Université de Montréal

Deep learning of representations and its application to computer vision

par Ian Goodfellow

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

Avril, 2014

Résumé

L'objectif de cette thèse par articles est de présenter modestement quelques étapes du parcours qui mènera (on espère) à une solution générale du problème de l'intelligence artificielle. Cette thèse contient quatre articles qui présentent chacun une différente nouvelle méthode d'inférence perceptive en utilisant l'apprentissage machine et, plus particulièrement, les réseaux neuronaux profonds. Chacun de ces documents met en évidence l'utilité de sa méthode proposée dans le cadre d'une tâche de vision par ordinateur. Ces méthodes sont applicables dans un contexte plus général, et dans certains cas elles ont été appliquées ailleurs, mais ceci ne sera pas abordé dans le contexte de cette de thèse.

Dans le premier article, nous présentons deux nouveaux algorithmes d'inférence variationelle pour le modèle génératif d'images appelé *codage parcimonieux "spike-and-slab" (CPSS)*. Ces méthodes d'inférence plus rapides nous permettent d'utiliser des modèles CPSS de tailles beaucoup plus grandes qu'auparavant. Nous démontrons qu'elles sont meilleures pour extraire des détecteur de caractéristiques quand très peu d'exemples étiquetés sont disponibles pour l'entraînement. Partant d'un modèle CPSS, nous construisons ensuite une architecture profonde, la *machine de Boltzmann profonde partiellement dirigée (MBP-PD)*. Ce modèle a été conçu de manière à simplifier d'entraînement des machines de Boltzmann profondes qui nécessitent normalement une phase de pré-entraînement glouton pour chaque couche. Ce problème est réglé dans une certaine mesure, mais le coût d'inférence dans le nouveau modèle est relativement trop élevé pour permettre de l'utiliser de manière pratique.

Dans le deuxième article, nous revenons au problème d'entraînement joint de machines de Boltzmann profondes. Cette fois, au lieu de changer de famille de modèles, nous introduisons un nouveau critère d'entraînement qui donne naissance aux *machines de Boltzmann profondes à multiples prédictions (MBP-MP)*. Les MBP-MP sont entraînables en une seule étape et ont un meilleur taux de succès en classification que les MBP classiques. Elles s'entraînent aussi avec des méthodes variationelles standard au lieu de nécessiter un classificateur discriminant pour obtenir un bon taux de succès en classification. Par contre, un des inconvénients de tels modèles est leur incapacité de générer des échantillons, mais ceci n'est pas trop grave puisque la performance de classification des machines de Boltzmann profondes n'est plus une priorité étant donné les dernières avancées en apprentissage supervisé. Malgré cela, les MBP-MP demeurent intéressantes parce qu'elles sont capable d'accomplir certaines tâches que des modèles purement supervisés ne peuvent pas faire, telles que celle de classifier des données incomplètes ou encore celle de combler intelligemment l'information manquante dans ces données incomplètes.

Le travail présenté dans cette thèse s'est déroulé au milieu d'une période de transformations importantes du domaine de l'apprentissage à réseaux neuronaux profonds qui a été déclenchée par la découverte de l'algorithme de "dropout" par Geoffrey Hinton. Dropout rend possible un entraînement purement supervisé d'architectures de propagation unidirectionnel sans être exposé au danger de surentraînement. Le troisième article présenté dans cette thèse introduit une nouvelle fonction d'activation spécialement conçue pour aller avec l'algorithme de Dropout. Cette fonction d'activation, appelée *maxout*, permet l'utilisation de aggrégation multi-canal dans un contexte d'apprentissage purement supervisé. Nous démontrons comment plusieurs tâches de reconnaissance d'objets sont mieux accomplies par l'utilisation de *maxout*.

Pour terminer, sont présentons un vrai cas d'utilisation dans l'industrie pour la transcription d'adresses de maisons à plusieurs chiffres. En combinant maxout avec une nouvelle sorte de couche de sortie pour des réseaux neuronaux de convolution, nous démontrons qu'il est possible d'atteindre un taux de succès comparable à celui des humains sur un ensemble de données coriace constitué de photos prises par les voitures de Google. Ce système a été déployé avec succès chez Google pour lire environ cent million d'adresses de maisons.

Mots-clés: réseau de neurones, apprentissage profond, apprentissage non supervisé, apprentissage supervisé, apprentissage semi-supervisé, machines de Boltzmann, les modèles basés sur l'énergie, l'inference variationnel, l'apprentissage variationnel, le codage parcimonieux, réseaux neuronaux de convolution, la fonction d'activation, "dropout," la reconnaissance d'objets, transcription, reconnaissance optique de caractères, géocodage, entrées manquantes

Summary

The goal of this thesis is to present a few small steps along the road to solving general artificial intelligence. This is a thesis by articles containing four articles. Each of these articles presents a new method for performing perceptual inference using machine learning and deep architectures. Each of these papers demonstrates the utility of the proposed method in the context of a computer vision task. The methods are more generally applicable and in some cases have been applied to other kinds of tasks, but this thesis does not explore such applications.

In the first article, we present two fast new variational inference algorithms for a generative model of images known as *spike-and-slab sparse coding (S3C)*. These faster inference algorithms allow us to scale spike-and-slab sparse coding to unprecedented problem sizes and show that it is a superior feature extractor for object recognition tasks when very few labeled examples are available. We then build a new deep architecture, the *partially-directed deep Boltzmann machine (PD-DBM)* on top of the S3C model. This model was designed to simplify the training procedure for deep Boltzmann machines, which previously required a greedy layer-wise pretraining procedure. This model partially succeeds at solving this problem, but the cost of inference in the new model is high enough that it makes scaling the model to serious applications difficult.

In the second article, we revisit the problem of jointly training deep Boltzmann machines. This time, rather than changing the model family, we present a new training criterion, resulting in *multi-prediction deep Boltzmann machines (MP-DBMs)*. MP-DBMs may be trained in a single stage and obtain better classification accuracy than traditional DBMs. They also are able to classify well using standard variational inference techniques, rather than requiring a separate, specialized, discriminatively trained classifier to obtain good classification performance. However, this comes at the cost of the model not being able to generate good samples. The classification performance of deep Boltzmann machines is no longer especially interesting following recent advances in supervised learning, but the MP-DBM remains interesting because it can perform tasks that purely supervised models cannot, such as classification in the presence of missing inputs and imputation of missing inputs.

The general zeitgeist of deep learning research changed dramatically during the midst of the work on this thesis with the introduction of Geoffrey Hinton's dropout algorithm. Dropout permits purely supervised training of feedforward architectures with little overfitting. The third paper in this thesis presents a new activation function for feedforward neural networks which was explicitly designed to work well with dropout. This activation function, called *maxout*, makes it possible to learn architectures that leverage the benefits of cross-channel pooling in a purely

supervised manner. We demonstrate improvements on several object recognition tasks using this activation function.

Finally, we solve a real world task: transcription of photos of multi-digit house numbers for geo-coding. Using maxout units and a new kind of output layer for convolutional neural networks, we demonstrate human level accuracy (with limited coverage) on a challenging real-world dataset. This system has been deployed at Google and successfully used to transcribe nearly 100 million house numbers.

Keywords: neural network, deep learning, unsupervised learning, supervised learning, semi-supervised learning, Boltzmann machines, energy-based models, variational inference, variational learning, feature learning, sparse coding, convolutional networks, activation function, dropout, pooling, object recognition, transcription, optical character recognition, inpainting, missing inputs

Contents

List of Figures

List of Tables

List of Abbreviations

AIS	Annealed Importance Sampling
CD	Contrastive Divergence
CNN	Convolutional Neural Network
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
EBM	Energy-Based Model
EM	Expectation Maximization
(GP)-GPU	(General Purpose) Graphics Processing Unit
GSN	Generative Stochastic Network
I.I.D	Independent and Identically-Distributed
KL	Kullback-Leibler
LBFGS	Limited-memory Boyden-Fletcher-Goldfarb-Shanno algorithm
MAP	Maximum a posteriori
mcRBM	Mean-Covariance Restricted Boltzmann Machine
MLP	Multi-Layer Perceptron
MP-DBM	Multi-Prediction Deep Boltzmann Machine
MPT	Multi-Prediction Training
NADE	Neural Autoregressive Distribution Estimator
NN	Neural Network
OCR	Optical Character Recognition
OMP	Orthogonal Matching Pursuit
PCD	Persistent Contrastive Divergence (also SML)
PD-DBM	Partially Directed Deep Boltzmann Machine

PDF	Probability Density Function
PWL	Piece-Wise Linear
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machine
SRBM	Semi-Restricted Boltzmann Machine
S3C	Spike-and-Slab Sparse Coding
SC	Sparse Coding
SGD	Stochastic Gradient Descent
SML	Stochastic Maximum Likelihood (also PCD)
ssRBM	Spike & Slab Restricted Boltzmann Machine
SVHN	Street View House Numbers
SVM	Support Vector Machine
ZCA	Zero-phase Component Analysis

Acknowledgments

I'd like to thank many people who helped me along my path to writing this thesis.

I'd especially like to thank my thesis advisor, Yoshua Bengio, for taking me under his wing, and for running a lab where so many researchers are so free to explore creative ideas. I'd also like to thank my co-advisor Aaron Courville, for all of his advice and knowledge he has shared with me.

All of my co-authors–Aaron Courville, Yoshua Bengio, David Warde-Farley, Mehdi Mirza, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet–were a pleasure to work with, and I could not have written this thesis without them.

I'd like to thank several people at Stanford who were instrumental in getting me interested in machine learning and starting me along this path, including Jerry Cain, Andrew Ng, Daphne Koller, Ethan Dreyfuss, Stephen Gould, and Andrew Saxe.

I'd like to thank Google for awarding me the Google PhD Fellowship in Deep Learning. The fellowship has given me the freedom to spend time on projects like the Pylearn2 open source machine learning library and helping Yoshua write a textbook on deep learning.

I'd like to thank Frédéric Bastien for keeping all of the computing and software infrastructure at LISA running smoothly, and helping get convolutional networks running fast in Theano.

I'd like to thank Guillaume Alain and Nicolas Boulanger-Lewandowski for their help translating the summary of this thesis into French. I'd like to thank Kyungyun Cho, Guillaume Alain, and Paula Goodfellow for their feedback on drafts of this thesis. I'd like to thank David Warde-Farley and Nicolas Boulanger-Lewandowski for help with various LaTeX commands, and Guillaume Desjardins for letting me copy the basic LaTeX template for a Université de Montréal PhD thesis from his own.

Several members of the LISA lab made LISA a fun and intellectual atmosphere. I'd especially like to thank David Warde-Farley, Yann Dauphin, Mehdi Mirza, Li Yao, Guillaume Desjardins, James Bergstra, Razvan Pascanu, and Guillaume Alain for many good lunches, fun game nights, and interesting discussions.

I'd like to thank the people I worked with at Google for making my internship an enjoyable time and providing a lot of help and mentorship. In addition to my co-authors mentioned above, I'd especially like to thank Samy Bengio, Rajat Monga, Marc'Aurelio Ranzato, and Ilya Sutskever.

I'd like to thank my parents, Val and Paula Goodfellow, for raising me to value education education. My grandmother Jalaine was especially adamant that I pursuse a PhD.

Several people were very supportive in my personal life during the past four years. I'd like to thank Dumitru Erhan for letting me sublet his apartment when

I first arrived in Montréal. I'd like to thank David Warde-Farley for helping me throw my couch off my fourth story balcony. I'd like to thank my friend Sarah for the countless times she helped me with things like moving between apartments. I'd like to thank my friend and exercise partner Claire for helping me stay in shape while working hard on my research. I'd like to thank Daniela for all of her support and understanding.

1 Machine Learning

This thesis focuses on advancing the state of the art of machine perception, with a particular focus on computer vision. Computer vision and many other forms of machine perception are too difficult to solve by manually designing rules for processing inputs. Instead, some degree of learning is necessary. My personal view is that nearly the entire perception system should be learned.

Throughout the rest of this thesis, the narrator will be referred to as "we," rather than "I." This is because, as a thesis by articles, this thesis presents research conducted in a collaborative setting. It should be understood that the writing outside of the articles themselves is my own.

This chapter provides some background on machine learning in general. The subsequent chapters give more background on the particular kinds of machine learning used in the rest of the thesis. The remainder of the thesis presents the articles containing new methods.

1.1 Introduction to Machine Learning

Machine learning is the study of designing machines (or more commonly, software for general purpose machines) that can learn from data. This is useful for solving a variety of tasks, including computer vision, for which the solution is too difficult for a human software engineer to specify in terms of a fixed piece of software. Moreover, since learning is a critical part of intelligence, studying machine learning can shed light on the principles that govern intelligence.

But what exactly does it mean for a machine to learn? A commonly-cited definition is "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" (Mitchell, 1997) . One can

imagine a very wide variety of experiences E, tasks T, and performance measures P.

In this work, the experience E always includes the experience of observing a set of *examples* encoded in a *design matrix* $\mathbf{X} \in \mathbb{R}^{m \times n}$. Each of the m rows of $\mathbf{X}$ represents a different example which is described by n *features*. For computer vision tasks in which the examples are images, each feature is the intensity of a different pixel in the image.

For most but not all of the experiments in this thesis, the experience E also includes observing a label for each of the examples. For classification tasks such as object recognition, the labels are encoded in a vector $\mathbf{y} \in \{1, \ldots, k\}^m$, with element y_i specifying which of k object classes example i belongs to. Each numeric value in the domain of y_i corresponds to a real-world category, e.g. 0 can mean "dogs", 1 can mean "cats", 2 can mean "cars", etc.

In some experiments in this thesis, the label for each example is a vector, specifying a sequence of symbols to associate with each example. This is used in chapter 11 for transcribing multi-digit house numbers from photos.

Machine learning researchers study very many different tasks T. In this work, we explore the following tasks:

- *Density estimation*: In this task, the machine learning algorithm is asked to learn a function $p_{\text{model}} : \mathbb{R}^n \to \mathbb{R}$, where $p_{\text{model}}(\mathbf{x})$ can be interpreted as a probability density function on the space that the examples were drawn from. To do this task well (we'll specify exactly what that means when we discuss performance measures P), the algorithm needs to learn the structure of the data it has seen. It must know where examples cluster tightly and where they are unlikely to occur.
- *Imputation of missing values*: In this task, the machine learning algorithm is given a new example $\mathbf{x} \in \mathbb{R}^n$, but with some entries x_i of $\mathbf{x}$ missing. The algorithm must provide a prediction of the values of the missing entries. This task is closely related to density estimation, because it can be solved by learning $p_{\text{model}}(\mathbf{x})$ then conditioning on the observed entries of $\mathbf{x}$.
- *Classification*: In this task, the algorithm is asked to output a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$. Here $f(\mathbf{x})$ can be interpreted as an estimate of the category that $\mathbf{x}$ belongs to. There are other variants of the classification task, for example, where f outputs a probability distribution over classes, but this

thesis does not make any extensive use of the probability distribution over classes.

– *Classification with missing inputs* : This is similar to classification, except rather than providing a single classification function, the algorithm must learn a set of functions. Each function corresponds to classifying $\mathbf{x}$ with a different subset of its inputs missing.
– *Transcription* : This is similar to classification, except that the output is a sequence of symbols, rather than a single symbol.

Each of these tasks must be evaluated with a performance measure P. For the density estimation task, one could define a new set of examples $\mathbf{X}^{(\text{test})}$ and measure the probability of these examples according to the model. Evaluating the performance of a density estimation algorithm is difficult and we often turn to proxies for this value. For missing values imputation, we can measure the conditional probability the model assigns to the missing pixels in the test set, or some proxy thereof. For the classification and related tasks, one could define a set of labels $\mathbf{y}^{(\text{test})}$, and measure the classification accuracy of the model, i.e., the frequency with which $f(\mathbf{X}^{(\text{test})}_{i,:}) = y^{(\text{test})}_i$.

1.1.1 Generalization and the IID assumptions

An important aspect of the performance measures described above is that they both depend on a test set of data not seen during the learning process. This means that the learning algorithm must be able to *generalize* to new examples. Generalization is what makes machine learning different from optimization.

In order to be able to generalize from the training set to the test set, one needs to assume that there is some common structure in the data. The most commonly used set of assumptions are the *i.i.d.* assumptions . These assumptions state that the data is independently and identically distributed: each example is generated independently from the other examples, and each example is drawn from the same distribution p_{data} (Cover, 2006) . Formally,

$$p_{\text{data}}(\mathbf{X}, \mathbf{y}) = \Pi_i p_{\text{data}}(\mathbf{X}_{i:}, y_i).$$

This assumption is crucial to theoretically establishing that the procedures described in the subsequent subsections will generalize.

1.1.2 Maximum likelihood estimation

An extremely popular approach to machine learning is *maximum likelihood estimation*. In this approach, one defines a probabilistic model that is controlled by a set of parameters θ. The model provides a probability distribution $p_{\text{model}}(\mathbf{x}; \theta)$ over examples $\mathbf{x}$. (In this work we do not explore *non-parametric modeling* in which p is some function of the training set which can not be encoded in a fixed-length parameter vector) One can then use a statistical estimator to obtain the correct value of θ, drawn from set Θ of permissible values.

The estimator used in maximum likelihood is

$$\begin{aligned} \hat{\theta} &= \text{argmax}_{\theta \in \Theta} \Pi_i p_{\text{model}}(\mathbf{X}_{i:}; \theta) \\ &= \text{argmax}_{\theta \in \Theta} \sum_i \log p_{\text{model}}(\mathbf{X}_{i:}; \theta). \end{aligned}$$

In other words, the maximum likelihood estimation procedure is to pick the parameters that maximize the probability that the model will generate the training data. As shown above, one usually exploits the monotonically increasing property of the logarithm and instead optimizes the *log likelihood*, an alternative criterion which is maximized by the same value of θ. The log likelihood is more convenient to work with than the likelihood. As a product of several factors in the interval [0, 1], computing the likelihood on a digital computer often results in numerical underflow. The log likelihood avoids this difficulty. It also conveniently decomposes into a sum over separate examples, which makes many forms of mathematical analysis more convenient.

To justify the maximum likelihood estimation approach, assume that $p_{\text{data}}(\mathbf{x}) \in \{p_{\text{model}}(\mathbf{x}; \theta), \theta \in \Theta\}$. Given this and the i.i.d assumptions one can prove that in the limit of infinite data, the maximum likelihood estimator recovers a p_{model} that matches p_{data}. Note that we claim we can recover the true probability distribution, not the true value of θ. This is because the value of θ that was used to generate the training data cannot be determined if multiple values of θ correspond to the same probability distribution. The ability of the estimator to asymptotically recover the correct distribution is called *consistency* (Newey and McFadden, 1994).

Of course, to generalize well the maximum likelihood estimator must also do

well without infinite data. In the case of finite data, the maximum likelihood estimator is not always the best possible approach. In cases where very little data is available, maximum likelihood estimation of parametric models performs poorly compared to other approaches such as *Bayesian inference* (in which one makes new predictions by integrating over all possible values of θ). Unfortunately, the family of models for which this integral can be evaluated analytically is extremely limited. Bayesian inference usually entails computationally expensive Monte Carlo approximations. In practice, a commonly used middle ground between maximum likelihood and Bayesian inference is to use a *regularized* estimator, in which a single set of parameters is chosen in a way that makes their new predictions resemble those obtained by Bayesian inference. Typically this means maximizing a function with two terms, one term being the log likelihood of the data given θ and the other being the log likelihood of θ under some prior. The computational cost of such methods is usually roughly the same as that of maximum likelihood estimation.

In this work, we usually use maximum likelihood estimation only in situations where at least tens of thousands of examples are available, and we typically use at least one form of regularization. With this amount of data available, it is reasonable to expect maximum likelihood to do a good job of recovering θ, especially when using regularization.

1.1.3 Optimization

Much of machine learning can be cast as optimization. In the case of maximum likelihood estimation, one can define an objective function given by the log likelihood

$$\ell(\theta) = \sum_i \log p_{\text{model}}(\mathbf{X}_{i:}; \theta)$$

and solve the optimization problem

$$\begin{aligned} &\text{maximize } \ell(\theta) \\ &\text{subject to } \theta \in \Theta. \end{aligned}$$

Sometimes this can be done simply by analytically solving $\nabla_\theta \ell(\theta) = 0$ for θ. Other times, there is no closed-form solution to that equation and the solution must be obtained by an iterative optimization method.

One of the simplest iterative optimization methods is *gradient ascent*. This algorithm is based on the observation that $\nabla_\theta \ell(\theta)$ gives the direction in which ℓ increases most rapidly in a local neighborhood around θ. The idea is to take small steps in the direction of the gradient.

On iteration t of the gradient ascent algorithm, we compute the updated value of θ using the following rule:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha(t)\nabla_\theta \ell(\theta)$$

where $\alpha(t)$ is a positive scalar controlling the size of the step (Bishop, 2006, Chapter 3) . α is commonly referred to as the *learning rate*.

Gradient ascent may be expensive if there is a lot of redundancy in the dataset. It may take only a small number of examples to get a good estimate of the direction of the gradient from the current value of θ but gradient ascent will compute the gradient contribution of every single example in the dataset. As an extreme case, consider the behavior of gradient ascent when all examples in the training set are the same as each other. In this case, the cost of computing the gradient is m times what is necessary to obtain the correct step direction.

An alternative algorithm resolves this problem. In *stochastic gradient ascent* (Bishop, 2006, Chapter 3) , use the following update rule:

$$\theta^{(t)} = \theta^{(t-1)} + \alpha(t)\nabla_\theta \sum_{i \in \mathbb{S}} \log p_{\text{model}}(\mathbf{X}_{i:}; \theta)$$

where $\mathbb{S}$ is a random subset of $\{1, \ldots, m\}$. The randomly selected training examples are called a *minibatch*. Typically used minibatch sizes range from 1 to 128.

Stochastic gradient descent is widely believed to have other beneficial characteristics besides reducing redundant computations, but not all of these are well-characterized, and we do not explore them here.

When training deep neural nets, it is important to enhance stochastic gradient ascent with a technique called *momentum*. Momentum is a computationally inexpensive modification of stochastic gradient ascent where the parameters move with a velocity that is influenced by the gradient at each step:

$$\mathbf{v}^{(t)} = \mu(t)\mathbf{v}^{(t-1)} + \alpha(t)\nabla_\theta \sum_{i \in \mathbb{S}} \log p_{\text{model}}(\mathbf{X}_{i:}; \theta)$$

$$\theta^{(t)} = \theta^{(t-1)} + \mathbf{v}^{(t)}$$

While standard gradient ascent follows the steepest direction at each step, momentum partially accounts for the curvature of the function. Sutskever et al. (2013) showed that this simple method can perform as well as much more complicated second order methods like Hessian-free optimization (Nocedal and Wright, 2006; Martens, 2010).

In this introduction we have presented the optimization in terms of ascending the log likelihood, but in practice the optimization technique is most broadly known as *stochastic gradient descent* (SGD). In this case, the learning rule is presented as descending a cost function. One can of course ascend the log likelihood by descending the negative log likelihood.

A machine learning practictioner has two main ways of influencing the results of training a model with a gradient method.

One is picking the function $\alpha(t)$ that determines how the learning rate evolves over time (and the $\mu(t)$ function when using momentum). Constant α often works well, as does a linearly decreasing $\alpha(t)$. For $\mu(t)$, it is often effective to begin at 0.5 and linearly increase to a value around 0.9.

The other parameter under the practitioner's control is the convergence criterion. A common practice is to halt if $\ell(\theta)$ (evaluated on a held-out validation set) does not increase very much after some number of passes through the dataset. In some cases it is infeasible to compute $\ell(\theta)$ but learning is possible so long as one can compute $\nabla_\theta \ell(\theta)$ or a reasonable approximation thereof. In these cases we must design other proxies to use to determine convergence.

