is that an element often cannot be represented compactly. Representative models for this class include k-nearest neighbours (k-NN) and k-means. These models will not be covered in-depth in this thesis.

1.3 Optimization

Optimization is the process through which a good function f is selected from the model family $\mathcal{F}$. Optimizing a model is also known as learning or training the model. Properly choosing the optimization method determines how efficiently we will find good functions f. Thus in practice, the optimization method limits the kind of functions that can be learned. If it is not efficient enough, even if there are good functions in $\mathcal{F}$, we might not be able to find them.

In parametric models, optimization is mostly done using gradient based methods. The parametric function f in Equation and the loss $\mathcal{L}$ are usually differentiable with respect to the parameters θ. This allows us to perform steepest descent with the first-order gradient. In neural networks, the backpropagation algorithm can be used to obtain the gradients efficiently (LeCun et al., 1998). Gradient descent is a local search method that relies on calculating the gradient of $\hat{R}$ to find good directions to move in parameter space. $-\frac{\partial \hat{R}}{\partial \theta}$ tell us in which direction to move θ to most decrease the empirical risk $\hat{R}$. Optimization starts at a random point in parameter space. Often this is either $\theta_i = 0$ or $\theta \sim \mathcal{N}(\mu, \sigma^2)$. Optimization is an iterative procedure that gradually moves in parameter space until it approximately reaches the minimum ($\frac{\partial \hat{R}}{\partial \theta} = 0$). At each step t the parameters are updated such that

$$\theta^{t+1} = \theta^t - \eta \frac{\partial \hat{R}(\theta^t)}{\partial \theta^t}.$$

The value η is called the learning rate. It controls how fast optimization will move in parameter space. It cannot be set too high because that would cause the optimization process to oscillate wildly in the parameter space. It cannot be too small either because then learning might be impractically slow. The learning rate must be found through a process of trial and error. Optimization must be performed to convergence with different learning rates and the learning rate resulting in the best empirical risk $\hat{R}$ is chosen. Usually a separate set of examples $\mathcal{V}$ called the validation set is used to find the best learning rate. The learning rate is refered to as a *hyper-parameter* because it is like an extra parameter to be optimized over.

The are more powerful optimization methods that make use of high-order derivatives of the function. For instance, the Newton methods relies on the curvature information in the Hessian to automatically adjust a learning rate for each parameter. These more powerful methods will be covered to some extent in Chapter 10.

1.4 Regularization

Regularization is used to prevent overfitting. Regularization helps the model to extend or generalize to unseen examples. To do so, it relies on an *inductive bias* to choose which model will generalize well. An inductive bias is a set of assumptions about the function to be learned. We must choose the function which corresponds best with the data and the inductive bias.

Most regularizers can be understood from Occam's razor. It states that *among competing hypotheses, the hypothesis with the fewest assumptions should be selected.* It was popularised by the philosopher William of Ockham in the 13th century. For example, consider the problem of the existence of Mugs. On on the hand, we can hypothesize they have been created by humans, on the other we can believe they were created by Leprechauns. Occam's razor tells us to believe the former because it doesn't assume the existence of never before seen magical Leprechauns. In terms of machine learning, Occam's razor dictates to select the model with the lowest complexity. The complexity of the model can be measured in various ways. The most common way is to use a Tikhonov regularization (Tikhonov and Arsenin,

1977) which measures the p-norm of the parameters

$$\Omega(\theta) = \|\theta\|_p.$$

The regularizer is taken into account by optimizing both the empirical risk and the measure of complexity

$$\hat{f} = \arg\min_{f_\theta \in \mathcal{F}} \hat{R}(f_\theta) + \Omega(\theta).$$

Usually, the magnitude of the parameters θ can be said to correlate with the degree of belief of the model in some pattern. When the 1-norm is used, then the regularization ensures that we believe in as few things as possible. Regardless of how much we believe in them. If the 2-norm is used then it prevents believing in anything too strongly. Both of these assumptions may be beneficial, depending on the task. Another inductive bias is that a good model should use the minimum amount of input features. This can be implemented through feature selection algorithms (Kira and Rendell, 1992).

There has been a breakthrough in regularization led by Hinton et al. (2006a); Bengio et al. (2007a) with the appearance of a new powerful data-dependent regularization method. In these models, the inductive bias is given by modelling the distribution of the data. This has led to further research in this domain (Tikhonov and Arsenin, 1977; Hinton et al., 2012; Wang and Manning, 2013; Zeiler and Fergus, 2013). This renewed interest for regularization is due to the renewed interest in deep neural networks. In Chapter 8 we will explore two new data-dependent regularizations for deep neural nets.

1.5 Supervised Learning

Supervised learning algorithms predict a label $\mathbf{y}$ from an input $\mathbf{x}$. Labelled datasets have the form $\{(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^N, \mathbf{y}^N)\}$. The goal of supervised learning algorithms is to recover a function $f : X \to Y$ which maps from the input space X to the target space Y. Functions of this form are known as classifiers. Classifiers are used in some of the most popular applications of machine learning. For example,

in automatic speech recognition the inputs $\mathbf{x}$ are acoustic frames and the labels $\mathbf{y}$ are phonemes.

1.5.1 Naive Bayes

The naive Bayes classifier is one of the simplest machine learning algorithms. The purpose of the algorithm is to learn a function $p(\mathbf{y}|\mathbf{x})$ which takes an input $\mathbf{x}$ and assigns it a label $\mathbf{y}$. The label is one of k possible labels. Using Bayes theorem we can rewrite the conditional as

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y})p(\mathbf{x}|\mathbf{y})}{p(\mathbf{x})}$$

The model is called naive because the different input dimensions are considered conditionally independent. This gives us

$$\arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{y}) \prod_{i=1}^{n} p(x_i|\mathbf{y}).$$

We can see that predicting $p(\mathbf{y}|\mathbf{x})$ in this model relies on learning the marginal probability of each class $p(y_k)$ and the conditional probability of each input given the class label $p(x_i|y_k)$. Therefore, the parameters of this model are $\theta = (p(y_k), p(x_i|y_k))$. The parameters are learned essentially by counting the frequencies of each probabilities in the training data. This model is used for simple problems with little data, most often in natural language processing, because the independence assumption is too strong for most real-world problems.

1.5.2 Logistic regression

The logistic regression is a very popular machine learning algorithm for classification. The logistic regression is expressed as

$$p(\mathbf{y}|\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the logistic function, $\mathbf{W} \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^n$. m is the number of classes and n is the number of input features. The logistic regression uses a linear projection followed by a logistic to perform classification. The loss

function is given by

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\log p(\mathbf{y}|\mathbf{x}) = -\mathbf{y}^T \log p(\mathbf{y}|\mathbf{x}) - (1 - \mathbf{y}^T) \log(1 - p(\mathbf{y}|\mathbf{x}))$$

where $\mathbf{y}$ is a one-hot vector. For binary classification this simplifies to the cross-entropy function. It measures the distance between the true probability distribution and the model distribution. The parameters of this model are found by performing gradient descent on the empirical risk. The logistic regression is widely used because it has the benefit of convex optimization and it can scale to large datasets and large input sizes. However, it is limited to learn linear class boundaries. Thus it is important to use good feature extractors before feeding the data to the logistic regression. A common practice in computer vision is to use handcrafted or off-the-shelf features like SIFT Lowe (1999).

1.5.3 Deep Neural Networks

This type of algorithm will be studied throughout this thesis. They receive a more in-depth treatment in Section 2.1.

1.6 Unsupervised Learning

Unsupervised learning is the task of finding hidden structure in unlabelled data. Unlabelled datasets have the form $\{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ where $\mathbf{x}$ are the input. Unlike in supervised learning there is no direct label to be predicted. It is up to the practitioner to decide on a proxy task that will help learn structure in the input. There are many possible proxy tasks, but they usually rely on one of 3 principles for learning structure.

The first is the idea of clustering. Clustering algorithms will group similar examples in the same cluster or partition of the input space. Similarity is often defined as the distance in Euclidean space, but other more appropriate similarity measure can be used. Such algorithms include k-means and Gaussian mixture models (GMMs). In general the clustering approach will learn the biggest modes in the data. This is useful in some cases, but it may be too crude for many tasks.

Another idea is auto-association. A model with latent factors can be trained to reconstruct the input. This will force the latent factors to encode information that describes the example well. The latent factors can then be used as a new representation of the input. The various regularized auto-encoders described in Chapter 2.3 fall into this category. Using these models as features extractors has been shown to help classification on various benchmarks (Vincent et al., 2010; Rifai et al., 2011).

Finally there is the idea of density estimation. Density estimation algorithms will estimate the density of examples in the dataset. This will force the model to learn about how probability is distributed in the input space. If the density model has latent factors, they can be used as a representation. Interestingly, clustering can be thought as a very crude way to do density estimation. As we will see in Section 2.3.3 auto-encoders with proper regularization have also been shown to do density estimation. This goes to show these three learning principles are quite related. The RBMs (Chapter 6.2) and other graphical models are good examples of density estimators.

2 Deep Learning

Deep learning is a novel approach to machine learning that relies on learning multiple levels of features. Hinton et al. (2006a); Bengio et al. (2007a) were the first papers to show significant gains training deep neural networks. These early papers relied on unsupervised learning algorithms for feature extraction. The features learned by stacks of models such as restricted Boltzmann machines and auto-encoders were used as input to traditional classifiers. A particularly successful alternative was to use the parameters of these models as initialization for a deep neural net. This novel initialization method called pretraining was in fact shown to be a novel data-dependent regularizer (Erhan et al., 2010). The renewed interest for deep neural nets has led to a lot of exploration and set new benchmarks notably in automatic speech recognition (Dahl et al., 2012) and object recognition (Krizhevsky et al., 2012). It has been discovered since then that unsupervised pretraining is not the only way train deep neural networks. Pretraining is useful as a regularizer that helps reduce over-fitting. Other regularizers have recently been found to successfully train deep neural networks (Hinton et al., 2012; Goodfellow, 2013; Wang and Manning, 2013; Zeiler and Fergus, 2013). However, the core idea remains to train deep features hierachies with some form of regularization to ensure good generalization.

Section 2.1 describes deep neural networks. Section 6.2 introduces the restricted Boltzmann machine. The various regularized auto-encoders are described in Section 2.3. Section 2.5 presents some of the research explaining why pretraining has been found to be helpful for deep networks. Finally, Section 2.6 and 2.7 discuss new developments and challenges in the field of deep learning.

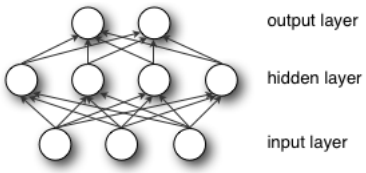


Figure 2.1: Graphical depiction of a one layer neural network (DNN). Image reproduced from Bengio (2009b).

2.1 Deep neural networks

A deep neural network is the combination of a logistic regressor with a learnt non-linear feature transform Ψ. This approach alleviates the problem of having to extract meaningful features through hand-crafting or off-the-shelf methods. The feature transform Ψ is learnt so that the it makes the input more linearly separable. The feature transform Ψ can be thought of as stack of logistic regressors where the targets are not known. The output units of these intermediary logistic regressions are known as *hidden* units because of this. A process known as *back-propagation* is used to propagate down the error from the top logistic regressors whose labels are known to the ones in the feature transform Ψ. This will cause the hidden units to help decrease the overall error of the network.

A single hidden layer neural network (illustrated in Figure 2.1) is typically characterized by the following equations

$$\begin{aligned} h_1(\mathbf{x}) &= \psi_1(\mathbf{W}^1\mathbf{x} + b^1) \\ o(\mathbf{h}) &= \psi_2(\mathbf{W}^2\mathbf{h} + b^2) \end{aligned}$$

The parameters $\mathbf{W}^1, \mathbf{W}^2$ are called the weight matrices. They represent patterns the hidden units are sensitive to and b^1, b^2 are offsets. The dot product is used to measure the distance between these patterns and the input. A unit will be strongly activated if the input is strongly co-linear with the pattern in it's weight vector (with the right offset).

The so-called activation functions ψ_1, ψ_2 are non-linear element-wise scalar functions that control the behaviour of the unit. Without these non-linear functions ($\psi_1(x) = x$) the dot products could be collapsed and the resulting input to output function would be linear. In other words, these functions are what allows the

network to learn non-linear functions of its input. Popular choices are the logistic sigmoid ($\frac{1}{1+e^{-x}}$), the tanh, and the rectified linear unit ($\max(0, x)$) (Glorot et al., 2011a). It is common to train networks with the logistic sigmoid as the activation for hidden units ψ_1 but several papers have reported better performance using the rectified linear units Glorot et al. (2011a); Hinton et al. (2012). Finding and justifying activation functions for the hidden layers is still an open research subject (Goodfellow et al., 2013).

The output activation function ψ_2 could be any function but the choice is typically guided by the type of the targets. For regression problems a linear activation is often used ($\psi_2(x) = x$). As we shall see this is justified in some cases because the network predicts the mean of a gaussian. For 1-of-k classification problems a softmax function is used. The softmax is defined by the following function

$$p(y_k|x_i) = \frac{e^{x_k}}{\sum_j e^{x_j}}$$

The output is a proper probability because the normalization factor over classes ensures that $\sum p(y_k|\mathbf{h}) = 1$. As we shall see, combined with the proper loss function, the probabilistic justification is that we are learning a multinomial distribution over the labels.

2.1.1 Approximation power

In a one layer network the feature transform Ψ is represented by the function h_1. We can add hidden layers by feeding h_1 as input to another hidden layer h_2 with different parameters and feeding h_2 to the output layer. A one hidden layer network is a universal approximator Hornik et al. (1989). Given enough hidden units, it can approximate any function. Adding hidden layers will help the network learn functions more efficiently. If the function to be learned can be factorized into sub-procedures then a deeper neural network will be exponentially smaller then it's shallow counterpart (Delalleau and Bengio, 2011; Pascanu et al., 2014).

2.1.2 The power of distributed representations

The representation Ψ learned by the DNN can be quite powerful. To better understand why, we can compare with an algorithm like Gaussian mixture models

(GMM). In the GMM, the feature representation Ψ describes how well the example matches each Gaussian of the mixture. The class of the example is affected mostly by the closest mean. Thus we can say that a GMM with N means and $O(N)$ parameters partitions the space into N sections.

In comparison, the hidden units forming Ψ are like logistic regressions but none of the output units are mutually exclusive. Each unit in Ψ partitions the space independently into two partitions. These binary partitions give rise to many others through combinations. Thus a DNN with N hidden units and $O(N)$ parameters partitions the input space into 2^N partitions. This allows the DNN to encode the properties of the examples by disentangling factors of variation (Bengio, 2009b). Disentangling the factors of variation means encoding each factor of variation in a different dimension of Ψ. This will allow classification on a specific property or a combination of them to be a linear operation in the space Ψ. The advantage of such a representation is that it would be more compact and would allow for better classification results.

Remarkably, disentangling factors of variation allows non-local generalization. Generalization in local methods like k-NN and decision trees is done by extrapolating from nearby examples. Local methods cannot generalize to zones in the input space that are far from training examples. With non-local generalization the network can properly operate with unseen examples. It is made possible because the network identifies properties or subspaces where the unseen example may be similar to the training examples. For example, a network may never have seen an example of a blue dog, but if the network disentangles the color from the object, the output logistic can properly classify the example as a dog.

2.1.3 Practical details

The weight matrices $\mathbf{W} \in \mathbb{R}^{m \times n}$ can be initialized with a uniform distribution $[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}]$ for tanh activation units and ReLU units Glorot and Bengio (2010). This initialization will ensure that early during training the gradients have zero-mean and unit standard deviation which will make it faster LeCun et al. (1998).

For regression, the loss function is usually the mean-squared error (MSE)

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = \|\mathbf{y} - o(h_1(\mathbf{x}))\|^2$$

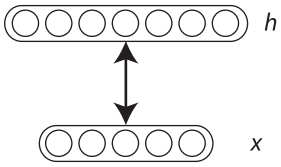


Figure 2.2: Graphical model of the restricted Boltzmann machine (RBM). Image reproduced from Bengio (2009b).

when the targets $\mathbf{y}$ are continuous unnormalized values. Combined with a linear output layer this amounts to predict the means of a gaussian. The loss function used for binomial probability vectors is the cross-entropy

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\mathbf{y}^T \log o(h_1(\mathbf{x})) - (1 - \mathbf{y}^T) \log(1 - o(h_1(\mathbf{x})))$$

where $\mathbf{y}$ is a binary vector. If the activation of the output layer is a softmax, $o(h_1(\mathbf{x}))$ will learn the conditional $p(\mathbf{y}_i = 1|\mathbf{x})$. The gradients for the output logistic regression is the same as the gradient for a regular logistic regression. The gradients for the hiddens layers are found using the chain rule (Rumelhart et al., 1986).

Deep neural nets are prone to overfitting because they can learn highly non-linear functions. An advance in regularization (Hinton et al., 2006a; Bengio et al., 2007a) has created a surge in interest for these models. Another cause of this resurgence is the appearance of fast graphical processing units (GPUs) that can run DNNs very fast.

2.2 Restricted Boltzmann machines

A restricted Boltzmann machine (RBM) is an undirected graphical model with binary variables (Hinton et al., 2006b): observed variables $\mathbf{x}$ and hidden variables $\mathbf{h}$. It is defined by the joint probability over $\{\mathbf{x}, \mathbf{h}\}$

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{\sum_{\mathbf{x}',\mathbf{h}'} e^{-E(\mathbf{x}',\mathbf{h}')}}$$

where the energy function E is given by

$$-E(\mathbf{x}, \mathbf{h}) = \mathbf{h}^T \mathbf{W}\mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{c}^T \mathbf{x}$$

with parameters $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$.

The Boltzmann machine is one of the first so-called generative models. The idea is to fit the parameters of the distribution $P(\mathbf{x}, \mathbf{h})$ so that they correspond to a real-world distribution of interest. We can then generate new examples from the model using sampling techniques like Gibbs sampling. It can also be used to answer questions about the distribution using its conditional distributions.

The distinguishing characteristic of the RBM is the presence of non-linear hidden units $\mathbf{h}$. It is these units that make the RBM one of the most powerful generative models (Salakhutdinov and Murray, 2008). The hidden units of the RBM can learn to represent hidden factors at play in the distribution. In images for example, it is clear that nearby pixels are related but the relation between distant pixels is quite complicated. A model that only incorporates interactions between visible variables would fail to capture these distant interactions. Instead, the RBM is able to detect edges and object parts as hidden factors that cause the activation of the pixels. These hidden factors can be thought of abstract conditions that are entangled to create what is captured by the visible units.

2.2.1 Conditionals

The restricted Boltzmann machine is distinguished from the Boltzmann machine by the lack of connections inside the set of visible units, and similarly for the hiddens. This gives the RBM closed-form conditionals which can be computed quite efficiently. This limitation does not prevent the RBM from being a universal approximator Le Roux and Bengio (2010).

The conditional distributions over observed and hidden variables are

$$P(h_j = 1|\mathbf{x}) = \sigma(\sum_i W_{ji} x_i + b_j)$$
$$P(v_i = 1|\mathbf{h}) = \sigma(\sum_j W_{ji} h_j + c_i)$$

with the logistic sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$. Computing both conditionals are in $O(mn)$

where m is the number of latent factors and n the number of visibles.

These conditionals are the base of inference in this model. We can use the RBM has a feature extractor through $P(\mathbf{h}|\mathbf{x})$. We can generate from the model by combining $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ with block Gibbs MCMC, described below.

2.2.2 Sampling

It is possible to sample new examples from the learned distribution $P(\mathbf{x})$. This is particularly useful for learning as we will see, but it also has practical applications Bengio et al. (2013). Samples are obtained by running a Markov chain to convergence. Typically the Gibbs sampling transition operator is used.

Gibbs sampling over a distribution $X = (X_1, \ldots, X_N)$ iterates over N sampling subsets of $X_i \sim P(X_i|X_{-i})$ where X_{-i} includes all the variables but the one at index i. This process is guaranteed to converge to the distribution on X (Bishop, 2006).

For the RBM, we can reduce the number of sampling sub-steps from N to 2 because there are two independent groups of units. We have for the visibles $P(x_i|\mathbf{x}_{-i}, \mathbf{h}) = P(x_i|\mathbf{h})$ and similarly for the hiddens. Sampling is done by alternating between $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$. This process is known as block Gibbs sampling because we are sampling blocks of variables at a time.

A step in the Markov chain is given by

$$\begin{aligned} \mathbf{h}^{t+1} &\sim P(\mathbf{h}|\mathbf{x}^t) \\ \mathbf{x}^{t+1} &\sim P(\mathbf{x}|\mathbf{h}^{t+1}) \end{aligned}$$

We can obtain a sample by iterating these steps until convergence which is guaranteed for $t = \infty$. In practice, a small number is steps is enough to reach the stationary distribution.

2.2.3 Learning

There are several principles that can be used to estimate the parameters of the RBM. These methods rely on different inductive principles, but the guiding principle is to match certain statistics of the target distribution and of the distribution of the model.

Stochastic maximum likelihood

The classic way of training this model is through maximum likelihood. This requires an expression for the likelihood of a sample. It can be found by marginalizing analytically over $\mathbf{h}$ to obtain

$$P(\mathbf{x}) = \frac{e^{-F(\mathbf{x})}}{\sum_{\mathbf{x}'} e^{-F(\mathbf{x}')}}$$

where the free energy F has the expression

$$-F(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \sum_j \log(1 + e^{\sum_i W_{ji}x_i + b_j})$$

Training the model is in principle as simple as following the gradient of the negative log-likelihood

$$-\frac{\partial \log P(\mathbf{x})}{\partial \theta} = E_{data}\left[\frac{\partial F(\mathbf{x})}{\partial \theta}\right] - E_{model}\left[\frac{\partial F(\mathbf{x})}{\partial \theta}\right]$$

However, this gradient is intractable because the second expectation is combinatorial. Stochastic Maximum Likelihood or SML (Younes, 1999; Tieleman, 2008) estimates this expectation using sample averages taken from a persistent MCMC chain (Tieleman, 2008). Starting from $\mathbf{x}^i$ a step in this chain is taken by sampling $\mathbf{h}^i \sim P(\mathbf{h}|\mathbf{x}^i)$, then we have $\mathbf{x}^{i+1} \sim P(\mathbf{x}|\mathbf{h}^i)$. SML-$k$ is the variant where k is the number of steps between parameter updates, with SML-1 being the simplest and most common choice, although better results (at greater computational expense) can be achieved with more steps.

Training the RBM using SML-1 is on the order of $O(dn)$ per update where d is the dimension of the input variables and n is the number of hidden variables. In the case of *high-dimensional sparse vectors* with p non-zeros, SML does not take advantage of the sparsity. More precisely, sampling $P(\mathbf{h}|\mathbf{x})$ (inference) can take advantage of sparsity and costs $O(pn)$ computations while "reconstruction", i.e., sampling from $P(\mathbf{x}|\mathbf{h})$ requires $O(dn)$ computations. Thus scaling to larger input sizes n yields a linear increase in training time even if the number of non-zeros p in the input remains constant.

Ratio matching

Ratio matching (Hyvärinen, 2007) is an estimation method for statistical models where the normalization constant is not known. It is similar to score matching (Hyvärinen, 2005) but applied on discrete data whereas score matching is limited to continuous inputs, and both are computationally simple and yield consistent estimators. Score matching estimates the parameters by matching the local directions of maximum likelihood near input points. The use of Ratio Matching in RBMs is of particular interest because their normalization constant is computationally intractable.

The core idea of ratio matching is to match ratios of probabilities between the data and the model. Thus Hyvärinen (2007) proposes to minimize the following objective function

$$P_x(\mathbf{x}) \sum_{i=1}^{d} \left[g\left(\frac{P_x(\mathbf{x})}{P_x(\bar{\mathbf{x}}^i)}\right) - g\left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)}\right)\right]^2 + \left[g\left(\frac{P_x(\bar{\mathbf{x}}^i)}{P_x(\mathbf{x})}\right) - g\left(\frac{P(\bar{\mathbf{x}}^i)}{P(\mathbf{x})}\right)\right]^2 \tag{2.1}$$

where P_x is the true probability distribution, P the distribution defined by the model, $g(x) = \frac{1}{1+x}$ is an activation function and $\bar{\mathbf{x}}^i = (x_1, x_2, \ldots, 1 - x_i, \ldots, x_d)$. In this form, we can see the similarity between score matching and ratio matching. The normalization constant is canceled because $\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} = \frac{e^{-F(\mathbf{x})}}{e^{-F(\bar{\mathbf{x}}^i)}}$, however this objective requires access to the true distribution P_x which is rarely available.

Hyvärinen (2007) shows that the Ratio Matching (RM) objective can be simplified up to constants in θ to

$$J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} \left(g\left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)}\right)\right)^2 \tag{2.2}$$

which does not require knowledge of the true distribution P_x. This objective can be described as ensuring that the training example $\mathbf{x}$ has the highest probability in the neighborhood of points at hamming distance 1.

In Chapter 6 we prove that one can rewrite Eq. 2.2 in a form reminiscent of auto-encoders:

$$J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} (x_i - P(x_i = 1 | \mathbf{x}_{-i}))^2. \tag{2.3}$$

This will be useful for reasoning about this estimator. The main difference with

auto-encoders is that each input variable is predicted by excluding it from the input.

Applying Equation 2.2 to the RBM we obtain $J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} \left(\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i))\right)^2$. The gradients have the familiar form

$$-\frac{\partial J_{RM}(\mathbf{x})}{\partial \theta} = \sum_{i=1}^{d} 2\eta_i \left[\frac{\partial F(\mathbf{x})}{\partial \theta} - \frac{\partial F(\bar{\mathbf{x}}^i)}{\partial \theta}\right] \tag{2.4}$$

with $\eta_i = \left(\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i))\right)^2 - \left(\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i))\right)^3$.

A naive implementation of this objective is $O(d^2 n)$ because it requires d computations of the free energy per example. This is much more expensive than SML, as noted by Marlin et al. (2010). Thankfully, as we argue here, it is possible to greatly reduce this complexity by reusing computation and taking advantage of the parametrization of RBMs. This can be done by saving the results of the computations $\alpha = \mathbf{c}^T\mathbf{x}$ and $\beta_j = \sum_i W_{ji}x_i + b_j$ when computing $F(\mathbf{x})$. The computation of $F(\bar{\mathbf{x}}^i)$ can be reduced to $O(n)$ with the formula $-F(\bar{\mathbf{x}}^i) = \alpha - (2x_i - 1)c_i + \sum_j \log(1 + e^{\beta_j - (2x_i - 1)W_{ji}})$. This implementation is $O(dn)$ which is the same complexity as SML. However, like SML, RM does not take advantage of sparsity in the input.

2.3 Regularized auto-encoders

A regularized auto-encoder (AE) is a feed-forward neural network trained to reconstruct its input. The original insight of the auto-encoder is that by rebuilding the input you use the input as the teaching signal in gradient descent. However, simply learning does not guarantee that the model will learn interesting patterns in the data. Classical auto-encoders limit the capacity by setting the number of latent factors below the dimension of the input, forming an *under-complete* representation. This restricted the model to learning a subspace of the principal components (Bourlard and Kamp, 1988). Recently, several models have been using different forms of regularization that allow learning richer features from the input (Vincent et al., 2010; Rifai et al., 2011). Notably these regularizers allow learning *over-complete* representations which have more features than inputs. Unlike RBMs the

AEs are not primarly motivated by probabilistic modelling. However, a recent line of papers Vincent (2011); Rifai et al. (2012); Alain and Bengio (2013) (one of which I've co-authored) has shown that regularized auto-encoders have a probabilistic interpretation.