It may seem intuitive to run the optimization process until the gradient on the the training set is near zero, indicating that we have reached a local maximum. In practice, doing so usually results in *overfitting*, a condition that occurs when the model memorizes spurious patterns in the training set and as a result obtains much worse accuracy on the test set. Generally, in machine learning applications, we care about performance on the test set, which we can estimate by monitoring performance on a held out validation set. The best criteria for deep learning usually are based on validation set performance. The main goal of such criteria is to

prevent overfitting, not to make sure that a maximum has been reached. A common approach is to store the parameters that have attained the best accuracy on the validation set, and stop training when no new best parameters have been found within some fixed number of update steps. At the end of training, we use the best stored parameters, not the last parameters visited by SGD.

Many other sophisticated optimization algorithms exist, but they have not proven as effective for deep learning as stochastic gradient and momentum have.

1.2 Supervised learning

Supervised learning is the class of learning problems where the desired output of the model on some training set is known in advance and supplied by a supervisor. One example of this is the aforementioned classification problem, where the learned model is a function $f(\mathbf{x})$ that maps examples $\mathbf{x}$ to category IDs. Another common supervised learning problem is *regression*. In the context of regression, the training set consists of a design matrix $\mathbf{X}$ and a vector of real-valued targets $\mathbf{y} \in \mathbb{R}^m$ (or a matrix of outputs in the case of multiple output targets for each example). In this work, we do not study regression.

It is possible to solve the classification problem using maximum likelihood estimation and stochastic gradient ascent. One simply fits a model $p(y \mid \mathbf{x}; \theta)$ or $p(\mathbf{x}, y; \theta)$ to the training set, and returns $f(x) = \text{argmax}_y p(y \mid \mathbf{x})$.

The maximum likelihood approach is the one most commonly employed in deep learning. We describe deep supervised learning in more detail in chapter 3.

Among shallow learning models, one of the best known supervised learning approaches is the support vector machine.

1.2.1 Support vector machines and statistical learning theory

The *support vector machine* (SVM) is a widely used model and associated learning algorithm for supervised learning. SVMs may be used to solve both regression (Drucker et al., 1996) and classification (Cortes and Vapnik, 1995) problems. We found classification SVMs useful for some of the work described in this thesis.

When solving the classification problem, an SVM discriminates between two classes. In order to solve a k-class classification problem, one may train k different SVMs. SVM i learns to discriminate class i from the other $k-1$ classes. This is called *one-against-all* classification (bo Duan and Keerthi, 2005). Other methods of solving multi-class problems exist, but this is the one we use in the current work.

When training a basic two-class SVM it is conventional to regard labels y_i as drawn from $\{-1, 1\}$. This makes some of the algebraic expressions that follow more compact. Examples belonging to class 1 are referred to as *positive examples* while examples belong to class -1 are known as *negative examples*.

The SVM works by finding a hyperplane that separates the positive examples from the negative examples as well as possible (different kinds of SVMs have different ways of quantifying "as well as possible," and the simplest form of SVM is only applicable to data that can be separated perfectly). This hyperplane is parameterized by a vector $\mathbf{w}$ and a scalar b. The classification function is $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$. In order to obtain good generalization, none of the examples should lie very close to the hyperplane. If a training example lies close to the hyperplane, a similar test example might cross the hyperplane and receive a different label. To this end, SVM training algorithms try to ensure that $y(\mathbf{w}^\top \mathbf{x} + b) \geq 1$ for all training examples $\mathbf{x}$.

SVMs are commonly used with the *kernel trick*. The kernel trick replaces dot products $\mathbf{x}^\top \mathbf{z}$ with evaluations of a *kernel function* $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$. All operations that the SVM and its training algorithm perform on the input can be written in terms of dot products $\mathbf{x}^\top \mathbf{z}$. By replacing these dot products with $K(\mathbf{x}, \mathbf{z})$, one can train the SVM in ϕ-mapped space rather than the original space. Clever choices of K allow the use of high-dimensional, even infinite-dimensional, ϕ. In the case of non-linear ϕ, the SVM will have a non-linear decision boundary rather than a separating hyperplane in the original data space. Its decision function will still be a hyperplane in ϕ-mapped space. While the kernel trick is popular, it has many disadvantages, including requiring the training algorithm to be adapted in ways that reduce its ability to scale to very large numbers of training examples. In this work we do not use the kernel trick.

Various methods of training SVMs exist. We found that a variant called the L2-SVM (Keerthi et al., 2005) is easy to train and obtains the best generalization on the tasks we consider here. The L2-SVM is controlled by a *regularization parameter* C.

C must be positive and it determines the cost of misclassifying a training example. Larger values of C mean that the SVM will learn to have higher accuracy on the training set. Too large of a value of C can however result in overfitting.

Formally, the L2-SVM training algorithm is to solve the following optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}(||\mathbf{w}||^2 + b^2) + \frac{C}{2}||\xi||^2 \\ &\text{subject to } y_i(\mathbf{X}_{i,:}\mathbf{w} + b) \geq 1 - \xi_i \ \ \forall i \end{aligned}$$

where each ξ_i is an introduced auxiliary variable measuring how far example i comes from satisfying the margin condition. This optimization problem may be solved efficiently by solving analytically for ξ, substituting the expression for ξ into the objective function to obtain an unconstrained problem, and applying an iterative optimization algorithm called LBFGS (Byrd et al., 1995).

In section 1.1.2, we saw that the concept of asymptotic consistency of statistical estimators provides some justificiation for using maximum likelihood estimation as a machine learning algorithm that generalizes to new data. SVMs have a different theoretical justification that is more directly related to the classification task and better developed for the case where there is a small amount of labeled data.

Results from *statistical learning theory* (Vapnik, 1999) show that by solving the SVM optimization problem, we can guarantee that the SVM's accuracy on the test set is likely to be reasonably similar to its accuracy on the training set. More formally, suppose the i.i.d assumptions hold, the SVM is trained on m examples consisting of n features each, and the SVM misclassifies $\hat{\epsilon}$ of the training set. Let ϵ represent the proportion of examples drawn from p_{data} that the SVM misclassifies (i.e., its error rate on an infinitely large test set). For any $\delta \in (0, 1)$ we can guarantee (Vapnik and Chervonenkis, 1971)

$$\text{with probability } 1 - \delta, \epsilon \leq \hat{\epsilon} + \sqrt{\frac{(n+1)\log(\frac{2m}{n+1} + 1) - \log(\frac{\delta}{4})}{m}}$$

This is a conservative bound; it applies to any p_{data} and any classifier based on a separating hyperplane. Real-world distributions usually result in much better test set performance. It is also possible to obtain tighter bounds that are specific

to SVMs.

1.3 Unsupervised learning

An unsupervised learning problem is one where the learning algorithm is not provided with labels y; it is provided only with the design matrix of examples $\mathbf{X}$. The goal of an unsupervised learning algorithm is to discover something about the structure of p_{data}.

Unsupervised learning need not be explicitly probabilistic. Many unsupervised learning algorithms are rather geometrical in nature.

A few common types of unsupervised learning include

- *Density estimation*, in which the learning algorithm attempts to recover p_{data}. Knowing p_{data} is useful for a variety of purposes, such as making predictions. Another application is *anomaly detection*. For example, a credit card company might suspect fraud if a purchase seems very unlikely given a model of a customer's spending habits. Examples of models used for density estimation include the *mixture of Gaussians* (Titterington et al., 1985) model.
- *Manifold learning*, in which the learning algorithm tries to explain the data as lying on a low-dimensional manifold embedded in the original space. Distance along such manifolds often gives a more meaningful way to measure the similarity of two examples than distance in the original space does. One example of such a model is the contractive autoencoder (Rifai et al., 2011).
- *Clustering*, in which the learning algorithm attempts to discover a set of categories that the data can be divided into neatly. For example, an online store might cluster its customers based on their purchasing habits. When a new customer buys one item, the store can see which cluster of previous customers tends to buy that item the most, and recommend other items bought by customers in that cluster. Examples of clustering algorithms include k-means (Steinhaus, 1957) and mean-shift (Fukunaga and Hostetler, 1975) clustering.

These are not necessarily mutually exclusive categories (density estimation is commonly but not always used to achieve all of the other). Nor are all of their goals clearly defined (a dataset of carrots, oranges, radishes, and apples could equally well

be divided into two clusters consisting of fruit and vegetables or into two clusters consisting of orange objects and red objects).

As an example, the mixture of Gaussians model supposes that the data can be divided into k different categories. A latent variable $h \in \{1, \ldots, k\}$ whose distribution is governed by a parameter $\mathbf{c}$ identifies which category a given example belongs to. The distribution over members of category i is given by a multivariate Gaussian distribution with mean $\mu^{(i)}$ and covariance matrix $\mathbf{\Sigma}^{(i)}$. Often $\mathbf{\Sigma}$ is restricted to be a diagonal matrix for computational and statistical reasons. The complete generative model is:

$$\begin{aligned} p(h = i) &= c_i \\ p(x \mid h) &= \mathcal{N}(\mathbf{x} \mid \mu^{(h)}, \mathbf{\Sigma}^{(h)}). \end{aligned}$$

This model can be fit with straightforward maximum likelihood estimation techniques. Fitting the model accomplishes both a density estimation task and a clustering task– an example $\mathbf{x}$ belongs to the cluster $\text{argmax}_h p(h \mid \mathbf{x})$.

1.4 Feature learning

Feature learning (also known as *representation learning*) is an important strategy in machine learning. Many learning problems become "easier" if the inputs $\mathbf{x}$ are transformed to a new set of inputs $\phi(\mathbf{x})$. Properly designed feature mappings ϕ can reduce both overfitting and underfitting. However, it can be difficult to explicitly design good functions ϕ. Feature learning refers to learning the feature mapping ϕ. All of the work in this thesis employs this strategy in one way or another.

As an example, consider fitting a linear SVM to the dataset depicted in Fig 1.1. In the original space, the SVM cannot represent the right decision boundary. In the transformed space, it is easy to separate the data.

In this example, ϕ was mostly helpful because it overcame a problem with the linear SVM's *representational capacity*–even with infinite data, the SVM simply has no way of specifying the right decision boundary to separate the data. Most practical applications of feature learning also aim to improve statistical efficiency.

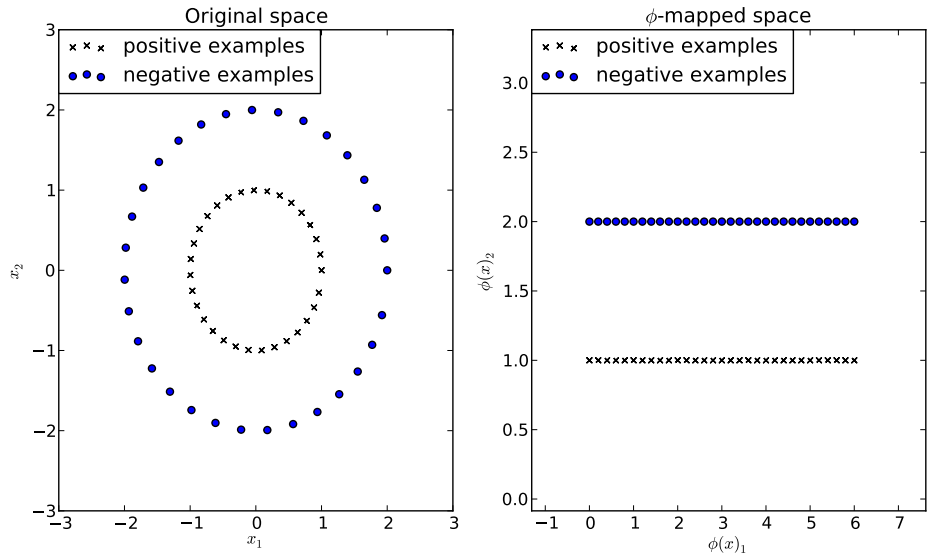


Figure 1.1: Left: An example dataset for an SVM. Right: The same dataset transformed by $\phi(x)$, where ϕ is conversion to polar coordinates.

Many feature learning algorithms are based on unsupervised learning, and can learn a reasonably useful mapping ϕ without any labeled data. This allows hybrid learning systems that can improve performance on supervised learning tasks by learning features on unlabeled data. One of the main reasons this approach is beneficial is that unlabeled data is usually more abundant. For example, an unsupervised learning algorithm trained on a large amount of images of cats and cars might discover features that are indicator variables for concepts like "has ears" or "has wheels." A classifier trained with these high-level input features then needs few labeled examples in order to generalize well.

Even when all of the available examples are labeled, training the features to model the input can provide some regularization.

Unsupervised feature learning is also useful because it allows the model to be split into pieces and trained one component at a time, even if each individual component cannot be meaningfully associated with an output target. For example, if we divide a 32×32 pixel image of a cat into a collection of small 6×6 pixel image patches, many of these patches do not contain any portion of the cat at all and those that do contain some portion of the cat probably do not contain enough information to identify it. We therefore cannot associate each image patch with a

label, so supervised learning cannot make progress with the input divided up in this way. Unsupervised learning can still learn good descriptions of each image patch, allowing us to learn thousands of features per image patch. When extracted at all locations in the image, this corresponds to millions of features per image. Learning these millions of features on a per-patch basis greatly reduces the computational cost of training such a system. This patch-based learning approach has been used in several practical applications (Lee et al., 2009; Coates et al., 2011) and is exploited in this thesis.

A closely related idea to feature learning is *deep learning* (Bengio, 2009). In deep learning, the feature extractor ϕ is formed by composing several simpler mappings:

$$\phi(\mathbf{x}) = \phi^{(L)}(\phi^{(L-1)}(\dots\phi^{(1)}(\mathbf{x})))$$

where L is the total number of mappings. Each mapping $\phi^{(i)}$ is known as a *layer*. The composite feature extractor ϕ is considered "deep" because the computational graph describing it has several of these layers. Each layer of a deep learning system can be thought of as being analogous to a line of code in a program–each layer references the results of earlier layers, and complicated tasks can be accomplished by running multiple simple layers in sequence. For example, see Fig. 1.2.

Deep learning was popularized by the success of deep belief networks (Hinton et al., 2006), stacked autoencoders (Bengio et al., 2007), and stacked denoising autoencoders (Vincent et al., 2008). In these approaches to deep learning, each sub-mapping $\phi^{(i)}$ is trained in isolation. This is known as *greedy layer-wise pretraining*. This pretraining is usually followed by *joint fine-tuning* of the entire system.

Since this style of deep learning system is formed by composing shallow learners, a popular form of deep learning research is devising new shallow learners. Some examples of recent work in developing shallow learners for feature learning includes work with sparse coding (Raina et al., 2007), restricted Boltzmann machines (RBMs) (Hinton et al., 2006; Courville et al., 2011a), the aforementioned autoencoder-based methods, and hybrids of autoencoders and sparse coding (Kavukcuoglu et al., 2010). The spike-and-slab sparse coding work we introduce in chapter 5 can be seen as a continuation of this line of research.

Other approaches to deep learning involve training the entire deep learning system simultaneously. This is the approach we use in the remainder of this thesis.

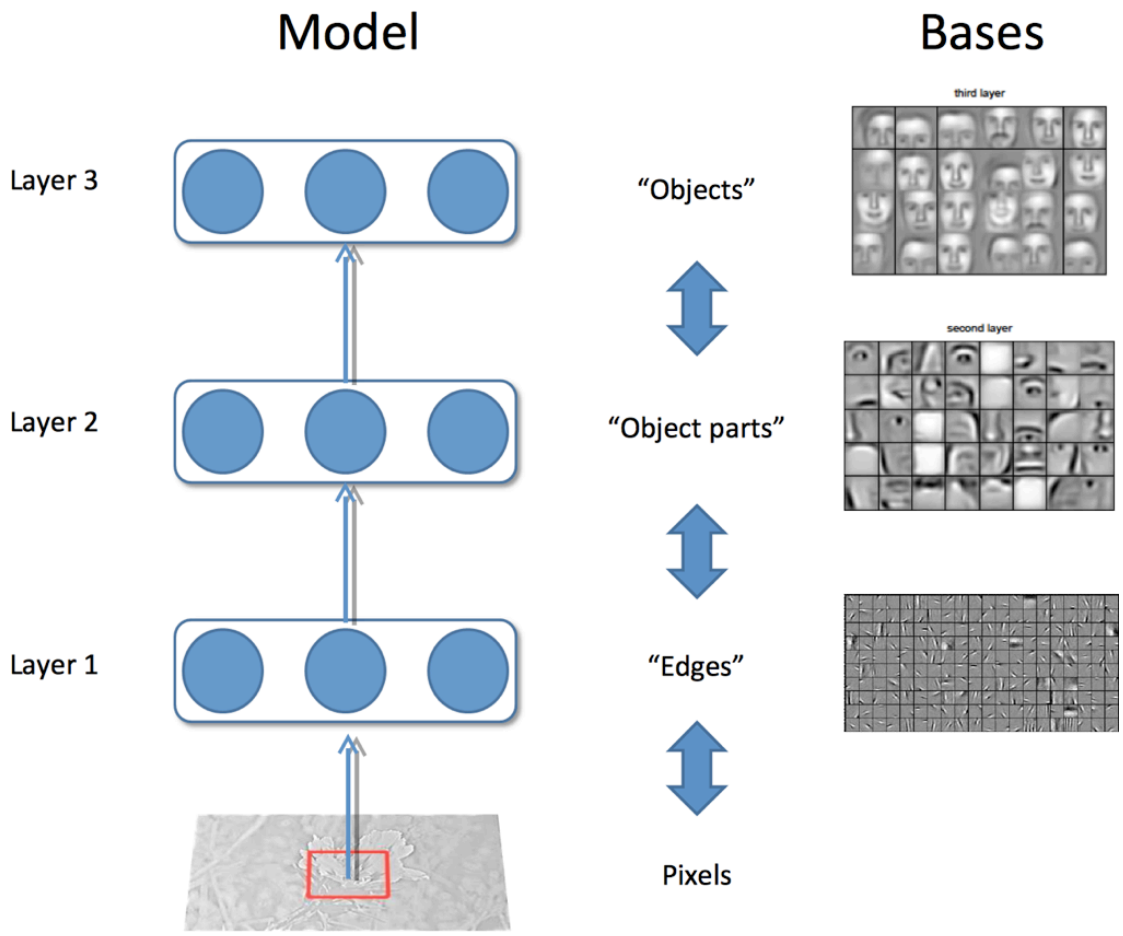


Figure 1.2: *Deep learning example*: When trained on images, the first layer of a deep learning system operates on the pixels and usually extracts some sort of edges from the image. The second layer operates on this representation in terms of edges and might extract small object parts that can be described as collections of small numbers of edges. The third layer operates on this representation in terms of object parts and might extract entire objects that can be described as collections of small numbers of object parts. The exact results depend on the algorithm employed, model architecture, and formatting of the dataset. *(This image was joint work with Honglak Lee and Andrew Saxe, originally prepared for an oral presentation of (Goodfellow et al., 2009))*

2 Structured Probabilistic Models

Chapter 1 presented some of the basic ideas of probabilistic modeling with maximum likelihood estimation. This chapter explores these ideas in greater depth, applying maximum likelihood estimation to more complicated models that require us to introduce approximations.

Sections 2.1 and 2.2 describe two ways of representing structure in a probabilistic model. Viewing probabilistic models as containing simplifying structure is a crucial cognitive tool that motivates design choices throughout the rest of this thesis. Section 2.3 explains a basic design choice about how to represent complicated interactions between multiple units.

Section 2.4 explains how to train models for which the likelihood cannot be computed using sampling-based approximations to the gradient of the log likelihood. Other approximate methods of training are possible for these models but the strategies detailed in this section are the ones that are used in this thesis.

Section 2.5, demonstrates how models with an intractable posterior distribution over their latent variables can be trained using variational approximations. Again, other approximations are possible, so the presentation here focuses on the methods actually used in the present work.

Finally, I'll discuss combining both forms of approximation in section 2.6.

2.1 Directed models

In general, a probability distribution over a vector-valued variable $\mathbf{x}$ represents probabilistic interactions between all of the variables. Suppose that $\mathbf{x} \in \{1, \ldots, k\}^n$. To parameterize a fully general $P(\mathbf{x})$ on discrete data like this is requires a table containing $k^n - 1$ entries! (one entry for all but one of the members

of the outcome space, with the probability of the last entry determined by the constraint that a probability distribution sum to 1)

Fortunately, most probability distributions we actually work with in practice do not involve all possible interactions between all possible variables. Many variables interact with each other only indirectly. This allows us to greatly simplify our representation of the distribution.

Probabilistic models that exploit this idea are called *structured probabilistic models*, because they represent the variables as belonging to a structure that restricts their ability to interact directly. Structure enables a model to do its job with fewer parameters, thus reducing the computational cost of storing it and increasing its statistical efficiency. It also reduces the computational cost of performing operations like computing marginal or conditional distributions over subsets of the variables (Koller and Friedman, 2009).

A common form of structured probabilistic model is the Bayesian network (Pearl, 1985). A Bayesian network is defined by a directed acyclic graph $\mathcal{G}$ whose vertices are the random variables in the model, and a set of *local conditional probability distributions* $p(x_i \mid \mathrm{Pa}_{\mathcal{G}}(x_i))$ where $\mathrm{Pa}_{\mathcal{G}}(x_i)$ returns the parents of x_i in $\mathcal{G}$.

The probability distribution over $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \Pi_i p(x_i \mid \mathrm{Pa}_{\mathcal{G}}(x_i)).$$

So long as each variable has few parents in the graph, the distribution can be represented with very few parameters. Simple restrictions on the graph structure can also guarantee that operations like computing marginal or conditional distributions over subsets of variables are efficient.

2.2 Undirected models

Some interactions between variables may not be well-captured by local conditional probability distributions. For example, when modeling the pixels in an image, there is no clear reason for one pixel to be a parent of the other; their interactions are basically symmetrical.

A Markov network (Kindermann, 1980) is a structured graphical model defined on an undirected graph $\mathcal{G}$. For each clique $\mathcal{C}$ in the graph, a factor $\phi(\mathcal{C})$ measures the affinity of the variables in that clique for being in each of their possible states. The factors are constrained to be non-negative. Together they define an *unnormalized probability distribution*

$$\tilde{p}(\mathbf{x}) = \Pi_{\mathcal{C} \in \mathcal{G}} \phi(\mathcal{C}).$$

The unnormalized probability distribution is efficient to work with so long as all the cliques are small.

Obtaining the normalized probability distribution may be costly. To do so, one must compute the *partition function* Z (though Z is conventionally written without arguments, it is in fact a function of whatever parameters govern each of the ϕ functions). Since

$$Z = \int_{\mathbf{x}} \tilde{p}(\mathbf{x}) d\mathbf{x}$$

it may be intractable to compute for high-dimensional $\mathbf{x}$, depending on the structure of $\mathcal{G}$ and the functional form of the ϕs.

Many interesting theoretical results about undirected models depend on the assumption that $\forall \mathbf{x}, \tilde{p}(\mathbf{x}) > 0$. A convenient way to enforce this to use an *energy-based model* (EBM) where

$$\tilde{p}(x) = \exp(-E(\mathbf{x}))$$

and $E(x)$ is known as the *energy function*. This can still be interpreted as a standard Markov network; the exponentation makes each term in the energy function correspond to a factor for a different clique. The $-$ sign isn't strictly necessary from a computational point of view (and some machine learning researchers have tried to do without it, e.g. (Smolensky, 1986)). It is a commonly used convention inherited from statistical phyiscs, along with the terms "energy function" and "partition function."

Some results in this chapter are presented in terms of energy-based models. For these results, the theory doesn't hold if $\tilde{p}(\mathbf{x}) = 0$ for some $\mathbf{x}$. Note that a directed graphical model may be encoded as an energy-based model so long as this condition is respected.

2.2.1 Sampling

Drawing a sample $\mathbf{x}$ from the probability distribution $p(\mathbf{x})$ defined by a structured model is an important operation. We briefly describe how to sample from directed models and EBMs here. For more detail, see (Koller and Friedman, 2009).

Sampling from a directed model is straightforward, assuming that one can sample from each of the conditional probability distributions. The procedure used in this case is called *ancestral sampling*. One simply draws samples of each of the variables in the network in an order that respects the network topology, i.e., before sampling a variable x_i from $P(x_i \mid \text{Pa}_{x_i})$, sample each of the members of Pa_{x_i}. This defines an efficient means of sampling all variables with a single pass through the network.

Sampling from an EBM is not straightforward. Suppose we have an EBM defining a distribution $p(a, b)$. In order to sample a, we must draw it from $p(a \mid b)$, and in order to sample b, we must draw it from $p(b \mid a)$. This "chicken and egg" problem means we can no longer use ancestral sampling. Since $\mathcal{G}$ is no longer directed and acyclical, we don't have a way of ordering the variables such that every variable can be sampled given only variables that come earlier in the ordering.

It turns out that we can sample from an EBM, but we can not generally do so with a single pass through the network. Instead we need to sample using a Markov chain. A Markov chain is defined by a *state* $\mathbf{x}$ and a transition distribution $T(\mathbf{x}' \mid \mathbf{x})$. Running the Markov chain means repeatedly updating the state $\mathbf{x}$ to a value $\mathbf{x}'$ sampled from $T(\mathbf{x}' \mid \mathbf{x})$.

Under certain distributions, a Markov chain is eventually guaranteed to draw $\mathbf{x}$ from an equilibrium distribution $\pi(\mathbf{x}')$, defined by the condition

$$\forall \mathbf{x}', \pi(\mathbf{x}') = \sum_{\mathbf{x}} T(\mathbf{x}' \mid \mathbf{x}) \pi(\mathbf{x}).$$

This condition guarantees that repeated applications of the transition sampling procedure don't change the distribution over the state of the Markov chain. Running the Markov chain until it reaches its equilibrium distribution is called "burning in" the Markov chain.

Unfortunately, there is no theory to predict how many steps the Markov chain must run before reaching its equilibrium distribution, nor any way to tell for sure that this event has happened. Also, even though successive samples come from the

same distribution, they are highly correlated with each other, so to obtain multiple independent samples one should run the Markov chain for several steps between collecting each sample. Markov chains tend to get stuck in a single mode of $\pi(\mathbf{x})$ for several steps. The speed with which a Markov chain moves from mode to mode is called its mixing rate. Since burning in a Markov chain and getting it to mix well may take several sampling steps, sampling correctly from an EBM is still a somewhat costly procedure.

Of course, all of this depends on ensuring $\pi(\mathbf{x}) = p(\mathbf{x})$. Fortunately, this is easy so long as $p(\mathbf{x})$ is defined by an EBM. The simplest method is to use *Gibbs sampling*, in which sampling from $T(\mathbf{x}' \mid \mathbf{x})$ is accomplished by selecting one variable x_i and sampling it from p conditioned on its neighbors in $\mathcal{G}$. It is also possible to sample several variables at the same time so long as they are conditionally independent given all of their neighbors.

2.3 Latent variables

Most of this thesis concerns models that have two types of variables: observed or "visible" variables $\mathbf{v}$ and latent or "hidden" variables $\mathbf{h}$. $\mathbf{v}$ corresponds to the variables actually provided in the design matrix $\mathbf{X}$ during training. $\mathbf{h}$ consists of variables that are introduced to the model in order to help it explain the structure in $\mathbf{v}$. Generally the exact semantics of $\mathbf{h}$ depend on the model parameters and are created by the learning algorithm. The motivation for this is twofold.

2.3.1 Latent variables versus structure learning

Often the different elements of $\mathbf{v}$ are highly dependent on each other. A good model of $\mathbf{v}$ which did not contain any latent variables would need to have very large numbers of parents per node in a Bayesian network or very large cliques in a Markov network. Just representing these higher order interactions is costly–both in a computational sense, because the number of parameters that must be stored in memory scales exponentially with the number of members in a clique, but also in a statistical sense, because this exponential number of parameters requires a wealth of data to estimate accurately.

There is also the problem of learning which variables need to be in such large cliques. An entire field of machine learning called *structure learning* is devoted to this problem . Most structure learning techniques involve fitting a model with a specific structure to the data, assigning it some score that rewards high training set accuracy and penalizes model complexity, then greedily adding or subtracting an edge from the graph in a way that is expected to increase the score. See (Koller and Friedman, 2009) for details of several approaches.

Using latent variables mostly avoids the problem of learning structure. A fixed structure over visible and hidden variables can use direct interactions between visible and hidden units to impose indirect interactions between visible units. Using simple parameter learning techniques we can learn a model with a fixed structure that imputes the right structure on the marginal $p(\mathbf{v})$. Of course, one still has the problem of determining the amount of latent variables and their connectivity, but it is usually not as important to determine the absolutely optimal model architecture when using latent variables as when using structure learning on fully observed models. Usually, in the context of deep learning and latent variable models, the architecture is controlled by a small number of hyperparameters, which are searched relatively coarsely.

2.3.2 Latent variables for feature learning

Another advantage of using latent variables is that they often develop useful semantics. As discussed in section 1.3, the mixture of Gaussians model learns a latent variable that corresponds to which category of examples the input was drawn from. Other more sophisticated models with more latent variables can create even richer descriptions of the input. Most of the approaches mentioned in section 1.4 accomplish feature learning by learning latent variables. Often, given some model of $\mathbf{v}$ and $\mathbf{h}$, it turns out that $\mathbb{E}[\mathbf{h} \mid \mathbf{v}]$ or $\text{argmax}_{\mathbf{h}} p(\mathbf{h}, \mathbf{v})$ is a good feature mapping for $\mathbf{v}$.

2.4 Stochastic approximations to maximum likelihood

Consider an energy-based model $p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h}))$.

Suppose that the partition function Z cannot be computed. This model may still be useful. As explained in section 2.2.1, one can still draw samples from this model, perhaps even efficiently. One might also be able to compute the ratio of the probability of two events, $p(\mathbf{v}, \mathbf{h})/p(\mathbf{v}', \mathbf{h}')$, or the posterior $p(\mathbf{h} \mid \mathbf{v})$, which as shown in 2.3 could be useful as a set of features to describe $\mathbf{v}$.

Given that such a model is useful, learning one is a desirable capability. However, our primary method of learning models is maximum likelihood estimation. As seen in section 1.1.3, this involves computing

$$\nabla_\theta \log p(\mathbf{v}).$$

Unfortunately, if we expand the definition of $p(\mathbf{v})$, we see that this expression contains Z:

$$\nabla_\theta \log \tilde{p}(v) - \nabla_\theta \log Z.$$

Since Z is intractable, there doesn't seem to be much hope of computing $\nabla_\theta \log Z$.

Fortunately, so long as Leibniz's rule applies, a sampling trick can approximate the gradient:

$$
\begin{aligned}
&\frac{\partial}{\partial \theta_i} \log Z \\
&= \frac{\partial}{\partial \theta_i} \log \int_{\mathbf{v}} \int_{\mathbf{h}} \tilde{p}(\mathbf{v}, \mathbf{h}) d\mathbf{h} d\mathbf{v} \\
&= \frac{\frac{\partial}{\partial \theta_i} \int_{\mathbf{v}} \int_{\mathbf{h}} \tilde{p}(\mathbf{v}, \mathbf{h}) d\mathbf{h} d\mathbf{v}}{\int_{\mathbf{v}} \int_{\mathbf{h}} \tilde{p}(\mathbf{v}, \mathbf{h}) d\mathbf{h} d\mathbf{v}} \\
&= \frac{1}{Z} \int_{\mathbf{v}} \int_{\mathbf{h}} \frac{\partial}{\partial \theta_i} \exp(-E(\mathbf{v}, \mathbf{h})) d\mathbf{h} d\mathbf{v} \\
&= -\frac{1}{Z} \int_{\mathbf{v}} \int_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})) \frac{\partial}{\partial \theta_i} E(\mathbf{v}, \mathbf{h}) d\mathbf{h} d\mathbf{v} \\
&= -\mathbb{E}_{\mathbf{v}, \mathbf{h}}[\frac{\partial}{\partial \theta_i} E(\mathbf{v}, \mathbf{h})]
\end{aligned}
$$

The expectation can be approximated by drawing samples of $\mathbf{v}$ and $\mathbf{h}$, but this of course raises the question of how to set up the Markov chain in a way that yields a good approximation and is efficient.

The naive approach is to initialize a new Markov chain and run it to its equilibrium distribution on every step of stochastic gradient ascent. Unfortunately, that is too expensive.

One solution to this problem is *contrastive divergence* (CD-k) (Hinton, 2002). This approach makes use of several Markov chains in parallel, one per example in the minibatch. At each learning step, each Markov chain is initialized with the corresponding data example and run for k steps. Typically $k = 1$. Clearly this approach only explores parts of space that are near the data points. This procedure generally results in the model's distribution having about the right shape near the data points, but the model may inadvertently learn to represent other modes far from the data.

Another approach is known alternatively as *stochastic maximum likelihood* (SML) (Younes, 1998) or *persistent contrastive divergence* (PCD) (Tieleman, 2008). This approach also makes use of parallel Markov chains but each is initialized only once, at the start of training. The state of each chain is sampled once per gradient ascent step. This approach depends on the assumption that the learning rate is small enough that the Markov chains will remain at their equilibrium distribution

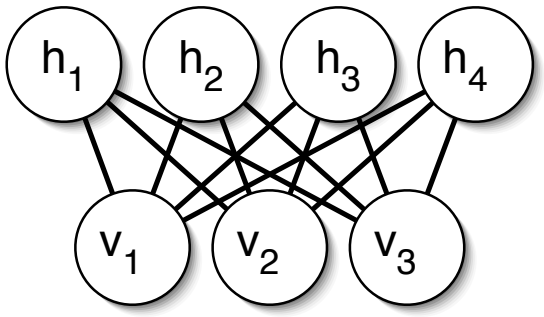


Figure 2.1: An example RBM drawn as a Markov network

even though that equilibrium distribution is continually changing. The advantage of SML over CD is that each Markov chain is updated for several steps, and consequently should explore all of the model's modes. This enables SML to suppress modes that are far from the data that CD might overlook.

2.4.1 Example: The restricted Boltzmann machine

The restricted Boltzmann machine (RBM) (Smolensky, 1986) is an example of a model that has intractable Z (Long and Servedio, 2010) yet may be trained using the techniques described in this section (Hinton, 2002).

It is an energy-based model with binary visible and hidden units. Its energy function is

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h} - \mathbf{v}^\top \mathbf{W} \mathbf{h}$$

where $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{W}$ are unconstrained, real-valued, learnable parameters. The model is depicted graphically in Fig. 2.1. As this figure makes clear, an important aspect of this model is that there are no direct interactions between any two visible units or between any two hidden units (hence the "restricted"; a general Boltzmann machine may have arbitrary connections).

The restrictions on the RBM structure yield the nice properties

$$p(\mathbf{h} \mid \mathbf{v}) = \Pi_i p(h_i \mid \mathbf{v})$$

and

$$p(\mathbf{v} \mid \mathbf{h}) = \Pi_i p(v_i \mid \mathbf{h}).$$

The individual conditionals are simple to compute as well, for example

$$p(h_i = 1 \mid \mathbf{v}) = \sigma\left(\mathbf{v}^\top \mathbf{W}_i + \mathbf{b}_i\right)$$

where σ is the logistic sigmoid function.

Together these properties allow for efficient block Gibbs sampling, alternating between sampling all of $\mathbf{h}$ simultaneously and sampling all of $\mathbf{v}$ simultaneously.

Since the energy function itself is just a linear function of the parameters, it is easy to take the needed derivatives. For example,

$$\frac{\partial}{\partial W_{ij}} E(\mathbf{v}, \mathbf{h}) = -v_i h_j.$$

These two properties–efficient Gibbs sampling and efficient derivatives– make it possible to train the RBM with stochastic approximations to $\nabla_\theta \log Z$.

2.5 Variational approximations

Another common difficulty in probabilistic modeling is that for many models the posterior distribution $p(\mathbf{h} \mid \mathbf{v})$ is infeasible to compute or even represent. Alternately, it may be infeasible to take expectations with respect to this distribution.

This poses problems for our goal outlined in section 2.3.2 of using $\mathbb{E}[\mathbf{h} \mid \mathbf{v}]$ as features. It also usually means that maximum likelihood estimation is infeasible. As shown in (Neal and Hinton, 1999), maximizing $p(\mathbf{v})$ is equivalent to maximizing

$$\mathbb{E}_{\mathbf{h} \sim P(\mathbf{h}|\mathbf{v})} \log P(\mathbf{v}, \mathbf{h}).$$

Fortunately, variational approximations provide a solution to both of these difficulties.

2.5.1 Variational learning

For any distribution $Q(\mathbf{h})$, the log likelihood may be decomposed (Neal and Hinton, 1999) into two terms.

$$\log p(\mathbf{v}) = D_{KL}(Q(\mathbf{h}) \| p(\mathbf{h} \mid \mathbf{v})) + \mathcal{L}(\mathbf{v}, Q).$$

Here, D_{KL} is the *Kullback-Leibler (KL) divergence* (Kullback and Leibler, 1951) and $\mathcal{L}$ is the negative *variational free energy*. The KL divergence is guaranteed to be non-negative, so this decomposition proves

$$\log p(\mathbf{v}) \geq \mathcal{L}(\mathbf{v}, Q).$$

$\mathcal{L}(\mathbf{v}, Q)$ is thus a lower bound on the log likelihood. The KL divergence measures the difference between two distributions, and goes to 0 when the two distributions are the same. Thus this lower bound is tight when $Q(\mathbf{h}) = P(\mathbf{h} \mid \mathbf{v})$. Consequently, one can maximize $\mathcal{L}(\mathbf{v}, Q)$ as a proxy for $\log p(\mathbf{v})$. Note that this maximization will involve modifying both the distribution Q (to make the lower bound tighter) and the parameters controlling p (to optimize the model using the bound).

In order to maximize $\mathcal{L}$, let's examine its functional form:

$$\mathcal{L}(\mathbf{v}, Q) = \mathbb{E}_{\mathbf{h} \sim Q}[\log P(\mathbf{v}, \mathbf{h})] + H_Q(\mathbf{h})$$

where $H_Q(\mathbf{h})$ is the Shannon entropy (Cover, 2006) of $\mathbf{h}$ under Q.

Since computing $\mathcal{L}(\mathbf{v}, Q)$ involves taking an expectation with respect to Q, it is necessary to restrict Q in order to make the expectation tractable. A particularly elegant way to restrict Q is to require it to take the form of a graphical model with a specific graph structure $\mathcal{G}$ (Saul and Jordan, 1996). A common approach is to use the *mean field approximation*

$$Q(\mathbf{h}) = \Pi_i Q(h_i)$$

which corresponds to a $\mathcal{G}$ with no edges. So long as inference in Q remains tractable, one can obtain better approximations by using a more complicated $\mathcal{G}$. This approach is known as *structured variational approximation*.

2.5.2 Variational inference

A common operation is to compute the Q that minimizes $D_{KL}(Q(\mathbf{h})\|P(\mathbf{h} \mid \mathbf{v}))$. This is necessary for extracting features $\mathbb{E}[\mathbf{h} \mid \mathbf{v}]$. It is also a common inner-loop to variational learning algorithms which alternate between optimizing $\mathcal{L}(\mathbf{v}, Q)$ with respect to Q and optimizing it with respect to the model parameters.

This operation is called *variational inference* (Koller and Friedman, 2009) because in the general case it involves solving a calculus of variations problem. Calculus of variations is the study of optimizing functionals. A functional is a mapping much like a function, except that a functional takes a function as its input. In variational inference, the functional being minimized is the KL divergence. The function being optimized is the distribution Q. Note that in the special case where none of the $\mathbf{h}$ variables is continuous, Q is merely a vector and may be optimized with traditional calculus techniques.

Usually variational inference involves using calculus of variations to find the functional form of Q, followed by an iterative procedure to find the parameters of Q. Consider the following example from (Bishop, 2006).

Suppose $\mathbf{h} \in \mathbb{R}^2$ and $p(\mathbf{h} \mid \mathbf{v}) = \mathcal{N}(\mathbf{h} \mid \mu, \beta^{-}1)$ (for the purpose of simplicity, in this example, the hidden units do not actually interact with the visible units).

Constrain Q with the mean field assumption $Q(\mathbf{h}) = Q(h_1)Q(h_2)$. Using calculus of variations one may then show

$$Q(\mathbf{h}_i) = \mathcal{N}(h_i \mid \hat{h}_i, 1/\beta_{i,i}).$$

In other words, the fact that $p(\mathbf{h} \mid \mathbf{v})$ is jointly Gaussian implies that the correct Q is also Gaussian. We never assumed that Q was Gaussian, only that it was factorial. The Gaussian nature of Q had to be derived via calculus of variations.

There is still an unknown: the mean of Q, $\hat{\mathbf{h}}$. This is an example of a *variational parameter*, a parameter controlling Q that cannot be found analytically. These parameters must be obtained by an iterative optimization procedure. Gradient descent would work, but is a slow and expensive procedure to use in the inner loop of a learning algorithm. Typically it is faster to optimize these parameters by iterating between fixed point equations.

2.6 Combining approximations: The deep Boltzmann machine

This chapter has described the tools needed to fit a very broad class of probabilistic models. Which tool to use depends on which aspects of the log-likelihood are problematic.

For the simplest distributions p, the log likelihood is tractable, and the model can be fit with a straightforward application of maximum likelihood estimation and gradient ascent as described in chapter 1.

This chapter has shown to implement probabilistic models in two different difficult cases. In the case where Z is intractable, one may still use maximum likelihood estimation via the sampling approximation techniques described in section 2.4. In the case where $p(\mathbf{h} \mid \mathbf{v})$ is intractable, one may still train the model using the negative variational free energy rather than the likelihood, as described in 2.5.

It is also possible that *both* of these difficulties will arise. An example of this occurs with the *deep Boltzmann machine* (Salakhutdinov and Hinton, 2009), which is a sequence of RBMs chained together with undirected connections. The model is depicted graphically in Fig. 2.2.

This model still has the same problem with computing the partition function as the simpler RBM does. It has also discarded the restricted structure that made $P(\mathbf{h} \mid \mathbf{v})$ easy to represent in the RBM. The typical way to train the DBM is to minimize the variational free energy rather than maximize the likelihood. Of course, the variational free energy still depends on the partition function, so it is necessary to use sampling techniques to approximate its gradient.

We put all of these techniques to use in chapter 5.2.2 when we introduce our own partially-directed deep Boltzmann machine.

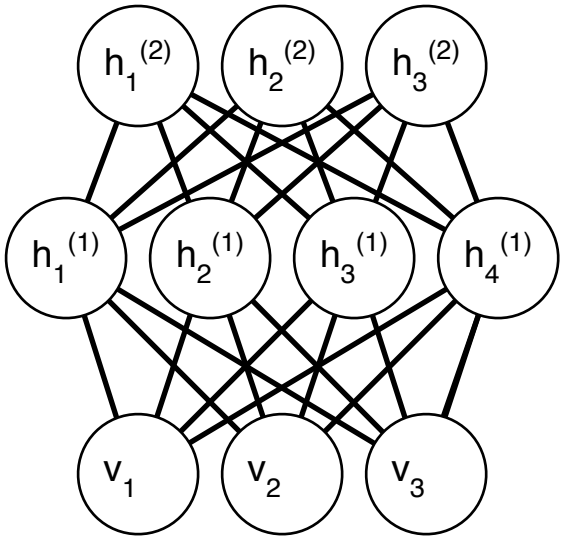


Figure 2.2: An example graph of a deep Boltzmann machine

3 Supervised deep learning

The current version of deep learning models that is most widely used to solve difficult engineering problems for industrial scale applications is based on purely supervised learning.

The standard deep learning model is the *multilayer perceptron (MLP)* , also known as the feed-forward neural network (Rumelhart et al., 1986a). This consists of a neural network that takes some input $\mathbf{x}$ and composes together transformations defined by several layers to produce an output:

$$f(\mathbf{x}) = f_L(f_{L-1}(\dots f_1(\mathbf{x})))$$

Each layer typically provides a matrix of learnable parameters $\mathbf{W}$ and a vector of learnable parameters $\mathbf{b}$ defining an affine transformation of the input. If each layer consisted only of an affine transformation, the entire function f would also be affine, so each layer also includes some fixed non-linear *activation function* g:

$$f_i(\mathbf{v}) = g(\mathbf{W}\mathbf{v} + \mathbf{b}).$$

Such models are motivated by a philosophy called *connectionism* (McClelland et al., 1986). The idea behind connectionism is that an individual neuron in an animal or a human being is not capable of doing anything interesting in isolation, but populations of neurons acting together can achieve intelligent behavior. Similarly, a single unit of a hidden layer in an MLP is useless, but any function can be approximated with any desired non-zero amount of error by an MLP with sufficiently many hidden units (Cybenko, 1989; Stinchcombe and White, 1989; Hornik et al., 1989). One can view humans and animals as a *proof of concept* illustrating that different amounts of intelligent behavior are possible with the amount of computational resources possessed by different species of animals. It may be possible for us to design algorithms that use hardware even more efficiently.

A special case of MLPs is the *convolutional neural network* (CNN) (Fukushima,

1980; LeCun et al., 1998). These networks restrict the structure of $\mathbf{W}$ and $\mathbf{b}$ for some of their layers. Specifically, when the input to the layer can be seen as samples taken on a grid in some space (for example, a rasterized image is a set of samples of brightness values collected by a 2-D grid of sensors, an audio recording is a set of samples of amplitude values collected on a 1-D grid throughout time, etc.), $\mathbf{W}$ may be restricted to define a discrete convolution rather than a general matrix multiplication. This makes $\mathbf{Wx}$ become *equivariant* to translations of the input. Because convolutions use the same parameters at every location, this significantly reduces the number of parameters that need to be stored and learned, improving both the model's memory requirements and statistical efficiency. Typically, the kernel used for the discrete convolution is small, meaning that the network is sparsely connected, which further reduces the number of parameters, and reduces the runtime of inference and backpropagation in the network.

Convolutional networks also typically include some kind of *spatial pooling* in their activation functions g; this refers to taking summary statistics over small spatial regions of the output in order to make the final output invariant to small spatial translations of the input. CNNs are very successful for image processing applications in modern commercial settings (Krizhevsky et al., 2012; Zeiler and Fergus, 2013b; Szegedy et al., 2013; Goodfellow et al., 2014).

MLPs and CNNs may be trained using stochastic gradient descent and momentum. The gradient, as defined by the chain rule of differentiation, contains very many terms. Fortunately, the different elements of the gradient contain many common subexpressions. Using a dynamic programming approach, one can avoid re-computing these subexpressions in order to compute the gradient efficiently. This idea is the basis of the *backpropagation* algorithm commonly used to compute the gradient (Bryson et al., 1963; Werbos, 1974; Rumelhart et al., 1986b). Not all modern approaches necessarily use the backpropagation algorithm *per se* (different choices of how to set up the dynamic programming process yield different speed-memory tradeoffs) but all do use symbolic differentiation strategies that employ the same basic dynamic programming technique.

Supervised deep learning has existed for decades but did not work well enough to solve many commercially intresting problems until recently. This is for three major reasons:

First, until recently, the hardware and software infrastructure available did not

allow for training of sufficiently large networks. If we refer back to the biological inspiration for connectionism, and view biological intelligence as a proof of concept giving some indication of what we can hope to achieve by simulating different amounts of neurons, Fig. 3.1 shows that until recently our networks were smaller than even the most primitive of biological nervous systems. As shown in Fig. 3.2, machine learning models were able to compensate for this somewhat by being nearly as densely connected as biological systems. However, we still lag absolute scale of the human nervous system by many orders of magnitude. It is only recently, with the introduction of GP-GPU (Bergstra et al., 2010; Bastien et al., 2012) and distributed (Dean et al., 2012) implementations of machine learning software that we have started to approach the necessary scale. Maxout, presented in chapter 9, can be seen as an attempt to increase the amount of feature detectors in the model without requiring a proportional increase in connections between feature detectors.