2.3.1 Denoising auto-encoders

The DAE is a learning algorithm for unsupervised feature extraction Vincent et al. (2010): it is provided with a stochastically corrupted input and trained to reconstruct the original clean input. Its training criterion can be shown to relate to several training criteria for density models of the input through Score Matching Hyvärinen (2005); Vincent (2011). It is also possible to show that the denoising objective yields to learning about the data distribution by using local moments and learning the score (Alain and Bengio, 2013). Alain and Bengio (2013) shows that the difference vector between the reconstruction and the input is the model's guess as to the direction of greatest increase in the likelihood, whereas the difference vector between the noisy corrupted input and the clean original is nature's hint of a direction of greatest increase in likelihood (since a noisy version of a training example is very likely to have a much lower probability under the data generating distribution than the original). It can also be shown that the DAE is extracting a representation that tries to preserve as much as possible of the information in the input (Vincent et al., 2010).

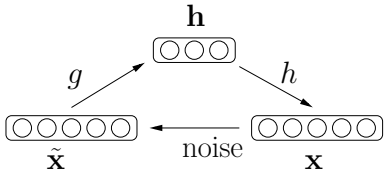


Figure 2.3: Schematic of the Denoising Auto-Encoder

The denoising auto-encoder reconstruction $f(\mathbf{x}) = h(g(\mathbf{x}))$ is composed of an encoder function $g(\cdot)$ and a decoder function $h(\cdot)$ (see Figure 2.3). During training, the input vector $\mathbf{x} \in [0,1]^d$ is partially and randomly corrupted into the vector $\tilde{\mathbf{x}}$. The encoder takes $\tilde{\mathbf{x}}$ and maps it into a hidden representation $\mathbf{h} \in [0,1]^{d'}$. The decoder takes the representation $\mathbf{h}$ and maps it back to a vector $\mathbf{z}$ in the input space ($[0,1]^d$ in our case). The DAE is trained to map a corrupted input $\tilde{\mathbf{x}}$ into the

original input $\mathbf{x}$ such that $g(h(\tilde{\mathbf{x}})) \approx \mathbf{x}$. This forces the code $\mathbf{h}$ to capture important and robust features of $\mathbf{x}$. Many corruption processes are possible, but they should have the property of generally producing *less plausible* examples. Typically, inputs are corrupted by randomly setting elements of $\mathbf{x}$ to 0 or 1, or adding Gaussian noise. The typical shallow encoder has the form

$$\begin{aligned} \mathbf{a}_1 &= \mathbf{W}^{(1)} \cdot \tilde{\mathbf{x}} + \mathbf{b}^{(1)} \\ \mathbf{h} &= s_1(\mathbf{a}_1) \end{aligned} \tag{2.5}$$

where s_a is a non-linear function like the sigmoid $s_a(u) = 1/(1 + \exp(-u))$, $\mathbf{W}^{(1)}$ is a $d' \times d$ weight matrix and $\mathbf{b}^{(1)}$ is a $d' \times 1$ vector. The function computed by the decoder is

$$\begin{aligned} \mathbf{a}_2 &= \mathbf{W}^{(2)} \cdot \mathbf{h} + \mathbf{b}^{(2)} \\ \mathbf{z} &= s_2(\mathbf{a}_2) \end{aligned} \tag{2.6}$$

Where $\mathbf{W}^{(2)}$ is a $d \times d'$ weight matrix and $\mathbf{b}^{(2)}$ is a $d \times 1$ vector.

Training

Given a dataset $D_n = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)})$, the parameters ($\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$) are trained by stochastic gradient descent to minimize a negative log-likelihood

$$\hat{R}(f, D_n) = \frac{1}{n} \sum_{i}^{n} - \log p(\mathbf{x}^{(i)} | f(\tilde{\mathbf{x}}^{(i)}) = \frac{1}{n} \sum_{i}^{n} d(\mathbf{x}^{(i)}, f(\tilde{\mathbf{x}}^{(i)}))$$

where d is a measure of the log-likelihood.

The measure that is typically used for binary vectors or vectors of binomial probabilities is the cross-entropy (or Bernouilli log-likelihood).

$$d(\mathbf{x}, \mathbf{z}) = -\sum_{k}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)]$$

The L2 distance is preferred when the input is continuous and without bounds because it corresponds to a Gaussian log-likelihood:

$$d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$$

The training updates of the denoising auto-encoder with a single layer neural net is in $O(mn)$ where m is the dimension of the input and n the number of hidden units.

The noise distribution is chosen through cross-validation. In most cases the binomial distribution will lead to better performance for binary vectors and the Gaussian will work better for continuous inputs. The level of the noise is an important hyper-parameter in this model. Theory (Alain and Bengio, 2013) suggests that the score is best estimated with a small noise but in practice the noise levels are quite high. Typical values for the probability the binomial corruption are between 0 and 1, and the optimal standard deviation when using gaussian noise is also usually found between 0 and 1.

Justification as a generative model

Vincent (2011) showed that a particular parametrization of the denoising auto-encoder is equivalent to the application of score matching of to a particular generative model. The connection was quite brittle but it planted the idea that the DAE could have a proper theoretical justification. The link was also interesting because it shed light on what is captured by the DAE to model the distribution. Score matching (Hyvärinen, 2005) is a parameter estimation method that relies on matching a local statistic called the score $\frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}}$ between the model distribution and the data distribution. The score indicates the direction of highest likelihood increase around the example. The link to score matching is evidence towards the idea that the simple and efficient denoising objective learns the directions of highest likelihoods.

Alain and Bengio (2013) establishes a more general link between denoising auto-encoders and generative models. Assuming only small noise and a mean squared error reconstruction, they show that after training the score is given by

$$\frac{\partial \log p(\mathbf{x})}{\partial \mathbf{x}} = \frac{r(\mathbf{x}) - \mathbf{x}}{\sigma^2} + O(\sigma^2)$$

The probability distribution can be recovered by integrating over the score. In practice this is inefficient, but recovering the distribution is not required to use the DAE has a feature extractor. Bengio et al. (2013) further generalized these results to arbitrary noise and arbitrary parametrization of the denoising auto-encoder.

2.3.2 Contractive auto-encoders

The contractive auto-encoder (Rifai et al., 2011) (CAE) is an unsupervised learning algorithm for feature extraction which uses a Tikhonov regularization (Tikhonov and Arsenin, 1977) on the learned features. The regularization helps achieve robust features that have been found to reach state-of-the-art performance on several benchmarks (Rifai et al., 2011, 2012). The CAE learns about the input distribution by capturing the local directions of variations around the input points. The CAE comprises an encoder function

$$\mathbf{h} = h(\mathbf{x}) = \sigma(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}^{(1)})$$

and a decoder function

$$\mathbf{z} = r(\mathbf{h}) = \sigma(\mathbf{W}^T \cdot \mathbf{h} + \mathbf{b}^{(2)}).$$

The distinctive characteristic of the CAE is that the loss combines the reconstruction error with a penalty on the Frobenius norm of the Jacobian of the hidden mapping

$$\hat{R}(f, D_n) = \frac{1}{n} \sum_{i}^{n} d(\mathbf{x}^{(i)}, r(h(\mathbf{x}^{(i)}))) + \lambda \left\| \frac{\partial h(\mathbf{x}^{(i)})}{\partial \mathbf{x}^{(i)}} \right\|^2 .$$

The hyper-parameter λ is to be cross-validated and optimal values are typically within 0 and 1. The training objective is in $O(mn)$ where m is the dimension of the input and n the number of hidden units.

The CAE captures information about the input through the two opposing forces in its loss function. The reconstruction is a term that guarantees that features conserve information about the input. It can be thought of as a soft way to ensure the encoder is a bijective mapping (Le et al., 2011), at least near the training examples. The Jacobian penalty encourages the model to be invariant and in the limit with $\lambda \to \infty$ it would force the model to learn a constant feature mapping. The Jacobian penalty is known as contractive and gives the model its name.

The Jacobian measures the variation of each hidden unit with respect to variations in the input $\frac{\partial h_i}{\partial x_i} = h_i(1 - h_i)W_{ij}$. Minizing this variation can be achieved either by saturating the hidden units (setting them close to 0 or 1) or reducing the norm of the weights. However, the CAE cannot simply reduce the norm of

the weights because the same weights are used for encoding and decoding. This is known as *tied* weights. The model must therefore learn to saturate and ignore certain variations in the input.

The contraction forces the model to discard as many directions of variations as possible. The reconstruction forces the model to keep the directions that occur in the data. The result is that the features $h_i(\mathbf{x})$ are only sensitive to directions in the input that actually occur in the data. (Rifai et al., 2011) has shown that these correpond to tangents of the data manifold. The tangents can be recovered by an eigen-decomposition of the Jacobian where the eigen-vectors with non-zero eigen-values are the local tangents at that point. As we will show in Chapter 8 this insight can be used to further regularize a deep neural network.

2.3.3 Links between auto-encoders and RBMs

RBMs have generally been preferred to auto-encoders because they have more theoretical justification. However, the auto-encoders are generally simpler to understand and implement. A simple look at the conditionals of the RBM and the encoder/decoder of the AEs suggest that the models are similar in some way. Elucidating the links between these two model families has been the subject of several papers Bengio and Delalleau (2009); Vincent (2011); Alain and Bengio (2013); Swersky et al. (2011).

Vincent (2010); Alain and Bengio (2013) have confirmed the intuition that certain auto-encoders capture the probability distribution through local statistics. This result is generalized by Bengio et al. (2013) who shows that auto-encoders with arbitrary noise and arbitrary parametrizations are generative models. This constitutes a very flexible framework for generative models.

Swersky et al. (2011) has shown that applying score matching to any parameterization of an RBM will lead to an auto-encoder. This encourages a new way to approach modeling with unsupervised models. The probabilistic framework of the RBM can be used to reason about the models but they can be implemented simply by transforming them into auto-encoder form using score matching.

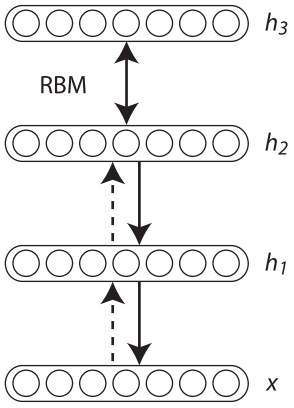


Figure 2.4: Graphical model of the deep belief network (DBN). Image reproduced from Bengio (2009b).

2.4 Stacking RBMs and AEs

The RBMs and AEs form basic learning modules that can be stacked to create deep architectures (Hinton et al., 2006a; Bengio et al., 2007a). Stacking these feature extractors will allow each layer to gradually learn more abstract features. Experiments have shown that these more abstract features can be useful on several classification problems (Le et al., 2012; Dahl et al., 2012).

Traditionally a feature extraction pipeline will include many layers of hand-crafted or off-the-shelf features. Popular off-the-shelf features for vision include SIFT (Lowe, 1999) for example. One ground-breaking idea pioneered by Hinton et al. (2006a); Bengio et al. (2007a) is to learn these features automatically using unsupervised data which is often easy to obtain. These features will have the advantage of being more tuned to the data and often the classification task. A paper which I've co-authored (Bengio et al., 2013) also shows deep algorithms lead to better mixing during sampling.

2.4.1 Deep belief nets

Stacking RBMs leads to a generative model called the deep belief net (DBN). The training algorithm is a greedy iterative procedure. The RBM at layer l is trained on pseudo-data sampled from the posterior $p(\mathbf{h}|\mathbf{x})$ of the model at layer

$l-1$. This results in the following generative model

$$p(v, h^{(1)}, \cdots, h^{(L)}) = \left[\prod_{l=1}^{L-1} p(h^{(l-1)}|h^{(l)})\right] p(h^{(L-1)}, h^{(L)}).$$

Surprisingly this model combines both directed and undirected connections even though the basic RBM is undirected (Figure 2.4). The last layer has undirected connections while the other layers are directed top-down. Exact inference in the model is intractable because of the depth and the presence of the directed connections. The features can be extracted by propagating the posterior over hiddens upward in the model. Hinton et al. (2006a) suggest using the last layer of the DBN as features or using the DBN to pretrain a deep neural network. In the case of pretraining, the parameters of the trained DBN are used as a starting point for the optimization of the DNN.

2.4.2 Stacked auto-encoders

Recent work (Bengio et al., 2007a; Vincent et al., 2010; Rifai et al., 2011) has shown that several regularized auto-encoders can take advantage of depth. The earliest work (Bengio et al., 2007a) generalizes the result of (Hinton et al., 2006a) with RBMs to classical auto-encoders. Even without regularization the classical auto-encoder trained with a greedy layer-wise fashion is able to reduce the error on MNIST from 2.4% to 1.4% Bengio et al. (2007a). In most work with auto-encoders (Bengio et al., 2007a; Vincent et al., 2010; Rifai et al., 2011) a greedy layer-wise scheme is used to obtain a deep architecture. This means training an auto-encoder at layer l on the representation learned by the auto-encoder at layer $l-1$. There has been interest for deep auto-encoders that are trained globally but this is still an open research area. A difficulty of this approach is that of optimizing such a deep auto-encoder.

2.5 Why does pretraining work?

Experiments have shown that pretraining helps mainly as a regularizer and as an aid to optimization to some extent (Erhan et al., 2010). As a regularization,

the prior enforced by pretraining is that the supervised task is based on the factors that explain some of the variations salient in the input data. In traditional machine learning, we assume the targets are directly related to the observations and we model this directly. In the case of pretraining for supervised tasks, we assume both the observations and the targets are caused by latent factors. In this framework, it makes sense to model the latent factors. Moreover, the targets are typically one-hot vectors which convey little information about the patterns to be detected. Learning the distribution of the observation has the benefit of being very rich in information because the targets are dense vectors. This allows the unsupervised model to quickly recover the latent factors (Erhan et al., 2010). When the parameters of the unsupervised model are used to initialize a neural network they set it up in a local basin of attraction. The pretraining process will saturate the hidden units making it more difficult to move far in optimization space. This will force the optimizer to seek a minimum that is close to the initialization (Erhan et al., 2010). Some evidence suggests that pretraining is also useful to optimization (Erhan et al., 2010). However, my experiments on datasets larger than those considered in (Erhan et al., 2010) have shown that pretraining does not help in these regimes (over 1 billion examples), which contradicts the notion that it helps optimization.

2.6 Beyond pretraining

Glorot et al. (2011a) has gone beyond the breakthrough of (Hinton et al., 2006a; Bengio et al., 2007a) by showing yet another way to train deep neural nets. It is a sizeable departure from both (Hinton et al., 2006a; Bengio et al., 2007a) because it does not rely on unsupervised pretraining in any way. This makes the approach simpler to use for practitioners and more applicable to very large-scale labeled datasets. Glorot et al. (2011a) shows that DNN with ReLU activation units can be trained without layer-wise pretraining. Hinton et al. (2012) further explores this direction using a new regularization. The idea is to set with probability p a random subset of the hidden units to 0 during the training of the DNN. At test time, the parameters are divided by p as a correction. This procedure helps to fight co-adaptation in the hidden units. Another explanation proposed by (Hinton et al., 2012) is that dropout is an efficient way to do bagging with an exponential

amount of networks with shared parameters. An element of this bag corresponds to a network with a particular subset of its units *dropped-out*. Dividing by p can be thought of as a crude way to average over all possible networks. Several other works have started exploring alternative ways to train deep networks without layer-wise pretraining (Goodfellow, 2013; Wang and Manning, 2013; Zeiler and Fergus, 2013).

Another possible way beyond pretraining is joint optimization of a discriminative generative model (**?**). This approach relies on optimizing an objective related to the pseudo-likelihood of a deep Boltzmann machine (Salakhutdinov and Hinton, 2009a). This would eliminate the cumbersome pretraining phase by combining it with supervised finetuning. Another advantage is that it would allow the features at different layers to learn to cooperate. The regularization occurs by forcing the hidden features to generate both the labels and the observations. Rasmus et al. (2014) has shown impressive results using a related approach, reaching 0.60% on MNIST.

In Chapter 8 we propose a geometric way to regularize deep neural networks that does not require pretraining. The idea is to penalize the Lipshitz constant of a deep neural network, thus leveraging the basic idea of smoothness to achieve generalization.

2.7 Challenges

There are numerous open questions in deep learning. One significant question is *what makes a good representation?* There have been many serious attempts at answering this question experimentally (Bengio et al., 2007a; Vincent et al., 2010; Rifai et al., 2011). Experimental evidence has shown that the specific encoding for the representation may be as important as the principle used to learn it (Coates and Ng, 2011). Another important issue is *how to learn tasks where examples cannot easily be represented by vectors*. This will be crucial in creating systems that can properly handle natural language. Unlike images, sentences cannot easily be represented by a fixed vector because they have variable length. New approaches Socher et al. (2011); Sutskever (2012) have recently been proposed but it remains an open problem. The question that is the focus of this document is *how to scale these algorithms to large-scale problems?* On one hand, the unsupervised learning

algorithms used in deep learning do not scale well with input size. This limits the application of deep learning in natural language processing where the inputs can be quite large. On the other hand, training networks with large hidden representations is not well studied. Anecdotal evidence suggests diminishing returns for larger networks.

3 Prologue to first article

3.1 Article Detail

Large-scale learning of embeddings with reconstruction sampling. Yann N. Dauphin, Xavier Glorot, Yoshua Bengio. Proceedings of the *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*.

Personal Contribution.

Yoshua Bengio proposed the original idea that led to this paper. I was responsible for the practical implementation of the method. This required implementing numerous missing sparse operators in the Theano library (Bastien et al., 2012). Xavier Glorot ran the experiments on one dataset, while I ran the experiments on a second dataset. Xavier Glorot and I collaborated closely to tune the algorithm to obtain good results. I contributed heavily to the writing, with Yoshua Bengio writing the introductory sections.

3.2 Context

This paper was motivated by removing a roadblock for denoising auto-encoders in natural language processing. The deep learning breakthrough of Hinton and Salakhutdinov (2006) had shown that using unsupervised algorithms to pretrain deep neural nets led to superior results. This observation had been applied to vision problems (Lee et al., 2009) and speech recognition (Dahl et al., 2010). In comparison, deep learning were not generating as much interest and breakthroughs in natural language processing. One drawback to unsupervised learning methods used in deep learning is that they were not capable of leveraging sparsity. This is crucial for applications to text because they often deal with large sparse vectors.

It appeared that solving this problem could allow much wider use of deep learning in this context.

3.3 Contributions

This paper allowed training denoising auto-encoders with very large sparse inputs. While the method is approximate, we demonstrate that it does not degrade the quality of the mode significantly. We observed that speed-ups of over an order of magnitude could be obtained without degradation in our experiments.

We also contributed experiments demonstrating the effectiveness of deep learning for natural language processing. We achieved two state-of-the-art results in this paper. Following this, Glorot et al. (2011c) noted that the use of reconstruction sampling allowed using a larger vocabulary and thus allowed better performance on an important sentiment classification benchmark. The method was also used to win the Unsupervised Learning and Transfer Learning Challenge (Mesnil et al., 2012). The task here was to learn representations for text that supported good transfer between domains. Finally, the method has helped garner more interest in using unsupervised methods with textual data (Deoras and Sarikaya, 2013; Lauly et al., 2014; Vincent, 2014).

4 Scaling DAEs to high-dimensional sparse inputs with importance sampling

In recent years, there has been a surge of interest for unsupervised representation learning algorithms, often for the purpose of building deep hierarchies of features [1]. See Bengio (2009b) for a recent review of Deep Learning algorithms, which are based on unsupervised learning of representations, one layer at a time, in order to build more abstract higher-level representations by the composition of lower-level ones. These representations are often used as input for classifiers, and measuring classification error is a good way, also chosen here, for evaluating the usefulness of these representations. One problem with these unsupervised feature learning approaches is that they often require computing a mapping from the learned representation back into the input space, e.g., either to reconstruct the input, denoise it, or stochastically generate it. Consider learning tasks where the input space is huge and sparse, as in many Natural Language Processing (NLP) tasks. In that case, computing the representation of the input vector is very cheap because one only needs to visit the non-zero entries of the input vector, i.e., multiply a very large dense matrix by a very sparse vector. However, **reconstructing a huge sparse vector** involves computing values for all the elements of that vector, and this can be much more expensive. For example with a bag-of-words representation of a 100-word paragraph and a vocabulary size of 100,000 words, computing the reconstruction from the representation is 1000 times more expensive than computing the representation itself.

The **main contribution** of this work starts from a very simple idea: train to **reconstruct only the non-zeros and a random subset of the zeros.** This introduces a bias in the reconstruction error (giving more weight to non-zeros than to zeros), which can be potentially beneficial or detrimental, but that can be corrected by a reweighting of error terms. The idea has also been refined in the context of the Denoising Auto-encoder, used for unsupervised learning of the

1. see NIPS'2010 Workshop on Deep Learning and Unsupervised Feature Learning, http://deeplearningworkshopnips2010.wordpress.com/

embeddings in our experiments. Instead of focusing only on the non-zeros of the uncorrupted input, we include also the non-zeros of the corrupted input, in order to sample the inputs on which the error is most likely to be large (since this minimizes the variance of our sampling-based estimator).

4.1 Related Work

There has been much previous work on learning embeddings for NLP. See Bengio (2008) for a review in the context of neural-network based models, which are related to the approach described here. The foundation of these ideas is the connectionist idea of distributed representations, even for symbolic data Hinton (1986). In NLP, this idea has been explored in particular using linear embeddings of words, e.g., with Latent Semantic Indexing (LSI) and related approaches Deerwester et al. (1990); Schutze (1993). Keep in mind that this is essentially based on finding principal components of the co-occurrence matrix (words co-occurring in the same document), and that a linear auto-encoder with squared reconstruction error with a bag-of-words representation would find the same representation as LSI. The combination of connectionist ideas and NLP was first explored at the character level, e.g. Miikkulainen and Dyer (1991), and later for words and language models Bengio et al. (2001) with a neural probabilistic language model. S of such models. Bengio et al. (2001) showed that these models could substantially improve on models based purely on n-grams in terms of perplexity, while Schwenk and Gauvain (2002) showed how to exploit these language models in state-of-the-art speech recognition systems. A core computational limitation of these models is that the neural network prediction (e.g., of the next word given previous words) consists of a probability for each word in the vocabulary, which makes computation scale with vocabulary size. In early work, this was addressed by limiting the vocabulary of the predicted words (and possibly using a cheaper predictor such as n-grams for the other ones).

In order to address this computational limitation and scale to larger vocabularies and larger datasets, two kinds of approaches were introduced in the past: using a tree structure for the predictions, or using sampling to visit only a few of the possible words. The approach introduced here is of the second kind. Tree-structured predictors are based on learning a class hierarchy and require only visiting the path

from the root to the leaf corresponding to the observed word Morin and Bengio (2005); Mnih and Hinton (2009). Sampling-based algorithms rely on stochastic approximations of the gradient which only require to compute the prediction on a small subset of the words Bengio and Sénécal (2003, 2008); Collobert and Weston (2008). Whereas the early attempts Bengio and Sénécal (2003, 2008) are focused on correctly estimating conditional probabilities (for the next word), Collobert and Weston (2008) only try to *rank* the words, with a criterion that can be written as a sum over words (comparing the score of the observed word with the score of any other word). This sum can be estimated by a Monte-Carlo sample. This works even with a single sample in the context of stochastic gradient descent, where we do a very large number of stochastic updates but each of them is small, hence averaging out much of the sampling noise. Whereas all these focused on predicting the next word, we focus here on reconstructing an input bag-of-words, or more generally a very sparse high-dimensional vector, since this kind of reconstruction is a basic requirement for many Deep Learning algorithms. We are not trying to predict word probabilities, only to learn good embeddings (which are used as part of a classifier) so we do not really need the reconstruction outputs to be calibrated probabilities.

4.2 Denoising Auto-Encoders

4.2.1 Introduction

In this paper we have applied the proposed idea of sampling reconstructions in the context of the Denoising Auto-Encoder (DAE) as the building block for training deep architectures, because our preliminary experiments found that a particular form of DAE surpassed the state-of-the-art in a text categorization task of sentiment analysis Glorot et al. (2011b). The DAE is a learning algorithm for unsupervised feature extraction Vincent et al. (2008): it is provided with a stochastically corrupted input and trained to reconstruct the original clean input. Its training criterion can be shown to relate to several training criteria for density models of the input, either via bounds Vincent et al. (2008) or through Score Matching Hyvärinen (2005); Vincent (2010). Intuitively, the difference vector between the reconstruction and the input is the model's guess as to the direction of greatest increase in the

likelihood, whereas the difference vector between the noisy corrupted input and the clean original is nature's hint of a direction of greatest increase in likelihood (since a noisy version of a training example is very likely to have a much lower probability under the data generating distribution than the original). It can also be shown that the DAE is extracting a representation that tries to preserve as much as possible of the information in the input Vincent et al. (2008).

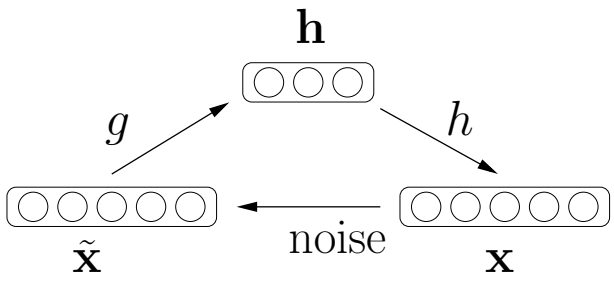


Figure 4.1: Schematic of the Denoising Auto-Encoder

The Denoising Auto-Encoder reconstruction $f(\mathbf{x}) = h(g(\mathbf{x}))$ is composed of an encoder function $g(\cdot)$ and a decoder function $h(\cdot)$ (see Figure 4.1). During training, the input vector $\mathbf{x} \in [0,1]^{d_x}$ is partially and randomly corrupted into the vector $\tilde{\mathbf{x}}$. The encoder takes $\tilde{\mathbf{x}}$ and maps it into a hidden representation $\mathbf{h} \in [0,1]^{d_h}$. The decoder takes the representation $\mathbf{h}$ and maps it back to a vector $\mathbf{z}$ in the input space ($[0,1]^{d_x}$ in our case). The DAE is trained to map a corrupted input $\tilde{\mathbf{x}}$ into the original input $\mathbf{x}$ such that $g(h(\tilde{\mathbf{x}})) \approx \mathbf{x}$. This forces the code $\mathbf{h}$ to capture important and robust features of $\mathbf{x}$. Many corruption processes are possible, but they should have the property of generally producing *less plausible* examples. Typically, inputs are corrupted by randomly setting elements of $\mathbf{x}$ to 0 or 1, or adding Gaussian noise. The encoder function used in Vincent et al. (2008) is

$$\begin{aligned} \mathbf{a}_1 &= \mathbf{W}^{(1)}\tilde{\mathbf{x}} + \mathbf{b}^{(1)} \\ \mathbf{h} &= s_1(\mathbf{a}_1) \end{aligned} \tag{4.1}$$

where s_a is a non-linear function like the sigmoid $s_a(u) = 1/(1+\exp(-u))$, $\mathbf{W}^{(1)}$ is a $d_h \times d_x$ weight matrix and $\mathbf{b}^{(1)}$ is a $d_h \times 1$ vector. In Vincent et al. (2008) the

function computed by the decoder is

$$\begin{aligned} \mathbf{a}_2 &= \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \\ \mathbf{z} &= s_2(\mathbf{a}_2) \end{aligned} \tag{4.2}$$

Where $\mathbf{W}^{(2)}$ is a $d_x \times d_h$ weight matrix and $\mathbf{b}^{(2)}$ is a $d_x \times 1$ vector.