Second, SGD often produces less than satisfactory results for the types of activation functions g that were used until recently. The use of the rectified linear activation function (Jarrett et al., 2009; Glorot et al., 2011)

$$g(\mathbf{z})_i = \max\{z_i, 0\}$$

made MLPs and CNNs significantly easier to optimize. In chapter 9.8 we show how the maxout activation function results in further improvements in the ease of optimizing a deep network.

Finally, we have only recently been able to overcome the problem of overfitting in rectified linear networks. Previous approaches to preventing overfitting relied on unsupervised pretraining, and no unsupervised pretraining method has been shown to work especially well for deep rectifier networks. We are now able to regularize rectifier networks effectively using the dropout (Hinton et al., 2012) algorithm. In the commercial setting, overfitting can also be eliminated by brute force, simply by using the various resources available to a large company to generate a large dataset.

In chapter 11 we demonstrate the application of the modern supervised deep learning techniques described in this chapter to a real world task on a dataset of commercial interest.

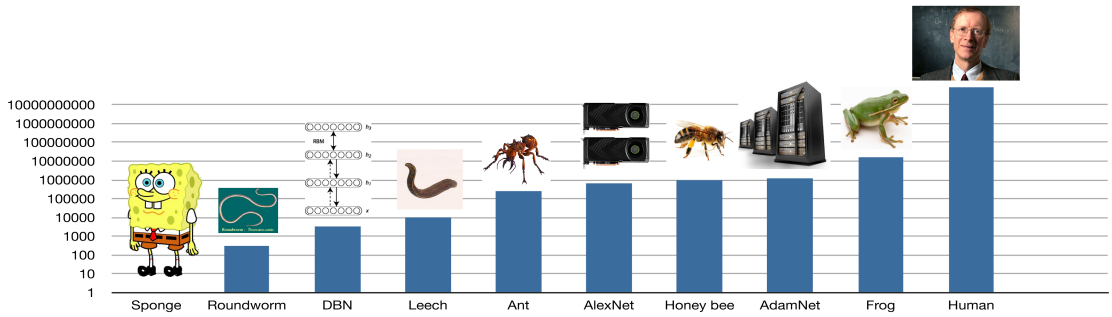


Figure 3.1: *Number of neurons in animals and machine learning models:* In the context of a machine learning model, a "neuron" refers to a hidden unit, which may in fact represent a considerably simpler functional unit than a biological neuron. One of the key tenets of connectionism is the idea that individual neurons are not particularly useful, but large populations of neurons can exhibit intelligent behaviors. Until very recently, artificial neural networks contained fewer "neurons" than even the most primitive of animals, making it somewhat wondrous that they worked at all. Modern neural networks employ about the same number of neurons as large insects, suggesting that further advances might be possible just by scaling to the greater amounts of neurons used by vertebrate animals. Current machine learning models are several orders of magnitude smaller than the human brain. Estimates of the number of neurons in various animals taken from http://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons. "DBN" refers to (Hinton et al., 2006). "AlexNet" refers to (Krizhevsky and Hinton, 2009). "AdamNet" refers to (Coates et al., 2013). Images are not my own.

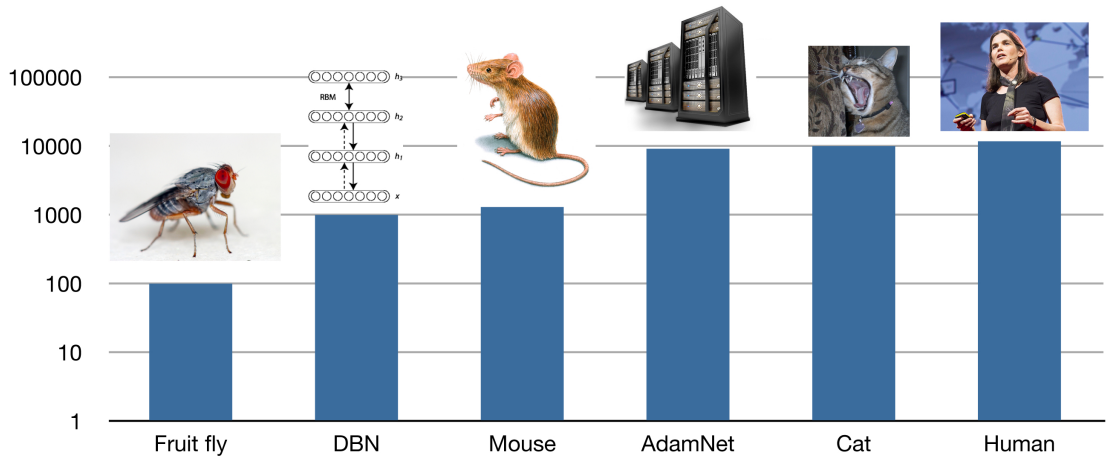


Figure 3.2: *Average number of connections per neuron in animals and machine learning models:* Part of the success of machine learning models despite their low number of neurons may be due to the comparatively high number of connections between neurons in machine learning models. In fact, machine learning models are not far from human levels of connectivity. This suggests that technologies that increase the total number of features in a model while reducing computational cost by limiting connectivity may be very effective. This explains the success of models that employ sparse connectivity and pooling, such as convolutional networks, especially maxout networks (see chapter 9). Estimates of the average number of connections per neuron obtained by dividing the number of neurons by the number of synapses listed at http://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons. "DBN" refers to (Hinton et al., 2006). "AdamNet" refers to (Coates et al., 2013). Images, with the exception of the photo of my cat, Stripey, are not my own.

4 Prologue to First Article

4.1 Article Details

Scaling up Spike-and-Slab Models for Unsupervised Feature Learning. Ian J. Goodfellow, Aaron Courville, and Yoshua Bengio. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*(8), 1902-1914.

Personal Contribution. The idea that a structured variational inference in a sparse coding model could provide an effective means of feature extraction was my own idea. Aaron Courville suggested using spike-and-slab sparse coding as the base model; my original idea was to use binary sparse coding. Aaron Courville also suggested one of the two inference algorithms presented in the paper, the method based on conjugate gradient descent. Aaron Courville and I developed the equations necessary for inference and learning jointly. The partially directed deep Boltzmann machine was my own idea. I implemented all of the necessary software and performed all of the experiments. I wrote the majority of the paper, with significant contributions to the writing from both Aaron Courville and Yoshua Bengio. I produced all of the figures.

4.2 Context

At the time that we wrote this article, the state of the art approach to object recognition on many datasets was to train a high-dimensional dictionary describing an image patch, extract the features of this dictionary at all locations in the image, pool the extracted features, and classify the resulting feature vector with an SVM. This pipeline was popularized by Coates et al. (2011). Later, Coates and Ng (2011) showed that, in this pipeline, sparse coding yields more regularized features than other feature extraction methods. This motivated us to explore applications of

sparse coding. In this article, we develop a means of scaling inference in the more regularized spike-and-slab sparse coding model to unprecedented problem sizes.

To this day, the RBM (Smolensky, 1986) remains the most successful model of natural images. Unfortunately, real-valued RBMs such as the mcRBM (Ranzato and Hinton, 2010) and the spike-and-slab RBM (Courville et al., 2011b) are notoriously difficult to train. Part of our motivation for developing S3C was to provide an easy means of learning a first-layer model on real-valued data, which could then provide binary random variables as input to a more well-established model for the second layer.

4.3 Contributions

The contribution of this paper is the introduction of two algorithms for performing fast inference in the spike-and-slab sparse coding (S3C) probabilistic model. The improved speed of these algorithms on parallel architectures allows us to scale S3C to unprecedented problem sizes. In particular, S3C is now a viable feature extractor for object recognition problems. It is most useful when labeled data is scarce. We demonstrated its utility in this regime by using S3C to win a transfer learning competition.

4.4 Recent Developments

Following the invention of dropout (Hinton et al., 2012), the importance of unsupervised feature learning as a means of regularizing classifiers has significantly diminished. This has reduced the practical utility of the work presented in this article, but S3C may become useful again when computer hardware advances to the point that we are able to run models larger than can be successfully regularized with dropout alone.

5 Scaling up Spike-and-Slab Models for Unsupervised Feature Learning

5.1 Introduction

It is difficult to overstate the importance of the quality of the input features to supervised learning algorithms. A supervised learning algorithm is given a set of examples $V = \{v^{(1)}, \ldots, v^{(m)}\}$ and associated labels $\{y^{(1)}, \ldots, y^{(m)}\}$ from which it learns a mapping from v to y that can predict the labels y of new unlabeled examples v. The difficulty of this task is strongly influenced by the choice of *representation*, or the feature set used to encode the input examples v. The premise of unsupervised feature discovery is that, by learning the structure of V, we can discover a feature mapping $\phi(v)$ that renders standard supervised learning algorithms, such as the support vector machine, more effective. Because $\phi(v)$ can be learned from unlabeled data, unsupervised feature discovery can be used for semi-supervised learning (where many more unlabeled examples than labeled examples are available) or transfer learning (where the classifier will be evaluated on only a subset of the categories present in the training data).

When adopting a *deep learning* (Bengio, 2009) approach, the feature learning algorithm should discover a ϕ that consists of the composition of several simple feature mappings, each of which transforms the output of the earlier mappings in order to incrementally disentangle the factors of variation present in the data. Deep learning methods are typically created by repeatedly composing together shallow unsupervised feature learners. Examples of shallow models applied to feature discovery include sparse coding (Raina et al., 2007), restricted Boltzmann machines (RBMs) (Hinton et al., 2006; Courville et al., 2011a), various autoencoder-based models (Bengio et al., 2007; Vincent et al., 2008), and hybrids of autoencoders and sparse coding (Kavukcuoglu et al., 2010).

In this paper, we describe how to use a model which we call *spike-and-slab sparse coding* (S3C) as an efficient feature learning algorithm. We also demonstrate how to

construct a new deep model, the *partially directed deep Boltzmann machine* (PD-DBM) with S3C as its first layer. Both are models of real-valued data, and as such are well-suited to modeling images, or image-like data, such as audio that has been preprocessed into an image-like space (Deng et al., 2010). In this paper, we focus on applying these models to object recognition.

Single-layer convolutional models based on simple thresholded linear feature extractors are currently among the state-of-the-art performers on the CIFAR-10 object recognition dataset (Coates and Ng, 2011; Jia and Huang, 2011). However, the CIFAR-10 dataset contains 5,000 labels per class, and this amount of labeled data can be inconvenient or expensive to obtain for applications requiring more than 10 classes. Previous work has shown that the performance of a simple thresholded linear feature set degrades sharply in accuracy as the number of labeled examples decreases (Coates and Ng, 2011).

We introduce the use of the S3C model as a feature extractor in order to make features more robust to this degradation. This is motivated by the observation that sparse coding performs relatively well when the number of labeled examples is low (Coates and Ng, 2011). Sparse coding inference invokes a competition among the features to *explain* the data and therefore, relative to simple thresholded linear feature extractors, acts as a more regularized feature extraction scheme. We speculate that this additional regularization is responsible for its improved performance in the low-labeled-data regime. S3C can be considered as employing an alternative regularization for feature extraction where, unlike sparse coding, the sparsity prior is decoupled from the magnitude of the non-zero, real-valued feature values.

The S3C generative model can be viewed as a hybrid of sparse coding and the recently introduced spike-and-slab RBM (Courville et al., 2011b). Like the spike-and-slab RBM (ssRBM), S3C possesses a layer of hidden units composed of real-valued *slab* variables and binary *spike* variables. The binary spike variables are well suited as inputs to subsequent layers in a deep model. However, like sparse coding and unlike the ssRBM, S3C can be interpreted as a *directed* graphical model, implying that features in S3C compete with each other to explain the input. As we show, S3C can be derived either from sparse coding by replacing the factorial Laplace prior with a factorial spike-and-slab prior, or from the ssRBM, by simply adding a term to its energy function that causes the hidden units to compete with each other.

We hypothesize that S3C features have a stronger regularizing effect than sparse coding features due to the greater sparsity in the spike-and-slab prior relative to the Laplace prior. We validate this hypothesis by showing that S3C has superior performance when labeled data is scarce. We present results on the the CIFAR-10 and CIFAR-100 object classification datasets. We also describe how we used S3C to win a transfer learning challenge.

The major technical challenge in using S3C is that exact inference over the posterior of the latent layer is intractable. We derive an efficient structured variational approximation to the posterior distribution and use it to perform approximate inference as well as learning as part of a variational Expectation Maximization (EM) procedure (Saul and Jordan, 1996). Our inference algorithm allows us to scale inference and learning in the spike-and-slab coding model to the large problem sizes required for state-of-the-art object recognition.

Our use of a variational approximation for inference distinguishes S3C from standard sparse coding schemes where maximum *a posteriori* (MAP) inference is typically used. It also allows us to naturally incorporate S3C as a module of a deeper model. We introduce learning rules for the resulting PD-DBM, describe some of its interesting theoretical properties, and demonstrate how this model can be trained jointly by a single algorithm, rather than requiring the traditional greedy learning algorithm that consists of composing individually trained components (Salakhutdinov and Hinton, 2009). The ability to jointly train deep models in a single unified learning stage has the advantage that it allows the units in higher layers to influence the entire learning process at the lower layers. We anticipate that this property may become essential in the future as the size of the models increases. Consider an extremely large deep model, with size sufficient that it requires sparse connections. When this model is trained jointly, the feedback from the units in higher layers will cause units in lower layers to naturally group themselves so that each higher layer unit receives all of the information it needs in its sparse receptive field. Even in small, densely connected models, greedy training may get caught in local optimal that joint training can avoid.

5.2 Models

We now describe the models considered in this paper. We first study a model we call the spike-and-slab sparse coding (S3C) model. This model has appeared previously in the literature in a variety of different domains (Lücke and Sheikh, 2011; Garrigues and Olshausen, 2008; Mohamed et al., 2012; Zhou et al., 2009; Titsias and Lázaro-Gredilla, 2011). Next, we describe a way to incorporate S3C into a deeper model, with the primary goal of obtaining a better generative model.

5.2.1 The spike-and-slab sparse coding model

The spike-and-slab sparse coding model consists of latent binary *spike* variables $h \in \{0,1\}^N$, latent real-valued *slab* variables $s \in \mathbb{R}^N$, and real-valued visible vector $v \in \mathbb{R}^D$ generated according to this process:

$\forall i \in \{1,\ldots,N\}, d \in \{1,\ldots,D\}$,

$$\begin{aligned} p(h_i = 1) &= \sigma(b_i) \\ p(s_i \mid h_i) &= \mathcal{N}(s_i \mid h_i \mu_i, \alpha_{ii}^{-1}) \\ p(v_d \mid s, h) &= \mathcal{N}(v_d \mid W_{d:}(h \circ s), \beta_{dd}^{-1}) \end{aligned} \tag{5.1}$$

where σ is the logistic sigmoid function, b is a set of biases on h, μ and W govern the linear dependence of s on h and v on s respectively, α and β are diagonal precision matrices of their respective conditionals, and $h \circ s$ denotes the element-wise product of h and s.

To avoid overparameterizing the distribution, we constrain the columns of W to have unit norm, as in sparse coding. We restrict α to be a diagonal matrix and β to be a diagonal matrix or a scalar. We refer to the variables h_i and s_i as jointly defining the i^{th} hidden unit, so that there are a total of N rather than $2N$ hidden units. The state of a hidden unit is best understood as $h_i s_i$, that is, the spike variables gate the slab variables [1].

5.2.2 The partially directed deep Boltzmann machine model

As described above, the S3C prior is factorial over the hidden units ($h_i s_i$ pairs). Distributions such as the distribution over natural images are rarely well described

1. We can essentially recover h_i and s_i from $h_i s_i$ since $s_i = 0$ has zero measure.

by simple independent factor models, and so we expect that S3C will likely be a poor generative model for the kinds of data that we wish to consider. We now show one way of incorporating S3C into a deeper model, with the primary goal of obtaining a better generative model. If we assume that μ becomes large relative to α, then the primary structure we need to model is in h. We therefore propose placing a DBM prior rather than a factorial prior on h. The resulting model can be viewed as a deep Boltzmann machine with directed connections at the bottom layer. We call the resulting model a partially directed deep Boltzmann machine (PD-DBM).

The PD-DBM model consists of an observed input vector $v \in \mathbb{R}^D$, a vector of slab variables $s \in \mathbb{R}^{N_0}$, and a set of binary vectors $\boldsymbol{h} = \{h^{(0)}, \ldots, h^{(L)}\}$ where $h^{(l)} \in \{0,1\}^{N_l}$ and L is the number of layers added on top of the S3C model.

The model is parameterized by β, α, and μ, which play the same roles as in S3C. The parameters $W^{(l)}$ and $b^{(l)}$, $l \in \{0, \ldots, L\}$ provide the weights and biases of both the S3C model and the DBM prior attached to it.

Together, the complete model implements the following probability distribution:

$$P_{\text{PD-DBM}}(v, s, \boldsymbol{h}) = P_{\text{S3C}}(v, s | h^{(0)}) P_{\text{DBM}}(\boldsymbol{h})$$

where

$$P_{\text{DBM}}(\boldsymbol{h}) \propto \exp\left(- \sum_{l=0}^{L} b^{(l)T} h^{(l)} - \sum_{l=1}^{L} h^{(l-1)T} W^{(l)} h^{(l)} \right).$$

A version of the model with three hidden layers ($L = 2$) is depicted graphically in Fig. 5.1.

Besides admitting a straightforward learning algorithm, the PD-DBM has several useful properties:

- The partition function exists for all parameter settings. This is not true of the spike-and-slab restricted Boltzmann machine (ssRBM), which is a very good generative model of natural images (Courville et al., 2011b).
- The model family is a universal approximator. The DBM portion, which is a universal approximator of binary distributions (Le Roux and Bengio, 2008), can implement a one-hot prior on $h^{(0)}$, thus turning the overall model into a mixture of Gaussians, which is a universal approximator of real-valued distributions (Titterington et al., 1985).
- Inference of the posterior involves feedforward, feedback, and lateral connec-

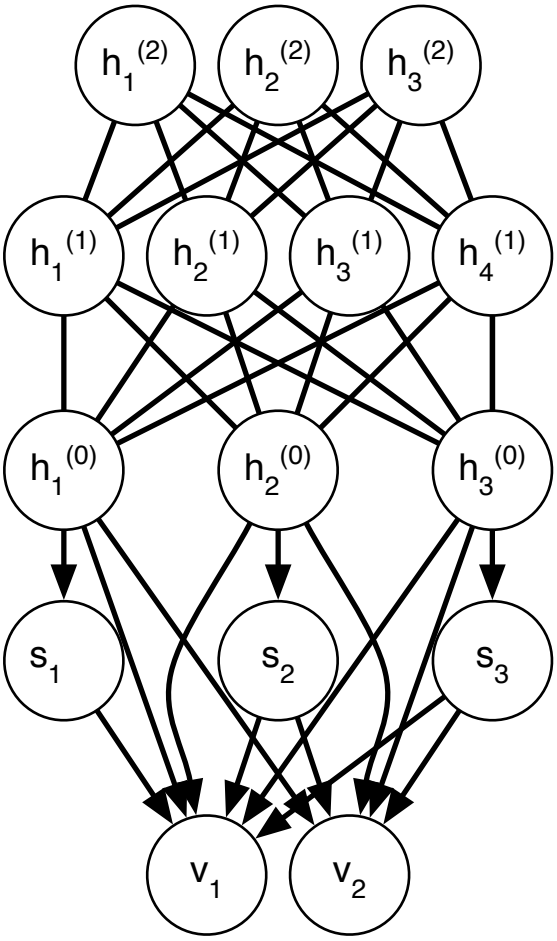


Figure 5.1: A graphical model depicting an example PD-DBM.

tions. This increases the biological plausibility of the model, and enables it to learn and exploit several rich kinds of interactions between features. The lateral interactions make the lower level features compete to explain the input, and the top-down influences help to obtain the correct representations of ambiguous input.

5.3 Learning procedures

Maximum likelihood learning is intractable for both models. S3C suffers from an intractable posterior distribution over the latent variables. In addition to an intractable posterior distribution, the PD-DBM suffers from an intractable partition function.

We follow the variational learning approach used by Salakhutdinov and Hinton (2009) to train DBMs: rather than maximizing the log likelihood, we maximize a variational lower bound on the log likelihood. In the case of the PD-DBM we must do so using a stochastic approximation of the gradient.

The basic strategy of variational learning is to approximate the true posterior $P(h, s \mid v)$ with a simpler distribution $Q(h, s)$. The choice of Q induces a lower bound on the log likelihood called the negative variational free energy. The term of the negative variational free energy that depends on the model parameters is

$$\begin{aligned} &\mathbb{E}_{s,h\sim Q}[\log P(v, s, h)] \\ =& -\mathbb{E}_{s,h\sim Q}[\log P(v \mid s, h^{(0)}) + \log P(s \mid h) + \log P(h)] \end{aligned}$$

In the case of S3C, this bound is tractable, and can be optimized in a straightforward manner. It is even possible to use variational EM (Saul and Jordan, 1996) to make large, closed-form jumps in parameter space. However, we find gradient ascent learning to be preferable in practice, due to the computational expense of the closed-form solution, which involves estimating and inverting the covariance matrix of all of the hidden units.

In the case of the PD-DBM, the objective function is not tractable because the partition function of the DBM portion of the model is not tractable. We can use contrastive divergence (Hinton, 2000) or stochastic maximum likelihood (Younes, 1998; Tieleman, 2008) to make a sampling-based approximation to the

DBM partition function's contribution to the gradient. Thus, unlike S3C, we must do gradient-based learning rather than closed-form parameter updates. However, the PD-DBM model still has some nice properties in that only a subset of the variables must be sampled during training. The factors of the partition function originating from the S3C portion of the model are still tractable. In particular, training does not ever require sampling real-valued variables. This is a nice property because it means that the gradient estimates are bounded for fixed parameters and data. When sampling real-valued variables, it is possible for the sampling procedure to make gradient estimates arbitrarily large.

We found that using the "true gradient" (Douglas et al., 1999) method to be useful for learning with the norm constraint on W. We also found that using momentum (Hinton, 2010) is very important for learning PD-DBMs.

5.3.1 Avoiding greedy pretraining

Deep models are commonly pretrained in a greedy layerwise fashion. For example, a DBM is usually initialized from a stack of RBMs, with one RBM trained on the data and each of the other RBMs trained on samples of the previous RBM's hidden layer.

Any greedy training procedure can obviously get stuck in a local minimum. Avoiding the need for greedy training could thus result in better models. For example, when pretraining with an RBM, the lack of explaining away in the posterior prevents the first layer from learning nearly parallel weight vectors, since these would result in similar activations (up to the bias term, which could simply make one unit always less active than the other). Even though the deeper layers of the DBM could implement the explaining away needed for these weight vectors to function correctly (ie, to have the one that resembles the input the most activate, and inhibit the other unit), the greedy learning procedure does not have the opportunity to learn such weight vectors.

Previous efforts at jointly training even two layer DBMs on MNIST have failed (Salakhutdinov and Hinton, 2009; Desjardins et al., 2012; Montavon and Müller, 2012). Typically, the jointly trained DBM does not make good use of the second layer, either because the second layer weights are very small or because they contain several duplicate weights focused on a small subset of first layer units that became

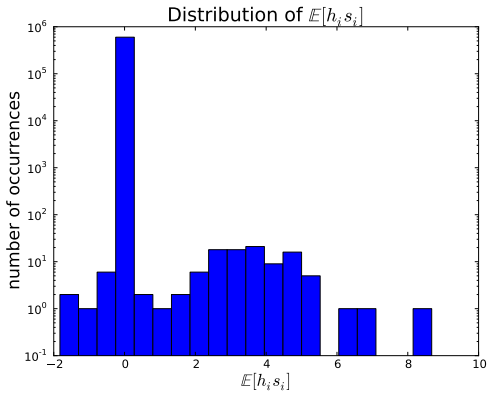


Figure 5.2: This example histogram of $\mathbb{E}_Q[h_i s_i]$ shows that Q is a sparse distribution. For this 6,000 hidden unit S3C model trained on 6×6 image patches, $Q(h_i = 1) < .01$ 99.7% of the time.

active early during training. We hypothesize that this is because the second layer hidden units in a DBM must both learn to model correlations in the first layer induced by the data and to counteract correlations in the first layer induced by the model family. When the second layer weights are set to 0, the DBM prior acts to correlate hidden units that have similar weight vectors (see Section 5.5.2).

The PD-DBM model avoids this problem. When the second layer weights are set to 0, the first layer hidden units are independent in the PD-DBM prior (essentially the S3C prior). The second layer thus has only one task: to model the correlations between first layer units induced by the data. As we will show, this hypothesis is supported by the fact that we are able to succesfully train a two layer PD-DBM without greedy pre-training.

5.4 Inference procedures

The goal of variational inference is to maximize the lower bound on the log likelihood with respect to the approximate distribution Q over the unobserved variables. This is accomplished by selecting the Q that minimizes the Kullback–Leibler divergence:

$$\mathcal{D}_{KL}(Q(h, s) \| P(h, s|v)) \tag{5.2}$$

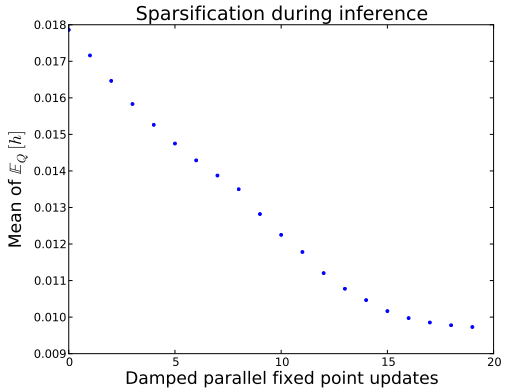


Figure 5.3: The explaining-away effect makes the S3C representation become more sparse with each damped iteration of the variational inference fixed point equations.

where $Q(h, s)$ is drawn from a restricted family of distributions. This family can be chosen to ensure that learning and inference with Q is tractable.

Variational inference can be seen as analogous to the encoding step of the traditional sparse coding algorithm. The key difference is that while sparse coding approximates the true posterior with a MAP point estimate of the latent variables, variational inference approximates the true posterior everywhere with the distribution Q.

5.4.1 Variational inference for S3C

When working with S3C, we constrain Q to be drawn from the family $Q(h, s) = \Pi_i Q(h_i, s_i)$. This is a richer approximation than the fully factorized family used in the mean field approximation. It allows us to capture the tight correlation between each spike variable and its corresponding slab variable while still allowing simple and efficient inference in the approximating distribution. It also avoids a pathological condition in the mean field distribution where $Q(s_i)$ can never be updated if $Q(h_i) = 0$.

Observing that eq. (5.2) is an instance of the Euler-Lagrange equation, we find that the solution must take the form

$$\begin{aligned} Q(h_i = 1) &= \hat{h}_i, \\ Q(s_i \mid h_i) &= \mathcal{N}(s_i \mid h_i \hat{s}_i,\ (\alpha_i + h_i W_i^\top \beta W_i)^{-1}) \end{aligned} \tag{5.3}$$

where $\hat{h}_i$ and $\hat{s}_i$ must be found by an iterative process. In a typical applica-

tion of variational inference, the iterative process consists of sequentially applying fixed point equations that give the optimal value of the parameters $\hat{h}_i$ and $\hat{s}_i$ for one factor $Q(h_i, s_i)$ given the value all of the other factors' parameters. This is for example the approach taken by Titsias and Lázaro-Gredilla (2011) who independently developed a variational inference procedure for the same problem. This process is only guaranteed to decrease the KL divergence if applied to each factor sequentially, i.e. first updating $\hat{h}_1$ and $\hat{s}_1$ to optimize $Q(h_1, s_1)$, then updating $\hat{h}_2$ and $\hat{s}_2$ to optimize $Q(h_2, s_2)$, and so on. In a typical application of variational inference, the optimal values for each update are simply given by the solutions to the Euler-Lagrange equations. For S3C, we make three deviations from this standard approach.

Because we apply S3C to very large-scale problems, we need an algorithm that can fully exploit the benefits of parallel hardware such as GPUs. Sequential updates across all N factors require far too much run-time to be competitive in this regime.

We have considered two different methods that enable parallel updates to all units. In the first method, we start each iteration by partially minimizing the KL divergence with respect to $\hat{s}$. The terms of the KL divergence that depend on $\hat{s}$ make up a quadratic function so this can be minimized via conjugate gradient descent. We implement conjugate gradient descent efficiently by using the R-operator (Pearlmutter, 1994) to perform Hessian-vector products rather than computing the entire Hessian explicitly (Schraudolph, 2002). This step is guaranteed to improve the KL divergence on each iteration. We next update $\hat{h}$ in parallel, shrinking the update by a damping coefficient. This approach is not guaranteed to decrease the KL divergence on each iteration but it is a widely applied approach that works well in practice (Koller and Friedman, 2009).

With the second method (Algorithm 1), we find in practice that we obtain faster convergence, reaching equally good solutions by replacing the conjugate gradient update to $\hat{s}$ with a more heuristic approach. We use a parallel damped update on $\hat{s}$ much like what we do for $\hat{h}$. In this case we make an additional heuristic modification to the update rule which is made necessary by the unbounded nature of $\hat{s}$. We clip the update to $\hat{s}$ so that if $\hat{s}_{\text{new}}$ has the opposite sign from $\hat{s}$, its magnitude is at most $\rho|\hat{s}|$. In all of our experiments we used $\rho = 0.5$ but any value in $[0, 1]$ is sensible. This prevents a case where multiple mutually inhibitory s units inhibit each other so strongly that rather than being driven to 0 they change

sign and actually increase in magnitude. This case is a failure mode of the parallel updates that can result in $\hat{s}$ amplifying without bound if clipping is not used.

Note that Algorithm 1 does not specify a convergence criterion. Many convergence criteria are possible–the convergence criterion could be based on the norm of the gradient of the KL divergence with respect to the variational parameters, the amount that the KL divergence has decreased in the last iteration, or the amount that the variational parameters have changed in the final iteration. Salakhutdinov and Hinton (2009) use the third approach when training deep Boltzmann machines and we find that it works well for S3C and the PD-DBM as well.

Algorithm 1 `Fixed-Point Inference`

Initialize $\hat{h}(0) = \sigma(b)$, $\hat{s}(0) = \mu$, and $k = 0$.
while not converged **do**
Compute the individually optimal value $\hat{s}_i^*$ for each i simultaneously:

$$\hat{s}_i^* = \frac{\mu_i \alpha_{ii} + v^\top \beta W_i - W_i \beta \left[\sum_{j \neq i} W_j \hat{h}_j \hat{s}_j(k)\right]}{\alpha_{ii} + W_i^\top \beta W_i}$$

Clip reflections by assigning

$$c_i = \rho \text{sign}(\hat{s}_i^*) |\hat{s}_i(k)|$$

for all i such that $\text{sign}(\hat{s}_i^*) \neq \text{sign}(\hat{s}_i(k))$ and $|\hat{s}_i^*| > \rho |\hat{s}_i(k)|$, and assigning $c_i = \hat{s}_i^*$ for all other i.
Damp the updates by assigning

$$\hat{s}(k+1)_i = \eta_s c + (1 - \eta_s)\hat{s}(k)$$

where $\eta_s \in (0, 1]$.
Compute the individually optimal values for $\hat{h}$:

$$z_i = \left(v - \sum_{j \neq i} W_j \hat{s}_j(k+1)\hat{h}_j(k) - \frac{1}{2} W_i \hat{s}_i(k+1)\right)^\top \beta W_i \hat{s}_i(k+1) + b_i - \frac{1}{2}\alpha_{ii}(\hat{s}_i(k+1) - \mu_i)^2 - \frac{1}{2}\log(\alpha_{ii} + W_i^\top \beta W_i) + \frac{1}{2}\log(\alpha_{ii})$$

$$\hat{h}_i^* = \sigma(z)$$

Damp the update to $\hat{h}$:

$$\hat{h}(k+1) = \eta_h \hat{h}^* + (1 - \eta_h)\hat{h}(k)$$

$k \leftarrow k + 1$
end while

We include some visualizations that demonstrate the effect of our inference procedure. Figure 5.2 shows that it produces a sparse representation. Figure 5.3 shows that the explaining-away effect incrementally makes the representation more sparse. Figure 5.4 shows that the inference procedure increases the negative variational free energy.

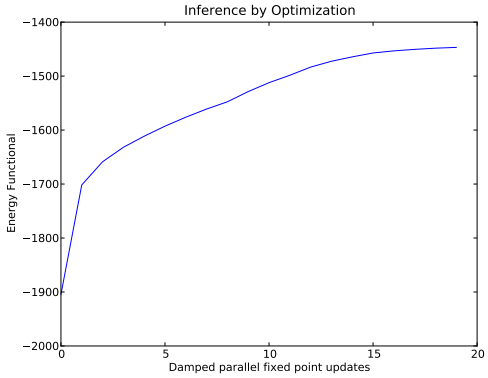


Figure 5.4: The negative variational free energy of a batch of 5000 image patches increases during the course of variational inference.

5.4.2 Variational inference for the PD-DBM

Inference in the PD-DBM is very similar to inference in S3C. We use the variational family

$$Q(s, h) = \Pi_{i=1}^{N_0} Q(s_i, h_i^{(0)}) \Pi_{l=1}^{L} \Pi_{i=1}^{N_l} Q(h_i^{(l)})$$

whose solutions take the form

$$Q(h_i^{(l)} = 1) = \hat{h}_i^{(l)},$$
$$Q(s_i \mid h_i^{(0)}) = \mathcal{N}(s_i \mid h_i^{(0)} \hat{s}_i,\ (\alpha_i + h_i W_i^\top \beta W_i)^{-1}).$$

We apply more or less the same inference procedure as in S3C. On each update step we update either $\hat{s}$ or $\hat{h}^{(l)}$ for some value of l. The update to $\hat{s}$ is exactly the same as in S3C. The update to $\hat{h}^{(0)}$ changes slightly to incorporate top-down influence from $\hat{h}^{(1)}$. When computing the individually optimal values of the elements of $\hat{h}^{(0)}$ we use the following fixed-point formula:

$$\hat{h}_i^{(0)*} = \sigma(z_i + W^{(1)} \hat{h}^{(1)})$$

The update to $\hat{h}^{(l)}$ for $l > 0$ is simple; it is the same as the mean field update in the DBM. No damping is necessary for this update. The conditional independence properties of the DBM guarantee that the optimal values of the elements of $\hat{h}^{(l)}$ do not depend on each other, so the individually optimal values are globally optimal (for a given $\hat{h}^{(l-1)}$ and $\hat{h}^{(l+1)}$). The update is given by

$$\hat{h}^{(l)*} = \sigma\left(b^{(l)} + \hat{h}^{(l-1)T} W^{(l)} + W^{(l+1)} \hat{h}^{(l+1)}\right)$$

where the term for layer $l + 1$ is dropped if $l + 1 > L$.

5.5 Comparison to other feature encoding methods

Here we compare S3C as a feature discovery algorithm to other popular approaches. We describing how S3C occupies a middle ground between two of these methods, sparse coding and the ssRBM, while avoiding many of the respective disadvantages when applied as feature discovery algorithms.

5.5.1 Comparison to sparse coding

Sparse coding (Olshausen and Field, 1997) has been widely used to discover features for classification (Raina et al., 2007). Recently Coates and Ng (2011) showed that this approach achieves excellent performance on the CIFAR-10 object recognition dataset. Sparse coding refers to a class of generative models where the observed data v is normally distributed given a set of continuous latent variables s and a dictionary matrix W: $v \sim \mathcal{N}(Ws, \sigma\mathbb{I})$. Sparse coding places a factorial and heavy tailed prior distribution over s (e.g. a Cauchy or Laplace distribution) chosen to encourage the mode of the posterior $p(s \mid v)$ to be sparse. One can derive the S3C model from sparse coding by replacing the factorial Cauchy or Laplace prior with a spike-and-slab prior.

One drawback of sparse coding is that the latent variables are not merely encouraged to be sparse; they are encouraged to remain close to 0, even when they are active. This kind of regularization is not necessarily undesirable, but in the case of simple but popular priors such as the Laplace prior (corresponding to an L_1 penalty on the latent variables s), the degree of regularization on active units is confounded with the degree of sparsity. There is little reason to believe that in realistic settings, these two types of complexity control should be so tightly bound together. The S3C model avoids this issue by controlling the sparsity of units via the b parameter that determines how likely each spike unit is to be active, while separately controlling the magnitude of active units via the μ and α parameters that govern the distribution over s. Sparse coding has no parameter analogous to μ and cannot control these aspects of the posterior independently.

Another drawback of sparse coding is that the factors are not actually sparse in the generative distribution. Indeed, each factor is zero with probability zero. The

features extracted by sparse coding are only sparse because they are obtained via MAP inference. In the S3C model, the spike variables ensure that each factor is zero with non-zero probability in the generative distribution. Since this places a greater restriction on the code variables, we hypothesize that S3C features provide more of a regularizing effect when solving classification problems.

Sparse coding is also difficult to integrate into a deep generative model of data such as natural images. While Yu et al. (2011) and Zeiler et al. (2011) have recently shown some success at learning hierarchical sparse coding, our goal is to integrate the feature extraction scheme into a proven generative model framework such as the deep Boltzmann machine (Salakhutdinov and Hinton, 2009). Existing inference schemes known to work well in the DBM-type (deep Boltzmann machine) setting are all either sample-based or are based on variational approximations to the model posteriors, while sparse coding schemes typically employ MAP inference. Our use of variational inference makes the S3C framework well suited to integrate into the known successful strategies for learning and inference in DBM models. In fact, the compatibility of the S3C and DBM inference procedures is confirmed by the success of the PD-DBM inference procedure. It is not obvious how one can employ a variational inference strategy to standard sparse coding with the goal of achieving sparse feature encoding.

Sparse coding models can be learned efficiently by alternately running MAP inference for several examples and then making a large, closed-form updates to the parameters. The same approach is also possible with S3C, and is in fact more principled since it is based on maximizing a variational lower bound rather than the MAP approximation. We do not explore this learning method for S3C in this paper.

5.5.2 Comparison to restricted Boltzmann machines

The S3C model also resembles another class of models commonly used for feature discovery: the RBM. An RBM (Smolensky, 1986) is a model defined through an energy function that describes the interactions between the observed data variables and a set of latent variables. It is possible to interpret the S3C as an energy-based model, by rearranging $p(v, s, h)$ to take the form $\exp\{-E(v, s, h)\}/Z$, with

the following energy function:

$$E(v, s, h) = \frac{1}{2}\left(v - \sum_i W_i s_i h_i\right)^\top \beta \left(v - \sum_i W_i s_i h_i\right) + \frac{1}{2}\sum_{i=1}^{N} \alpha_i (s_i - \mu_i h_i)^2 - \sum_{i=1}^{N} b_i h_i, \tag{5.4}$$

The ssRBM model family is a good starting point for S3C because it has demonstrated both reasonable performance as a feature discovery scheme and remarkable performance as a generative model (Courville et al., 2011b). Within the ssRBM family, S3C's closest relative is a variant of the μ-ssRBM, defined by the following energy function:

$$E(v, s, h) = -\sum_{i=1}^{N} v^T \beta W_i s_i h_i + \frac{1}{2} v^T \beta v + \frac{1}{2}\sum_{i=1}^{N} \alpha_i (s_i - \mu_i h_i)^2 - \sum_{i=1}^{N} b_i h_i, \tag{5.5}$$

where the variables and parameters are defined identically to those in S3C. Comparison of equations 5.4 and 5.5 reveals that the simple addition of a latent factor interaction term $\frac{1}{2}(h \circ s)^\top W^\top \beta W (h \circ s)$ to the ssRBM energy function turns the ssRBM into the S3C model. With the inclusion of this term S3C moves from an undirected ssRBM model to the directed graphical model described in equation (5.1). This change from undirected modeling to directed modeling has three important effects, that we describe in the following paragraphs:

The effect on the partition function: The most immediate consequence of the transition to directed modeling is that the partition function becomes tractable. Because the RBM parition function is intractable, most training algorithms for the RBM require making stochastic approximations to the partition function, the same as our learning procedure for the PD-DBM does. Since the S3C partition function is tractable, we can follow its true gradient, which provides one advantage over the RBM. The partition function of S3C is also guaranteed to exist for all possible settings of the model parameters, which is not true of the ssRBM. In the ssRBM, for some parameter values, it is possible for $p(s, v \mid h)$ to take the form of a normal distribution whose covariance matrix is not positive definite. Courville et al. (2011b) have explored resolving this issue by constraining the parameters, but this

was found to hurt classification performance.

The effect on the posterior: RBMs have a factorial posterior, but S3C and sparse coding have a complicated posterior due to the "explaining away" effect. For this reason, RBMs can use exact inference and maximum likelihood estimation. Models with an intractable posterior such as S3C and DBMs must use approximate inference and are often trained with a variational lower bound on the likelihood.

The RBMs factorial posterior means that features defined by similar basis functions will have similar activations, while in directed models, similar features will compete so that only the most relevant features will remain significantly active. As shown by Coates and Ng (2011), the sparse Gaussian RBM is not a very good feature extractor – the set of basis functions W learned by the RBM actually work better for supervised learning when these parameters are plugged into a sparse coding model than when the RBM itself is used for feature extraction. We think this is due to the factorial posterior. In the vastly overcomplete setting, being able to selectively activate a small set of features that cooperate to explain the input likely provides S3C a major advantage in discriminative capability.

Considerations of biological plausibility also motivate the use of a model with a complicated posterior. As described in (Hyvärinen et al., 2009), a phenomenon called "end stopping" similar to explaining away has been observed in V1 simple cells. End-stopping occurs when an edge detector is inhibited when retinal cells near the ends of the edge it detects are stimulated. The inhibition occurs due to lateral interactions with other simple cells, and is a major motivation for the lateral interactions present in the sparse coding posterior.

The effect on the prior: The addition of the interaction term causes S3C to have a factorial prior. This probably makes it a poor generative model, but this is not a problem for the purpose of feature discovery. Moreover, the quality of the generative model can be improved by incorporating S3C into a deeper architecture, as we will show.

RBMs were designed with a nonfactorial prior because factor models with factorial priors are generally known to result in poor generative models. However, in the case of real-valued data, typical RBM priors are not especially useful. For

example, the ssRBM variant described in eq. (5.5) has the following prior:

$$p(s,h) \propto \exp\left\{\frac{1}{2}\left(\sum_{i=1}^{N} W_i s_i h_i\right)^{\top} \beta \left(\sum_{i=1}^{N} W_i s_i h_i\right) - \frac{1}{2}\sum_{i=1}^{N} \alpha_i \left(s_i - \mu_i h_i\right)^2 + \sum_{i=1}^{N} b_i h_i\right\}.$$

It is readily apparent from the first term (all other terms factorize across hidden units) that this prior acts to correlate units that have similar basis vectors, which is almost certainly not a desirable property for feature extraction tasks. Indeed it is this nature of the RBM prior that causes both the desirable (easy computation) and undesirable (no explaining away) properties of the posterior.

5.5.3 Other related work

The notion of a spike-and-slab prior was established in statistics by Mitchell and Beauchamp (1988). Outside the context of unsupervised feature discovery for supervised learning, the basic form of the S3C model (i.e. a spike-and-slab latent factor model) has appeared a number of times in different domains (Lücke and Sheikh, 2011; Garrigues and Olshausen, 2008; Mohamed et al., 2012; Zhou et al., 2009; Titsias and Lázaro-Gredilla, 2011). To this literature, we contribute an inference scheme that scales to the kinds of object classifications tasks that we consider. We outline this inference scheme next.

5.6 Runtime results

Our inference scheme achieves very good computational performance, both in terms of memory consumption and in terms of run-time. The computational bottleneck in our classification pipeline is SVM training, not feature learning or feature extraction.

Comparing the computational cost of our inference scheme to others is a difficult task because it could be confounded by differences in implementation and because it is not clear exactly what sparse coding problem is equivalent to an equivalent spike-and-slab sparse coding problem. However, we observed informally during our

supervised learning experiments that feature extraction using S3C took roughly the same amount of time as feature extraction using sparse coding.

In Fig. 5.5, we show that our improvements to spike-and-slab inference performance allow us to scale spike-and-slab modeling to the problem sizes needed for object recognition tasks. Previous work on spike-and-slab modeling was not able to use similar amounts of hidden units or training examples.

As a large-scale test of our inference scheme's ability, we trained over 8,000 densely-connected filters on full 32×32 color images. A visualization of the learned filters is shown in Fig. 5.6. This test demonstrated that our approach scales well to large (over 3,000 dimensional) inputs, though it is not yet known how to use features for classification as effectively as patch-based features which can be incorporated into a convolutional architecture with pooling. For comparison, to our knowledge the largest image patches used in previous spike-and-slab models with lateral interactions were 16×16 (Garrigues and Olshausen, 2008).

Finally, we performed a series of experiments to compare our heuristic method of updating $\hat{s}$ with the conjugate gradient method of updating $\hat{s}$. The conjugate gradient method is guaranteed to reduce the KL divergence on each update to $\hat{s}$. The heuristic method has no such guarantee. These experiments provide an empirical justification for the use of the heuristic method.

We considered three different models, each on a different dataset. We used MNIST (LeCun et al., 1998), CIFAR-100 (Krizhevsky and Hinton, 2009) , and whitened 6×6 patches drawn from CIFAR-100 as the three datasets.

Because we wish to compare different inference algorithms and inference affects learning, we did not want to compare the algorithms on models whose parameters were the result of learning. Instead we obtained the value of W by drawing randomly selected patches ranging in size from 6×6 to the full image size for each dataset. This provides a data-driven version of W with some of the same properties like local support that learned filters tend to have. None of the examples used to initialize W were used in the later timing experiments. We initialized b, μ, α, and β randomly. We used 400 hidden units for some experiments and 1600 units for others, to investigate the effect of overcompleteness on runtime.

For each inference scheme considered, we found the fastest possible variant obtainable via a two-dimensional grid search over η_h and either η_s in the case of the heuristic method or the number of conjugate gradient steps to apply per $\hat{s}$

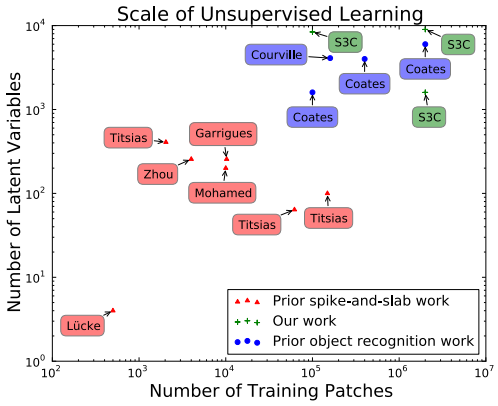


Figure 5.5: Our inference scheme enables us to extend spike-and-slab modeling from small problems to the scale needed for object recognition. Previous object recognition work is from (Coates and Ng, 2011; Courville et al., 2011b). Previous spike-and-slab work is from (Mohamed et al., 2012; Zhou et al., 2009; Garrigues and Olshausen, 2008; Lücke and Sheikh, 2011; Titsias and Lázaro-Gredilla, 2011).