4.2.2 Training

Given a dataset $D_n = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)})$, the parameters ($\mathbf{W}^{(1)}$, $\mathbf{b}^{(1)}$, $\mathbf{W}^{(2)}$, $\mathbf{b}^{(2)}$) are trained by stochastic gradient descent, following Vincent et al. (2008), to minimize the cross-entropy

$$\begin{aligned} \hat{R}(f, D_n) &= \frac{1}{n}\sum_{i}^{n} L(\mathbf{x}^{(i)}, f(\tilde{\mathbf{x}}^{(i)})) \\ L(\mathbf{x}, \mathbf{z}) &= \sum_{k}^{d} H(\mathbf{x}_k, \mathbf{z}_k) \\ &= \sum_{k}^{d} -[\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k)\log(1 - \mathbf{z}_k)] \end{aligned}$$

where H is the cross-entropy, $\mathbf{x}$ and $\mathbf{z}$ are considered as *vectors of binomial probabilities*.

4.2.3 Motivation

The Denoising Auto-Encoder can learn a representation from unlabeled data, but it can be later fine-tuned using labeled data. The ability to exploit large quantities of unlabeled data is very important because labeled data are usually scarce. Obtaining labeled data usually requires paying for the manual labeling of unlabeled samples. Furthermore, in the context of Natural Language Processing, the World Wide Web is a gold mine of unlabeled data. In contrast, using a purely supervised training approach (i.e. SVMs, CRFs) can only exploit the scarce labeled data.

The hypothesis that has been confirmed earlier for specific datasets Vincent et al. (2008) is that the representation $\mathbf{h}$ learned by the DAE makes the statistical structure of the input clearer, in the sense that it can be advantageous for

initializing a supervised classifier. It has been shown that Auto-Encoders (especially deep ones) go beyond Principal Component Analysis (PCA) by capturing multi-modal interactions in the input distribution Japkowicz et al. (2000); Hinton and Salakhutdinov (2006). In other words, the encoder learns to project $\mathbf{x}$ into a space $\mathbf{h}$ where the original factors of variation of the data tend to be better separated. Experimental results show that using $\mathbf{h}$ instead of $\mathbf{x}$ as input to a classifier can significantly help achieve better generalization Erhan et al. (2010); Larochelle et al. (2007).

4.3 Scaling the Denoising Auto-Encoder

4.3.1 Challenges

The dot products involved in the training of the DAE are expensive. The computations involved in g, h, the gradient ∇g through g, and the gradient ∇h through h are all in $O(d_x \times d_h)$, where d_x is the size of the sparse input vector and d_h is the size of the representation code.

This is problematic in the context of Natural Language Processing because the desired input size d_x may be in the millions.

4.3.2 Scaling the Encoder: Sparse Dot Product

We can take advantage of sparsity in any dot product $\mathbf{u} \cdot \mathbf{v}$ because the null elements $\mathbf{u}_i$ or $\mathbf{v}_i$ do not influence the result. This is also true for the matrix-vector product. Therefore, we can reduce the cost of the dot product by computing only the operations linked to non-zero elements, i.e., $g \in O(d_{NNZ} \times d_h)$, where d_{NNZ} is the number of non-zero elements in $\mathbf{x}$. The theoretical speed-up would be ${}^{d_x}/_{d_{NNZ}}$. This also applies to the gradient with respect to $\mathbf{W}^{(1)}$ (${}^{\partial \hat{R}}/_{\partial \mathbf{W}^{(1)}} = {}^{\partial \hat{R}}/_{\partial \mathbf{a}_1}\mathbf{x}'$). In practice, the speed-up is smaller because working with dense vector and matrix multiplications can be done more efficiently on modern computers, i.e., there is an overhead for handling sparse vectors. In our experiments we have found the overhead to be on the order of 50%. On the other hand, the biggest loss in comparison to a dense implementation comes from losing the use of BLAS'

optimized matrix-matrix product (GEMM), when training the model by showing one minibatch at a time (e.g. 10 in our experiments). The speedup from the dense matrix-matrix multiplication is on the order of 3 in our experiments, hence for the sparse computation to be advantageous, the sparsity level must be high enough to compensate for these two disadvantages).

4.3.3 Scaling the Decoder: Reconstruction Sampling

We introduce *reconstruction sampling* to make the decoder scalable. The idea is to calculate the reconstruction cost L based only on a sub-sample of the input units:

$$\hat{L}(\mathbf{x}, \mathbf{z}) = \sum_{k}^{d} \frac{\hat{\mathbf{p}}_k}{\mathbf{q}_k} H(\mathbf{x}_k, \mathbf{z}_k)$$

where we introduce $\hat{\mathbf{p}} \in \{0, 1\}^{d_x}$ with $\hat{\mathbf{p}} \sim P(\hat{\mathbf{p}}|\mathbf{x})$, and scalar weights $1/\mathbf{q}$. The sampling pattern $\hat{\mathbf{p}}$ controls whether a given input unit will participate in the learning objective for this presentation of the example $\mathbf{x}$. If training iterates through examples in the training set, the next time $\mathbf{x}$ is seen again, a different pattern $\hat{\mathbf{p}}$ may be sampled. In this paper, we have found that an effective sampling procedure is to choose $P(\hat{\mathbf{p}}|\mathbf{x})$ to reconstruct all non-zero inputs and a set of randomly chosen zero inputs. The average number of sampled units is defined as d_{SMP}.

The scalar weights $1/\mathbf{q}_k$ allow us to compensate for non-uniform choices of the sampling probabilities for the corresponding binary random variables $\hat{\mathbf{p}}_k$. If $\mathbf{q}_k = E[\hat{\mathbf{p}}_k|k, \mathbf{x}, \tilde{\mathbf{x}}]$, then this is an **importance sampling** scheme, i.e., the expected cost is guaranteed to be unchanged by the sampling procedure since $E[\frac{\hat{\mathbf{p}}_k}{\mathbf{q}_k}|k, \mathbf{x}, \tilde{\mathbf{x}}] = 1$. This is related to but different from Bengio and Sénécal (2003), in which $\mathbf{x}$ is a one-hot vector indicating what is the next word, and there is less information about which bits it matters most to sample.

We propose $\hat{L}$ instead of L as a (stochastic) training objective because it can be computed more efficiently, and we empirically find that it yields similar solutions for the same number of training updates. If $\mathbf{p}_k = 0$, then $\mathbf{z}_k$ need not be computed since it does not influence the cost $\hat{L}$. Calculating each $\mathbf{z}_k = s_2(\mathbf{W}_k^{(2)}\mathbf{h} + \mathbf{b}_k^{(2)})$ is on the order of $O(d_h)$. Therefore, computing only the units $\mathbf{z}_k$ that are sampled yields

$$h \in O(d_{SMP} \times d_h)$$

with the expected speed-up of ${}^{d_x}/_{d_{SMP}}$.

The gradients for $\hat{L}$ can also be calculated more efficiently. The gradient for the elements $\mathbf{z}_k$ where $\mathbf{p}_k = 0$ is null, so $\frac{\partial \hat{R}}{\partial \mathbf{z}}$ contains only d_{SMP} non-zero values. We calculate the gradients using the sparse dot product presented in section 4.3.2:

$$\begin{aligned} \frac{\partial \hat{R}}{\partial \mathbf{W}^{(2)}} &= \frac{\partial \hat{R}}{\partial \mathbf{a}_2}\mathbf{h}' \\ \frac{\partial \hat{R}}{\partial \mathbf{h}} &= \mathbf{W}^{(2)\prime}\frac{\partial \hat{R}}{\partial \mathbf{a}_2} \end{aligned}$$

The speed-up for both operations is on the order of ${}^{d_x}/_{d_{SMP}}$.

Sampling probabilities The optimal sampling probabilities $P(\hat{\mathbf{p}}|\mathbf{x})$ are those that yield the minimum variance of the estimator (under the assumption that we choose the weights $1/\mathbf{q}$ in order to get an unbiased estimator), since the total error of the sampling-based estimator is variance plus bias squared (and we are setting the bias to 0). Like for importance sampling, the minimum variance is achieved when $P(\hat{\mathbf{p}}|\mathbf{x})$ is proportional to the absolute value of the original distribution (uniform here) times the integrand, which here is just the reconstruction loss. Hence we should ideally pick those bits k on which the model is most likely to make a large error, but of course we do not know that before we sample which ones to reconstruct. The heuristic we propose is to always pick those bits k on which either $\mathbf{x}_k = 1$ or $\tilde{\mathbf{x}}_k = 1$, and to pick the same number of bits randomly from the remainder. Let $\mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}}) = \{k : \mathbf{x}_k = 1 \text{ or } \tilde{\mathbf{x}}_k = 1\}$. Then we choose to reconstruct unit k with probability

$$P(\hat{\mathbf{p}}_k = 1|\mathbf{x}_k) = \begin{cases} 1 & \text{if } k \in \mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}}) \\ |\mathcal{C}(\mathbf{x}, \tilde{\mathbf{x}})|/d_x & otherwise \end{cases} \tag{4.3}$$

The motivation for this heuristic is that because of the input sparsity, the 1's tend to come more as a surprise than 0's, and hence yield a larger reconstruction error. Regarding the cases where the auto-encoder input $\mathbf{x}_k = 1$ when $\mathbf{x}_k \neq 1$, these also tend to yield large errors, because the auto-encoder has to uncover the fact that those bits were flipped due to the corruption process, and cannot just copy them from the input. A smaller-variance estimator could probably be obtained by numerically estimating the average error associated to different bits depending on whether or not it is a 1 or a 0 in $\mathbf{x}_k$ and $\tilde{\mathbf{x}}_k$, but we have found that with

this simple scheme we achiExperimental Results on Amazon (small set)eve the same accuracy curve (as a function of number of updates) as with the dense (not sampled) training scheme, hence there is not much room left for improvement. In fact, it is questionable whether perfectly unbiasing the sampling scheme (i.e. choosing corrections $\mathbf{q}_k = P(\hat{\mathbf{p}}_k = 1)$) is what most helps produce the most useful embeddings, e.g., as measured by classification error from the learned intermediate features, on a predictive task of interest. For example, it could be argued that in the case of sparse input vectors, the non-zero inputs provide more important information and that error on them should be penalized more, which would argue in favor of choosing weights $1/\mathbf{q}_k$ constant (e.g. 1). We therefore experiment with both the unbiased scheme (eq. 6.7) and a biased scheme ($\mathbf{q}_k = 1$).

4.4 Implementation

4.4.1 Encoder

We implement the encoder as:

$$\mathbf{h} = s_1(\texttt{SparseDot}_{CSR}(\tilde{\mathbf{x}}, \mathbf{W}^{(1)}) + \mathbf{b}^{(1)})$$

$\texttt{SparseDot}_{CSR}(\mathbf{A}, \mathbf{B}) = \mathbf{AB}$. Note the operation is transposed compared to equation 4.1. In this setting, $\mathbf{W}^{(1)}$ is a $d_x \times d_h$, $\mathbf{b}^{(1)}$ is $1 \times d_h$. $\texttt{SparseDot}_{CSR}$ is more efficient when the sparse operand appears first. The input $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are stored in Compressed Sparse Row (CSR) format.

The gradient is given by:

$$\frac{\partial \hat{R}}{\partial \mathbf{W}^{(1)}} = \texttt{SparseDot}_{CSC}(\mathbf{x}', \frac{\partial \hat{R}}{\partial \mathbf{a}_1})$$

Where $\mathbf{x}^T$ is in Compressed Sparse Column (CSC) format.

For reference, the implementation of $\texttt{SparseDot}_{CSR}$ is given in Algorithm 1. The implementation of $\texttt{SparseDot}_{CSC}$ is similar. $\texttt{NON-ZERO-INDICES}(\mathbf{u})$ returns the set of non-zero indices in the row vector $\mathbf{u}$. $\texttt{AXPY}(\alpha, \mathbf{A}, \mathbf{B}) = \alpha\mathbf{A} + \mathbf{B}$ is part of the BLAS programming interface Lawson et al. (1979).

In our experiments, `AXPY` is provided by the highly optimized Goto BLAS Goto and Geijn (2008). Note that this is an important optimization. Typical implementations of Auto-Encoders rely on BLAS for their dot products, our operations must also leverage BLAS to be competitive.

Algorithm 1 $\texttt{SparseDot}_{CSR}(\mathbf{A}, \mathbf{B})$

Input: $\mathbf{A} = [A_{ij}]_{M \times K}$, $\mathbf{B} = [B_{ij}]_{K \times N}$
Output: $\mathbf{C} = [C_{ij}]_{M \times N}$

for $m = 1$ **to** M **do**
  for all $k \in \texttt{NON-ZERO-INDICES}(\mathbf{x}_m)$ **do**
    $\mathbf{C}_m \leftarrow \texttt{AXPY}(\mathbf{A}_{mk}, \mathbf{B}_k, \mathbf{C}_m)$
  end for
end for

4.4.2 Decoder

The decoder is implemented as:

$$\mathbf{z} = s_2(\texttt{SamplingDot}(\mathbf{h}, \mathbf{W}^{(2)}, \hat{\mathbf{p}}) + \mathbf{b}^{(2)})$$

$\texttt{SamplingDot}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ outputs $\mathbf{C} \circ (\mathbf{AB})$, where $\circ$ is element-wise multiplication. In comparison with equation 4.2, the operations are transposed and $\mathbf{W}^{(2)}$ is supposed to be $d_h \times d_x$ while $\mathbf{b}^{(2)}$ is $1 \times d_x$.

The key differences between `SamplingDot` and $\mathbf{C} \circ (\mathbf{AB})$ are:

1. The values in $\mathbf{AB}$ set to 0 by the element-wise product with $\mathbf{C}$ are not calculated.
2. The $\mathbf{B}$ matrix is expected to be $d_x \times d_h$ instead of $d_h \times d_x$. In other words, `SamplingDot` assumes $\mathbf{B}$ is transposed. This allows cache-friendly traversal of that matrix. This is especially important because in our setting $\mathbf{B}$ is a huge matrix.

$\mathbf{W}^{(2)}$ is stored as $d_x \times d_h$ instead of $d_h \times d_x$.

The gradients are calculated as:

$$\begin{aligned}\frac{\partial \hat{R}}{\partial \mathbf{W}^{(2)}} &= \texttt{SparseDot}_{Dense}(\frac{\partial \hat{R}}{\partial \mathbf{a}_2}', \mathbf{h}) \\ \frac{\partial \hat{R}}{\partial \mathbf{h}} &= \texttt{SparseDot}_{Dense}(\frac{\partial \hat{R}}{\partial \mathbf{a}_2}, \mathbf{W}^{(2)})\end{aligned}$$

$\texttt{SparseDot}_{Dense}$ calculates the dot product between a sparse matrix represented in dense format and a dense matrix. As explained in section 4.3.3, $\frac{\partial \hat{R}}{\partial \mathbf{a}_2}$ contains very few non-zero elements. It isn't converted into a sparse representation because the conversion is expensive and would have to be performed for each training update.

The implementation for `SamplingDot` is given in Algorithm 2. The implementation of $\texttt{SparseDot}_{Dense}$ is similar to Algorithm 1. $\texttt{DOT}(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$ is the vector dot product. It is part of the BLAS programming interface and is implemented by Goto BLAS in our experiments.

Algorithm 2 `SamplingDot`($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$)

Input: $\mathbf{A} = [A_{ij}]_{M \times K}$, $\mathbf{B} = [B_{ij}]_{N \times K}$, $\mathbf{C} = [C_{ij}]_{M \times N}$
Output: $\mathbf{D} = [D_{ij}]_{M \times N}$

for $m = 1$ **to** M **do**
  for $n = 1$ **to** N **do**
    if $\mathbf{C}_{mn} \neq 0$ **then**
      $\mathbf{D}_m \leftarrow \texttt{DOT}(\mathbf{A}_m, \mathbf{B}_n)$
    end if
  end for
end for

4.5 Experiments

We perform two sets of experiments on two popular NLP datasets. First, we show the properties and effectiveness of our approach in a setting where we can compare with the non-sampled version of the training algorithm for DAEs. Second, we train large-scale models on the Amazon dataset.

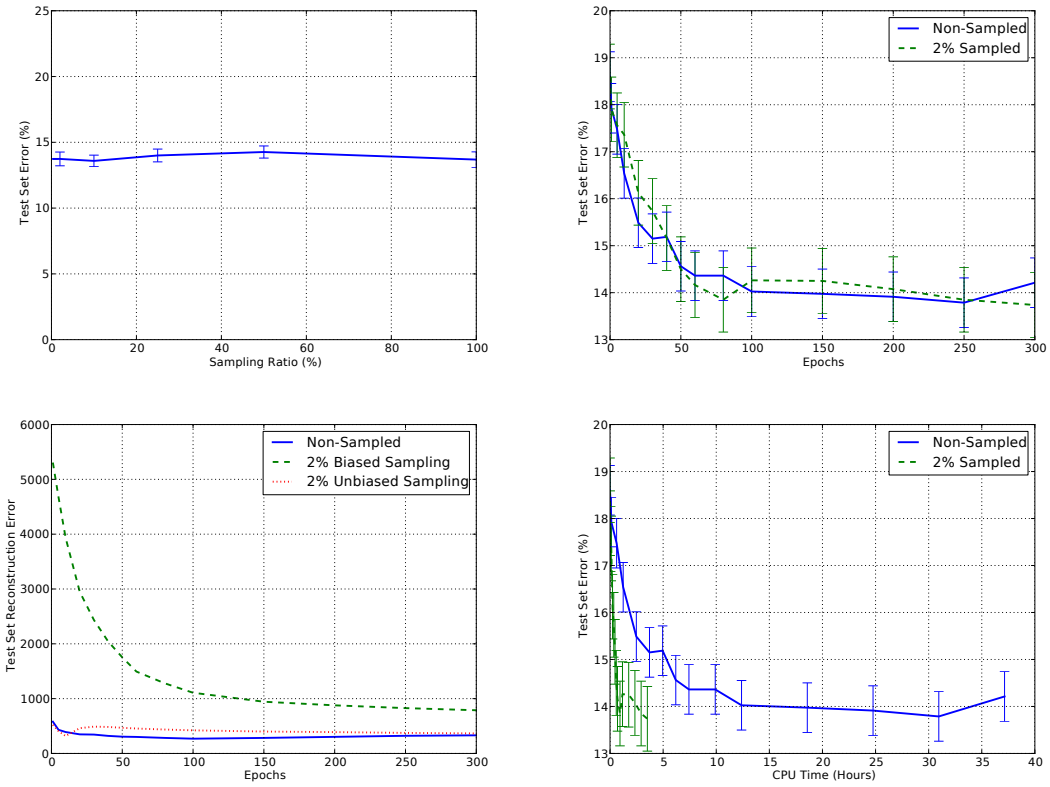


Figure 4.2: Experimental Results on Amazon (small set). Increasing the sampling approximation does not hurt classification error, but yields a 10.5x speedup. The biased estimator gets a worse reconstruction error, but not the unbiased one, and both convergence curves (in terms of training epochs) are similar for all models.

Amazon Multi-Domain Sentiment Dataset. Sentiment analysis aims to determine the judgment of a writer given a textual comment. We investigate our reconstruction sampling method on the Amazon sentiment analysis data set, introduced by Blitzer et al. (2007). It proposes a collection of more than $340,000$ product reviews on 25 different domains. For tractability, a smaller and more controlled dataset has been released, containing four different domains, with 1000 positives and 1000 negatives examples for each domain and a few thousands of unlabeled data. Our experiments will be conducted on both versions, we will refer to this last version as "small Amazon" and to the complete set as "full Amazon."

Reuters Corpus Volume I (RCV1) is a popular benchmark for document classification Lewis et al. (2004). It consists of over 800,000 real-world news wire stories

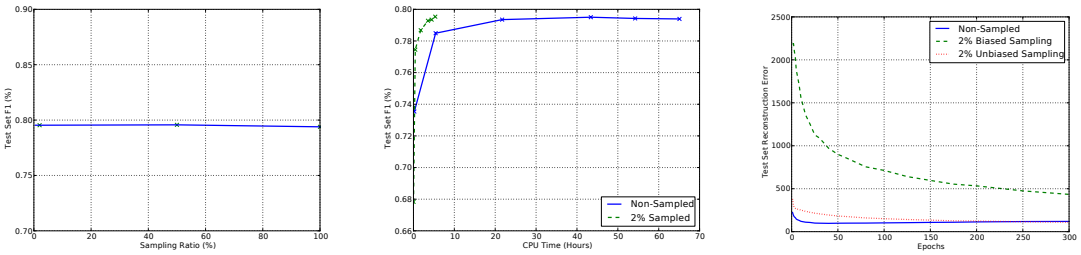


Figure 4.3: Experimental Results on RCV1. Increasing the sampling approximation does not hurt test F1, but yields a 12x speedup. The biased estimator gets a worse reconstruction error, but not the unbiased one.

represented in bag-of-words vectors with 47,236 dimensions. The dataset is split into a training set with 23,149 documents and a test set with 781,265 documents. The are three categories of labels to predict: Topics, Industries and Regions. There are 103 non-mutually exclusive topics. We focus our experiment on predicting the topics of documents. As proposed by Lewis et al. (2004) the performance measure is the $F_{1.0}$ over the test set.

Training Methodology. In our experiments, we perform unsupervised feature extraction using DAEs and we use these features as input to classifier. On the Amazon dataset, we train linear SVMs Fan et al. (2008). On the RCV1 dataset, a logistic regression is used.

We train a set of baseline DAEs that perform no sampling as well as a set of DAEs that have multiple levels of sampling. On the small Amazon and RCV1 dataset, we reduce the vocabulary to the 5000 most frequent input tokens in order to make the training of the baseline practical. On the full Amazon dataset, we kept 25,000 dimensions.

DAEs are trained with a minibatch size of 10. We reserve 10% of the training set of each dataset as a validation set. All hyper-parameters, for the DAEs and the classifiers, are chosen based on the performance on the validation set. We monitor validation and test error at different training epochs.

On the Amazon datasets we train linear SVMs for sentiment classification on different domains (4 on the small Amazon and 7 on the large scale Amazon), The reported value is the averaged test error and its standard deviation across domains.

The experiments are run on a cluster of computers with a double quad-core

Intel(R) Xeon(R) CPU E5345@2.33GHz with 8Gb of RAM.

Results. The kinds of embeddings learned on the Amazon data is shown in Figure 4.5, where the learned representations are non-linearly mapped to 2 dimensions by t-SNE van der Maaten and Hinton (2008).

Sampling has no effect on the quality of the representation learned. In Figure 4.2 (top-left) and 4.3 (left), we plot the test set accuracy for different levels of sampling. We observe the DAEs trained by reconstructing only 2% of the input units give results as good as the DAEs trained without sampling. However, we have found that the representation never reaches the same quality when the sampling probability doesn't depend on $\mathbf{x}$ (i.e. $P(\hat{\mathbf{p}}_k = 1) = d_{SMP}/d_x$).

One epoch is a training pass through the training set. Figure 4.2 (top-right) shows that the DAEs trained using sampling converge as fast in terms of epochs (i.e. in term of training updates) as the DAE trained without. The sampling DAE is trained to reconstruct only 2% of the input units. While initially the baseline DAE converges faster, after a few epochs both DAEs exhibit similar convergence. In order to assess the quality of the obtained results, we compared the averaged test error over the 4 domains with published results by Blitzer et al. (2007), we obtained an averaged test error of 13.7%, whereas they reported 16.7%.

Figure 4.2 (bottom-right) and 4.3 (middle) show that the training of DAEs using sampling is much faster. In particular, we compare the learning curve of a DAE that reconstructs only 2% of the input units and the baseline DAE. The sampling DAE converges 10.5x faster on the Amazon dataset and 12x faster on the RCV1 dataset.

Figure 4.2 (bottom-left) and 4.3 (right) show the effect of the term $\mathbf{q}$ on the convergence of the reconstruction cost. In the biased DAE we set $\mathbf{q}_k = 1$ and in the unbiased DAE we use $\mathbf{q}_k = E[\hat{\mathbf{p}}_k | k, \mathbf{x}, \tilde{\mathbf{x}}]$. This experiment shows that the DAE trained with the unbiased objective converges to the same reconstruction cost as the baseline while the unbiased version does not (since it minimizes a different cost). However, the networks using the biased and unbiased objectives converge similarly in terms of the quality of the representation.

Figure 4.4 shows the speed-up and training curves obtained on the full Amazon dataset, where the sampled reconstruction model converges 22 times faster, and reconstructing about 0.5% of the inputs. Keep in mind that the baseline dense

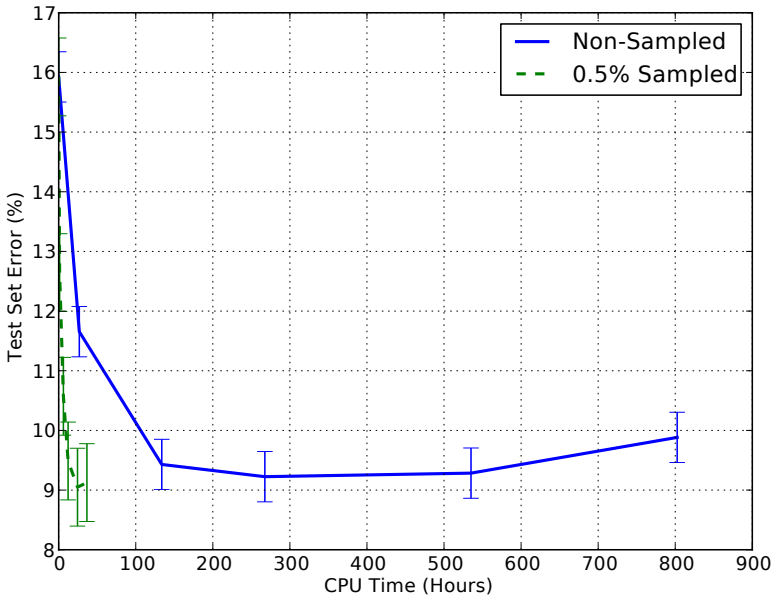


Figure 4.4: Experimental Results on Full Amazon set: test error vs CPU time. The speed-up is about 22x.

training has been optimized already for speed (e.g., choosing the minibatch size and optimized code).

4.6 Conclusion

We have introduced a very simple optimization to speed-up training of unsupervised learning algorithms such as auto-encoders when the input vectors are very large and very sparse. The basic idea is to reconstruct only the non-zero's and a random subsample of the zero's of the input vector. A weighting scheme similar to importance sampling yields an unbiased estimator. On a dataset with a large input size we have found speed-up's of up to 22x, even comparing to optimized dense computation (using minibatches and BLAS' optimized matrix-matrix multiplications). We expect much larger speed-ups will be obtained in applications involving very large sparse input vectors, where the degree of sparsity is even larger than those tested here (2% and .5%).