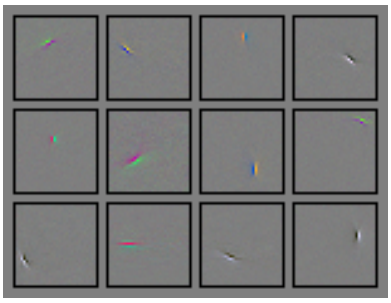


Figure 5.6: Example filters from a dictionary of over 8,000 learned on full 32x32 images.

update in the case of the conjugate gradient method. We used the same value of these parameters on every pair of update steps. It may be possible to obtain faster results by varying the parameters throughout the course of inference.

For these timing experiments, it is necessary to make sure that each algorithm is not able to appear faster by converging early to an incorrect solution. We thus replace the standard convergence criterion based on the size of the change in the variational parameters with a requirement that the KL divergence reach within 0.05 on average of our best estimate of the true minimum value of the KL divergence found by batch gradient descent.

All experiments were performed on an Nvidia Ge-Force GTX-580.

The results are summarized in Fig. 5.7.

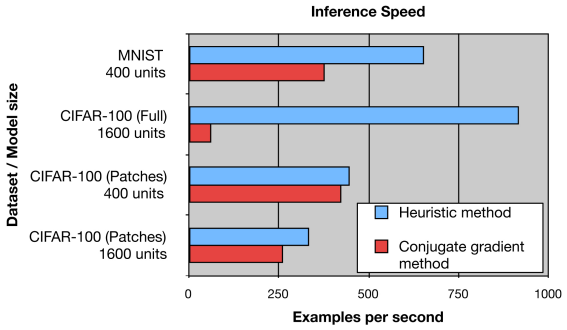


Figure 5.7: The inference speed for each method was computed based on the inference time for the same set of 100 examples from each dataset. The heuristic method is consistently faster than the conjugate gradient method. The conjugate gradient method is slowed more by problem size than the heuristic method is, as shown by the conjugate gradient method's low speed on the CIFAR-100 full image task. The heuristic method has a very low cost per iteration but is strongly affected by the strength of explaining-away interactions–moving from CIFAR-100 full images to CIFAR-100 patches actually slows it down because the degree of overcompleteness increases.

5.7 Classification results

Because S3C forms the basis of all further model development in this line of research, we concentrate on validating its value as a feature discovery algorithm. We conducted experiments to evaluate the usefulness of S3C features for supervised learning on the CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton, 2009) datasets. Both datasets consist of color images of objects such as animals and vehicles. Each contains 50,000 train and 10,000 test examples. CIFAR-10 contains 10 classes while CIFAR-100 contains 100 classes, so there are fewer labeled examples per class in the case of CIFAR-100.

For all experiments, we used the same overall procedure as Coates and Ng (2011) except for feature learning. CIFAR-10 consists of 32×32 images. We train our feature extractor on 6×6 contrast-normalized and ZCA-whitened patches from the training set (this preprocessing step is not necessary to obtain good performance with S3C; we included it primarily to facilitate comparison with other work). At test time, we extract features from all 6×6 patches on an image, then average-pool them. The average-pooling regions are arranged on a non-overlapping grid. Finally, we train an L2-SVM with a linear kernel on the pooled features.

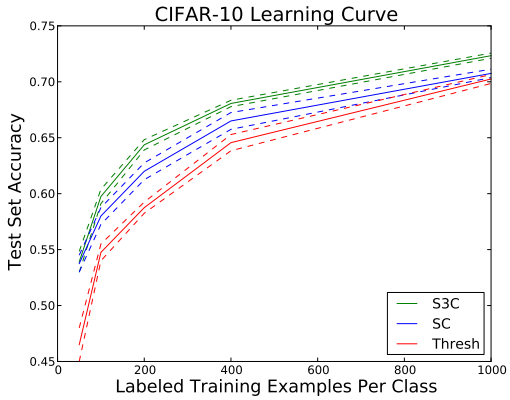


Figure 5.8: Semi-supervised classification accuracy on subsets of CIFAR-10. Thresholding, the best feature extractor on the full dataset, performs worse than sparse coding when few labels are available. S3C improves upon sparse coding's advantage.

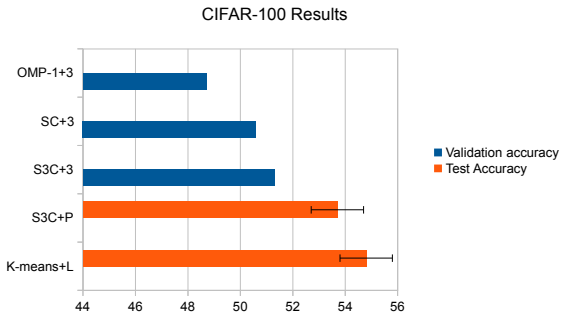


Figure 5.9: CIFAR-100 classification accuracy for various models. As expected, S3C outperforms SC (sparse coding) and and OMP-1. S3C with spatial pyramid pooling is near the state-of-the-art method, which uses a learned pooling structure.

5.7.1 CIFAR-10

We use CIFAR-10 to evaluate our hypothesis that S3C is similar to a more regularized version of sparse coding.

Coates and Ng (2011) used 1600 basis vectors in all of their sparse coding experiments. They post-processed the sparse coding feature vectors by splitting them into the positive and negative part for a total of 3200 features per average-pooling region. They average-pool on a 2×2 grid for a total of 12,800 features per image (i.e. each element of the 2×2 grid averages over a block with sides $\lceil (32 - 6 + 1)/2 \rceil$ or $\lfloor (32 - 6 + 1)/2 \rfloor$). We used $\mathbb{E}_Q[h]$ as our feature vector. Unlike the output of sparse coding, this does not have a negative part, so using a 2×2 grid we would have only 6,400 features. In order to compare with similar sizes of feature vectors we used a 3×3 pooling grid for a total of 14,400 features (i.e. each element of the 3×3 grid averages over 9×9 locations) when evaluating S3C. To ensure this is a fair means of comparison, we confirmed that running sparse coding with a 3×3 grid and absolute value rectification performs worse than sparse coding with a 2×2 grid and sign splitting (76.8% versus 77.9% on the validation set).

We tested the regularizing effect of S3C by training the SVM on small subsets of the CIFAR-10 training set, but using features that were learned on patches drawn from the entire CIFAR-10 train set. The results, summarized in Figure 5.8, show that S3C has the advantage over both thresholding and sparse coding for a wide range of amounts of labeled data. (In the extreme low-data limit, the confidence interval becomes too large to distinguish sparse coding from S3C).

On the full dataset, S3C achieves a test set accuracy of 78.3 ± 0.9 % with 95% confidence. Coates and Ng (2011) do not report test set accuracy for sparse coding with "natural encoding" (i.e., extracting features in a model whose parameters are all the same as in the model used for training) but sparse coding with different parameters for feature extraction than training achieves an accuracy of 78.8 ± 0.9% (Coates and Ng, 2011). Since we have not enhanced our performance by modifying parameters at feature extraction time these results seem to indicate that S3C is roughly equivalent to sparse coding for this classification task. S3C also outperforms ssRBMs, which require 4,096 basis vectors per patch and a 3×3 pooling grid to achieve 76.7 ± 0.9% accuracy. All of these approaches are close to the best result, using the pipeline from Coates and Ng (2011), of 81.5% achieved using thresholding of linear features learned with OMP-1. These results show that

S3C is a useful feature extractor that performs comparably to the best approaches when large amounts of labeled data are available.

5.7.2 CIFAR-100

Having verified that S3C features help to regularize a classifier, we proceed to use them to improve performance on the CIFAR-100 dataset, which has ten times as many classes and ten times fewer labeled examples per class. We compare S3C to two other feature extraction methods: OMP-1 with thresholding, which Coates and Ng (2011) found to be the best feature extractor on CIFAR-10, and sparse coding, which is known to perform well when less labeled data is available. We evaluated only a single set of hyperparameters for S3C. For sparse coding and OMP-1 we searched over the same set of hyperparameters as Coates and Ng (2011) did: $\{0.5, 0.75, 1.0, 1.25, 1.25\}$ for the sparse coding penalty and $\{0.1, 0.25, 0.5, 1.0\}$ for the thresholding value. In order to use a comparable amount of computational resources in all cases, we used at most 1600 hidden units and a 3×3 pooling grid for all three methods. For S3C, this was the only feature encoding we evaluated. For SC (sparse coding) and OMP-1, which double their number of features via sign splitting, we also evaluated 2×2 pooling with 1600 latent variables and 3×3 pooling with 800 latent variables to be sure the models do not suffer from overfitting caused by the larger feature set. These results are summarized in Fig. 5.9.

The best result to our knowledge on CIFAR-100 is $54.8 \pm 1\%$ (Jia and Huang, 2011), achieved using a learned pooling structure on top of "triangle code" features from a dictionary learned using k-means. This feature extractor is very similar to thresholded OMP-1 features and is known to perform slightly worse on CIFAR-10. The validation set results, which all use the same control pooling layer, in Fig. 5.9 show that S3C is the best known detector layer on CIFAR-100. Using a pooling strategy of concatenating 1×1, 2×2 and 3×3 pooled features we achieve a test set accuracy of $53.7 \pm 1\%$.

5.7.3 Transfer learning challenge

For the NIPS 2011 Workshop on Challenges in Learning Hierarchical Models (Le et al., 2011), the organizers proposed a transfer learning competition. This competition used a dataset consisting of 32×32 color images, including 100,000

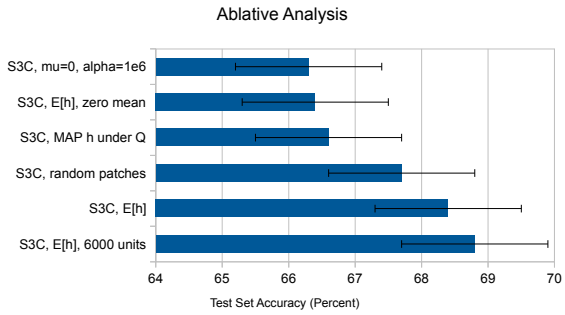


Figure 5.10: Performance of several limited variants of S3C

unlabeled examples, 50,000 labeled examples of 100 object classes not present in the test set, and 120 labeled examples of 10 object classes present in the test set. The test set was not made public until after the competition. We recognized this contest as a chance to demonstrate S3C's ability to perform well with extremely small amounts of labeled data. We chose to disregard the 50,000 labels and treat this as a semi-supervised learning task.

We applied the same approach as on the CIFAR datasets, albeit with a small modification to the SVM training procedure. Due to the small labeled dataset size, we used leave-one-out cross validation rather than five fold cross validation.

We won the competition, with a test set accuracy of 48.6 %. We do not have any information about the competing entries, other than that we outperformed them. Our test set accuracy was tied with a method run by the contest organizers, based on a combination of methods (Coates et al., 2011; Le et al., 2011). Since these methods do not use transfer learning either, this suggests that the contest primarily provides evidence that S3C is a powerful semi-supervised learning tool.

5.7.4 Ablative analysis

In order to better understand which aspects of our S3C object classification method are most important to obtaining good performance, we conducted a series of ablative analysis experiments. For these experiments we trained on 5,000 labels of the STL-10 dataset (Coates et al., 2011). Previous work on the STL-10 dataset is based on training on 1,000 label subsets of the training set, so the performance numbers in this section should only be compared to each other, not to previous work. The results are presented in Fig. 5.10.

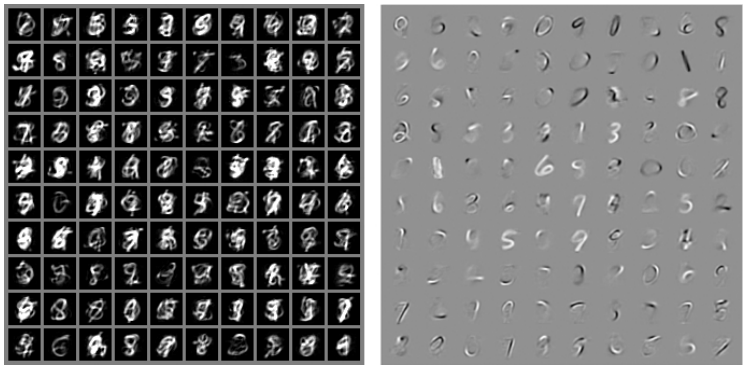


Figure 5.11: *Left:* Samples drawn from an S3C model trained on MNIST. *Right:* The filters used by this S3C model.

Our best-performing method uses $\mathbb{E}_Q[h]$ as features. This allows us to abstract out the s variables, so that they achieve a form of per-component brightness invariance. Our experiments show that including the s variables or using MAP inference in Q rather than an expectation hurts classification performance. We experimented with fixing μ to 0 so that s is regularized to be small as well as sparse, as in sparse coding. We found that this hurts performance even more. Lastly we experimented with replacing S3C learning with simply assigning W to be a set of randomly selected patches from the training set. We call this approach S3C-RP. We found that this does not impair performance much, so learning is not very important compared to our inference algorithm. This is consistent with Coates and Ng (2011)'s observations that the feature extractor matters more than the learning algorithm and that learning matters less for large numbers of hidden units.

5.8 Sampling results

In order to demonstrate the improvements in the generative modeling capability conferred by adding a DBM prior on h, we trained an S3C model and a PD-DBM model on the MNIST dataset. We chose to use MNIST for this portion of the experiments because it is easy for a human observer to qualitatively judge whether samples come from the same distribution as this dataset.

Figure 5.12: *Left:* Samples drawn from a PD-DBM model trained on MNIST using joint training only. *Center:* Samples drawn from a DBM model of the same size, trained using greedy layerwise pretraining followed by joint training. *Right:* Samples drawn from a DBM trained using joint training only

For the PD-DBM, we used $L = 1$, for a total of two hidden layers. We did not use greedy, layerwise pretraining– the entire model was learned jointly. Such joint learning without greedy pretraining has never been accomplished with similar deep models such as DBMs or DBNs.

The S3C samples and basis vectors are shown in Fig. 5.11. The samples do not resemble digits, suggesting that S3C has failed to model the data. However, inspection of the S3C filters shows that S3C has learned a good basis set for representing MNIST digits using digit templates, pen strokes, etc. It simply does not have the correct prior on these bases and as a result activates subsets of them that do not correspond to MNIST digits. The PD-DBM samples clearly resemble digits, as shown in Fig. 5.12. For comparison, Fig. 5.12 also shows samples from two DBMs. In all cases we display the expected value of the visible units given the hidden units.

The first DBM was trained by running the demo code that accompanies (Salakhutdinov and Hinton, 2009). We used the same number of *units* in each layer in order to make these models comparable (500 in the first layer and 1,000 in the second). This means that the PD-DBM has a slightly greater number of *parameters* than the DBM, since the first layer units of the PD-DBM have both mean and precision parameters while the first layer units of the DBM have only a bias parameter. Note that the DBM operates on a binarized version of MNIST while S3C and the PD-DBM regard MNIST as real-valued. Additionally, the DBM demo code uses

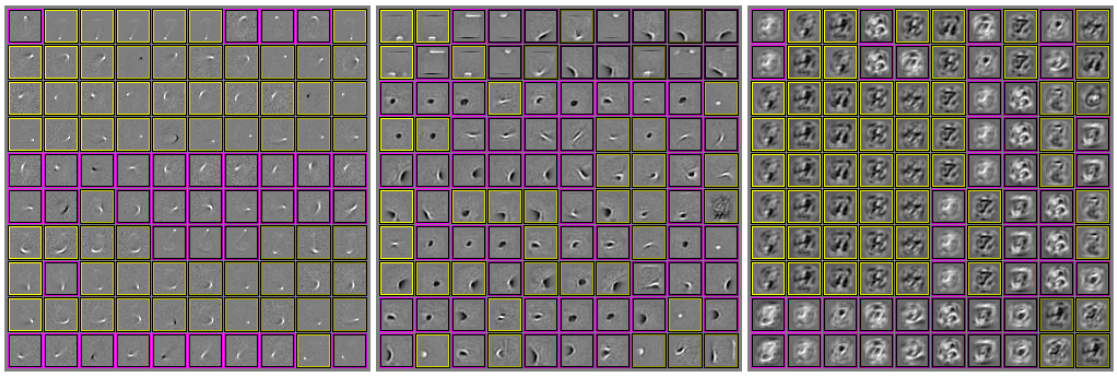


Figure 5.13: Each panel shows a visualization of the weights for a different model. Each row represents a different second layer hidden unit. We show ten units for each model corresponding to those with the largest weight vector norm. Within each row, we plot the weight vectors for the ten most strongly connected first layer units. Black corresponds to inhibition, white to excitation, and gray to zero weight. This figure is best viewed in color–units plotted with a yellow border have excitatory second layer weights while units plotted with a magenta border have inhibitory second layer weights. *Left:* PD-DBM model trained jointly. Note that each row contains many similar filters. This is how the second layer weights achieve invariance to some transformations such as image translation. This is one way that deep architectures are able to disentangle factors of variation. One can also see how the second layer helps implement the correct prior for the generative task. For example, the unit plotted in the first row excites filters used to draw 7s and inhibits filters used to draw 1s. Also, observe that the first layer filters are much more localized and contained fewer templates than those in Fig. 5.11 right. This suggests that joint training has a significant effect on the quality of the first layer weights; greedy pretraining would have attempted to solve the generative task with more templates due to S3C's independent prior. *Center:* DBM model with greedy pretraining followed by joint training. These weights show the same disentangling and invariance properties as those of the PD-DBM. Note that the filters have more black areas. This is because the RBM must use inhibitory weights to limit hidden unit activities, while S3C accomplishes the same purpose via the explaining-away effect. *Right:* DBM with joint training only. Note that many of the second layer weight vectors are duplicates of each other. This is because the second layer has a pathological tendency to focus on modeling a handful of first-layer units that learn interesting responses earliest in learning.

the MNIST labels during generative training while the PD-DBM and S3C were not trained with the benefit of the labels. The DBM demo code is hardcoded to pretrain the first layer for 100 epochs, the second layer for 200 epochs, and then jointly train the DBM for 300 epochs. We trained the PD-DBM starting from a random initialization for 350 epochs.

The second DBM was trained using two modifications from the demo code in order to train it in as similar a fashion to our PD-DBM model as possible: first, it was trained without access to labels, and second, it did not receive any pretraining. This model was trained for only 230 epochs because it had already converged to a bad local optimum by this time. This DBM is included to provide an example of how DBM training fails when greedy layerwise pretraining is not used. DBM training can fail in a variety of ways and no example should be considered representative of all of them.

To analyze the differences between these models, we display a visualization of the weights of the models that shows how the layers interact in Fig. 5.13.

5.9 Conclusion

We have motivated the use of the S3C model for unsupervised feature discovery. We have described a variational approximation scheme that makes it feasible to perform learning and inference in large-scale S3C and PD-DBM models. We have demonstrated that S3C is an effective feature discovery algorithm for both supervised and semi-supervised learning with small amounts of labeled data. This work addresses two scaling problems: the computation problem of scaling spike-and-slab sparse coding to the problem sizes used in object recognition, and the problem of scaling object recognition techniques to work with more classes. We demonstrate that this work can be extended to a deep architecture using a similar inference procedure, and show that the deeper architecture is better able to model the input distribution. Remarkably, this deep architecture does not require greedy training, unlike its DBM predecessor.

Acknowledgments

This work was funded by DARPA and NSERC. The authors would like to thank Pascal Vincent for helpful discussions. The computation done for this work was conducted in part on computers of RESMIQ, Clumeq and SharcNet. We would like to thank the developers of Theano (Bergstra et al., 2010) and pylearn2 (Warde-Farley et al., 2011).

6 Prologue to Second Article

6.1 Article Details

Multi-Prediction Deep Boltzmann Machines. Ian J. Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. In *Advances in Neural Information Processing Systems 26 (NIPS '13)*, pp. 646-654.

Personal Contribution. The basic ideas of multi-prediction training and the multi-inference trick were my own. The details were refined with guidance from Aaron Courville and Yoshua Bengio. Yoshua Bengio discovered the connection to generative stochastic networks. I implemented the training and inference procedures. Mehdi Mirza assisted with hyperparameter search and creation of the figures. I did most of the writing, with assistance from Aaron Courville and Yoshua Bengio.

6.2 Context

See chapter 2.6 for an overview of deep Boltzmann machines. Deep Boltzmann machines are probabilistic models that are among the best performers in terms of likelihood on datasets such as MNIST. DBMs have also proven their value for tasks such as classification. At the time that we began this work, pretraining via unsupervised learning was the dominant strategy for obtaining a well-regularized, high capacity classifier. Today, purely supervised networks outperform pretrained networks on most tasks, but on the MNIST dataset, the deep Boltzmann machine remains the basis for the state of the art classification method (Hinton et al., 2012).

Unfortunately, deep Boltzmann machines are difficult to train. They require multiple training stages, including a troublingly greedy layer-wise pretraining stage.

Practitioners seeking to apply deep Boltzmann machines must have a good theoretical and intuitive understanding of Markov chain Monte Carlo sampling in order to diagnose problems with the model that arise during hyperparameter search. Moreover, to obtain good classification results, inference in the generative model defined by the deep Boltzmann machine is insufficient. Instead, the deep Boltzmann machine must be used as a feature extractor for a specialized classifier. This dependence on a specialized classifier subtracts from the usefulness of the deep Boltzmann machine, since inference cannot be used to fill in arbitrary subsets of variables with high accuracy, as one would expect from a probabilistic model.

6.3 Contributions

The primary contribution of this paper is to introduce a new means of training the deep Boltzmann machine. This new method allows the deep Boltzmann machine to be trained in a single stage and results in a model that can classify well simply by using approximate inference, without needing to train dedicated classifiers for specific inference problems. This comes at the cost of the model not being able to generate good samples, but it makes the model useful for engineering tasks such as imputing missing values or classifying despite missing inputs. It also simplifies the process of training the model as a classifier.

6.4 Recent Developments

Since the development of this model, Uria et al. (2013) have developed a criterion similar to multi-prediction training. Rather than using a family of recurrent nets, it uses a family of NADE models.

7 Multi-Prediction Deep Boltzmann Machines

7.1 Introduction

A deep Boltzmann machine (DBM) (Salakhutdinov and Hinton, 2009) is a structured probabilistic model consisting of many layers of random variables, most of which are latent. DBMs are well established as generative models and as feature learning algorithms for classifiers.

Exact inference in a DBM is intractable. DBMs are usually used as feature learners, where the mean field expectations of the hidden units are used as input features to a separate classifier, such as an MLP or logistic regression. To some extent, this erodes the utility of the DBM as a probabilistic model–it can generate good samples, and provides good features for deterministic models, but it has not proven especially useful for solving inference problems such as predicting class labels given input features or completing missing input features.

Another drawback to the DBM is the complexity of training it. Typically it is trained in a greedy, layerwise fashion, by training a stack of RBMs. Training each RBM to model samples from the previous RBM's posterior distribution increases a variational lower bound on the likelihood of the DBM, and serves as a good way to initialize the joint model. Training the DBM from a random initialization generally does not work. It can be difficult for practitioners to tell whether a given lower layer RBM is a good starting point to build a larger model.

We propose a new way of training deep Boltzmann machines called *multi-prediction training* (MPT). MPT uses the mean field equations for the DBM to induce recurrent nets that are then trained to solve different inference tasks. The resulting trained MP-DBM model can be viewed either as a single probabilistic model trained with a variational criterion, or as a family of recurrent nets that solve related inference tasks.

We find empirically [1] that the MP-DBM does not require greedy layerwise training, so its performance on the final task can be monitored from the start. This makes it more suitable than the DBM for practitioners who do not have extensive experience with layerwise pretraining techniques or Markov chains. Anyone with experience minimizing non-convex functions should find MP-DBM training familiar and straightforward. Moreover, we show that inference in the MP-DBM is useful– the MP-DBM does not need an extra classifier built on top of its learned features to obtain good inference accuracy. We show that it outperforms the DBM at solving a variety of inference tasks including classification, classification with missing inputs, and prediction of randomly selected subsets of variables. Specifically, we use the MP-DBM to outperform the classification results reported for the standard DBM by Salakhutdinov and Hinton (2009) on both the MNIST handwritten character dataset (LeCun et al., 1998) and the NORB object recognition dataset (LeCun et al., 2004).

7.2 Review of deep Boltzmann machines

Typically, a DBM contains a set of D input features v that are called the *visible units* because they are always observed during both training and evaluation. When a class label is present the DBM typically represents it with a discrete-valued label unit y. The unit y is observed (on examples for which it is available) during training, but typically is not available at test time. The DBM also contains several latent variables that are never observed. These *hidden units* are usually organized into L layers $h^{(i)}$ of size $N_i, i \in \{1, \ldots, L\}$, with each unit in a layer conditionally independent of the other units in the layer given the neighboring layers.

The DBM is trained to maximize the mean field lower bound on $\log P(v, y)$. Unfortunately, training the entire model simultaneously does not seem to be feasible. See (Goodfellow et al., 2013) for an example of a DBM that has failed to learn using the naive training algorithm. Salakhutdinov and Hinton (2009) found that for their joint training procedure to work, the DBM must first be initialized by training one layer at a time. After each layer is trained as an RBM, the RBMs can

1. Code and hyperparameters available at http://www-etud.iro.umontreal.ca/~goodfeli/mp_dbm.html

be modified slightly, assembled into a DBM, and the DBM may be trained with PCD (Younes, 1998; Tieleman, 2008) and mean field. In order to achieve good classification results, an MLP designed specifically to predict y from v must be trained on top of the DBM model. Simply running mean field inference to predict y given v in the DBM model does not work nearly as well. See figure 7.1 for a graphical description of the training procedure used by Salakhutdinov and Hinton (2009).

The standard approach to training a DBM requires training $L+2$ different models using $L+2$ different objective functions, and does not yield a single model that excels at answering all queries. Our proposed approach requires training only one model with only one objective function, and the resulting model outperforms previous approaches at answering many kinds of queries (classification, classification with missing inputs, predicting arbitrary subsets of variables given the complementary subset).

7.3 Motivation

There are numerous reasons to prefer a single-model, single-training stage approach to deep Boltzmann machine learning:

1. **Optimization** As a greedy optimization procedure, layerwise training may be suboptimal. Small-scale experimental work has demonstrated this to be the case for deep belief networks (Arnold and Ollivier, 2012).

   In general, for layerwise training to be optimal, the training procedure for each layer must take into account the influence that the deeper layers will provide. The layerwise initialization procedure simply does not attempt to be optimal.

   The procedures used by Le Roux and Bengio (2008); Arnold and Ollivier (2012) make an optimistic assumption that the deeper layers will be able to implement the best possible prior on the current layer's hidden units. This approach is not immediately applicable to Boltzmann machines because it is specified in terms of learning the parameters of $P(h^{(i-1)}|h^{(i)})$ assuming that the parameters of the $P(h^{(i)})$ will be set optimally later. In a DBM the

symmetrical nature of the interactions between units means that these two distributions share parameters, so it is not possible to set the parameters of the one distribution, leave them fixed for the remainder of learning, and then set the parameters of the other distribution. Moreover, model architectures incorporating design features such as sparse connections, pooling, or factored multilinear interactions make it difficult to predict how best to structure one layer's hidden units in order for the next layer to make good use of them.

2. **Probabilistic modeling** Using multiple models and having some models specialized for exactly one task (like predicting y from v) loses some of the benefit of probabilistic modeling. If we have one model that excels at all tasks, we can use inference in this model to answer arbitrary queries, perform classification with missing inputs, and so on. The standard DBM training procedure gives this up by training a rich probabilistic model and then using it as just a feature extractor for an MLP.

3. **Simplicity** Needing to implement multiple models and training stages makes the cost of developing software with DBMs greater, and makes using them more cumbersome. Beyond the software engineering considerations, it can be difficult to monitor training and tell what kind of results during layerwise RBM pretraining will correspond to good DBM classification accuracy later. Our joint training procedure allows the user to monitor the model's ability of interest (usually ability to classify y given v) from the very start of training.

7.4 Methods

We now described the new methods proposed in this paper, and some pre-existing methods that we compare against.

7.4.1 Multi-prediction Training

Our proposed approach is to directly train the DBM to be good at solving all possible variational inference problems. We call this *multi-prediction training* because the procedure involves training the model to predict any subset of variables given the complement of that subset of variables.

Let $\mathcal{O}$ be a vector containing all variables that are observed during training. For a purely unsupervised learning task, $\mathcal{O}$ is just v itself. In the supervised setting, $\mathcal{O} = [v, y]^\top$. Note that y won't be observed at test time, only training time. Let $\mathcal{D}$ be the training set, i.e. a collection of values of $\mathcal{O}$. Let S be a sequence of subsets of the possible indices of $\mathcal{O}$. Let Q_i be the variational (e.g., mean-field) approximation to the joint of O_{S_i} and h given $\mathcal{O}_{-S_i}$.

$$Q_i(\mathcal{O}_{S_i}, h) = \text{argmin}_Q D_{KL}\left(Q(\mathcal{O}_{S_i}, h) \| P(\mathcal{O}_{S_i}, h \mid \mathcal{O}_{-S_i})\right).$$

In all of the experiments presented in this paper, Q is constrained to be factorial, though one could design model families for which it makes sense to use richer structure in Q. Note that there is not an explicit formula for Q; Q must be computed by an iterative optimization process. In order to accomplish this minimization, we run the mean field fixed point equations to convergence. Because each fixed point update uses the output of a previous fixed point update as input, this optimization procedure can be viewed as a recurrent neural network. (To simplify implementation, we don't explicitly test for convergence, but run the recurrent net for a pre-specified number of iterations that is chosen to be high enough that the net usually converges)

We train the MP-DBM by using minibatch stochastic gradient descent on the *multi-prediction* (MP) objective function

$$J(\mathcal{D}, \theta) = -\sum_{\mathcal{O} \in \mathcal{D}} \sum_i \log Q_i(\mathcal{O}_{S_i})$$

In other words, the criterion for a single example $\mathcal{O}$ is a sum of several terms, with term i measuring the model's ability to predict (through a variational approximation) a subset of the variables in the training set, $\mathcal{O}_{S_i}$, given the remainder of the observed variables, $\mathcal{O}_{-S_i}$.

During SGD training, we sample minibatches of values of $\mathcal{O}$ and S_i. Sampling $\mathcal{O}$ just means drawing an example from the training set. Sampling an S_i uniformly simply requires sampling one bit (1 with probability 0.5) for each variable, to determine whether that variable should be an input to the inference procedure or a prediction target. To compute the gradient, we simply backprop the error derivatives of J through the recurrent net defining Q.

See Fig. 7.2 for a graphical description of this training procedure, and Fig. 7.3

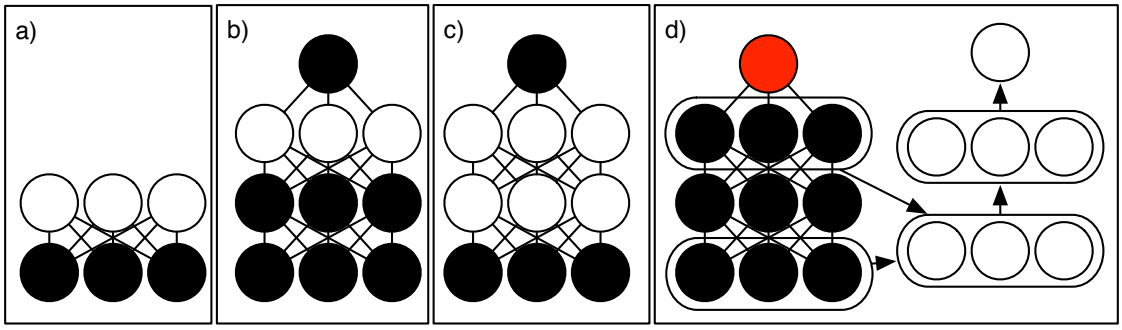


Figure 7.1: The training procedure used by Salakhutdinov and Hinton (2009) on MNIST. a) Train an RBM to maximize $\log P(v)$ using CD. b) Train another RBM to maximize $\log P(h^{(1)}, y)$ where $h^{(1)}$ is drawn from the first RBM's posterior. c) Stitch the two RBMs into one DBM. Train the DBM to maximize $\log P(v, y)$. d) Delete y from the model (don't marginalize it out, just remove the layer from the model). Make an MLP with inputs v and the mean field expectations of $h^{(1)}$ and $h^{(2)}$. Fix the DBM parameters. Initialize the MLP parameters based on the DBM parameters. Train the MLP parameters to predict y.

for an example of the inference procedure run on MNIST digits.

This training procedure is similar to one introduced by Brakel et al. (2013) for time-series models. The primary difference is that we use $\log Q$ as the loss function, while Brakel et al. (2013) apply hard-coded loss functions such as mean squared error to the predictions of the missing values.

7.4.2 The Multi-Inference Trick

Mean field inference can be expensive due to needing to run the fixed point equations several times in order to reach convergence. In order to reduce this computational expense, it is possible to train using fewer mean field iterations than required to reach convergence. In this case, we are no longer necessarily minimizing J as written, but rather doing partial training of a large number of fixed-iteration recurrent nets that solve related problems.

We can approximately take the geometric mean over all predicted distributions Q (for different subsets S_i) and renormalize in order to combine the predictions of all of these recurrent nets. This way, imperfections in the training procedure are averaged out, and we are able to solve inference tasks even if the corresponding recurrent net was never sampled during MP training.

In order to approximate this average efficiently, we simply take the geometric mean at each step of inference, instead of attempting to take the correct geometric

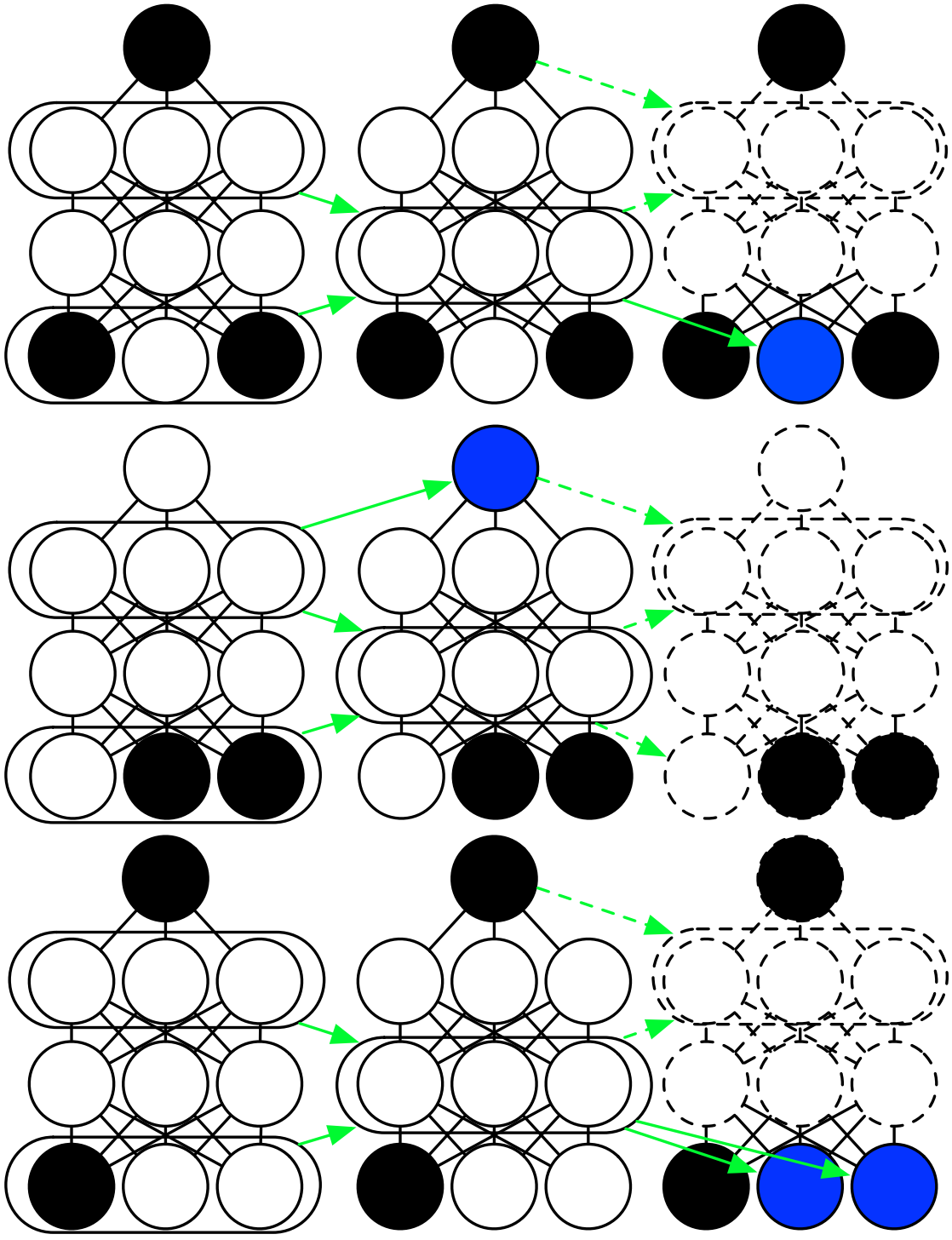


Figure 7.2: *Multi-prediction training*: This diagram shows the neural nets instantiated to do multi-prediction training on one minibatch of data. The three rows show three different examples. Black circles represent variables the net is allowed to oberve. Blue circles represent prediction targets. Green arrows represent computational dependencies. Each column shows a single mean field fixed point update. Each mean field iteration consists of two fixed point updates. Here we show only one iteration to save space, but in a real application MP training should be run with 5-15 iterations.

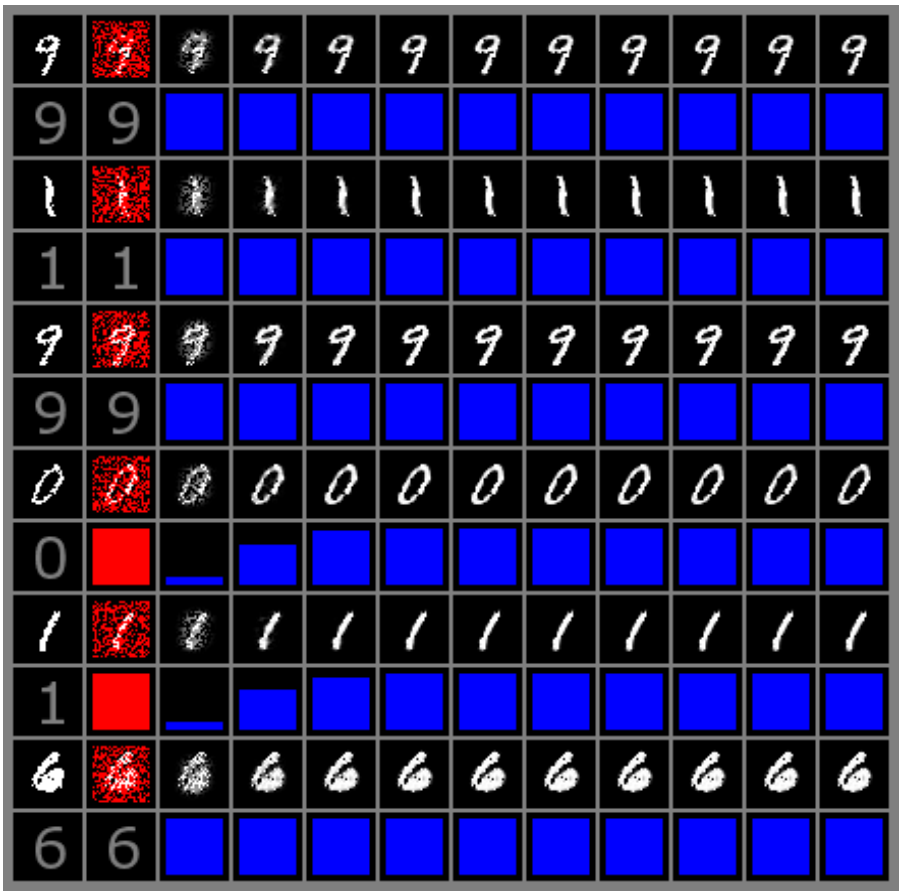


Figure 7.3: *Mean field inference applied to MNIST digits.* Within each pair of rows, the upper row shows pixels and the lower row shows class labels. The first column shows a complete, labeled example. The second column shows information to be masked out, using red pixels to indicate information that is removed. The subsequent columns show steps of mean field. The images show the pixels being filled back in by the mean field inference, and the blue bars show the probability of the correct class under the mean field posterior.

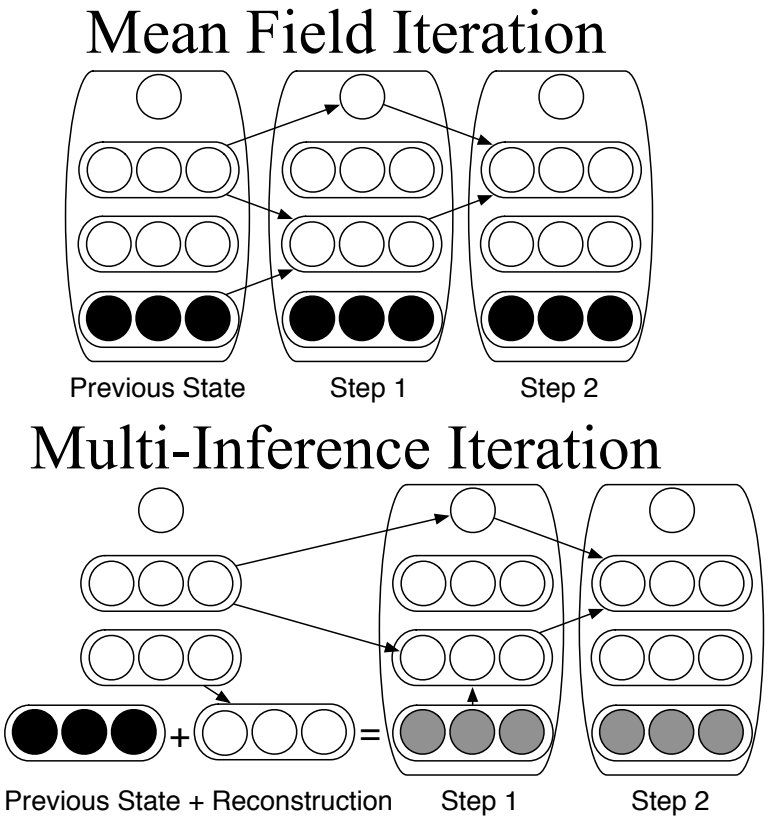


Figure 7.4: *Multi-inference trick:* When estimating y given v, a mean field iteration consists of first applying a mean field update to $h^{(1)}$ and y, then applying one to $h^{(2)}$. To use the multi-inference trick, start the iteration by computing r as the mean field update v would receive if it were not observed. Then use $0.5(r + v)$ in place of v and run a regular mean field iteration.

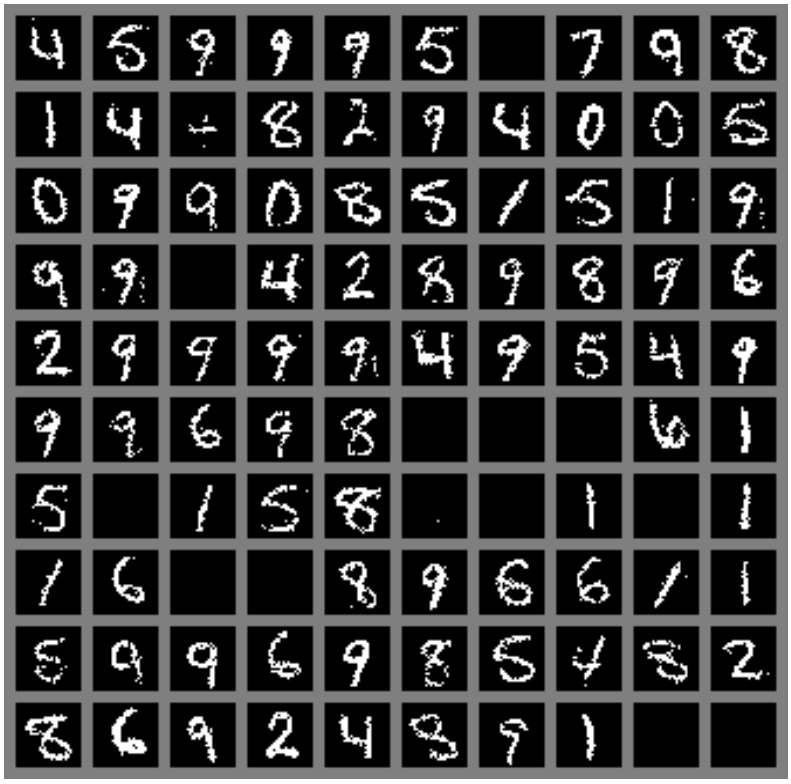


Figure 7.5: Samples generated by alternately sampling S_i uniformly and sampling $\mathcal{O}_{-S_i}$ from $Q_i(\mathcal{O}_{-S_i})$.

mean of the entire inference process. See Fig. 7.4 for a graphical depiction of the method. This is the same type of approximation used to take the average over several MLP predictions when using dropout (Hinton et al., 2012). Here, the averaging rule is slightly different. In dropout, the different MLPs we average over either include or exclude each variable. To take the geometric mean over a unit h_j that receives input from v_i, we average together the contribution $v_i W_{ij}$ from the model that contains v_i and the contribution 0 from the model that does not. The final contribution from v_i is $0.5 v_i W_{ij}$ so the dropout model averaging rule is to run an MLP with the weights divided by 2.

For the multi-inference trick, each recurrent net we average over solves a different inference problem. In half of the problems, v_i is observed, and contributes $v_i W_{ij}$ to h_j's total input. In the other half of the problems, v_i is inferred. In contrast to dropout, v_i is never completely absent. If we represent the mean field estimate of v_i with r_i, then in this case that unit contributes $r_i W_{ij}$ to h_j's total input. To run multi-inference, we thus replace references to v with $0.5(v + r)$, where r is updated at each mean field iteration. The main benefit to this approach is that it gives a good way to incorporate information from many recurrent nets trained in slightly different ways. If the recurrent net corresponding to the desired inference task is somewhat suboptimal due to not having been sampled enough during training, its defects can be oftened be remedied by averaging its predictions with those of other similar recurrent nets. The multi-inference trick can also be understood as including an input denoising step built into the inference. In practice, multi-inference mostly seems to be beneficial if the network was trained without letting mean field run to convergence. When the model was trained with converged mean field, each recurrent net is just solving an optimization problem in a graphical model, and it doesn't matter whether every recurrent net has been individually trained. The multi-inference trick is mostly useful as a cheap alternative when getting the absolute best possible test set accuracy is not as important as fast training and evaluation.

7.4.3 Justification and advantages

In the case where we run the recurrent net for predicting Q to convergence, the multi-prediction training algorithm follows the gradient of the objective function

J. This can be viewed as a mean field approximation to the generalized pseudolikelihood.

While both pseudolikelihood and likelihood are asymptotically consistent estimators, their behavior in the limited data case is different. Maximum likelihood should be better if the overall goal is to draw realistic samples from the model, but generalized pseudolikelihood can often be better for training a model to answer queries conditioning on sets similar to the S_i used during training.

Note that our variational approximation is not quite the same as the way variational approximations are usually applied. We use variational inference to ensure that the distributions we shape using backprop are as close as possible to the true conditionals. This is different from the usual approach to variational learning, where Q is used to define a lower bound on the log likelihood and variational inference is used to make the bound as tight as possible.