Figure 4.5: Embeddings learned on the Amazon sentiment data for a randomly selected set of word stems. Colors indicate the Amazon domain, showing that the embedding large scale (left) naturally discovers these categories. Right: zoom showing semantically similar words grouped near each other, on the topic of electronics.

5 Prologue to second article

5.1 Article Detail

Stochastic Ratio Matching of RBMs for Sparse High-Dimensional Inputs. Yann N. Dauphin, Yoshua Bengio. Proceedings of the *Advances in Neural Information Processing Systems 26 (NIPS'13)*.

Personal Contribution.

I proposed the idea that allowed extending the reconstruction sampling (Dauphin et al., 2011) method to restricted Boltzmann machines. I implemented the algorithm and performed all the experiments. I was responsible for writing the description of the algorithm and of ratio matching as well as the experimental section. Yoshua Bengio wrote the introductory sections and did general editing of the paper.

5.2 Context

Before this paper was published, it was costly to train binomial RBMs with large sparse inputs. Dauphin et al. (2011) had proposed a solution for the denoising auto-encoders, but this method did not apply directly to RBMs. The crux of the difficulty resided in the fact that it was hard to subsample the input without biasing the Gibbs chain. Dahl et al. (2012b) introduced a method for training RBMs with softmax inputs. The method replaces the traditional Gibbs chain with iterations of Metropolis-Hastings sampling. This allowed training on word observations in the form of n-grams. The experiments showed this approach was quite useful in practice. This made the application of RBMs to language modelling tractable. However, it left open the question of training on general high-dimensional sparse distributions.

5.3 Contributions

The paper extends the work described in (Dauphin et al., 2011) to binomial RBMs. We were able to reformulate the objective ratio matching in a manner that is similar to auto-encoders. Using this, we were able to devise a good importance sampling distribution inspired by (Dauphin et al., 2011). We showed that this technique can be successfully unbiased using importance weights. Experimentally, the RBMs trained using the proposed method of stochastic ratio matching are able to match the performance of RBMs trained with either ratio matching or stochastic maximum likelihood. The RBMs were evaluated based on their estimated negative log-likelihood on two natural language processing datasets. The stochastic ratio matching method led to orders of magnitude speed-ups. When used to pretrain neural networks, we obtained state-of-the-art results on two topic classification datasets.

6 Scaling RBMs to high-dimensional sparse inputs with importance sampling

Unsupervised feature learning algorithms have recently attracted much attention, with the promise of letting the data guide the discovery of good representations. In particular, unsupervised feature learning is an important component of many Deep Learning algorithms (Bengio, 2009a), such as those based on auto-encoders (Bengio et al., 2007b) and Restricted Boltzmann Machines or RBMs (Hinton et al., 2006b). Deep Learning of representations involves the discovery of several levels of representation, with some algorithms able to exploit unlabeled examples and unsupervised or semi-supervised learning.

Whereas Deep Learning has mostly been applied to computer vision and speech recognition, an important set of application areas involve *high-dimensional sparse input vectors*, for example in some Natural Language Processing tasks (such as the text categorization tasks tackled here), as well as in information retrieval and other web-related applications where a very large number of rarely non-zero features can be devised. We would like learning algorithms whose computational requirements grow with the number of non-zeros in the input but not with the total number of features. Unfortunately, auto-encoders and RBMs are computationally inconvenient when it comes to handling such high-dimensional sparse input vectors, because they require a form of *reconstruction* of the input vector, *for all the elements of the input vector, even the ones that were zero.*

In Section 6.1, we recapitulate the Reconstruction Sampling algorithm (Dauphin et al., 2011) that was proposed to handle that problem in the case of auto-encoder variants. The basic idea is to use an importance sampling scheme to stochastically select a subset of the input elements to reconstruct, and importance weights to obtain an unbiased estimator of the reconstruction error gradient.

In this paper, we are interested in extending these ideas to the realm of RBMs. In Section 6.2 we briefly review the basics of RBMs and the Gibbs chain involved in training them. Ratio matching (Hyvärinen, 2007), is an inductive principle and training criterion that can be applied to train RBMs but does not require

a Gibbs chain. In Section 6.3, we present and justify a novel algorithm based on ratio matching order to achieve our objective of taking advantage of highly sparse inputs. The new algorithm is called *Stochastic Ratio Matching* or SRM. In Section 6.5 we present a wide array of experimental results demonstrating the successful application of Stochastic Ratio Matching, both in terms of computational performance (flat growth of computation as the number of non-zeros is increased, linear speedup with respect to regular training) and in terms of generalization performance: the state-of-the-art on two text classification benchmarks is achieved and surpassed. An interesting and unexpected result is that we find the *biased version* of the algorithm (without reweighting) to yield more discriminant features.

6.1 Reconstruction Sampling

An auto-encoder learns an encoder function f mapping inputs x to features $h = f(x)$, and a decoding or reconstruction function g such that $g(f(x)) \approx x$ for training examples x. See Bengio et al. (2012) for a review. In particular, with the denoising auto-encoder, x is stochastically corrupted into $\tilde{x}$ (e.g. by flipping some bits) and trained to make $g(f(\tilde{x})) \approx x$. To avoid the expensive reconstruction $g(h)$ when the input is very high-dimensional, Dauphin et al. (2011) propose that for each example, a small random subset of the input elements be selected for which $g_i(h)$ and the associated reconstruction error is computed. To make the corresponding estimator of reconstruction error (and its gradient) unbiased, they propose to use an importance weighting scheme whereby the loss on the i-th input is weighted by the inverse of the probability that it be selected. To reduce the variance of the estimator, they propose to always reconstruct the i-th input if it was one of the non-zeros in x or in $\tilde{x}$, and to choose uniformly at random an equal number of zero elements. They show that the unbiased estimator yields the expected linear speedup in training time compared to the deterministic gradient computation, while maintaining good performance for unsupervised feature learning. We would like to extend similar ideas to RBMs.

6.2 Restricted Boltzmann Machines

A restricted Boltzmann machine (RBM) is an undirected graphical model with binary variables (Hinton et al., 2006b): observed variables $\mathbf{x}$ and hidden variables $\mathbf{h}$. In this model, the hidden variables help uncover higher order correlations in the data.

The energy takes the form

$$-E(\mathbf{x}, \mathbf{h}) = \mathbf{h}^T \mathbf{W} \mathbf{x} + \mathbf{b}^T \mathbf{h} + \mathbf{c}^T \mathbf{x}$$

with parameters $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$.

The RBM can be trained by following the gradient of the negative log-likelihood

$$-\frac{\partial \log P(\mathbf{x})}{\partial \theta} = E_{data}\left[\frac{\partial F(\mathbf{x})}{\partial \theta}\right] - E_{model}\left[\frac{\partial F(\mathbf{x})}{\partial \theta}\right]$$

where $F(x)$ is the free energy (unnormalized log-probability associated with $P(x)$). However, this gradient is intractable because the second expectation is combinatorial. Stochastic Maximum Likelihood or SML (Younes, 1999; Tieleman, 2008) estimates this expectation using sample averages taken from a persistent MCMC chain (Tieleman, 2008). Starting from $\mathbf{x}^i$ a step in this chain is taken by sampling $\mathbf{h}^i \sim P(\mathbf{h}|\mathbf{x}^i)$, then we have $\mathbf{x}^{i+1} \sim P(\mathbf{x}|\mathbf{h}^i)$. SML-$k$ is the variant where k is the number of steps between parameter updates, with SML-1 being the simplest and most common choice, although better results (at greater computational expense) can be achieved with more steps.

Training the RBM using SML-1 is on the order of $O(dn)$ where d is the dimension of the input variables and n is the number of hidden variables. In the case of *high-dimensional sparse vectors* with p non-zeros, SML does not take advantage of the sparsity. More precisely, sampling $P(\mathbf{h}|\mathbf{x})$ (inference) can take advantage of sparsity and costs $O(pn)$ computations while "reconstruction", i.e., sampling from $P(\mathbf{x}|\mathbf{h})$ requires $O(dn)$ computations. Thus scaling to larger input sizes n yields a linear increase in training time even if the number of non-zeros p in the input remains constant.

6.3 Ratio Matching

Ratio matching (Hyvärinen, 2007) is an estimation method for statistical models where the normalization constant is not known. It is similar to score matching (Hyvärinen, 2005) but applied on discrete data whereas score matching is limited to continuous inputs, and both are computationally simple and yield consistent estimators. The use of Ratio Matching in RBMs is of particular interest because their normalization constant is computationally intractable.

The core idea of ratio matching is to match ratios of probabilities between the data and the model. Thus Hyvärinen (2007) proposes to minimize the following objective function

$$P_x(\mathbf{x}) \sum_{i=1}^{d} \left[g\left(\frac{P_x(\mathbf{x})}{P_x(\bar{\mathbf{x}}^i)}\right) - g\left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)}\right)\right]^2 + \left[g\left(\frac{P_x(\bar{\mathbf{x}}^i)}{P_x(\mathbf{x})}\right) - g\left(\frac{P(\bar{\mathbf{x}}^i)}{P(\mathbf{x})}\right)\right]^2 \tag{6.1}$$

where P_x is the true probability distribution, P the distribution defined by the model, $g(x) = \frac{1}{1+x}$ is an activation function and $\bar{\mathbf{x}}^i = (x_1, x_2, \ldots, 1 - x_i, \ldots, x_d)$. In this form, we can see the similarity between score matching and ratio matching. The normalization constant is canceled because $\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} = \frac{e^{-F(\mathbf{x})}}{e^{-F(\bar{\mathbf{x}}^i)}}$, however this objective requires access to the true distribution P_x which is rarely available.

Hyvärinen (2007) shows that the Ratio Matching (RM) objective can be simplified to

$$J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} \left(g\left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)}\right)\right)^2 \tag{6.2}$$

which does not require knowledge of the true distribution P_x. This objective can be described as ensuring that the training example $\mathbf{x}$ has the highest probability in the neighborhood of points at hamming distance 1.

We propose to rewrite Eq. 6.2 in a form reminiscent of auto-encoders:

$$J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} (x_i - P(x_i = 1 | \mathbf{x}_{-i}))^2. \tag{6.3}$$

This will be useful for reasoning about this estimator. The main difference with auto-encoders is that each input variable is predicted by excluding it from the input.

Applying Equation 6.2 to the RBM we obtain $J_{RM}(\mathbf{x}) = \sum_{i=1}^{d} \left(\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i))\right)^2$.

The gradients have the familiar form

$$-\frac{\partial J_{RM}(\mathbf{x})}{\partial \theta} = \sum_{i=1}^{d} 2\eta_i \left[\frac{\partial F(\mathbf{x})}{\partial \theta} - \frac{\partial F(\bar{\mathbf{x}}^i)}{\partial \theta} \right] \tag{6.4}$$

with $\eta_i = (\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i)))^2 - (\sigma(F(\mathbf{x}) - F(\bar{\mathbf{x}}^i)))^3$.

A naive implementation of this objective is $O(d^2 n)$ because it requires d computations of the free energy per example. This is much more expensive than SML as noted by Marlin et al. (2010). Thankfully, as we argue here, it is possible to greatly reduce this complexity by reusing computation and taking advantage of the parametrization of RBMs. This can be done by saving the results of the computations $\alpha = \mathbf{c}^T\mathbf{x}$ and $\beta_j = \sum_i W_{ji}x_i + b_j$ when computing $F(\mathbf{x})$. The computation of $F(\bar{\mathbf{x}}^i)$ can be reduced to $O(n)$ with the formula $-F(\bar{\mathbf{x}}^i) = \alpha - (2x_i - 1)c_i + \sum_j \log(1 + e^{\beta_j - (2x_i - 1)W_{ji}})$. This implementation is $O(dn)$ which is the same complexity as SML. However, like SML, RM does not take advantage of sparsity in the input.

6.4 Stochastic Ratio Matching

We propose Stochastic Ratio Matching (SRM) as a more efficient form of ratio matching for high-dimensional sparse distributions. The ratio matching objective requires the summation of d terms in $O(n)$. The basic idea of SRM is to estimate this sum using a very small fraction of the terms, randomly chosen. If we rewrite the ratio matching objective as an expectation over a discrete distribution

$$J_{RM}(\mathbf{x}) = d \sum_{i=1}^{d} \frac{1}{d} g^2 \left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} \right) = dE \left[g^2 \left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} \right) \right] \tag{6.5}$$

we can use Monte Carlo methods to estimate J_{RM} without computing all the terms in Equation 6.2. However, in practice this estimator has a high variance. Thus it is a poor estimator, especially if we want to use very few Monte Carlo samples. The solution proposed for SRM is to use an Importance Sampling scheme to obtain a lower variance estimator of J_{RM}. Combining Monte Carlo with importance

sampling, we obtain the SRM objective

$$J_{SRM}(\mathbf{x}) = \sum_{i=1}^{d} \frac{\gamma_i}{E[\gamma_i]} g^2 \left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} \right) \tag{6.6}$$

where $\gamma_i \sim P(\gamma_i = 1|\mathbf{x})$ is the so-called proposal distribution of our importance sampling scheme. The proposal distribution determines which terms will be used to estimate the objective since only the terms where $\gamma_i = 1$ are non-zero. $J_{SRM}(\mathbf{x})$ is an unbiased estimator of $J_{RM}(\mathbf{x})$, i.e.,

$$\begin{aligned} E[J_{SRM}(\mathbf{x})] &= \sum_{i=1}^{d} \frac{E[\gamma_i]}{E[\gamma_i]} g^2 \left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} \right) \\ &= J_{RM}(\mathbf{x}) \end{aligned}$$

The intuition behind importance sampling is that the variance of the estimator can be reduced by focusing sampling on the largest terms of the expectation. More precisely, it is possible to show that the variance of the estimator is minimized when $P(\gamma_i = 1|\mathbf{x}) \propto g^2(P(\mathbf{x})/P(\bar{\mathbf{x}}^i))$. Thus we would like the probability $P(\gamma_i = 1|\mathbf{x})$ to reflect how large the error $(x_i - P(x_i = 1|\mathbf{x}_{-i}))^2$ will be. The challenge is finding a good approximation for $(x_i - P(x_i = 1|\mathbf{x}_{-i}))^2$ and to define a proposal distribution that is efficient to sample from.

Following Dauphin et al. (2011), we propose such a distribution for high-dimensional sparse distributions. In these types of distributions the marginals $P_x(x_i = 1)$ are very small. They can easily be learned by the biases $\mathbf{c}$ of the model, and may even be initialized very close to their optimal value. Once the marginals are learned, the model will likely only make wrong predictions when $P_x(x_i = 1|\mathbf{x}_{-i})$ differs significantly from $P_x(x_i = 1)$. If $x_i = 0$ then the error $(0 - P(x_i = 1|\mathbf{x}_{-i}))^2$ is likely small because the model has a high bias towards $P(x_i = 0)$. Conversely, the error will be high when $x_i = 1$. In other words, the model will mostly make errors for terms where $x_i = 1$ and a small number of dimensions where $x_i = 0$. We can use this to define the heuristic proposal distribution

$$P(\gamma_i = 1|\mathbf{x}) = \begin{cases} 1 & \text{if } x_i = 1 \\ p/(d - \sum_j 1_{x_j > 0}) & otherwise \end{cases} \tag{6.7}$$

where p is the average number of non-zeros in the data. The idea is to always

sample the terms where $x_i = 1$ and a subset of k of the $(d - \sum_j 1_{x_j>0})$ remaining terms where $x_i = 0$. Note that if we sampled the γ_i independently, we would get $E[k] = p$.

However, instead of sampling those γ_i bits independently, we find that much smaller variance is obtained by sampling a number of zeros k that is **constant** for all examples, i.e., $k = p$. A random k can cause very significant variance in the gradients and this makes stochastic gradient descent more difficult. In our experiments we set $k = p = E[\sum_j 1_{x_j>0}]$ which is a small number by definition of these sparse distributions, and guarantees that computation costs will remain constant as n increases for a fixed number of non-zeros. The computational cost of SRM per training example is $O(pn)$, as opposed to $O(dn)$ for RM. While simple, we find that this heuristic proposal distribution works well in practice, as shown below.

For comparison, we also perform experiments with a biased version of Equation 6.6

$$J_{BiasedSRM}(\mathbf{x}) = \sum_{i=1}^{d} \gamma_i g^2 \left(\frac{P(\mathbf{x})}{P(\bar{\mathbf{x}}^i)} \right). \tag{6.8}$$

This will allow us to gauge the effectiveness of our importance weights for unbiasing the objective. The biased objective can be thought as down-weighting the ratios where $x_i = 0$ by a factor of $E[\gamma_i]$.

SRM is related to previous work (Dahl et al., 2012a) on applying RBMs to high-dimensional sparse inputs, more precisely multinomial observations, e.g., one K-ary multinomial for each word in an n-gram window. A careful choice of Metropolis-Hastings transitions replaces Gibbs transitions and allows to handle large vocabularies. In comparison, SRM is geared towards general sparse vectors and involves an extremely simple procedure without MCMC.

6.5 Experimental Results

In this section, we demonstrate the effectiveness of SRM for training RBMs. Additionally, we show that RBMs are useful features extractors for topic classification.

Datasets We have performed experiments with the Reuters Corpus Volume I (RCV1) and 20 Newsgroups (20 NG). RCV1 is a benchmark for document classification of over 800,000 news wire stories (Lewis et al., 2004). The documents are represented as bag-of-words vectors with 47,236 dimensions. The training set contains 23,149 documents and the test set has 781,265. While there are 3 types of labels for the documents, we focus on the task of predicting the topic. There are a set of 103 non-mutually exclusive topics for a document. We report the performance using the $F_{1.0}$ measure for comparison with the state of the art. 20 Newsgroups is a collection of Usenet posts composing a training set of 11,269 examples and 7505 test examples. The bag-of-words vectors contain 61,188 dimensions. The postings are to be classified into one of 20 categories. We use the *by-date* train/test split which ensures that the training set contains postings preceding the test examples in time. Following Larochelle et al. (2012), we report the classification error and for a fair comparison we use the same preprocessing [1].

Methodology We compare the different estimation methods for the RBM based on the log-likelihoods they achieve. To do this we use Annealed Importance Sampling or AIS (Salakhutdinov and Murray, 2008). For all models we average 100 AIS runs with 10,000 uniformly spaced reverse temperatures β_k. We compare RBMs trained with ratio matching, stochastic ratio matching and biased stochastic ratio matching. We include experiments with RBMs trained with SML-1 for comparison.

Additionally, we provide experiments to motivate the use of high-dimensional RBMs in NLP. We use the RBM to pretrain the hidden layers of a feed-forward neural network (Hinton et al., 2006b). This acts as a regularization for the network and it helps optimization by initializing the network close to a good local minimum (Erhan et al., 2010).

The hyper-parameters are cross-validated on a validation set consisting of 5% of the training set. In our experiments with AIS, we use the validation log-likelihood as the objective. For classification, we use the discriminative performance on the validation set. The hyper-parameters are found using random search (Bergstra and Bengio, 2012a) with 64 trials per set of experiments. The learning rate for the RBMs is sampled from $10^{-[0,3]}$, the number of hidden units from $[500, 2000]$ and the number of training epochs from $[5, 20]$. The learning rate for the MLP is

1. http://qwone.com/~jason/20Newsgroups/20news-bydate-matlab.tgz

sampled from $10^{-[2,0]}$. It is trained for 32 epochs using early-stopping based on the validation set. We regularize the MLP by dropping out 50% of the hidden units during training (Hinton et al., 2012). We adapt the learning rate dynamically by multiplying it by 0.95 when the validation error increases.

All experiments are run on a cluster of double quad-core Intel Xeon E5345 running at 2.33Ghz with 2GB of RAM.

6.5.1 Using SRM to train RBMs

Table 6.1: Log-probabilities estimated by AIS for the RBMs trained with the different estimation methods. With a fixed budget of epochs, SRM achieves likelihoods on the test set comparable with RM and SML-1.

		Estimates		Avg. log-prob.	
		$\log \hat{Z}$	$\log(\hat{Z} \pm \hat{\sigma})$	Train	Test
RCV1	Biased SRM	1084.96	1079.66, 1085.65	-758.73	-793.20
	SRM	325.26	325.24, 325.27	-139.79	-151.30
	RM	499.88	499.48, 500.17	-119.98	**-147.32**
	SML-1	323.33	320.69, 323.99	-138.90	-153.50
20 NG	Biased SRM	1723.94	1718.65, 1724.63	-960.34	-1018.73
	SRM	546.52	546.55, 546.49	-178.39	-190.72
	RM	975.42	975.62, 975.18	-159.92	**-185.61**
	SML-1	612.15	611.68, 612.46	-173.56	-188.82

We can measure the effectiveness of SRM by comparing it with various estimation methods for the RBM. As the RBM is a generative model, we must compare these methods based on the log-likelihoods they achieve. Note that Dauphin et al. (2011) relies on the classification error because there is no accepted performance measure for DAEs. As both RM and SML scale badly with input dimension, we restrict the dimension of the dataset to the $p = 1,000$ most frequent words. We will describe experiments with all dimensions in the next section.

As seen in Table 6.1, SRM is a good estimator for training RBMs and is a good approximation of RM. We see that with the same budget of epochs SRM achieves log-likelihoods comparable with RM on both datasets. The striking difference of more than 500 nats with Biased SRM shows that the importance weights success-

fully unbias the estimator. Interestingly, we observe that RM is able to learn better generative models than SML-1 for both datasets. This is similar to Marlin et al. (2010) where Pseudolikelihood achieves better log-likelihood than SML on a subset of 20 newsgroups. We observe this is an optimization problem since the training log-likelihood is also higher than RM. One explanation is that SML-1 might experience mixing problems (Bengio et al., 2013).

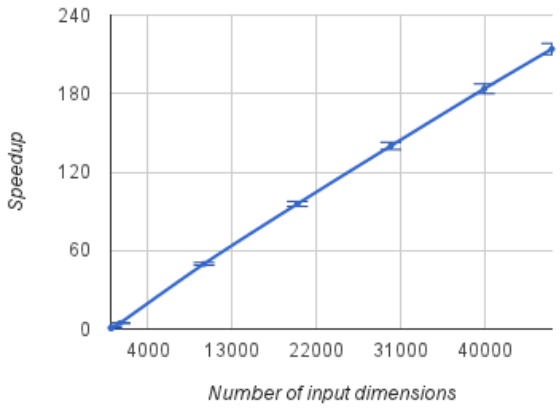


Figure 6.1: Average speedup in the calculation of gradients by using the *SRM* objective compared to *RM*. The speed-up is linear and reaches up to 2 orders of magnitude.

Figure 6.1 shows that as expected SRM achieves a linear speed-up compared to RM, reaching speed-ups of 2 orders of magnitude. In fact, we observed that the computation time of the gradients for RM scale linearly with the size of the input while the computation time of SRM remains fairly constant because the number of non-zeros varies little. This is an important property of SRM which makes it suitable for very large scale inputs.

The importance sampling scheme of SRM (Equation 6.7) relies on the hypothesis that terms where $x_i = 1$ produce a larger gradient than terms where $x_i = 0$. We can verify this by monitoring the average gradients during learning on RCV1. Figure 6.2 demonstrates that the average gradients for the terms where $x_i = 1$ is 2 orders of magnitudes larger than those where $x_i = 0$. This confirms the hypothesis underlying the sampling scheme of SRM.

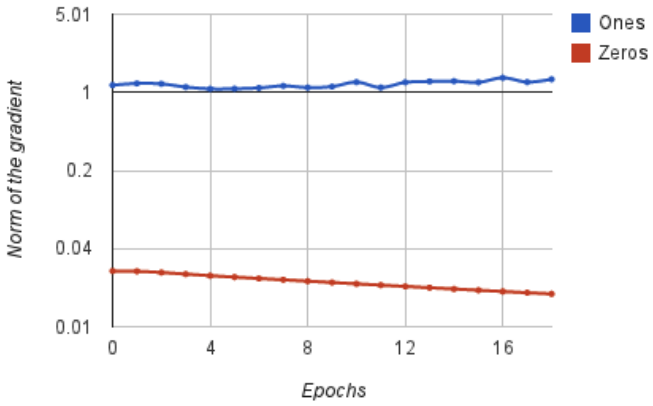


Figure 6.2: Average norm of the gradients for the terms in Equation 6.2 where $x_i = 1$ and $x_i = 0$. Confirming the hypothesis for the proposal distribution the terms where $x_i = 1$ are 2 orders of magnitude larger.

6.5.2 Using RBMs as feature extractors for NLP

Having established that SRM is an efficient unbiased estimator of RM, we turn to the task of using RBMs not as generative models but as feature extractors. We find that keeping the bias in SRM is helpful for classification. This is similar to the known result that contrastive divergence, which is biased, yields better classification results than persistent contrastive divergence, which is unbiased. The bias increases the weight of non-zeros features. The superior performance of the biased objective suggests that the non-zero features contain more information about the classification task. In other words, for these tasks it's more important to focus on what is there than what is not there.

Table 6.2: Classification results on RCV1 with all 47,326 dimensions. The DBN trained with SRM achieves state-of-the-art performance.

Model	Test set F1
Rocchio	0.693
k-NN	0.765
SVM	0.816
SDA-MLP (Rec. sampling)	0.831
RBM-MLP (Unbiased SRM)	0.816
RBM-MLP (Biased SRM)	0.829
DBN-MLP (Biased SRM)	**0.836**

On RCV1, we train our models on all 47,326 dimensions. The RBM trained with SRM improves on the state-of-the-art (Lewis et al., 2004), as shown in Table 6.2. The total training time for this RBM using SRM is 57 minutes. We also train a Deep Belief Net (DBN) by stacking an RBM trained with SML on top of the RBMs learned with SRM. This type of 2-layer deep architecture is able to significantly improve the performance on that task (Table 6.2). In particular the DBN does significantly better than a stack of denoising auto-encoders we trained using biased reconstruction sampling (Dauphin et al., 2011), which appears as *SDA-MLP (Rec. Sampling)* in Table 6.2.

Table 6.3: Classification results on 20 Newsgroups with all 61,188 dimensions. Prior results from (Larochelle et al., 2012). The RBM trained with SRM achieves state-of-the-art results.

Model	Test set Error
SVM	32.8 %
MLP	28.2 %
RBM	24.9 %
HDRBM	21.9 %
DAE-MLP (Rec. sampling)	20.6 %
RBM-MLP (Biased SRM)	**20.5** %

We apply RBMs trained with SRM on 20 newsgroups with all 61,188 dimensions. We see in Table 6.3 that this approach improves the previous state-of-the-art by over 1% (Larochelle et al., 2012), beating non-pretrained MLPs and SVMs by close to 10 %. This result is closely followed by the DAE trained with reconstruction sampling which in our experiments reaches 20.6% test error. The simpler RBM trained by SRM is able to beat the more powerful HD-RBM model because it uses all the 61,188 dimensions.

6.6 Conclusion

We have proposed a very simple algorithm called Stochastic Ratio Matching (SRM) to take advantage of sparsity in high-dimensional data when training discrete RBMs. It can be used to estimate gradients in $O(np)$ computation where

p is the number of non-zeros, yielding linear speedup against the $O(nd)$ of Ratio Matching (RM) where d is the input size. It does so while providing an unbiased estimator of the ratio matching gradient. Using this efficient estimator we train RBMs as features extractors and achieve state-of-the-art results on 2 text classification benchmarks.

7 Prologue to third article

7.1 Article Detail

The manifold tangent classifier. Salah Rifai, Yann N. Dauphin, Pascal Vincent, Yoshua Bengio, Xavier Muller. Proceedings of the *Advances in Neural Information Processing Systems, 2011 (NIPS 2011)*.

Personal Contribution.

The original idea for this paper was conceived by combining two projects from Salah Rifai and me. Both Salah Rifai and I have agreed to share first-authorship, with Salah's name coming first. We are both responsible for the core idea of using the learned tangents of contractive auto-encoder for classification. Salah Rifai was in charge of training the contractive auto-encoders well for our experiments. I implemented the tangent propogation framework needed to make use of the extracted tangents and ran the semi-supervised and supervised learning experiments. Notably, I was responsible for the state-of-the-art results on MNIST.

7.2 Context

This paper arose from the general interest in obtaining or encouraging invariance in deep neural networks. Goodfellow et al. (2009) had observed that deep networks naturally learned more invariant representations with each added layer. However, it is not clear improve invariance to task-independent transformation. In the case of images, convolutional networks are a well accepted way of achieving certain invariances (Jarrett et al., 2009). The use of convolutional and max-pooling affords the network some translation and rotation invariance among others. It does not offer more high-level invariances or even task-specific invariances.

We were interested in a framework for invariance that allowed invariance over a large variety of transformations, and that was applicable to multiple domains. Simard et al. (1992) proposed a framework that fulfilled half these requirements. It allowed invariance to several geometric transformations, but they had to be known in advance. Thus, the paper only proposed transformations for the domain of images. It did not provide any specific insight for other domains, like natural language or speech and even images undergo transformations that are difficult to formalize. In this paper, we were interested in generalizing tangent propagation to any distribution by learning the invariances using a contractive auto-encoder.

7.3 Contributions

This paper was important to the understanding of the contractive auto-encoder in a manifold perspective. We proposed a method to extract tangents from the contractive auto-encoder. Experimentally, we were able to show that the tangents visually match the analytical tangents for vision problems. We showed that when applied to other domains, like natural language processing, the tangents still corresponded to credible transformations. These insights were key to the creation of a generative process for contractive auto-encoders (Rifai et al., 2012). This process moves by following the tangent directions in the latent space. This has in turn led to further discoveries about the generative nature of auto-encoders. Alain and Bengio (2013); Bengio et al. (2014) discovered that denoising auto-encoders learn local properties of the distribution and devised a new generative process.

We demonstrated experimentally that the tangents learned by the contractive auto-encoder were useful in creating task-specific invariances across several tasks. We confirmed that these invariances were especially important when the amount of labelled data is low. We showed through experiments on the Covertype dataset that this invariance framework extended to non-vision data. Furthermore, we obtained state-of-the-art results on the MNIST dataset.

8 Regularizing deep networks with a geometric approach

Much of machine learning research can be viewed as an exploration of ways to compensate for scarce prior knowledge about how to solve a specific task by extracting (usually implicit) knowledge from vast amounts of data. This is especially true of the search for generic learning algorithms that are to perform well on a wide range of domains for which they were not specifically tailored. While such an outlook precludes using much domain-specific knowledge in designing the algorithms, it can however be beneficial to leverage what might be called "generic" prior hypotheses, that appear likely to hold for a wide range of problems. The approach studied in the present work exploits three such prior hypotheses:

1. The **semi-supervised learning hypothesis**, according to which learning aspects of the input distribution $p(x)$ can improve models of the conditional distribution of the supervised target $p(y|x)$, i.e., $p(x)$ and $p(y|x)$ share something (Lasserre et al., 2006). This hypothesis underlies not only the strict semi-supervised setting where one has many more unlabeled examples at his disposal than labeled ones, but also the successful *unsupervised pre-training* approach for learning deep architectures, which has been shown to significantly improve supervised performance even without using *additional* unlabeled examples (Hinton et al., 2006a; Bengio, 2009b; Erhan et al., 2010).

2. The **(unsupervised) manifold hypothesis**, according to which real world data presented in high dimensional spaces is likely to concentrate in the vicinity of non-linear sub-manifolds of much lower dimensionality (Cayton, 2005; Narayanan and Mitter, 2010).

   This is believed to be the reason why it is possible to obtain sensible models even in high dimensions, in spite of the curse of dimensionality. It stems from the view that, from a given point of high probability density (such as a point form the training set), there are only a comparatively small number of local directions of transformations that will yield a point with equally high

probability density (that "looks" like the data), while a majority of directions would move towards less likely,"degraded", observations. Further support for this hypothesis can be argued from the observation that uniformly random generated points in high dimensional spaces almost never look like training data.

3. The **manifold hypothesis for classification**, according to which points of different classes are likely to concentrate along different sub-manifolds, separated by low density regions of the input space.

The recently proposed Contractive Auto-Encoder (CAE) algorithm (Rifai et al., 2011), based on the idea of encouraging the learned representation to be robust to small variations of the input, was shown to be very effective for unsupervised feature learning. Its successful application in the pre-training of deep neural networks is yet another illustration of what can be gained by adopting **hypothesis 1**. In addition, Rifai et al. (2011) propose, and show empirical evidence for, the hypothesis that the trade-off between reconstruction error and the pressure to be insensitive to variations in input space has an interesting consequence: It yields a mostly contractive mapping that, locally around each training point, remains substantially sensitive only to a few input directions (with different directions of sensitivity for different training points). This is taken as evidence that the algorithm indirectly exploits **hypothesis 2** and models a lower-dimensional manifold. Most of the directions to which the representation is substantially sensitive are thought to be directions tangent to the data-supporting manifold (those that locally define its *tangent space*).

The present work follows through on this interpretation, and investigates whether it is possible to use this information, that is presumably captured about manifold structure, to further improve classification performance by leveraging **hypothesis 3**. To that end, we extract a set of basis vectors for the local tangent space at each training point from the Contractive Auto-Encoder's learned parameters. This is obtained with a Singular Value Decomposition (SVD) of the Jacobian of the encoder that maps each input to its learned representation. Based on hypothesis 3, we then adopt the "generic prior" that class labels are likely to be insensitive to most directions within these local tangent spaces (ex: small translations, rotations or scalings usually do not change an image's class). Supervised classification algorithms that have been devised to efficiently exploit tangent directions given as

domain-specific prior-knowledge (Simard et al., 1992, 1993), can readily be used instead with our learned tangent spaces. In particular, we will show record-breaking improvements by using TangentProp for fine tuning CAE-pre-trained deep neural networks. To the best of our knowledge this is the first time that the implicit relationship between an unsupervised learned mapping and the tangent space of a manifold is rendered explicit and successfully exploited for the training of a classifier. This showcases a unified approach that simultaneously leverages all three "generic" prior hypotheses considered. Our experiments (see Section 8.5) show that this approach sets new records for domain-knowledge-free performance on several real-world classification problems. Remarkably, in some cases it even outperformed methods that use weak or strong domain-specific prior knowledge (e.g. convolutional networks and tangent distance based on a-priori known transformations). Naturally, this approach is even more likely to be beneficial for datasets where no prior knowledge is readily available.

8.1 Contractive auto-encoders (CAE)

We consider the problem of the unsupervised learning of a non-linear feature extractor from a dataset $\mathcal{D} = \{x_1, \ldots, x_n\}$. Examples $x_i \in \mathcal{R}^d$ are i.i.d. samples from an unknown distribution $p(x)$.

8.1.1 Traditional auto-encoders

The auto-encoder framework is one of the oldest and simplest techniques for the unsupervised learning of non-linear feature extractors. It learns an *encoder* function h, that maps an input $x \in \mathcal{R}^d$ to a hidden representation $h(x) \in \mathcal{R}^{d_h}$, jointly with a *decoder* function g, that maps h back to the input space as $r = g(h(x))$ the *reconstruction* of x. The encoder and decoder's parameters θ are learned by stochastic gradient descent to minimize the average *reconstruction error* $L(x, g(h(x)))$ for the examples of the training set. The objective being minimized is:

$$\mathcal{J}_{\text{AE}}(\theta) = \sum_{x \in \mathcal{D}} L(x, g(h(x))). \tag{8.1}$$

We will will use the most common forms of encoder, decoder, and reconstruction error:

Encoder: $h(x) = s(Wx + b_h)$, where s is the element-wise logistic sigmoid $s(z) = \frac{1}{1+e^{-z}}$. Parameters are a $d_h \times d$ weight matrix W and bias vector $b_h \in \mathcal{R}^{d_h}$.

Decoder: $r = g(h(x)) = s_2(W^T h(x) + b_r)$. Parameters are W^T (tied weights, shared with the encoder) and bias vector $b_r \in \mathcal{R}^d$. Activation function s_2 is either a logistic sigmoid ($s_2 = s$) or the identity (linear decoder).

Loss function: Either the squared error: $L(x, r) = \|x - r\|^2$ or Bernoulli cross-entropy: $L(x, r) = -\sum_{i=1}^{d} x_i \log(r_i) + (1 - x_i) \log(1 - r_i)$.

The set of parameters of such an auto-encoder is $\theta = \{W, b_h, b_r\}$.

A fully linear auto-encoder (without non-linearity s and s_2) with squared reconstruction error and $d_h < d$ will learn the same subspace as Principal Component Analysis, but this is no longer the case with non-linearities (Japkowicz et al., 2000) particularly when using tied weights.

Historically, auto-encoders were primarily viewed as a technique for dimensionality reduction, where a narrow bottleneck (i.e. $d_h < d$) was in effect acting as a capacity control mechanism. By contrast, recent successes (Bengio et al., 2007b; Ranzato et al., 2007; Kavukcuoglu et al., 2009; Vincent et al., 2010; Rifai et al., 2011) tend to rely on rich, oftentimes over-complete representations ($d_h > d$), so that more sophisticated forms of regularization are required to pressure the auto-encoder to extract relevant features and avoid trivial solutions. Several successful techniques aim at sparse representations (Ranzato et al., 2007; Kavukcuoglu et al., 2009; Goodfellow et al., 2009). Alternatively, denoising auto-encoders (Vincent et al., 2010) change the objective from mere reconstruction to that of denoising.

8.1.2 First order and higher order contractive auto-encoders

More recently, Rifai et al. (2011) introduced the Contractive Auto-Encoder (CAE), that encourages robustness of representation $h(x)$ to small variations of a training input x, by penalizing its *sensitivity* to that input, measured as the Frobenius norm of the encoder's Jacobian $J(x) = \frac{\partial h}{\partial x}(x)$. The regularized objective minimized by the CAE is the following:

$$\mathcal{J}_{\text{CAE}}(\theta) = \sum_{x \in \mathcal{D}} L(x, g(h(x))) + \lambda \|J(x)\|^2, \tag{8.2}$$

where λ is a non-negative regularization hyper-parameter that controls how strongly the norm of the Jacobian is penalized. Note that, with the traditional sigmoid encoder form given above, one can easily obtain the Jacobian of the encoder. Its j^{th} row is obtained form the j^{th} row of W as:

$$J(x)_j = \frac{\partial h_j(x)}{\partial x} = h_j(x)(1 - h_j(x))W_j. \tag{8.3}$$

Computing the extra penalty term (and its contribution to the gradient) is similar to computing the reconstruction error term (and its contribution to the gradient), thus relatively cheap.

It is also possible to penalize higher order derivatives (Hessian) by using a simple stochastic technique that eschews computing them explicitly, which would be prohibitive. It suffices to penalize differences between the Jacobian at x and the Jacobian at nearby points $\tilde{x} = x + \epsilon$ (stochastic corruptions of x). This yields the CAE+H (Rifai et al., 2011) variant with the following optimization objective:

$$\mathcal{J}_{\text{CAE+H}}(\theta) = \sum_{x \in \mathcal{D}} L(x, g(h(x))) + \lambda \left|\left|J(x)\right|\right|^2 + \gamma \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2 I)} \left[\left|\left|J(x) - J(x+\epsilon)\right|\right|^2\right], \tag{8.4}$$

where γ is an additional regularization hyper-parameters that controls how strongly we penalize local variations of the Jacobian, i.e. higher order derivatives. The expectation $\mathbb{E}$ is over Gaussian noise variable ϵ. In practice stochastic samples thereof are used for each stochastic gradient update. The CAE+H is the variant used for our experiments.

8.2 Characterizing the tangent bundle captured by a CAE

Rifai et al. (2011) reason that, while the regularization term encourages insensitivity of $h(x)$ in all input space directions, this pressure is counterbalanced by the need for accurate reconstruction, thus resulting in $h(x)$ being substantially sensitive *only* to the few input directions required to distinguish close by training points. The geometric interpretation is that these directions span the local tangent space

of the underlying manifold that supports the data. The *tangent bundle* of a smooth manifold is the manifold along with the set of tangent planes taken at all points on it. Each such tangent plane can be equipped with a local Euclidean coordinate system or *chart*. In topology, an *atlas* is a collection of such *charts* (like the locally Euclidean map in each page of a geographic atlas). Even though the set of charts may form a non-Euclidean manifold (e.g., a sphere), each chart is Euclidean.

8.2.1 Conditions for the feature mapping to define an atlas on a manifold

In order to obtain a proper atlas of charts, h must be a diffeomorphism. It must be smooth (C^∞) and invertible on open Euclidean balls on the manifold $\mathcal{M}$ around the training points. Smoothness is guaranteed because of our choice of parametrization (affine + sigmoid). Injectivity (different values of $h(x)$ correspond to different values of x) on the training examples is encouraged by minimizing reconstruction error (otherwise we cannot distinguish training examples x_i and x_j by only looking at $h(x_i)$ and $h(x_j)$). Since $h(x) = s(Wx + b_h)$ and s is invertible, using the definition of injectivity we get (by composing $h(x_i) = h(x_j)$ with s^{-1})

$$\forall i, j \; h(x_i) = h(x_j) \iff W\Delta_{ij} = 0$$

where $\Delta_{ij} = x_i - x_j$. In order to preserve the injectivity of h, W has to form a basis spanned by its rows W_k, where $\forall\, i, j \; \exists\, \alpha \in \mathcal{R}^{d_h}, \Delta_{ij} = \sum_k^{d_h} \alpha_k W_k$. With this condition satisfied, mapping h is injective in the subspace spanned by the variations in the training set. If we limit the domain of h to $h(\mathcal{X}) \subset (0,1)^{d_h}$ comprising values obtainable by h applied to some set $\mathcal{X}$, then we obtain surjectivity by definition, hence *bijectivity of h between the training set $\mathcal{D}$ and $h(\mathcal{D})$*. Let $\mathcal{M}_x$ be an open ball on the manifold $\mathcal{M}$ around training example x. By smoothness of the manifold $\mathcal{M}$ and of mapping h, we obtain bijectivity locally around the training examples (on the manifold) as well, i.e., between $\cup_{x\in\mathcal{D}}\mathcal{M}_x$ and $h(\cup_{x\in\mathcal{D}}\mathcal{M}_x)$.

8.2.2 Obtaining an atlas from the learned feature mapping

Now that we have necessary conditions for local invertibility of $h(x)$ for $x \in \mathcal{D}$, let us consider how to define the local chart around x from the nature of h. Because

h must be sensitive to changes from an example x_i to one of its neighbors x_j, but insensitive to other changes (because of the CAE penalty), we expect that this will be reflected in the spectrum of the Jacobian matrix $J(x) = \frac{\partial h(x)}{\partial x}$ at each training point x. In the ideal case where $J(x)$ has rank k, $h(x + \epsilon v)$ differs from $h(x)$ only if v is in the span of the singular vectors of $J(x)$ with non-zero singular value. In practice, $J(x)$ has many tiny singular values. Hence, we define a local chart around x using the Singular Value Decomposition of $J^T(x) = U(x)S(x)V^T(x)$ (where $U(x)$ and $V(x)$ are orthogonal and $S(x)$ is diagonal). The tangent plane $\mathcal{H}_x$ at x is given by the span of the set of principal singular vectors $\mathcal{B}_x$:

$$\mathcal{B}_x = \{U_{\cdot k}(x) | S_{kk}(x) > \epsilon\} \quad \text{and} \quad \mathcal{H}_x = \{x + v | v \in \text{span}(\mathcal{B}_x)\},$$

where $U_{\cdot k}(x)$ is the k-th column of $U(x)$, and $\text{span}(\{z_k\}) = \{x | x = \sum_k w_k z_k, w_k \in \mathcal{R}\}$. We can thus define an atlas $\mathcal{A}$ captured by h, based on the local linear approximation around each example:

$$\mathcal{A} = \{(\mathcal{M}_x, \phi_x) | x \in \mathcal{D}, \phi_x(\tilde{x}) = \mathcal{B}_x(\tilde{x} - x)\}. \tag{8.5}$$

Note that this way of obtaining an atlas can also be applied to subsequent layers of a deep network. It is thus possible to use a greedy layer-wise strategy to initialize a network with CAEs (Rifai et al., 2011) and obtain an atlas that corresponds to the nonlinear features computed at any layer.

8.3 Exploiting the learned tangent directions for classification

Using the previously defined charts for every point of the training set, we propose to use this *additional information* provided by unsupervised learning to improve the performance of the supervised task. In this we adopt the **manifold hypothesis for classification** mentioned in the introduction.

8.3.1 CAE-based tangent distance

One way of achieving this is to use a nearest neighbor classifier with a similarity criterion defined as the shortest distance between two hyperplanes (Simard et al., 1993). The tangents extracted on each points will allow us to shrink the distances between two samples when they can approximate each other by a linear combination of their local tangents. Following Simard et al. (1993), we define the *tangent distance* between two points x and y as the distance between the two hyperplanes $\mathcal{H}_x, \mathcal{H}_y \subset \mathcal{R}^d$ spanned respectively by $\mathcal{B}_x$ and $\mathcal{B}_y$. Using the usual definition of distance between two spaces, $d(\mathcal{H}_x, \mathcal{H}_y) = \inf\{\|z-w\|^2 | / \ (z, w) \in \mathcal{H}_x \times \mathcal{H}_y\}$, we obtain the solution for this convex problem by solving a system of linear equations (Simard et al., 1993). This procedure corresponds to allowing the considered points x and y to move along the directions spanned by their associated local charts. Their distance is then evaluated on the new coordinates where the distance is minimal. We can then use a nearest neighbor classifier based on this distance.

8.3.2 CAE-based tangent propagation

Nearest neighbor techniques are often impractical for large scale datasets because their computational requirements scale linearly with n for each test case. By contrast, once trained, neural networks yield fast responses for test cases. We can also leverage the extracted local charts when training a neural network. Following the *tangent propagation* approach of Simard et al. (1992), but exploiting our learned tangents, we encourage the output o of a neural network classifier to be insensitive to variations in the directions of the local chart of x by adding the following penalty to its supervised objective function:

$$\Omega(x) = \sum_{u \in \mathcal{B}_x} \left\| \frac{\partial o}{\partial x}(x) \ u \right\|^2 \tag{8.6}$$

Contribution of this term to the gradients of network parameters can be computed in $O(N_w)$, where N_w is the number of neural network weights.

8.3.3 The Manifold Tangent Classifier (MTC)

Putting it all together, here is the high level summary of how we build and train a deep network:

1. Train (unsupervised) a stack of K CAE+H layers (Eq. 8.4). Each is trained in turn on the representation learned by the previous layer.
2. For each $x_i \in \mathcal{D}$ compute the Jacobian of the last layer representation $J^{(K)}(x_i) = \frac{\partial h^{(K)}}{\partial x}(x_i)$ and its SVD [1]. Store the leading d_M singular vectors in set $\mathcal{B}_{x_i}$.
3. On top of the K pre-trained layers, stack an output layer of size the number of classes. Fine-tune the whole network for supervised classification [2] with an added tangent propagation penalty (Eq. 8.6), using for each x_i, tangent directions $\mathcal{B}_{x_i}$.

We call this deep learning algorithm the Manifold Tangent Classifier (MTC). Alternatively, instead of step 3, one can use the tangent vectors in $\mathcal{B}_{x_i}$ in a tangent distance nearest neighbors classifier.

8.4 Related prior work

Many **Non-Linear Manifold Learning** algorithms (Roweis and Saul, 2000; Tenenbaum et al., 2000) have been proposed which can automatically discover the main directions of variation around each training point, i.e., the tangent bundle. Most of these algorithms are non-parametric and local, i.e., explicitly parametrizing the tangent plane around each training point (with a separate set of parameters for each, or derived mostly from the set of training examples in every neighborhood), as most explicitly seen in **Manifold Parzen Windows** (Vincent and Bengio, 2003) and manifold **Charting** (Brand, 2003). See Bengio and Monperrus (2005)

1. $J^{(K)}$ is the product of the Jacobians of each encoder (see Eq. 8.3) in the stack. It suffices to compute its leading d_M SVD vectors and singular values. This is achieved in $O(d_M \times d \times d_h)$ per training example. For comparison, the cost of a forward propagation through a single MLP layer is $O(d \times d_h)$ per example.
2. A sigmoid output layer is preferred because computing its Jacobian is straightforward and efficient (Eq. 8.3). The supervised cost used is the cross entropy. Training is by stochastic gradient descent.

for a critique of local non-parametric manifold algorithms: they might require a number of training examples which grows exponentially with manifold dimension and curvature (more crooks and valleys in the manifold will require more examples). One attempt to generalize the manifold shape non-locally (Bengio et al., 2006) is based on explicitly predicting the tangent plane associated to any given point x, as a parametrized function of x. Note that these algorithms all explicitly exploit training set neighborhoods (see Figure 8.2), i.e. they use pairs or tuples of points, with the goal to explicitly model the tangent space, while it is modeled implicitly by the CAE's objective function (that is *not* based on pairs of points). More recently, the **Local Coordinate Coding** (LCC) algorithm (Yu et al., 2009) and its Local Tangent LCC variant (Yu and Zhang, 2010) were proposed to build a a local chart around each training example (with a local low-dimensional coordinate system around it) and use it to define a representation for each input x: the responsibility of each local chart/anchor in explaining input x and the coordinate of x in each local chart. That representation is then fed to a classifier and yield better generalization than x itself.

The tangent distance (Simard et al., 1993) and TangentProp (Simard et al., 1992) algorithms were initially designed to exploit prior domain-knowledge of directions of invariance (ex: knowledge that the class of an image should be invariant to small translations rotations or scalings in the image plane). However any algorithm able to output a chart for a training point might potentially be used, as we do here, to provide directions to a Tangent distance or TangentProp (Simard et al., 1992) based classifier. Our approach is nevertheless unique as the CAE's unsupervised feature learning capabilities are used *simultaneously* to provide a good initialization of deep network layers **and** a coherent non-local predictor of tangent spaces. TangentProp is itself closely related to the **Double Backpropagation** algorithm (Drucker and LeCun, 1992), in which one instead adds a penalty that is the sum of squared derivatives of the prediction error (with respect to the network input). Whereas TangentProp attempts to make the output insensitive to selected directions of change, the double backpropagation penalty term attempts to make the error at a training example invariant to changes in all directions. Since one is also trying to minimize the error at the training example, this amounts to making that minimization more robust, i.e., extend it to the *neighborhood* of the training examples. The term "double backpropagation" comes from idea that one

needs to compute derivatives through derivatives: the gradient of the penalty term with respect to the parameters can be computed by unfolding the computation of the regular backpropagation (on top of the forward propagation computation), and then computing (backwards) gradients through both the forward propagation and backpropagation sub-networks.

Also related is the **Semi-Supervised Embedding** algorithm (Weston et al., 2008). In addition to minimizing a supervised prediction error, it encourages each layer of representation of a deep architecture to be *invariant* when the training example is changed from x to a near neighbor of x in the training set. This algorithm works implicitly under the hypothesis that the variable y to predict from x is invariant to the local directions of change present between nearest neighbors. This is consistent with the **manifold hypothesis for classification** (hypothesis 3 mentioned in the introduction). Instead of removing variability along the local directions of variation, the **Contractive Auto-Encoder** (Rifai et al., 2011) *initially* finds a representation which is most *sensitive* to them, as we explained in section 8.1.

8.5 Experiments

We conducted experiments to evaluate our approach and the quality of the manifold tangents learned by the CAE, using a range of datasets from different domains:

MNIST is a dataset of 28×28 images of handwritten digits. The learning task is to predict the digit contained in the images. **Reuters Corpus Volume I** is a popular benchmark for document classification. It consists of 800,000 real-world news wire stories made available by Reuters. We used the 2000 most frequent words calculated on the whole dataset to create a bag-of-words vector representation. We used the LYRL2004 split to separate between a train and test set. **CIFAR-10** is a dataset of 70,000 32×32 RGB real-world images. It contains images of real-world objects (i.e. cars, animals) with all the variations present in natural images (i.e. backgrounds). **Forest Cover Type** is a large-scale database of cartographic variables for the prediction of forest cover types made available by the US Forest Service.

We investigate whether leveraging the CAE learned tangents leads to better classification performance on these problems, using the following methodology: Optimal hyper-parameters for (a stack of) CAEs are selected by cross-validation on a disjoint validation set extracted from the training set. The quality of the feature extractor and tangents captured by the CAEs is evaluated by initializing an neural network (MLP) with the same parameters and fine-tuning it by backpropagation on the supervised classification task. The optimal strength of the supervised TangentProp penalty and number of tangents d_M is also cross-validated.

Results

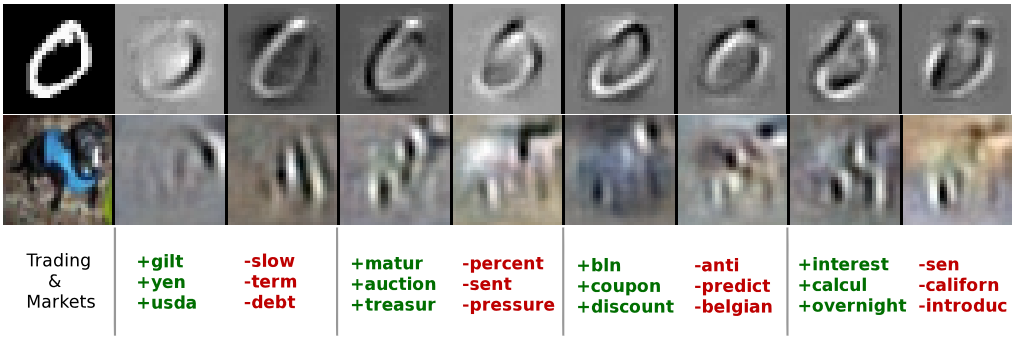


Figure 8.1: Visualisation of the tangents learned by the CAE for MNIST, CIFAR-10 and RCV1 (top to bottom). The left-most column is the example and the following columns are its tangents. On RCV1, we show the tangents of a document with the topic "Trading & Markets" (MCAT) with the negative terms in red(-) and the positive terms in green(+).

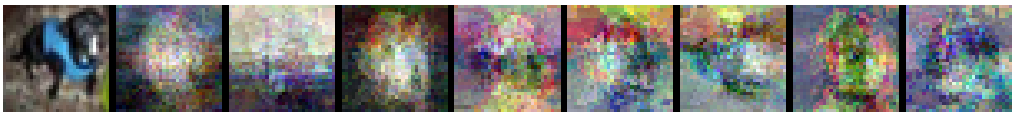


Figure 8.2: Tangents extracted by local PCA on CIFAR-10. This shows the limitation of approaches that rely on training set neighborhoods.

Figure 8.1 shows a visualization of the tangents learned by the CAE. On MNIST, the tangents mostly correspond to small geometrical transformations like translations and rotations. On CIFAR-10, the model also learns sensible tangents, which seem to correspond to changes in the *parts* of objects. The tangents on RCV1-v2 correspond to the addition or removal of similar words and removal of irrelevant words. We also note that extracting the tangents of the model is a way to visualize

what the model has learned about the structure of the manifold. Interestingly, we see that **hypothesis 3** holds for these datasets because most tangents do not change the class of the example.

Table 8.1: Classification accuracy on several datasets using KNN variants measured on 10,000 test examples with 1,000 training examples. The KNN is trained on the raw input vector using the Euclidean distance while the K-layer+KNN is computed on the representation learned by a K-layer CAE. The KNN+Tangents uses at every sample the local charts extracted from the 1-layer CAE to compute tangent distance.

	KNN	KNN+Tangents	1-Layer CAE+KNN	2-Layer CAE+KNN
MNIST	86.9	88.7	90.55	**91.15**
CIFAR-10	25.4	**26.5**	25.1	-
COVERTYPE	70.2	**70.98**	69.54	67.45

We use KNN using tangent distance to evaluate the quality of the learned tangents more objectively. Table 8.1 shows that using the tangents extracted from a CAE always lead to better performance than a traditional KNN.

Table 8.2: Semi-supervised classification error on the MNIST test set with 100, 600, 1000 and 3000 labeled training examples. We compare our method with results from (Weston et al., 2008; Ranzato et al., 2007; Salakhutdinov and Hinton, 2007).

	NN	SVM	CNN	TSVM	DBN-rNCA	EmbedNN	CAE	MTC
100	25.81	23.44	22.98	16.81	-	16.86	13.47	**12.03**
600	11.44	8.85	7.68	6.16	8.7	5.97	6.3	**5.13**
1000	10.7	7.77	6.45	5.38	-	5.73	4.77	**3.64**
3000	6.04	4.21	3.35	3.45	3.3	3.59	3.22	**2.57**

As described in section 8.3.2, the tangents extracted by the CAE can be used for fine-tuning the multilayer perceptron using tangent propagation, yielding our Manifold Tangent Classifier (MTC). As it is a semi-supervised approach, we evaluate its effectiveness with a varying amount of labeled examples on MNIST. Following Weston et al. (2008), the unsupervised feature extractor is trained on the full training set and the supervised classifier is trained on a restricted labeled set. Table 8.2 shows our results for a single hidden layer MLP initialized with CAE+H pretraining (noted CAE for brevity) and for the same classifier fine-tuned with tangent propagation (i.e. the *manifold tangent classifier* of section 8.3.3, noted MTC). The methods that do not leverage the semi-supervised learning hypothesis

(Support Vector Machines, traditional Neural Networks and Convolutional Neural Networks) give very poor performance when the amount of labeled data is low. In some cases, the methods that can learn from unlabeled data can reduce the classification error by half. The CAE gives better results than other approaches across almost the whole range considered. It shows that the features extracted from the rich unlabeled data distribution give a good inductive prior for the classification task. Note that the MTC consistently outperforms the CAE on this benchmark.

Table 8.3: Classification error on the MNIST test set with the full training set.

K-NN	NN	SVM	DBN	CAE	DBM	CNN	MTC
3.09%	1.60%	1.40%	1.17%	1.04%	0.95%	0.95%	**0.81**%

Table 8.3 shows our results on the full MNIST dataset with some results taken from (LeCun et al., 1999; Hinton et al., 2006a). The CAE in this figure is a two-layer deep network with 2000 units per layer pretrained with the CAE+H objective. The MTC uses the same stack of CAEs trained with tangent propagation using 15 tangents. The prior state of the art for the permutation invariant version of the task was set by the Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009b) at 0.95%. Using our approach, we reach 0.81% error on the test set. Remarkably, the MTC also outperforms the basic Convolutional Neural Network (CNN) even though the CNN exploits prior knowledge about vision using convolution and pooling to enhance the results.

Table 8.4: Classification error on the Forest CoverType dataset.

SVM	Distributed SVM	MTC
4.11%	3.46%	**3.13**%

We also trained a 4 layer MTC on the Forest CoverType dataset. Following Trebar and Steele (2008), we use the data split DS2-581 which contains over 500,000 training examples. The MTC yields the best performance for the classification task beating the previous state of the art held by the distributed SVM (mixture of several non-linear SVMs).

8.6 Conclusion

In this work, we have shown a new way to characterize a manifold by extracting a local chart at each data point based on the unsupervised feature mapping built with a deep learning approach. The developed Manifold Tangent Classifier successfully leverages three common "generic prior hypotheses" in a unified manner. It learns a meaningful representation that captures the structure of the manifold, and can leverage this knowledge to reach superior classification performance. On datasets from different domains, it successfully achieves state of the art performance.

9 Prologue to Fourth article

9.1 Article Detail

Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, Yoshua Bengio. Proceedings of the *Advances in Neural Information Processing Systems 27 (NIPS'14)*.

Personal Contribution.

The paper proposes an experimental validation of the work of theoretical physicists (Bray and Dean, 2007) and an optimization algorithm called saddle-free Newton. I contributed to both. Surya Ganguli proposed validating the theoretical predictions made by Bray and Dean (2007) about the distribution of saddle points. Razvan Pascanu and I were responsible for these experiments. I devised the experimental protocol. Razvan was responsible for obtaining results for one dataset while I labored on a second one. Razvan Pascanu had proposed using the square of the Hessian to better handle negative curvature. I determined experimentally that this was not optimal. I improved on the idea by proposing to use the absolute value of the Hessian. This quantity better conserves the curvature information. I ran the experiments using this algorithm using feedfoward networks. Caglar Gulcehre and Kyunghyun Cho ran the experiments with recurrent neural networks.

9.2 Context

It was commonly thought that the main difficulty in training deep neural networks was the presence of multiple local minima. This was considered a serious

drawback to using neural networks because the issue of multiple local minima cannot be resolved. For this reason, some researchers have avoided using neural networks. Concurrently to this, researchers in disparate litteratures have shown that some non-convex models have a relatively small amount of local minima. Bray and Dean (2007); Fyodorov and Williams (2007) were able to show that high-dimensional random gaussian fields possess an exponential amount of saddle points and exponentially few local minima. This ran counter to cursory intuition and stood in stark contrast with the beliefs of neural network researchers. One of the objectives of this paper was to see if these results would apply to neural networks. As such, there were no optimization algorithm expressly designed to allow escaping saddle points efficiently. The paper was also interested in proposing something in that department.

9.3 Contributions

To our knowledge the paper was the first experimental validation of the predictions made in either Bray and Dean (2007) or Fyodorov and Williams (2007). We demonstrated that there is a strong correlation between the index of a critical point and the training error. We also showed the eigenvalues of the Hessian follow Wagner's semi-circular distribution. This confirms the prediction that saddle points are prevalent in the optimization landscapes of neural networks. Following this experimental confirmation, Choromanska et al. (2014) was able to theoretically prove this for deep rectifier networks. This may be an important element to motivate the application of neural networks to wider domains. Kim et al. (2014) note this in their application of neural networks to search for gravitational-wave signals.

We also presented an optimization better suited to non-convex optimization. It attained better results than gradient descent on several problems. This has led to more research in trying to make better optimizers for non-convex problems. Ge et al. (2015) proposes modifications to gradient descent to guarantee global convergence. Hazan et al. (2015) introduces a new optimization method based on continuations. It allows avoiding critical points by smoothing the optimization surface.

10 Identifying the challenges in high-dimensional non-convex optimization

It is often the case that our geometric intuition, derived from experience within a low dimensional physical world, is inadequate for thinking about the geometry of typical error surfaces in high-dimensional spaces. To illustrate this, consider minimizing a randomly chosen error function of a single scalar variable, given by a single draw of a Gaussian process. (Rasmussen and Williams, 2005) have shown that such a random error function would have many local minima and maxima, with high probability over the choice of the function, but saddles would occur with negligible probability. On the other-hand, as we review below, typical, random Gaussian error functions over N scalar variables, or dimensions, are increasingly likely to have saddle points rather than local minima as N increases. Indeed the ratio of the number of saddle points to local minima increases *exponentially* with the dimensionality N.

A typical problem for both local minima and saddle-points is that they are often surrounded by plateaus of small curvature in the error. While gradient descent dynamics are repelled away from a saddle point to lower error by following directions of negative curvature, this repulsion can occur slowly due to the plateau. Second order methods, like the Newton method, are designed to rapidly descend plateaus surrounding local minima by multiplying the gradient steps with the inverse of the Hessian matrix. However, the Newton method does not treat saddle points appropriately; as argued below, saddle-points instead become *attractive* under the Newton dynamics.

Thus, given the proliferation of saddle points, not local minima, in high dimensional problems, the entire theoretical justification for quasi-Newton methods, i.e. the ability to rapidly descend to the bottom of a convex local minimum, becomes less relevant in high dimensional non-convex optimization. In this work, which is an extension of the previous report Pascanu et al. (2014), we first want to raise awareness of this issue, and second, propose an alternative approach to second-order optimization that aims to rapidly escape from saddle points. This algorithm

leverages second-order curvature information in a fundamentally different way than quasi-Newton methods, and also, in numerical experiments, outperforms them in some high dimensional problems involving deep or recurrent networks.

10.1 The prevalence of saddle points in high dimensions

Here we review arguments from disparate literatures suggesting that saddle points, not local minima, provide a fundamental impediment to rapid high dimensional non-convex optimization. One line of evidence comes from statistical physics. Bray and Dean (2007); Fyodorov and Williams (2007) study the nature of critical points of random Gaussian error functions on high dimensional continuous domains using replica theory (see Parisi (2007) for a recent review of this approach).

One particular result by Bray and Dean (2007) derives how critical points are distributed in the ϵ vs α plane, where α is the index, or the fraction of negative eigenvalues of the Hessian at the critical point, and ϵ is the error attained at the critical point. Within this plane, critical points concentrate on a monotonically increasing curve as α ranges from 0 to 1, implying a strong correlation between the error ϵ and the index α: the larger the error the larger the index. The probability of a critical point to be an $O(1)$ distance off the curve is exponentially small in the dimensionality N, for large N. This implies that critical points with error ϵ much larger than that of the global minimum, are exponentially likely to be saddle points, with the fraction of negative curvature directions being an increasing function of the error. Conversely, all local minima, which necessarily have index 0, are likely to have an error very close to that of the global minimum. Intuitively, *in high dimensions, the chance that all the directions around a critical point lead upward (positive curvature) is exponentially small* w.r.t. the number of dimensions, unless the critical point is the global minimum or stands at an error level close to it, i.e., it is unlikely one can find a way to go further down.

These results may also be understood via random matrix theory. We know that for a large Gaussian random matrix the eigenvalue distribution follows Wigner's famous semicircular law (Wigner, 1958), with both mode and mean at 0. The

probability of an eigenvalue to be positive or negative is thus $\frac{1}{2}$. Bray and Dean (2007) showed that the eigenvalues of the Hessian at a critical point are distributed in the same way, except that the semicircular spectrum is shifted by an amount determined by ϵ. For the global minimum, the spectrum is shifted so far right, that all eigenvalues are positive. As ϵ increases, the spectrum shifts to the left and accrues more negative eigenvalues as well as a density of eigenvalues around 0, indicating the typical presence of plateaus surrounding saddle points at large error. Such plateaus would slow the convergence of first order optimization methods, yielding the illusion of a local minimum.

The random matrix perspective also concisely and intuitively crystallizes the striking difference between the geometry of low and high dimensional error surfaces. For $N = 1$, an exact saddle point is a 0–probability event as it means randomly picking an eigenvalue of exactly 0. As N grows it becomes exponentially unlikely to randomly pick all eigenvalues to be positive or negative, and therefore most critical points are saddle points.

Fyodorov and Williams (2007) review qualitatively similar results derived for random error functions superimposed on a quadratic error surface. These works indicate that for typical, generic functions chosen from a random Gaussian ensemble of functions, local minima with high error are exponentially rare in the dimensionality of the problem, but saddle points with many negative and approximate plateau directions are exponentially likely. However, is this result for generic error landscapes applicable to the error landscapes of practical problems of interest?

Baldi and Hornik (1989) analyzed the error surface of a multilayer perceptron (MLP) with a single linear hidden layer. Such an error surface shows only saddle-points and *no* local minima. This result is qualitatively consistent with the observation made by Bray and Dean (2007). Indeed Saxe et al. (2014) analyzed the dynamics of learning in the presence of these saddle points, and showed that they arise due to scaling symmetries in the weight space of a deep linear MLP. These scaling symmetries enabled Saxe et al. (2014) to find new exact solutions to the nonlinear dynamics of learning in deep linear networks. These learning dynamics exhibit plateaus of high error followed by abrupt transitions to better performance. They qualitatively recapitulate aspects of the hierarchical development of semantic concepts in infants (Saxe et al., 2013).

In (Saad and Solla, 1995) the dynamics of stochastic gradient descent are an-

alyzed for soft committee machines. This work explores how well a student network can learn to imitate a randomly chosen teacher network. Importantly, it was observed that learning can go through an initial phase of *being trapped in the symmetric submanifold* of weight space. In this submanifold, the student's hidden units compute similar functions over the distribution of inputs. The slow learning dynamics within this submanifold originates from saddle point structures (caused by permutation symmetries among hidden units), and their associated plateaus (Rattray et al., 1998; Inoue et al., 2003). The exit from the plateau associated with the symmetric submanifold corresponds to the differentiation of the student's hidden units to mimic the teacher's hidden units. Interestingly, this exit from the plateau is achieved by following directions of negative curvature associated with a saddle point. sin directions perpendicular to the symmetric submanifold.

Mizutani and Dreyfus (2010) look at the effect of negative curvature on learning and implicitly at the effect of saddle points in the error surface. Their findings are similar. They show that the error surface of a single layer MLP has saddle points where the Hessian matrix is indefinite.

10.2 Experimental validation of the prevalence of saddle points

In this section, we experimentally test whether the theoretical predictions presented by Bray and Dean (2007) for random Gaussian fields hold for neural networks. To our knowledge, this is the first attempt to measure the relevant statistical properties of neural network error surfaces and to test if the theory developed for random Gaussian fields generalizes to such cases.

In particular, we are interested in how the critical points of a single layer MLP are distributed in the ϵ–α plane, and how the eigenvalues of the Hessian matrix at these critical points are distributed. We used a small MLP trained on a down-sampled version of MNIST and CIFAR-10. Newton method was used to identify critical points of the error function. The results are in Fig. 10.1. More details about the setup are provided in Appendix 10.8.3.

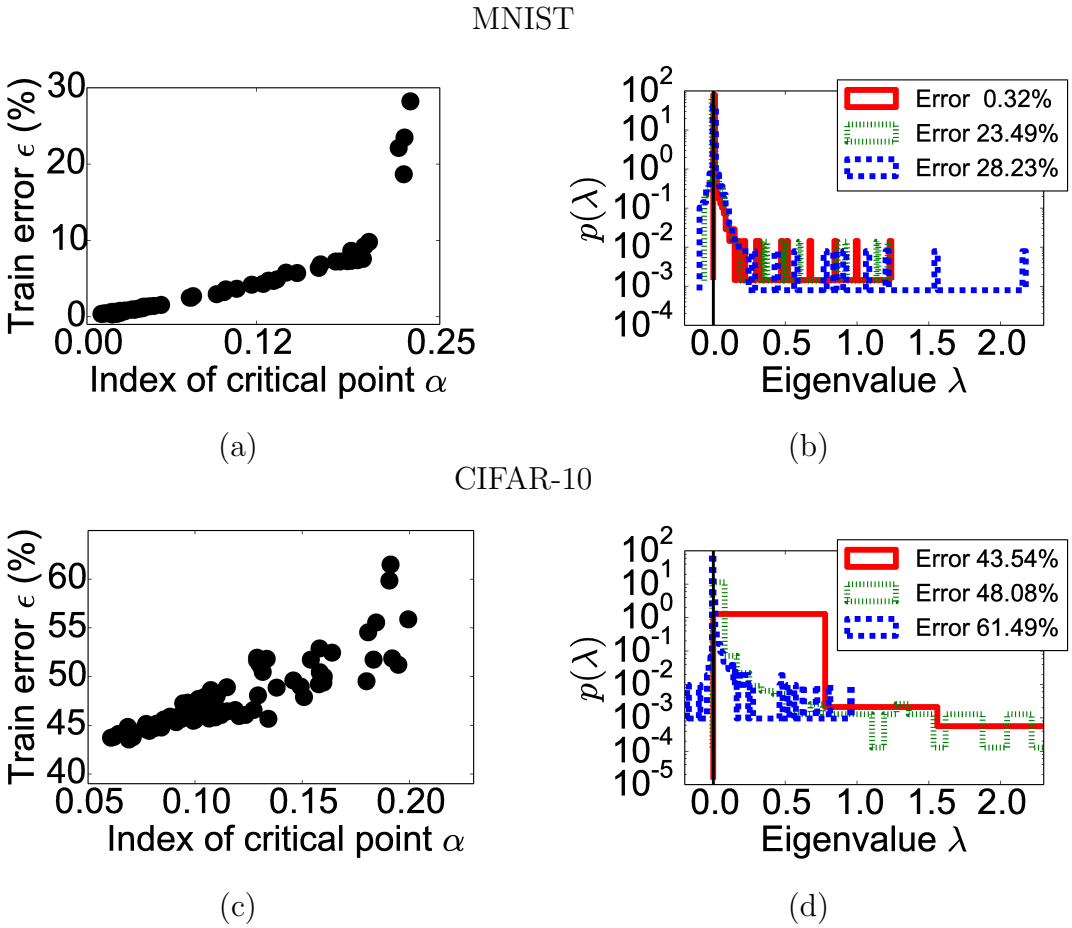


Figure 10.1: (a) and (c) show how critical points are distributed in the ϵ–α plane. Note that they concentrate along a monotonically increasing curve. (b) and (d) plot the distributions of eigenvalues of the Hessian at three different critical points. Note that the y axes are in logarithmic scale.

This empirical test confirms that the observations by Bray and Dean (2007) qualitatively hold for neural networks. Critical points concentrate along a monotonically increasing curve in the ϵ–α plane. Thus the prevalence of high error saddle points do indeed pose a severe problem for training neural networks. While the eigenvalues do not seem to be exactly distributed according to the semicircular law, their distribution does shift to the left as the error increases. The large mode at 0 indicates that there is a plateau around any critical point of the error function of a neural network.

10.3 Dynamics of optimization algorithms near saddle points

Given the prevalence of saddle points, it is important to understand how various optimization algorithms behave near them. Let us focus on non-degenerate saddle points for which the Hessian is not singular. These critical points can be locally analyzed by re-parameterizing the function according to Morse's lemma below (see chapter 7.3, Theorem 7.16 in Callahan (2010) or Appendix 10.8.2:

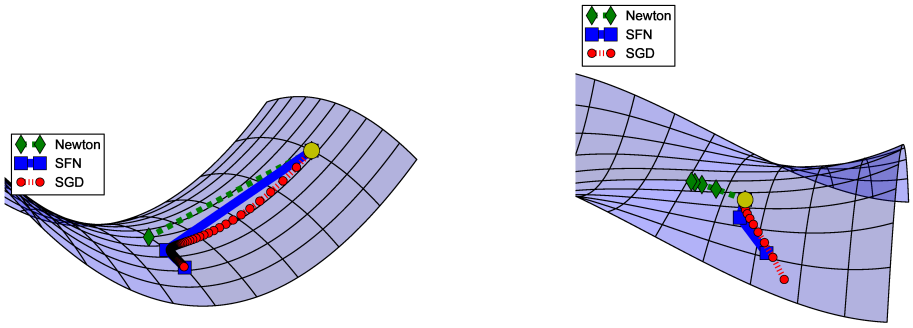
$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2}\sum_{i=1}^{n_\theta} \lambda_i \Delta \mathbf{v}_i^2, \tag{10.1}$$

where λ_i represents the ith eigenvalue of the Hessian, and $\Delta\mathbf{v}_i$ are the new parameters of the model corresponding to motion along the eigenvectors $\mathbf{e}_i$ of the Hessian of f at θ^*.

If finding the local minima of our function is the desired outcome of our optimization algorithm, we argue that an optimal algorithm would move away from the saddle point at a speed that is inverse proportional with the flatness of the error surface and hence depndented of how trustworthy this descent direction is further away from the current position.

A step of the *gradient descent* method always points away from the saddle point close to it (SGD in Fig. 10.2). Assuming equation (10.8) is a good approximation of our function we will analyze the optimality of the step according to how well the resulting $\Delta\mathbf{v}$ optimizes the right hand side of (10.8). If an eigenvalue λ_i is

positive (negative), then the step moves toward (away) from θ^* along $\Delta\mathbf{v}_i$ because the restriction of f to the corresponding eigenvector direction $\Delta\mathbf{v}_i$, achieves a minimum (maximum) at θ^*. The drawback of the gradient descent method is not the direction, but the *size* of the step along each eigenvector. The step, along any direction $\mathbf{e}_i$, is given by $-\lambda_i \Delta\mathbf{v}_i$, and so small steps are taken in directions corresponding to eigenvalues of small absolute value.



(a) (b)

Figure 10.2: Behaviors of different optimization methods near a saddle point for (a) classical saddle structure $5x^2 - y^2$; (b) monkey saddle structure $x^3 - 3xy^2$. The yellow dot indicates the starting point. SFN stands for the saddle-free Newton method we proposed.

The *Newton method* solves the slowness problem by rescaling the gradients in each direction with the inverse of the corresponding eigenvalue, yielding the step $-\Delta\mathbf{v}_i$. However, this approach can result in moving toward the saddle point. Specifically, if an eigenvalue is negative, the Newton step moves along the eigenvector in a direction *opposite* to the gradient descent step, and thus moves in the direction of θ^*. θ^* becomes an *attractor* for the Newton method (see Fig. 10.2), which can get stuck in this saddle point and not converge to a local minima. This justifies using the Newton method to find critical points of any index in Fig. 10.1.

A *trust region* approach is one approach of scaling second order methods to non-convex problems. In one such method, the Hessian is damped to remove negative curvature by adding a constant α to its diagonal, which is equivalent to adding α to each of its eigenvalues. If we project the new step along the different eigenvectors of the modified Hessian, it is equivalent to rescaling the projections of the gradient on this direction by the inverse of the modified eigenvalues $\lambda_i + \alpha$ yields the step $-\left({}^{\lambda_i}/_{\lambda_i+\alpha}\right)\Delta\mathbf{v}_i$. To ensure the algorithm does not converge to the saddle point,

one must increase the damping coefficient α enough so that $\lambda_{min} + \alpha > 0$ even for the most negative eigenvalue λ_{min}. This ensures that the modified Hessian is positive definnite. However, the drawback is again a potentially small step size in many eigen-directions incurred by a large damping factor α (the rescaling factors in each eigen-direction are not proportional to the curvature anymore).

Besides damping, another approach to deal with negative curvature is to ignore them. This can be done regardless of the approximation strategy used for the Newton method such as a truncated Newton method or a BFGS approximation (see Nocedal and Wright (2006) chapters 4 and 7). However, such algorithms cannot escape saddle points, as they ignore the very directions of negative curvature that must be followed to achieve escape.

Natural gradient descent is a first order method that relies on the curvature of the parameter manifold. That is, natural gradient descent takes a step that induces a constant change in the behaviour of the model as measured by the KL-divergence between the model before and after taking the step. The resulting algorithm is similar to the Newton method, except that it relies on the Fisher Information matrix $\mathbf{F}$.

It is argued by Rattray et al. (1998); Inoue et al. (2003) that natural gradient descent can address certain saddle point structures effectively. Specifically, it can resolve those saddle points arising from having units behaving very similarly. Mizutani and Dreyfus (2010), however, argue that natural gradient descent also suffers with negative curvature. One particular known issue is the over-realizable regime, where around the stationary solution θ^*, the Fisher matrix is rank-deficient. Numerically, this means that the Gauss-Newton direction can be orthogonal to the gradient at some distant point from θ^* (Mizutani and Dreyfus, 2010), causing optimization to converge to some non-stationary point. Another weakness is that the difference $\mathbf{S}$ between the Hessian and the Fisher Information Matrix can be large near certain saddle points that exhibit strong negative curvature. This means that the landscape close to these critical points may be dominated by $\mathbf{S}$, meaning that the rescaling provided by $\mathbf{F}^{-1}$ is not optimal in all directions.

The same is true for TONGA (Le Roux et al., 2007), an algorithm similar to natural gradient descent. It uses the covariance of the gradients as the rescaling factor. As these gradients vanish approaching a critical point, their covariance will result in much larger steps than needed near critical points.

10.4 Generalized trust region methods

In order to attack the saddle point problem, and overcome the deficiencies of the above methods, we will define a class of *generalized trust region methods*, and search for an algorithm within this space. This class involves a straightforward extension of classical trust region methods via two simple changes: (1) We allow the minimization of a first-order Taylor expansion of the function instead of always relying on a second-order Taylor expansion as is typically done in trust region methods, and (2) we replace the constraint on the norm of the step $\Delta\theta$ by a constraint on the distance between θ and $\theta + \Delta\theta$. Thus the choice of distance function and Taylor expansion order specifies an algorithm. If we define $\mathcal{T}_k(f, \theta, \Delta\theta)$ to indicate the k-th order Taylor series expansion of f around θ evaluated at $\theta+\Delta\theta$, then we can summarize a generalized trust region method as:

$$\Delta\theta = \arg\min_{\Delta\theta} \mathcal{T}_k\{f, \theta, \Delta\theta\} \quad \text{with } k \in \{1, 2\}\text{s. t. } d(\theta, \theta + \Delta\theta) \leq \Delta. \tag{10.2}$$

For example, the α-damped Newton method described above arises as a special case with $k = 2$ and $d(\theta, \theta + \Delta\theta) = ||\Delta\theta||_2^2$, where α is implicitly a function of Δ.

10.5 Attacking the saddle point problem

We now search for a solution to the saddle-point problem within the family of generalized trust region methods. In particular, the analysis of optimization algorithms near saddle points discussed in Sec. 10.3 suggests a simple heuristic solution: rescale the gradient along each eigen-direction $\mathbf{e}_i$ by $1/|\lambda_i|$. This achieves the same optimal rescaling as the Newton method, while preserving the sign of the gradient, thereby turning saddle points into repellers, not attractors, of the learning dynamics. The idea of taking the absolute value of the eigenvalues of the Hessian was suggested before. See, for example, (Nocedal and Wright, 2006, chapter 3.4) or Murray (2010, chapter 4.1). However, we are not aware of any proper justification of this algorithm or even a detailed exploration (empirical or otherwise) of this idea. One cannot simply replace $\mathbf{H}$ by $|\mathbf{H}|$, where $|\mathbf{H}|$ is the matrix obtained by taking the absolute value of each eigenvalue of $\mathbf{H}$, without proper justification. While we

might be able to argue that this heuristic modification does the right thing near critical points, is it still the right thing far away from the critical points? How can we express this step in terms of the existing methods ? Here we show this heuristic solution arises naturally from our generalized trust region approach.

Unlike classical trust region approaches, we consider minimizing a first-order Taylor expansion of the loss ($k = 1$ in Eq. (10.2)). This means that the curvature information has to come from the constraint by picking a suitable distance measure d (see Eq. (10.2)). Since the minimum of the first order approximation of f is at infinity, we know that this optimization dynamics will always jump to the border of the trust region. So we must ask how far from θ can we trust the first order approximation of f? One answer is to bound the discrepancy between the first and second order Taylor expansions of f by imposing the following constraint:

$$d(\theta, \theta + \Delta\theta) = \left| f(\theta) + \nabla f \Delta\theta + \frac{1}{2}\Delta\theta^\top \mathbf{H} \Delta\theta - f(\theta) - \nabla f \Delta\theta \right| = \frac{1}{2}\left|\Delta\theta^\top \mathbf{H} \Delta\theta\right| \leq \Delta, \tag{10.3}$$

where ∇f is the partial derivative of f with respect to θ and $\Delta \in \mathbb{R}$ is some small value that indicates how much discrepancy we are willing to accept. Note that the distance measure d takes into account the curvature of the function.

Eq. (10.3) is not easy to solve for $\Delta\theta$ in more than one dimension. Alternatively, one could take the square of the distance, but this would yield an optimization problem with a constraint that is quartic in $\Delta\theta$, and therefore also difficult to solve. We circumvent these difficulties through a Lemma:

Lemma 1. *Let* $\mathbf{A}$ *be a nonsingular square matrix in* $\mathbb{R}^n \times \mathbb{R}^n$*, and* $\mathbf{x} \in \mathbb{R}^n$ *be some vector. Then it holds that* $|\mathbf{x}^\top \mathbf{A} \mathbf{x}| \leq \mathbf{x}^\top |\mathbf{A}| \mathbf{x}$*, where* $|\mathbf{A}|$ *is the matrix obtained by taking the absolute value of each of the eigenvalues of* $\mathbf{A}$*.*

Proof. See Appendix 10.8.4 for the proof. □

Instead of the originally proposed distance measure in Eq. (10.3), we approximate the distance by its upper bound $\Delta\theta|\mathbf{H}|\Delta\theta$ based on Lemma 1. This results in the following generalized trust region method:

$$\Delta\theta = \arg\min_{\Delta\theta} f(\theta) + \nabla f \Delta\theta \quad \text{s. t. } \Delta\theta^\top |\mathbf{H}| \Delta\theta \leq \Delta. \tag{10.4}$$

Note that as discussed before, we can replace the inequality constraint with an equality one, as the first order approximation of f has a minimum at infinity

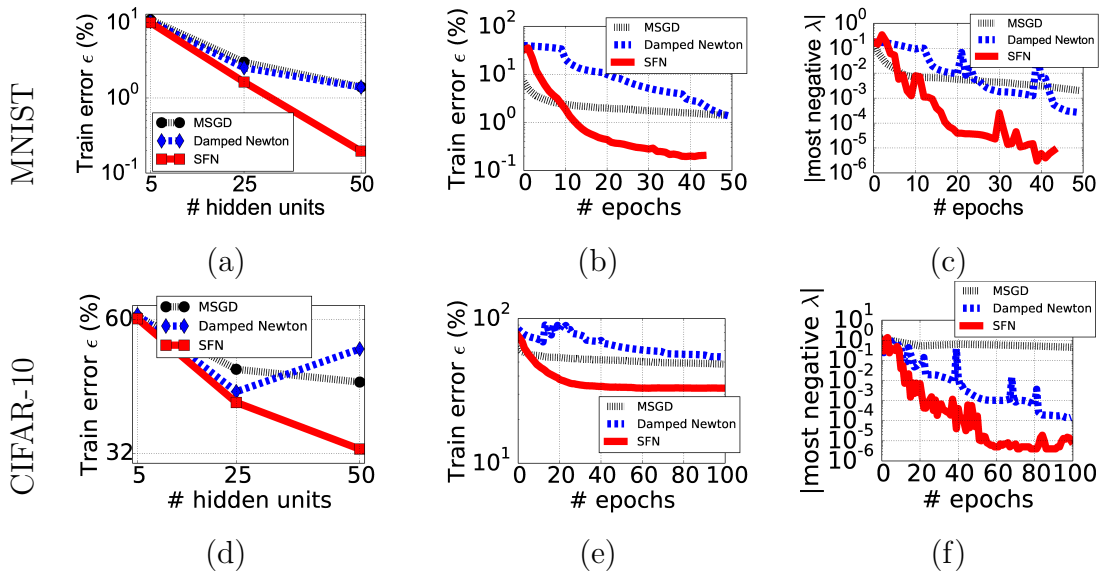


Figure 10.3: Empirical evaluation of different optimization algorithms for a single-layer MLP trained on the rescaled MNIST and CIFAR-10 dataset. In (a) and (d) we look at the minimum error obtained by the different algorithms considered as a function of the model size. (b) and (e) show the optimal training curves for the three algorithms. The error is plotted as a function of the number of epochs. (c) and (f) track the norm of the largest negative eigenvalue.

and the algorithm always jumps to the border of the trust region. Similar to (Pascanu and Bengio, 2014), we use Lagrange multipliers to obtain the solution of this constrained optimization. This gives (up to a scalar that we fold into the learning rate) a step of the form:

$$\Delta\theta = -\nabla f |\mathbf{H}|^{-1} \tag{10.5}$$

This algorithm, which we call the saddle-free Newton method (SFN), leverages curvature information in a fundamentally different way, to define the shape of the trust region, rather than Taylor expansion to second order, as in classical methods. Unlike gradient descent, it can move further (less) in the directions of low (high) curvature. It is identical to the Newton method when the Hessian is positive definite, but unlike the Newton method, it can escape saddle points. Furthermore, unlike gradient descent, the escape is rapid even along directions of weak negative curvature (see Fig. 10.2).

The exact implementation of this algorithm is intractable in a high dimensional problem, because it requires the exact computation of the Hessian. Instead we

Algorithm 3 Approximate saddle-free Newton

Input: Function $f(\theta)$ to minimize
  for $i = 1 \to M$ **do**
    $\mathbf{V} \leftarrow k$ Lanczos vectors of $\frac{\partial^2 f}{\partial \theta^2}$
    $s(\alpha) = f(\theta + \mathbf{V}\alpha)$
    $|\hat{\mathbf{H}}| \leftarrow \left|\frac{\partial^2 s}{\partial \alpha^2}\right|$ by using an eigen decomposition of $\hat{\mathbf{H}}$
    for $j = 1 \to m$ **do**
      $\mathbf{g} \leftarrow -\frac{\partial s}{\partial \alpha}$
      $\lambda \leftarrow \arg\min_\lambda s((|\hat{\mathbf{H}}| + \lambda \mathbf{I})^{-1}\mathbf{g})$
      $\theta \leftarrow \theta + \mathbf{V}(|\hat{\mathbf{H}}| + \lambda \mathbf{I})^{-1}\mathbf{g}$
    end for
  end for

use an approach similar to Krylov subspace descent (Vinyals and Povey, 2012). We optimize that function in a lower-dimensional Krylov subspace $\hat{f}(\alpha) = f(\theta + \alpha\mathbf{V})$. The k Krylov subspace vectors $\mathbf{V}$ are found through Lanczos iteration of the Hessian. These vectors will span the k biggest eigenvectors of the Hessian with high-probability. This reparametrization through α greatly reduces the dimensionality and allows us to use exact saddle-free Newton in the subspace. [1] See Alg. 3 for the pseudocode.

10.6 Experimental validation of the saddle-free Newton method

In this section, we empirically evaluate the theory suggesting the existence of many saddle points in high-dimensional functions by training neural networks.

10.6.1 Existence of Saddle Points in Neural Networks

In this section, we validate the existence of saddle points in the cost function of neural networks, and see how each of the algorithms we described earlier behaves near them. In order to minimize the effect of any type of approximation used in the

1. In the Krylov subspace, $\frac{\partial \hat{f}}{\partial \alpha} = \mathbf{V}\left(\frac{\partial f}{\partial \theta}\right)^\top$ and $\frac{\partial^2 \hat{f}}{\partial \alpha^2} = \mathbf{V}\left(\frac{\partial^2 f}{\partial \theta^2}\right)\mathbf{V}^\top$.

algorithms, we train small neural networks on the scaled-down version of MNIST and CIFAR-10, where we can compute the update directions by each algorithm exactly. Both MNIST and CIFAR-10 were downsampled to be of size 10×10.

We compare minibatch stochastic gradient descent (MSGD), damped Newton and the proposed saddle-free Newton method (SFN). The hyperparameters of SGD were selected via random search (Bergstra and Bengio, 2012b), and the damping coefficients for the damped Newton and saddle-free Newton [2] methods were selected from a small set at each update.

The theory suggests that the number of saddle points increases exponentially as the dimensionality of the function increases. From this, we expect that it becomes more likely for the conventional algorithms such as SGD and Newton methods to stop near saddle points, resulting in worse performance (on training samples). Figs. 10.3 (a) and (d) clearly confirm this. With the smallest network, all the algorithms perform comparably, but as the size grows, the saddle-free Newton algorithm outperforms the others by a large margin.

A closer look into the different behavior of each algorithm is presented in Figs. 10.3 (b) and (e) which show the evolution of training error over optimization. We can see that the proposed saddle-free Newton escapes, or does not get stuck at all, near a saddle point where both SGD and Newton methods appear trapped. Especially, at the 10-th epoch in the case of MNIST, we can observe the saddle-free Newton method rapidly escaping from the saddle point. Furthermore, Figs. 10.3 (c) and (f) provide evidence that the distribution of eigenvalues shifts more toward the right as error decreases for all algorithms, consistent with the theory of random error functions. The distribution shifts more for SFN, suggesting it can successfully avoid saddle-points on intermediary error (and large index).

2. Damping is used for numerical stability.

10.6.2 Effectiveness of saddle-free Newton Method in Deep Feedforward Neural Networks

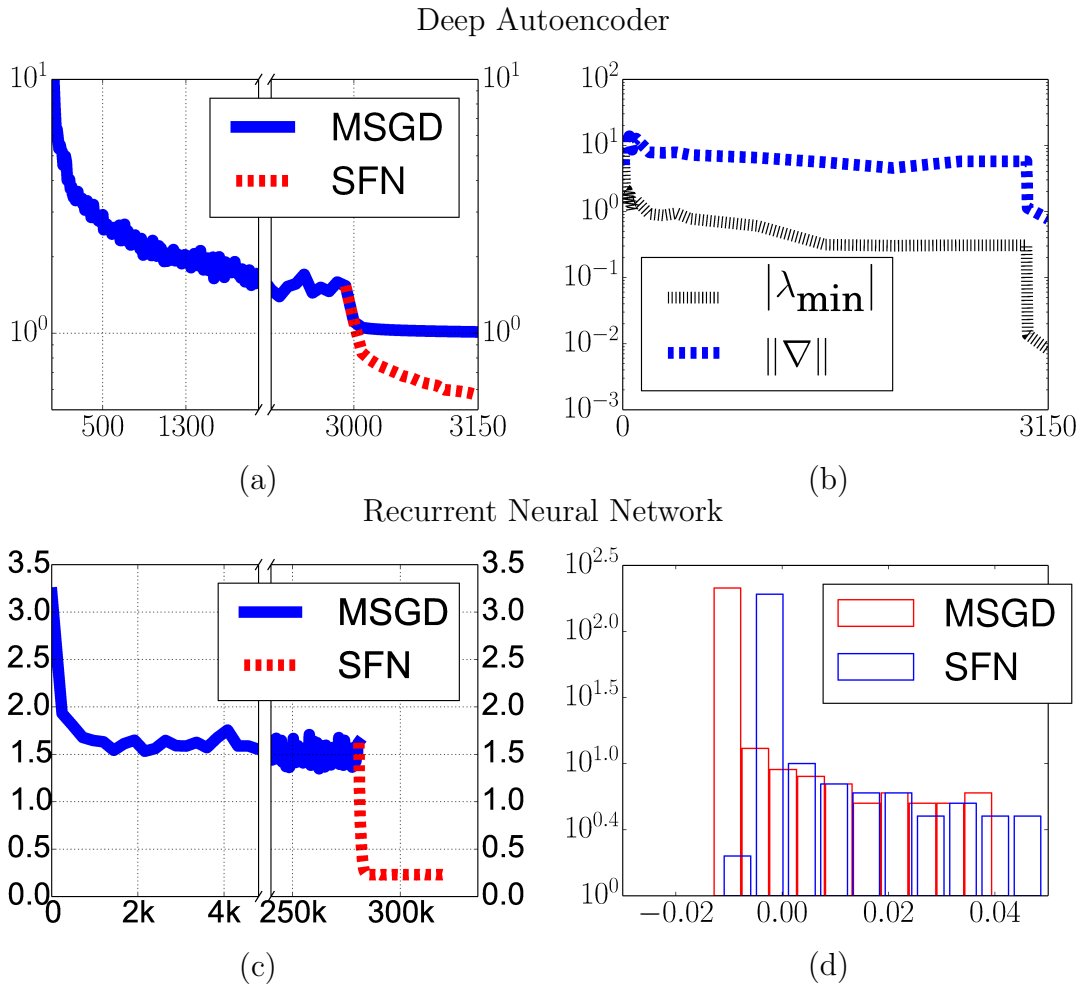


Figure 10.4: Empirical results on training deep autoencoders on MNIST and recurrent neural network on Penn Treebank. (a) and (c): The learning curve for SGD and SGD followed by saddle-free Newton method. (b) The evolution of the magnitude of the most negative eigenvalue and the norm of the gradients w.r.t. the number of epochs (deep autoencoder). (d) The distribution of eigenvalues of the RNN solutions found by SGD and the SGD continued with saddle-free Newton method.

Here, we further show the effectiveness of the proposed saddle-free Newton method in a larger neural network having seven hidden layers. The neural network is a deep autoencoder trained on (full-scale) MNIST and considered a standard benchmark problem for assessing the performance of optimization algorithms on neural networks (Sutskever et al., 2013). In this large-scale problem, we used the Krylov subspace descent approach described earlier with 500 subspace vectors.

We first trained the model with SGD and observed that learning stalls after achieving the mean-squared error (MSE) of 1.0. We then continued with the saddle-free Newton method which rapidly escaped the (approximate) plateau at which SGD was stuck (See Fig. 10.4 (a)). Furthermore, even in these large scale experiments, we were able to confirm that the distribution of Hessian eigenvalues shifts right as error decreases, and that the proposed saddle-free Newton algorithm accelerates this shift (See Fig. 10.4 (b)).

The model trained with SGD followed by the saddle-free Newton method was able to get the state-of-the-art MSE of 0.57 compared to the previous best error of 0.69 achieved by the Hessian-Free method (Martens, 2010). Saddle free Newton method does better.

10.6.3 Recurrent Neural Networks: Hard Optimization Problem

Recurrent neural networks are widely known to be more difficult to train than feedforward neural networks (see, e.g., Bengio et al., 1994; Pascanu et al., 2013). In practice they tend to underfit, and in this section, we want to test if the proposed saddle-free Newton method can help avoiding underfitting, assuming that that it is caused by saddle points. We trained a small recurrent neural network having 120 hidden units for the task of character-level language modeling on Penn Treebank corpus. Similarly to the previous experiment, we trained the model with SGD until it was clear that the learning stalled. From there on, training continued with the saddle-free Newton method.

In Fig. 10.4 (c), we see a trend similar to what we observed with the previous experiments using feedforward neural networks. The SGD stops progressing quickly and does not improve performance, suggesting that the algorithm is stuck in a plateau, possibly around a saddle point. As soon as we apply the proposed saddle-free Newton method, we see that the error drops significantly. Furthermore, Fig. 10.4 (d) clearly shows that the solution found by the saddle-free Newton has fewer negative eigenvalues, consistent with the theory of random Gaussian error functions. In addition to the saddle-free Newton method, we also tried continuing with the truncated Newton method with damping, however, without much success.

10.7 Conclusion

In summary, we have drawn from disparate literatures spanning statistical physics and random matrix theory to neural network theory, to argue that (a) non-convex error surfaces in high dimensional spaces generically suffer from a proliferation of saddle points, and (b) in contrast to conventional wisdom derived from low dimensional intuition, local minima with high error are exponentially rare in high dimensions. Moreover, we have provided the first experimental tests of these theories by performing new measurements of the statistical properties of critical points in neural network error surfaces. These tests were enabled by a novel application of Newton's method to search for critical points of any index (fraction of negative eigenvalues), and they confirmed the main qualitative prediction of theory that the index of a critical point tightly and positively correlates with its error level.

Motivated by this theory, we developed a framework of generalized trust region methods to search for algorithms that can rapidly escape saddle points. This framework allows us to leverage curvature information in a fundamentally different way than classical methods, by defining the shape of the trust region, rather than locally approximating the function to second order. Through further approximations, we derived an exceedingly simple algorithm, the saddle-free Newton method, which rescales gradients by the absolute value of the inverse Hessian. This algorithm had previously remained heuristic and theoretically unjustified, as well as numerically unexplored within the context of deep and recurrent neural networks. Our work shows that near saddle points it can achieve rapid escape by combining the best of gradient descent and Newton methods while avoiding the pitfalls of both. Moreover, through our generalized trust region approach, our work shows that this algorithm is sensible even far from saddle points. Finally, we demonstrate improved optimization on several neural network training problems.

For the future, we are mainly interested in two directions. The first direction is to explore methods beyond Kyrylov subspaces, such as one in (Sohl-Dickstein et al., 2014), that allow the saddle-free Newton method to scale to high dimensional problems, where we cannot easily compute the entire Hessian matrix. In the second direction, the theoretical properties of critical points in the problem of training a neural network will be further analyzed. More generally, it is likely that a deeper understanding of the statistical properties of high dimensional error surfaces will

guide the design of novel non-convex optimization algorithms that could impact many fields across science and engineering.

10.8 Appendix

10.8.1 Description of the different types of saddle-points

In general, consider an error function $f(\theta)$ where θ is an N dimensional continuous variable. A critical point is by definition a point θ where the gradient of $f(\theta)$ vanishes. All critical points of $f(\theta)$ can be further characterized by the curvature of the function in its vicinity, as described by the eigenvalues of the Hessian. Note that the Hessian is symmetric and hence the eigenvalues are real numbers. The following are the four possible scenarios:

- If all eigenvalues are non-zero and positive, then the critical point is a local minimum.
- If all eigenvalues are non-zero and negative, then the critical point is a local maximum.
- If the eigenvalues are non-zero and we have both positive and negative eigenvalues, then the critical point is a saddle point with a *min-max* structure (see Figure 10.5 (b)). That is, if we restrict the function f to the subspace spanned by the eigenvectors corresponding to positive (negative) eigenvalues, then the saddle point is a maximum (minimum) of this restriction.
- If the Hessian matrix is singular, then the *degenerate* critical point can be a saddle point, as it is, for example, for $\theta^3, \theta \in \mathbb{R}$ or for the monkey saddle (Figure 10.5 (a) and (c)). If it is a saddle, then, if we restrict θ to only change along the direction of singularity, the restricted function does not exhibit a minimum nor a maximum; it exhibits, to second order, a plateau. When moving from one side to other of the plateau, the eigenvalue corresponding to this picked direction generically changes sign, being exactly zero at the critical point. Note that an eigenvalue of zero can also indicate the presence of a gutter structure, a degenerate minimum, maximum or saddle, where a set of connected points are all minimum, maximum or saddle structures of the same shape and error. In Figure 10.5 (d) it is shaped as a circle. The

error function looks like the bottom of a wine bottle, where all points along this circle are minimum of equal value.

A plateau is an almost flat region in some direction. This structure is given by having the eigenvalues (which describe the curvature) corresponding to the directions of the plateau be *close to 0*, but *not exactly 0*. Or, additionally, by having a large discrepancy between the norm of the eigenvalues. This large difference would make the direction of "relative" small eigenvalues look like flat compared to the direction of large eigenvalues.

10.8.2 Reparametrization of the space around saddle-points

This reparametrization is given by taking a Taylor expansion of the function f around the critical point. If we assume that the Hessian is not singular, then there is a neighbourhood around this critical point where this approximation is reliable and, since the first order derivatives vanish, the Taylor expansion is given by:

$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2}(\Delta\theta)^\top \mathbf{H} \Delta\theta \tag{10.6}$$

Let us denote by $\mathbf{e}_1, \ldots, \mathbf{e}_{n_\theta}$ the eigenvectors of the Hessian $\mathbf{H}$ and by $\lambda_1, \ldots, \lambda_{n_\theta}$ the corresponding eigenvalues. We can now make a change of coordinates into the space span by these eigenvectors:

$$\Delta\mathbf{v} = \frac{1}{2} \begin{bmatrix} \mathbf{e}_1{}^\top \\ \ldots \\ \mathbf{e}_{n_\theta}{}^\top \end{bmatrix} \Delta\theta \tag{10.7}$$

$$f(\theta^* + \Delta\theta) = f(\theta^*) + \frac{1}{2} \sum_{i=1}^{n_\theta} \lambda_i (\mathbf{e}_i{}^\top \Delta\theta)^2 = f(\theta^*) + \sum_{i=1}^{n_\theta} \lambda_i \Delta\mathbf{v}_i^2 \tag{10.8}$$

10.8.3 Empirical exploration of properties of critical points

To obtain the plot on MNIST we used the Newton method to discover nearby critical points along the path taken by the saddle-free Newton algorithm. We consider 20 different runs of the saddle-free algorithm, each using a different random seed. We then run 200 jobs. The first 100 jobs are looking for critical points near the value of the parameters obtained after some random number of epochs (between 0 and 20) of a randomly selected run (among the 20 different runs) of

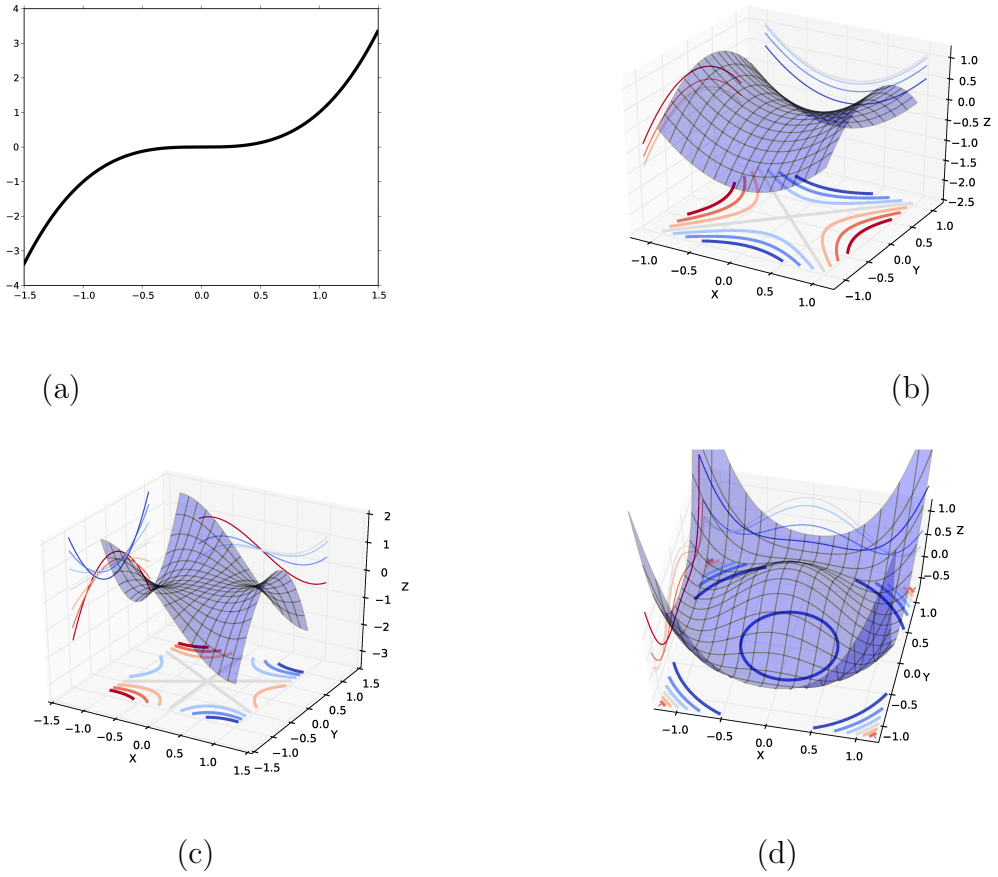


Figure 10.5: Illustrations of three different types of saddle points (a-c) plus a gutter structure (d). Note that for the gutter structure, any point from the circle $x^2 + y^2 = 1$ is a minimum. The shape of the function is that of the bottom of a bottle of wine. This means that the minimum is a "ring" instead of a single point. The Hessian is singular at any of these points. (c) shows a Monkey saddle where you have both a min-max structure as in (b) but also a 0 eigenvalue, which results, along some direction, in a shape similar to (a).

saddle-free Newton method. To this starting position uniform noise is added of small amplitude (the amplitude is randomly picked between the different values $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ The last 100 jobs look for critical points near uniformally sampled weights (the range of the weights is given by the unit cube). The task (dataset and model) is the same as the one used previously.

To obtain the plots on CIFAR, we have trained multiple 3-layer deep neural networks using SGD. The activation function of these networks is the tanh function. We saved the parameters of these networks for each epoch. We trained 100 networks with different parameter initializations between 10 and 300 epochs (chosen randomly). The networks were then trained using the Newton method to find a nearby critical point. This allows us to find many different critical points along the learning trajectories of the networks.

10.8.4 Proof of Lemma 1

Lemma 2. *Let* $\mathbf{A}$ *be a nonsingular square matrix in* $\mathbb{R}^n \times \mathbb{R}^n$*, and* $\mathbf{x} \in \mathbb{R}^n$ *be some vector. Then it holds that* $|\mathbf{x}^\top \mathbf{A}\mathbf{x}| \leq \mathbf{x}^\top |\mathbf{A}| \mathbf{x}$*, where* $|\mathbf{A}|$ *is the matrix obtained by taking the absolute value of each of the eigenvalues of* $\mathbf{A}$*.*

Proof. Let $\mathbf{e}_1, \dots \mathbf{e}_n$ be the different eigenvectors of $\mathbf{A}$ and $\lambda_1, \dots \lambda_n$ the corresponding eigenvalues. We now re-write the identity by expressing the vector $\mathbf{x}$ in terms of these eigenvalues:

$$|\mathbf{x}^\top \mathbf{A}\mathbf{x}| = \left| \sum_i (\mathbf{x}^\top \mathbf{e}_i) \mathbf{e}_i{}^\top \mathbf{A}\mathbf{x} \right| = \left| \sum_i (\mathbf{x}^\top \mathbf{e}_i) \lambda_i (\mathbf{e}_i{}^\top \mathbf{x}) \right| = \left| \sum_i \lambda_i (\mathbf{x}^\top \mathbf{e}_i)^2 \right|$$

We can now use the triangle inequality $|\sum_i x_i| \leq \sum_i |x_i|$ and get that

$$|\mathbf{x}^\top \mathbf{A}\mathbf{x}| \leq \sum_i |(\mathbf{x}^\top \mathbf{e}_i)^2 \lambda_i| = \sum_i (\mathbf{x}^\top \mathbf{e}_i) |\lambda_i| (\mathbf{e}_i{}^\top \mathbf{x}) = \mathbf{x}^\top |\mathbf{A}| \mathbf{x}$$

□

10.8.5 Implementation details for approximate saddle-free Newton

The Krylov subspace is obtained through a slightly modified Lanczos process (see Algorithm 4). The initial vector of the algorithm is the gradient of the model. As noted by Vinyals and Povey (2012), we found it was useful to include the previous search direction as the last vector of the subspace.

As described in the main paper, we have $\frac{\partial \hat{f}}{\partial \alpha} = \mathbf{V}\left(\frac{\partial f}{\partial \theta}\right)^\top$ and $\frac{\partial^2 \hat{f}}{\partial \alpha^2} = \mathbf{V}\left(\frac{\partial^2 f}{\partial \theta^2}\right)\mathbf{V}^\top$. Note that the calculation of the Hessian in the subspace can be greatly sped up by memorizing the vectors $\mathbf{V}_i \frac{\partial^2 f}{\partial \theta^2}$ during the Lanczos process. Once memorized, the Hessian is simply the product of the two matrices $\mathbf{V}$ and $\mathbf{V}_i \frac{\partial^2 f}{\partial \theta^2}$.

We have found that it is beneficial to perform multiple optimization steps within the subspace. We do not recompute the Hessian for these steps under the assumption that the Hessian will not change much.

Algorithm 4 Obtaining the Lanczos vectors

Input: $\mathbf{g} \leftarrow -\frac{\partial f}{\partial \theta}$
Input: $\Delta\theta$ (The past weight update)
  $\mathbf{V}_0 \leftarrow 0$
  $\mathbf{V}_1 \leftarrow \frac{\mathbf{g}}{\|\mathbf{g}\|}$
  $\beta_1 \leftarrow 0$
  for $i = 1 \rightarrow k-1$ **do**
    $\mathbf{w}_i \leftarrow \mathbf{V}_i \frac{\partial^2 f}{\partial \theta^2}$
    if $i = k-1$ **then**
      $\mathbf{w}_i \leftarrow \Delta\theta$
    end if
    $\alpha_i \leftarrow \mathbf{w}_i \mathbf{V}_i$
    $\mathbf{w}_i \leftarrow \mathbf{w}_i - \alpha_i \mathbf{V}_i - \beta_i \mathbf{V}_{i-1}$
    $\beta_{i+1} \leftarrow \|\mathbf{w}_i\|$
    $\mathbf{V}_{i+1} \leftarrow \frac{\mathbf{w}}{\|\mathbf{w}_i\|}$
  end for

10.8.6 Experiments

Existence of Saddle Points in Neural Networks

For feedforward networks using SGD, we choose the following hyperparameters using the random search strategy (Bergstra and Bengio, 2012b):

– Learning rate
– Size of minibatch
– Momentum coefficient

For random search, we draw 80 samples and pick the best one.

For both the Newton and saddle-free Newton methods, the damping coefficient is chosen at each update, to maximize the improvement, among $\{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.

Effectiveness of saddle-free Newton Method in Deep Neural Networks

The deep auto-encoder was first trained using the protocol used by Sutskever et al. (2013). In these experiments we use classical momentum.

Recurrent Neural Networks: Hard Optimization Problem

We initialized the recurrent weights of RNN to be orthogonal as suggested by Saxe et al. (2014). The number of hidden units of RNN is fixed to 120. For recurrent neural networks using SGD, we choose the following hyperparameters using the random search strategy:

– Learning rate
– Threshold for clipping the gradient (Pascanu et al., 2013)
– Momentum coefficient

For random search, we draw 64 samples and pick the best one. Just like in the experiment using feedforward neural networks, the damping coefficient of both the Newton and saddle-free Newton methods was chosen at each update, to maximize the improvement.

We clip the gradient and saddle-free update step if it exceeds certain threshold as suggested by Pascanu et al. (2013).

Since it is costly to compute the exact Hessian for RNN's, we used the eigenvalues of the Hessian in the Krylov subspace to plot the distribution of eigenvalues for Hessian matrix in Fig. 10.4 (d).

11 Conclusion

In this thesis we have proposed several new methods to help scale neural networks. Scaling is an important question for AI because current neural networks are still orders of magnitude smaller than those found in mice. As a case in point, scaling is one of the reasons behind the resurgence of neural networks in the mid-2000s. The state-of-the-art in several fields was broken by training much bigger but albeit classical neural networks on GPUs. Thus it stands to reason that further scaling these models will lead to improvements that will bring us closer to AI.

To this effect, we have proposed new methods in Chapters 3 and 6 to reduce the computational cost of several unsupervised learning models on high-dimensional sparse data. This is important to their application to problems in natural language processing. We have shown that these models can take advantage of sparsity through the use of importance sampling without decreasing the quality of the models learned. We observed speed-ups of orders of magnitude. What's more, the models trained using this approach achieved then state-of-the-art results, for instance of the Amazon sentiment analysis dataset and the RCV1 topic detection dataset.

Furthermore, we went beyond reducing the computational cost and proposed a new method to combat overfitting in neural networks. It is well known that increasing the capacity of a model can have detrimental consequences on its generalization. Therefore, proper generalization methods are important if we are to significantly scale these models. In chapter 8, we introduce a new kind of data-dependent regularizer. This regularizer relies on enforcing invariance of the classifier to learned transformations of the data. We were able to show that the transformations (rotation/translation) in several problems like MNIST, CIFAR and RCV1 could be learned in an unsupervised fashion using contractive auto-encoders. This method was able to reach then state-of-the-art results on the permutation invariance MNIST with 0.81% test error.

Finally, we have shown that training neural networks don't suffer as much from

local minima as previously thought. This may explain some of the success in training them in spite of the fears about local minima. We have confirmed experimentally in Chapter 10 the theoretical arguments brought forth by statistical physicists to show that there are exponentially more saddle points in high-dimensional loss surface than local minima. We have shown that properly addressing the challenges brought upon by saddle points with the saddle-free Newton method can significantly improve performance over the damped Newton method for non-convex problems.

As future work, I am working on a new optimizer tailored for non-convex problems. In particular, I am interested in devising an algorithm which can take better advantage of the computational resources at hand. Currently, training neural networks does not typically take advantage of multiple GPUs. Practitioners take advantage of additional resources by cross-validating different hyper-parameters on different machines. While that may have worked in the past, training on some of the current benchmarks takes up to a month. Thus it would be wise to use additional computational resources to actually reduce the training time by distributing training.

Bibliography

(-1). *Journal of Machine Learning Research*.

(-1). Advances in neural information processing systems 26 (nips'13). In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. NIPS Foundation (http://books.nips.cc).

(-1, April). International conference on learning representations 2014. In *International Conference on Learning Representations 2014(Conference Track)*.

(-1a). Proceedings of the 30th international conference on machine learning (icml'13). In *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. ACM.

(-1b). Proceedings of the twenty-eight international conference on machine learning (icml'11). In *Proceedings of theTwenty-eight International Conference on Machine Learning (ICML'11)*.

Alain, G. and Y. Bengio (2013). What regularized auto-encoders learn from the data generating distribution. In *International Conference on Learning Representations (ICLR'2013)*.

Baldi, P. and K. Hornik (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks 2*, 53–58.

Bastien, F., P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bengio, Y. (2008). Neural net language models. *Scholarpedia 3*(1), 3881.

Bengio, Y. (2009a). *Learning deep architectures for AI*. Now Publishers.

Bengio, Y. (2009b). Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*(1), 1–127. Also published as a book. Now Publishers, 2009.

Bengio, Y., A. Courville, and P. Vincent (2012). Representation learning: A review and new perspectives. Technical report, arXiv:1206.5538.

Bengio, Y., A. Courville, and P. Vincent (2013, August). Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 35*(8), 1798–1828.

Bengio, Y. and O. Delalleau (2009, June). Justifying and generalizing contrastive divergence. *Neural Computation 21*(6), 1601–1621.

Bengio, Y., R. Ducharme, and P. Vincent (2001). A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems 13 (NIPS'00)*, pp. 932–938. MIT Press.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007a). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pp. 153–160. MIT Press.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007b). Greedy layer-wise training of deep networks. In *NIPS'2006*.

Bengio, Y., H. Larochelle, and P. Vincent (2006). Non-local manifold Parzen windows. In Y. Weiss, B. Schölkopf, and J. Platt (Eds.), *Advances in Neural Information Processing Systems 18 (NIPS'05)*, pp. 115–122. MIT Press.

Bengio, Y., G. Mesnil, Y. Dauphin, and S. Rifai (2013). Better mixing via deep representations. See ICM (1a).

Bengio, Y. and M. Monperrus (2005). Non-local manifold tangent learning. In L. Saul, Y. Weiss, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 17 (NIPS'04)*, pp. 129–136. MIT Press.

Bengio, Y. and J.-S. Sénécal (2003). Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*.

Bengio, Y. and J.-S. Sénécal (2008). Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Trans. Neural Networks 19*(4), 713–722.

Bengio, Y., P. Simard, and P. Frasconi (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks 5*(2), 157–166. Special Issue on Recurrent Neural Networks, March 94.

Bengio, Y., E. Thibodeau-Laufer, and J. Yosinski (2014). Deep generative stochastic networks trainable by backprop. In *Proceedings of the Thirty-one International Conference on Machine Learning (ICML'14)*.

Bengio, Y., L. Yao, G. Alain, and P. Vincent (2013). Generalized denoising auto-encoders as generative models. See NIP (1).

Bergstra, J. and Y. Bengio (2012a, February). Random search for hyper-parameter optimization. *Journal of Machine Learning Research 13*, 281–305.

Bergstra, J. and Y. Bengio (2012b). Random search for hyper-parameter optimization. *J. Machine Learning Res. 13*, 281–305.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the Association for Computational Linguistics (ACL'07)*, pp. 440–447.

Bourlard, H. and Y. Kamp (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics 59*, 291–294.

Brand, M. (2003). Charting a manifold. In S. Becker, S. Thrun, and K. Obermayer (Eds.), *Advances in Neural Information Processing Systems 15 (NIPS'02)*, pp. 961–968. MIT Press.

Bray, A. J. and D. S. Dean (2007, Apr). Statistics of critical points of gaussian fields on large-dimensional spaces. *Phys. Rev. Lett. 98*, 150201.

Callahan, J. (2010). *Advanced Calculus: A Geometric View*. Undergraduate Texts in Mathematics. Springer.

Cayton, L. (2005). Algorithms for manifold learning. Technical Report CS2008-0923, UCSD.

Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2014). The loss surface of multilayer networks. *arXiv preprint arXiv:1412.0233*.

Coates, A. and A. Y. Ng (2011). The importance of encoding versus training with sparse coding and vector quantization. In *ICML'2011*.

Collobert, R. and J. Weston (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 160–167. ACM.

Dahl, G., R. Adams, and H. Larochelle (2012a, July). Training restricted boltzmann machines on word observations. In J. Langford and J. Pineau (Eds.), *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, New York, NY, USA, pp. 679–686. Omnipress.

Dahl, G. E., R. P. Adams, and H. Larochelle (2012b). Training restricted Boltzmann machines on word observations. In *ICML'2012*.

Dahl, G. E., M. Ranzato, A. Mohamed, and G. E. Hinton (2010). Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*.

Dahl, G. E., D. Yu, L. Deng, and A. Acero (2012). Context-dependent pre-trained deep neural networks for large vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing 20*(1), 33–42.

Dauphin, Y. and Y. Bengio (2013). Stochastic ratio matching of RBMs for sparse high-dimensional inputs. See NIP (1).

Dauphin, Y., X. Glorot, and Y. Bengio (2011, June). Large-scale learning of embeddings with reconstruction sampling. See ICM (1b).

Dauphin, Y., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014a). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS'2014*.

Dauphin, Y. N., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014b). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems*, pp. 2933–2941.

Deerwester, S., S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science 41*(6), 391–407.

Delalleau, O. and Y. Bengio (2011). Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*, pp. 666–674.

Deoras, A. and R. Sarikaya (2013). Deep belief network based semantic taggers for spoken language understanding. In *INTERSPEECH*, pp. 2713–2717.

Drucker, H. and Y. LeCun (1992). Improving generalisation performance using double back-propagation. *IEEE Transactions on Neural Networks 3*(6), 991–997.

Erhan, D., Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio (2010, February). Why does unsupervised pre-training help deep learning? SeeJML (1), pp. 625–660.

Erhan, D., A. Courville, Y. Bengio, and P. Vincent (2010, May). Why does unsupervised pre-training help deep learning? In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Volume 9, pp. 201–208.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). LIBLINEAR: A library for large linear classification. *9*, 1871–1874.

Fyodorov, Y. V. and I. Williams (2007). Replica symmetry breaking condition exposed by random matrix calculation of landscape complexity. *Journal of Statistical Physics 129*(5-6), 1081–1116.

Ge, R., F. Huang, C. Jin, and Y. Yuan (2015, March). Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition. *ArXiv e-prints*.

Glorot, X. and Y. Bengio (2010, May). Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Volume 9, pp. 249–256.

Glorot, X., A. Bordes, and Y. Bengio (2011a, April). Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

Glorot, X., A. Bordes, and Y. Bengio (2011b). Deep sparse rectifier neural networks.

Glorot, X., A. Bordes, and Y. Bengio (2011c, June). Domain adaptation for large-scale sentiment classification: A deep learning approach. See ICM (1b), pp. 97–110.

Goodfellow, I. (2013). Piecewise linear multilayer perceptrons and dropout. Technical report, arXiv:1301.5088.

Goodfellow, I., Q. Le, A. Saxe, and A. Ng (2009). Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 646–654.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. See ICM (1a), pp. 1319–1327.

Goto, K. and R. A. v. d. Geijn (2008, May). Anatomy of high-performance matrix multiplication. *ACM Transactions Mathematical Software 34*, 12:1–12:25.

Hayes-Roth, F., D. Waterman, and D. Lenat (1984). Building expert systems.

Hazan, E., K. Y. Levy, and S. Shalev-Swartz (2015, March). On Graduated Optimization for Stochastic Non-Convex Problems. *ArXiv e-prints*.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst 1986, pp. 1–12. Lawrence Erlbaum, Hillsdale.

Hinton, G. E., S. Osindero, and Y. Teh (2006a). A fast learning algorithm for deep belief nets. *Neural Computation 18*, 1527–1554.

Hinton, G. E., S. Osindero, and Y.-W. Teh (2006b). A fast learning algorithm for deep belief nets. *Neural Computation 18*, 1527–1554.

Hinton, G. E. and R. R. Salakhutdinov (2006, July). Reducing the dimensionality of data with neural networks. *Science 313*(5786), 504–507.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural networks 2*(5), 359–366.

Hyvärinen, A. (2005). Estimation of non-normalized statistical models using score matching. *Journal of Machine Learning Research 6*, 695–709.

Hyvärinen, A. (2007). Some extensions of score matching. *Computational Statistics and Data Analysis 51*, 2499–2512.

Inoue, M., H. Park, and M. Okada (2003). On-line learning theory of soft committee machines with correlated hidden units –steepest gradient descent and natural gradient descent–. *Journal of the Physical Society of Japan 72*(4), 805–810.

Japkowicz, N., S. J. Hanson, and M. A. Gluck (2000). Nonlinear autoassociation is not equivalent to PCA. *Neural Computation 12*(3), 531–545.

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153. IEEE.

Kavukcuoglu, K., M. Ranzato, R. Fergus, and Y. LeCun (2009). Learning invariant features through topographic filter maps. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'09)*, pp. 1605–1612. IEEE.

Kim, K., I. W. Harry, K. A. Hodge, Y.-M. Kim, C.-H. Lee, H. K. Lee, J. J. Oh, S. H. Oh, and E. J. Son (2014). Application of artificial neural network to search for gravitational-wave signals associated with short gamma-ray bursts. *arXiv preprint arXiv:1410.6878*.

Kira, K. and L. A. Rendell (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 129–134.

Krizhevsky, A., I. Sutskever, and G. Hinton (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Larochelle, H., D. Erhan, A. Courville, J. Bergstra, and Y. Bengio (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In Z. Ghahramani (Ed.), *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, pp. 473–480. ACM.

Larochelle, H., M. I. Mandel, R. Pascanu, and Y. Bengio (2012). Learning algorithms for the classification restricted boltzmann machine. *Journal of Machine Learning Research 13*, 643–669.

Lasserre, J. A., C. M. Bishop, and T. P. Minka (2006). Principled hybrids of generative and discriminative models. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'06)*, Washington, DC, USA, pp. 87–94. IEEE Computer Society.

Lauly, S., H. Larochelle, M. Khapra, B. Ravindran, V. C. Raykar, and A. Saha (2014). An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*, pp. 1853–1861.

Lawson, C. L., R. J. Hanson, D. R. Kincaid, and F. T. Krogh (1979, September). Basic linear algebra subprograms for fortran usage. *ACM Transactions Mathematical Software 5*, 308–323.

Le, Q., M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Ng (2012). Building high-level features using large scale unsupervised learning. In *ICML'2012*.

Le, Q. V., A. Karpenko, J. Ngiam, and A. Y. Ng (2011). ICA with reconstruction cost for efficient overcomplete feature learning. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems 24*, pp. 1017–1025.

Le Roux, N. and Y. Bengio (2010, August). Deep belief networks are compact universal approximators. *Neural Computation 22*(8), 2192–2207.

Le Roux, N., P.-A. Manzagol, and Y. Bengio (2007). Topmoumoute online natural gradient algorithm. *Advances in Neural Information Processing Systems*.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag.

LeCun, Y., P. Haffner, L. Bottou, and Y. Bengio (1999). Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pp. 319–345. Springer.

Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. Montreal, Canada: ACM.

Lewis, D. D., Y. Yang, T. G. Rose, and F. Li (2004, December). Rcv1: A new benchmark collection for text categorization research. *5*, 361–397.

Lewis, D. D., Y. Yang, T. G. Rose, F. Li, G. Dietterich, and F. Li (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research 5*, 361–397.

Lowe, D. (1999). Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, Volume 2, pp. 1150–1157 vol.2.

Marlin, B., K. Swersky, B. Chen, and N. de Freitas (2010). Inductive principles for restricted Boltzmann machine learning. In *Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, Volume 9, pp. 509–516.

Martens, J. (2010). Deep learning via hessian-free optimization. In *International Conference in Machine Learning*, pp. 735–742.

Mesnil, G., Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. J. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent, A. Courville, and J. Bergstra (2012). Unsupervised and transfer learning challenge: a deep learning approach. In I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver (Eds.), *JMLR W& CP: Proceedings of the Unsupervised and Transfer Learning challenge and workshop*, Volume 27, pp. 97–110.

Miikkulainen, R. and M. G. Dyer (1991). Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science 15*, 343–399.

Mikolov, T., A. Deoras, S. Kombrink, L. Burget, and J. Cernocky (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Proc. 12th annual conference of the international speech communication association (INTERSPEECH 2011)*.

Mizutani, E. and S. Dreyfus (2010). An analysis on negative curvature induced by singularity in multi-layer neural-network learning. In *Advances in Neural Information Processing Systems*, pp. 1669–1677.

Mnih, A. and G. E. Hinton (2009). A scalable hierarchical distributed language model. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 1081–1088.

Morin, F. and Y. Bengio (2005). Hierarchical probabilistic neural network language model. In R. G. Cowell and Z. Ghahramani (Eds.), *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, pp. 246–252.

Murray, W. (2010). Newton-type methods. Technical report, Department of Management Science and Engineering, Stanford University.

Narayanan, H. and S. Mitter (2010). Sample complexity of testing the manifold hypothesis. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 23*, pp. 1786–1794.

Nocedal, J. and S. Wright (2006). *Numerical Optimization*. Springer.

Parisi, G. (2007). Mean field theory of spin glasses: statistics and dynamics. Technical Report Arxiv 0706.0094.

Pascanu, R. and Y. Bengio (2014, April). Revisiting natural gradient for deep networks. See ICL (1).

Pascanu, R., Y. Dauphin, S. Ganguli, and Y. Bengio (2014). On the saddle point problem for non-convex optimization. Technical Report Arxiv 1405.4604.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). On the difficulty of training recurrent neural networks. In *ICML'2013*.

Pascanu, R., G. Montufar, and Y. Bengio (2014, April). On the number of inference regions of deep feed forward networks with piece-wise linear activations. See ICL (1).

Ranzato, M., F. Huang, Y. Boureau, and Y. LeCun (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press.

Ranzato, M., C. Poultney, S. Chopra, and Y. LeCun (2007). Efficient learning of sparse representations with an energy-based model. In *NIPS'2006*.

Rasmus, A., T. Raiko, and H. Valpola (2014). Denoising autoencoder with modulated lateral connections learns invariant representations of natural images. *CoRR abs/1412.7210*.

Rasmussen, C. E. and C. K. I. Williams (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Rattray, M., D. Saad, and S. I. Amari (1998, December). Natural Gradient Descent for On-Line Learning. *Physical Review Letters 81*(24), 5461–5464.

Rifai, S., Y. Bengio, A. Courville, P. Vincent, and M. Mirza (2012). Disentangling factors of variation for facial expression recognition. In *European Conference on Computer Vision*.

Rifai, S., Y. Bengio, Y. Dauphin, and P. Vincent (2012). A generative process for sampling contractive auto-encoders. In *Proceedings of the Twenty-nine International Conference on Machine Learning (ICML'12)*. ACM.

Rifai, S., Y. Dauphin, P. Vincent, Y. Bengio, and X. Muller (2011). The manifold tangent classifier. In *NIPS'2011*. Student paper award.

Rifai, S., G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot (2011). Higher order contractive auto-encoder. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011, June). Contractive auto-encoders: Explicit invariance during feature extraction. See ICM (1b).

Roweis, S. and L. K. Saul (2000, December). Nonlinear dimensionality reduction by locally linear embedding. *Science 290*(5500), 2323–2326.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature 323*, 533–536.

Saad, D. and S. A. Solla (1995, Oct). On-line learning in soft committee machines. *Physical Review E 52*, 4225–4243.

Salakhutdinov, R. and G. Hinton (2009a). Deep Boltzmann machines. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, Volume 8.

Salakhutdinov, R. and G. E. Hinton (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS'07)*, San Juan, Porto Rico. Omnipress.

Salakhutdinov, R. and G. E. Hinton (2009b). Deep Boltzmann machines. In *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS'09)*, Volume 5, pp. 448–455.

Salakhutdinov, R. and I. Murray (2008). On the quantitative analysis of deep belief networks. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings*

of the Twenty-fifth International Conference on Machine Learning (ICML'08), Volume 25, pp. 872–879. ACM.

Saxe, A., J. McClelland, and S. Ganguli (2013). Learning hierarchical category structure in deep neural networks. *Proceedings of the 35th annual meeting of the Cognitive Science Society*, 1271–1276.

Saxe, A., J. McClelland, and S. Ganguli (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *International Conference on Learning Representations*.

Schutze, H. (1993). Word space. In C. Giles, S. Hanson, and J. Cowan (Eds.), *Advances in Neural Information Processing Systems 5 (NIPS'92)*, San Mateo CA, pp. 895–902. Morgan Kaufmann.

Schwenk, H. and J.-L. Gauvain (2002). Connectionist language modeling for large vocabulary continuous speech recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Orlando, Florida, pp. 765–768.

Seide, F., G. Li, and D. Yu (2011). Conversational speech transcription using context-dependent deep neural networks. In *Interspeech 2011*, pp. 437–440.

Simard, P., B. Victorri, Y. LeCun, and J. Denker (1992). Tangent prop - A formalism for specifying selected invariances in an adaptive network. In J. M. S. Hanson and R. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4 (NIPS'91)*, San Mateo, CA, pp. 895–903. Morgan Kaufmann.

Simard, P. Y., Y. LeCun, and J. Denker (1993). Efficient pattern recognition using a new transformation distance. In C. Giles, S. Hanson, and J. Cowan (Eds.), *Advances in Neural Information Processing Systems 5 (NIPS'92)*, pp. 50–58. Morgan Kaufmann, San Mateo.

Socher, R., C. Manning, and A. Y. Ng (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning (ICML'2011)*.

Sohl-Dickstein, J., B. Poole, and S. Ganguli (2014). Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *ICML'2014*.

Sutskever, I. (2012). *Training Recurrent Neural Networks*. Ph. D. thesis, Department of computer science, University of Toronto.

Sutskever, I., J. Martens, G. E. Dahl, and G. E. Hinton (2013, May). On the importance of initialization and momentum in deep learning. In S. Dasgupta and D. Mcallester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Volume 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings.

Swersky, K., M. Ranzato, D. Buchman, B. Marlin, and N. de Freitas (2011). On autoencoders and score matching for energy based models. In *ICML'2011*. ACM.

Tenenbaum, J., V. de Silva, and J. C. Langford (2000, December). A global geometric framework for nonlinear dimensionality reduction. *Science 290*(5500), 2319–2323.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In *ICML'2008*, pp. 1064–1071.

Tikhonov, A. N. and V. Y. Arsenin (1977). *Solutions of Ill-posed Problems*. Washington D.C.: W. H. Winston.

Trebar, M. and N. Steele (2008). Application of distributed SVM architectures in classifying forest data cover types. *Computers and Electronics in Agriculture 63*(2), 119 – 130.

Valiant, L. G. (1984, November). A theory of the learnable. *Commun. ACM 27*(11), 1134–1142.

van der Maaten, L. and G. E. Hinton (2008, November). Visualizing data using t-SNE. *Journal of Machine Learning Research 9*, 2579–2605.

Vapnik, V. N. (1999). An overview of statistical learning theory. *Neural Networks, IEEE Transactions on 10*(5), 988–999.

Vincent, P. (2010, November). A connection between score matching and denoising autoencoders. Technical Report 1358, Université de Montréal, DIRO.

Vincent, P. (2011, July). A connection between score matching and denoising autoencoders. *Neural Computation 23*(7), 1661–1674.

Vincent, P. (2014). Efficient exact gradient update for training deep networks with very large sparse targets. *arXiv preprint arXiv:1412.7091*.

Vincent, P. and Y. Bengio (2003). Manifold Parzen windows. In *NIPS'2002*. MIT Press.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 1096–1103. ACM.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010, December). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. SeeJML (1), pp. 3371–3408.

Vinyals, O. and D. Povey (2012). Krylov Subspace Descent for Deep Learning. In *AISTATS*.

Wang, S. and C. Manning (2013). Fast dropout training. In *ICML'2013*.

Weston, J., F. Ratle, and R. Collobert (2008). Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, New York, NY, USA, pp. 1168–1175. ACM.

Wigner, E. P. (1958). On the distribution of the roots of certain symmetric matrices. *The Annals of Mathematics 67*(2), 325–327.

Younes, L. (1999). On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics and Stochastic Reports 65*(3), 177–228.

Yu, K. and T. Zhang (2010, June). Improved local coordinate coding using local tangents. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*. ACM.

Yu, K., T. Zhang, and Y. Gong (2009). Nonlinear learning using local coordinate coding. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and

A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22*, pp. 2223–2231.

Zeiler, M. D. and R. Fergus (2013). Stochastic pooling for regularization of deep convolutional neural networks. Technical Report Arxiv 1301.3557.