In the case where the recurrent net is not trained to convergence, there is an alternate way to justify MP training. Rather than doing variational learning on a single probabilistic model, the MP procedure trains a family of recurrent nets to solve related prediction problems by running for some fixed number of iterations. Each recurrent net is trained only on a subset of the data (and most recurrent nets are never trained at all, but only work because they share parameters with the others). In this case, the multi-inference trick allows us to justify MP training as approximately training an ensemble of recurrent nets using bagging.

Stoyanov et al. (2011) have observed that a training strategy similar to MPT (but lacking the multi-inference trick) is useful because it trains the model to work well with the inference approximations it will be evaluated with at test time. We find these properties to be useful as well. The choice of this type of variational learning combined with the underlying generalized pseudolikelihood objective makes an MP-DBM very well suited for solving approximate inference problems but not very well suited for sampling.

Our primary design consideration when developing multi-prediction training was ensuring that the learning rule was state-free. PCD training uses persistent Markov chains to estimate the gradient. These Markov chains are used to approximately sample from the model, and only sample from approximately the right distribution if the model parameters evolve slowly. The MP training rule does not make any reference to earlier training steps, and can be computed with no burn in.

This means that the accuracy of the MP gradient is not dependent on properties of the training algorithm such as the learning rate which can easily break PCD for many choices of the hyperparameters.

Another benefit of MP is that it is easy to obtain an unbiased estimate of the MP objective from a small number of samples of v and i. This is in contrast to the log likelihood, which requires estimating the log partition function. The best known method for doing so is AIS, which is relatively expensive (Neal, 2001). Cheap estimates of the objective function enable early stopping based on the MP-objective (though we generally use early stopping based on classification accuracy) and optimization based on line searches (though we do not explore that possibility in this paper).

7.4.4 Regularization

In order to obtain good generalization performance, Salakhutdinov and Hinton (2009) regularized both the weights and the activations of the network.

Salakhutdinov and Hinton (2009) regularize the weights using an L2 penalty. We find that for joint training, it is critically important to not do this (on the MNIST dataset, we were not able to find any MP-DBM hyperparameter configuration involving weight decay that performs as well as layerwise DBMs, but without weight decay MP-DBMs outperform DBMs). When the second layer weights are not trained well enough for them to be useful for modeling the data, the weight decay term will drive them to become very small, and they will never have an opportunity to recover. It is much better to use constraints on the norms of the columns of the weight vectors as done by Srebro and Shraibman (2005).

Salakhutdinov and Hinton (2009) regularize the activities of the hidden units with a somewhat complicated sparsity penalty. See `http://www.mit.edu/~rsalakhu/DBM.html` for details. We use $\max(|\mathbb{E}_{h \sim Q(h)}[h] - t| - \lambda, 0)$ and backpropagate this through the entire inference graph. t and λ are hyperparameters.

7.4.5 Related work: centering

Montavon and Müller (2012) showed that an alternative, "centered" representation of the DBM results in successful generative training without a greedy layerwise

pretraining step. However, centered DBMs have never been shown to have good classification performance. We therefore evaluate the classification performance of centering in this work. We consider two methods of variational PCD training. In one, we use Rao-Blackwellization (Blackwell, 1947; Kolmogorov, 1953; Rao, 1973) of the negative phase particles to reduce the variance of the negative phase. In the other variant ("centering+"), we use a special negative phase that Salakhutdinov and Hinton (2009) found useful. This negative phase uses a small amount of mean field, which reduces the variance further but introduces some bias, and has better symmetry with the positive phase. See `http://www.mit.edu/~rsalakhu/DBM.html` for details.

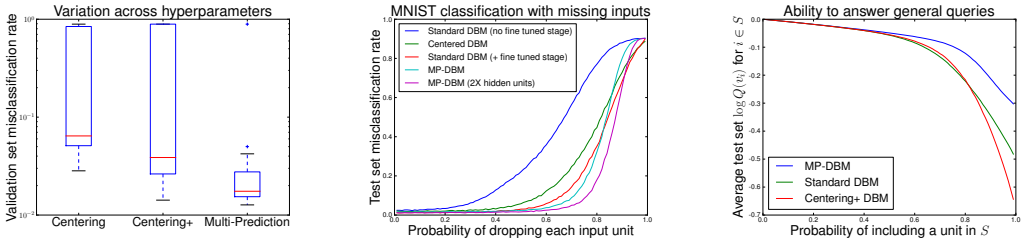
7.4.6 Sampling, and a connection to GSNs

The focus of this paper is solving inference problems, not generating samples, so we do not investigate the sampling properties of MP-DBMs extensively. However, it is interesting to note that an MP-DBM can be viewed as a collection of dependency networks (Heckerman et al., 2000) with shared parameters. Dependency networks are a special case of generative stochastic networks or GSNs (Bengio et al. (2013), section 3.4). This means that the MP-DBM is associated with a distribution arising out of the Markov chain in which at each step one samples an S_i uniformly and then samples $\mathcal{O}$ from $Q_i(\mathcal{O})$. Example samples are shown in figure 7.5. Furthermore, it means that if MPT is a consistent estimator of the conditional distributions, then MPT is a consistent estimator of the probability distribution defined by the stationary distribution of this Markov chain. Samples drawn by Gibbs sampling in the DBM model do not look as good (probably because the variational approximation is too damaging). This suggests that the perspective of the MP-DBM as a GSN merits further investigation.

7.5 Experiments

7.5.1 MNIST experiments

In order to compare MP training and centering to standard DBM performance, we cross-validated each of the new methods by running 25 training experiments for



(a) Cross-validation **(b)** Missing inputs **(c)** General queries

Figure 7.6: Quantitative results on MNIST: (a) During cross-validation, MP training performs well for most hyperparameters, while both centering and centering with the special negative phase do not perform as well and only perform well for a few hyperparameter values. Note that the vertical axis is on a log scale. (b) Generic inference tasks: When classifying with missing inputs, the MP-DBM outperforms the other DBMs for most amounts of missing inputs. (c) When using approximate inference to resolve general queries, the standard DBM, centered DBM, and MP-DBM all perform about the same when asked to predict a small number of variables. For larger queries, the MP-DBM performs the best.

each of three conditions: centered DBMs, centered DBMs with the special negative phase ("Centering+"), and MP training.

All three conditions visited exactly the same set of 25 hyperparameter values for the momentum schedule, sparsity regularization hyperparameters, weight and bias initialization hyperparameters, weight norm constraint values, and number of mean field iterations. The centered DBMs also required one additional hyperparameter, the number of Gibbs steps to run for variational PCD. We used different values of the learning rate for the different conditions, because the different conditions require different ranges of learning rate to perform well. We use the same size of model, minibatch and negative chain collection as Salakhutdinov and Hinton (2009), with 500 hidden units in the first layer, 1,000 hidden units in the second, 100 examples per minibatch, and 100 negative chains. The energy function for this model is

$$E(v,h,y) = -v^\top W^{(1)} h^{(1)} - h^{(1)T} W^{(2)} h^{(2)} - h^{(2)T} W^{(3)} y$$
$$-v^\top b^{(0)} - h^{(1)T} b^{(1)} - h^{(2)T} b^{(2)} - y^\top b^{(3)}.$$

See Fig. 7.6a for the results of cross-validation. On the validation set, MP training consistently performs better and is much less sensitive to hyperparameters than the other methods. This is likely because the state-free nature of the learning rule makes it perform better with settings of the learning rate and momentum schedule that result in the model distribution changing too fast for a method based on Markov chains to keep up.

When we add an MLP classifier (as shown in Fig. 7.1d), the best "Center-

ing+" DBM obtains a classification error of 1.22% on the test set. The best MP-DBM obtains a classification error of 0.88%. This compares to 0.95% obtained by Salakhutdinov and Hinton (2009).

If instead of adding an MLP to the model, we simply train a larger MP-DBM with twice as many hidden units in each layer, and apply the multi-inference trick, we obtain a classification error rate of 0.91%. In other words, we are able to classify nearly as well using a single large DBM and a generic inference procedure, rather than using a DBM followed by an entirely separate MLP model specialized for classification.

The original DBM was motivated primarily as a generative model with a high AIS score and as a means of initializing a classifier. Here we explore some more uses of the DBM as a generative model. Fig. 7.6b shows an evaluation of various DBM's ability to classify with missing inputs. Fig. 7.6c shows an evaluation of their ability to resolve queries about random subsets of variables. In both cases we find that the MP-DBM performs the best for most amounts of missing inputs.

7.5.2 NORB experiments

NORB consists of 96×96 binocular greyscale images of objects from five different categories, under a variety of pose and lighting conditions. Salakhutdinov and Hinton (2009) preprocessed the images by resampling them with bigger pixels near the border of the image, yielding an input vector of size 8,976. We used this preprocessing as well. Salakhutdinov and Hinton (2009) then trained an RBM with 4,000 binary hidden units and Gaussian visible units to preprocess the data into an all-binary representation, and trained a DBM with two hidden layers of 4,000 units each on this representation. Since the goal of this work is to provide a single unified model and training algorithm, we do not train a separate Gaussian RBM. Instead we train a single MP-DBM with Gaussian visible units and three hidden layers of 4,000 units each. The energy function for this model is

$$E(v,h,y) = -(v-\mu)^\top \beta W^{(1)} h^{(1)} - h^{(1)T} W^{(2)} h^{(2)} - h^{(2)T} W^{(3)} h^{(3)} - h^{(3)T} W^{(4)} y \\ + \frac{1}{2}(v-\mu)^\top \beta (v-\mu) - h^{(1)T} b^{(1)} - h^{(2)T} b^{(2)} - h^{(3)T} b^{(3)} - y^\top b^{(4)}.$$

where μ is a learned vector of visible unit means and β is a learned diagonal precision matrix.

By adding an MLP on top of the MP-DBM, following the same architecture as

Salakhutdinov and Hinton (2009), we were able to obtain a test set error of 10.6%. This is a slight improvement over the standard DBM's 10.8%.

On MNIST we were able to outperform the DBM without using the MLP classifier because we were able to train a larger MP-DBM. On NORB, the model size used by Salakhutdinov and Hinton (2009) is already as large as we are able to fit on most of our graphics cards, so we were not able to do the same for this dataset. It is possible to do better on NORB using convolution or synthetic transformations of the training data. We did not evaluate the effect of these techniques on the MP-DBM because our present goal is not to obtain state-of-the-art object recognition performance but only to verify that our joint training procedure works as well as the layerwise training procedure for DBMs. There is no public demo code available for the standard DBM on this dataset, and we were not able to reproduce the standard DBM results (layerwise DBM training requires significant experience and intuition). We therefore can't compare the MP-DBM to the original DBM in terms of answering general queries or classification with missing inputs on this dataset.

7.6 Conclusion

This paper has demonstrated that MP training and the multi-inference trick provide a means of training a single model, with a single stage of training, that matches the performance of standard DBMs but still works as a general probabilistic model, capable of handling missing inputs and answering general queries. We have verified that MP training outperforms the standard training procedure at classification on the MNIST and NORB datasets where the original DBM was first applied. We have shown that MP training works well with binary, Gaussian, and softmax units, as well as architectures with either two or three hidden layers. In future work, we hope to apply the MP-DBM to more practical applications, and explore techniques, such as dropout, that could improve its performance further.

Acknowledgments

We would like to thank the developers of Theano (Bergstra et al., 2010; Bastien et al., 2012), Pylearn2 (Goodfellow et al., 2013). We would also like to thank NSERC, Compute Canada, and Calcul Québec for providing computational resources.

8 Prologue to Third Article

8.1 Article Details

Maxout Networks. Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. *Proceedings of the 30th International Conference on Machine Learning (ICML '13)*, pp. 1319-1327.

Personal Contribution. The idea for the maxout activation function and using it in combination with dropout was my own. All of my co-authors suggested experiments for understanding the effectiveness of this combination. David Warde-Farley, Mehdi Mirza, and I jointly ran these more scientific experiments. I wrote the code for the activation function itself. David Warde-Farley and I jointly wrote the code to accelerate convolution in Theano along with help from other Theano developers. I found the best hyperparameters for both versions of the MNIST dataset. I found hyperparameters for non-augmented CIFAR-10 that improved upon the state of the art, and then David Warde-Farley improved upon mine further and obtained the state of the art on augmented CIFAR-10 as well. I found the best hyperparameters for CIFAR-100. Mehdi Mirza implemented the infrastructure code needed to handle large datasets in order to work with SVHN. He also implemented the preprocessing code for SVHN and found the best hyperparameters for this dataset. David Warde-Farley, Mehdi Mirza, and I each made figures for the paper. All authors contributed to the writing of the paper. I supplied the basic idea for the universal approximation proof, and Aaron Courville wrote the formal proof sketch.

8.2 Context

At the time that we wrote this article, the dropout (Hinton et al., 2012) algorithm had been recently introduced. In the excitement following the introduction

of dropout, we were interested in finding a model family that would synergize well with dropout. Dropout also presented the opportunity to train models that had previously suffered from overfitting. The maxout activation function, with its cross-channel pooling, is one such model.

8.3 Contributions

The main contribution of this paper is to introduce a new activation function for feedforward neural networks that significantly improves their performance. We also perform detailed experiments to explain these performance gains.

8.4 Recent Developments

Since its publication, this work has been frequently cited and expanded upon. Wang and Manning (2013) developed a fast analytical approximation to dropout and included formulas for using their method with maxout layers. Smirnov (2013) used maxout for whale call detection. Xie et al. (2013) used maxout to rank highly in a machine learning contest (Goodfellow et al., 2013). Miao et al. (2013) used maxout for low resource speech recognition. Cai et al. (2013) also applied maxout to speech recognition, and found that maxout works well even without dropout training, so long as the dataset is large and the number of pieces per maxout unit is low. Alsharif and Pineau (2013) used maxout networks to implement the conditional probability distributions in a larger graphical model used to transcribe text from photos. Goodfellow et al. (2014) used maxout networks directly to transcribe house numbers from photos, as presented in chapter 11.

9 Maxout Networks

9.1 Introduction

Dropout (Hinton et al., 2012) provides an inexpensive and simple means of both training a large ensemble of models that share parameters and approximately averaging together these models' predictions. Dropout applied to multilayer perceptrons and deep convolutional networks has improved the state of the art on tasks ranging from audio classification to very large scale object recognition (Hinton et al., 2012; Krizhevsky et al., 2012). While dropout is known to work well in practice, it has not previously been demonstrated to actually perform model averaging for deep architectures [1] . Dropout is generally viewed as an indiscriminately applicable tool that reliably yields a modest improvement in performance when applied to almost any model.

We argue that rather than using dropout as a slight performance enhancement applied to arbitrary models, the best performance may be obtained by directly designing a model that enhances dropout's abilities as a model averaging technique. Training using dropout differs significantly from previous approaches such as ordinary stochastic gradient descent. Dropout is most effective when taking relatively large steps in parameter space. In this regime, each update can be seen as making a significant update to a different model on a different subset of the training set. The ideal operating regime for dropout is when the overall training procedure resembles training an ensemble with bagging under parameter sharing constraints. This differs radically from the ideal stochastic gradient operating regime in which a single model makes steady progress via small steps. Another consideration is that dropout model averaging is only an approximation when applied to deep models. Explicitly designing models to minimize this approximation error may thus enhance dropout's performance as well.

1. Between submission and publication of this paper, we have learned that Srivastava (2013) performed experiments on this subject similar to ours.

We propose a simple model that we call *maxout* that has beneficial characteristics both for optimization and model averaging with dropout. We use this model in conjunction with dropout to set the state of the art on four benchmark datasets [2] .

9.2 Review of dropout

Dropout is a technique that can be applied to deterministic feedforward architectures that predict an output y given input vector v. These architectures contain a series of hidden layers $\mathbf{h} = \{h^{(1)}, \ldots, h^{(L)}\}$. Dropout trains an ensemble of models consisting of the set of all models that contain a subset of the variables in both v and $\mathbf{h}$. The same set of parameters θ is used to parameterize a family of distributions $p(y \mid v; \theta, \mu)$ where $\mu \in \mathcal{M}$ is a binary mask determining which variables to include in the model. On each presentation of a training example, we train a different sub-model by following the gradient of $\log p(y \mid v; \theta, \mu)$ for a different randomly sampled μ. For many parameterizations of p (such as most multilayer perceptrons) the instantiation of different sub-models $p(y \mid v; \theta, \mu)$ can be obtained by elementwise multiplication of v and $\mathbf{h}$ with the mask μ. Dropout training is similar to bagging (Breiman, 1994), where many different models are trained on different subsets of the data. Dropout training differs from bagging in that each model is trained for only one step and all of the models share parameters. For this training procedure to behave as if it is training an ensemble rather than a single model, each update must have a large effect, so that it makes the sub-model induced by that μ fit the current input v well.

The functional form becomes important when it comes time for the ensemble to make a prediction by averaging together all the sub-models' predictions. Most prior work on bagging averages with the arithmetic mean, but it is not obvious how to do so with the exponentially many models trained by dropout. Fortunately, some model families yield an inexpensive *geometric* mean. When $p(y \mid v; \theta) = \text{softmax}(v^\top W + b)$, the predictive distribution defined by renormalizing the geometric mean of $p(y \mid v; \theta, \mu)$ over $\mathcal{M}$ is simply given by $\text{softmax}(v^\top W/2 + b)$. In other words, the average prediction of exponentially many sub-models can be

2. Code and hyperparameters available at http://www-etud.iro.umontreal.ca/~goodfeli/maxout.html

computed simply by running the full model with the weights divided by 2. This result holds exactly in the case of a single layer softmax model. Previous work on dropout applies the same scheme in deeper architectures, such as multilayer perceptrons, where the $W/2$ method is only an approximation to the geometric mean. The approximation has not been characterized mathematically, but performs well in practice.

9.3 Description of maxout

The maxout model is simply a feed-forward achitecture, such as a multilayer perceptron or deep convolutional neural network, that uses a new type of activation function: the maxout unit. Given an input $x \in \mathbb{R}^d$ (x may be v, or may be a hidden layer's state), a maxout hidden layer implements the function

$$h_i(x) = \max_{j \in [1,k]} z_{ij}$$

where $z_{ij} = x^\top W_{\dots ij} + b_{ij}$, and $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are learned parameters. In a convolutional network, a maxout feature map can be constructed by taking the maximum across k affine feature maps (i.e., pool across channels, in addition spatial locations). When training with dropout, we perform the elementwise multiplication with the dropout mask immediately prior to the multiplication by the weights in all cases–we do not drop inputs to the max operator. A single maxout unit can be interpreted as making a piecewise linear approximation to an arbitrary convex function. Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit. See Fig. 9.1 for a graphical depiction of how this works.

Maxout abandons many of the mainstays of traditional activation function design. The representation it produces is not sparse at all (see Fig. 9.2), though the gradient is highly sparse and dropout will artificially sparsify the effective representation during training. While maxout may learn to saturate on one side or the other this is a measure zero event (so it is almost never bounded from above). While a significant proportion of parameter space corresponds to the function being bounded from below, maxout is not constrained to learn to be bounded at all. Maxout is locally linear almost everywhere, while many popular activation functions

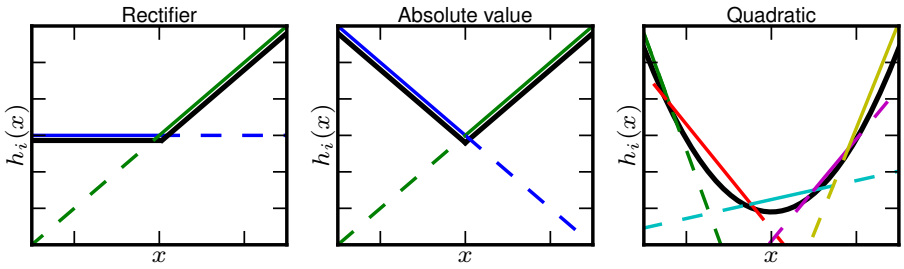


Figure 9.1: Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.

have signficant curvature. Given all of these departures from standard practice, it may seem surprising that maxout activation functions work at all, but we find that they are very robust and easy to train with dropout, and achieve excellent performance.

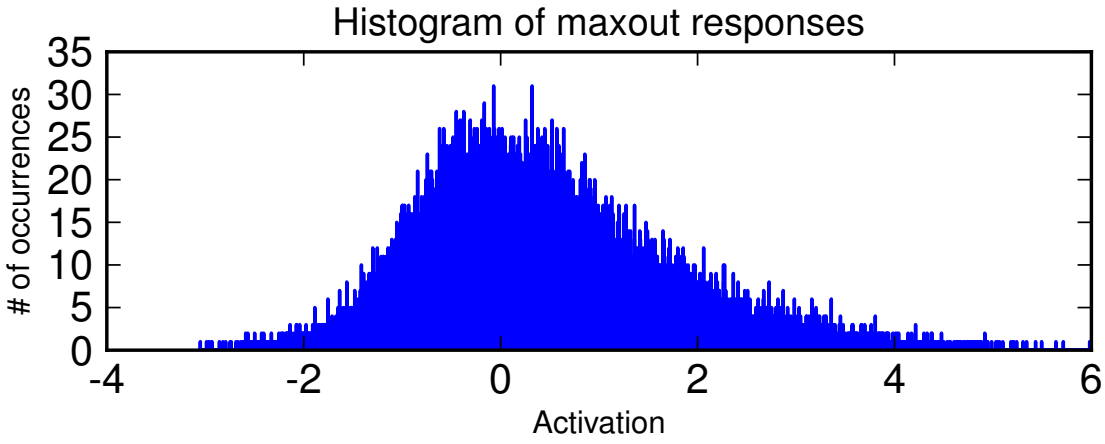


Figure 9.2: The activations of maxout units are not sparse.

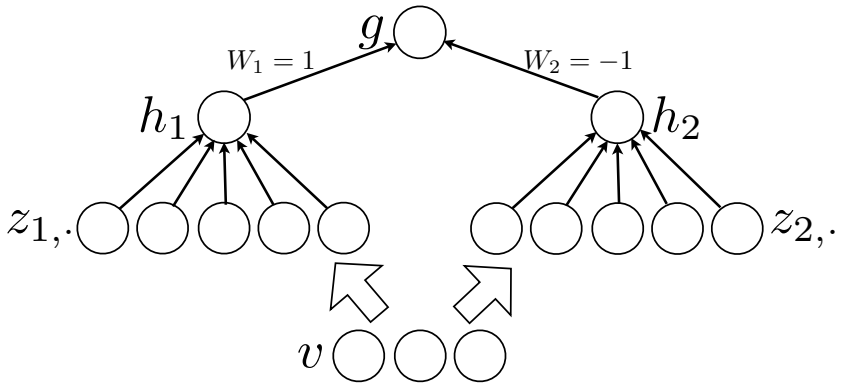


Figure 9.3: An MLP containing two maxout units can arbitrarily approximate any continuous function. The weights in the final layer can set g to be the difference of h_1 and h_2. If z_1 and z_2 are allowed to have arbitrarily high cardinality, h_1 and h_2 can approximate any convex function. g can thus approximate any continuous function due to being a difference of approximations of arbitrary convex functions.

9.4 Maxout is a universal approximator

A standard MLP with enough hidden units is a universal approximator. Similarly, maxout networks are universal approximators. Provided that each individual maxout unit may have arbitrarily many affine components, we show that a maxout model with just two hidden units can approximate, arbitrarily well, any continuous function of $v \in \mathbb{R}^n$. A diagram illustrating the basic idea of the proof is presented in Fig. 9.3.

Consider the continuous piecewise linear (PWL) function $g(v)$ consisting of k locally affine regions on $\mathbb{R}^n$.

Proposition 9.4.1. *(From Theorem 2.1 in Wang (2004)) For any positive integers m and n, there exist two groups of $n+1$-dimensional real-valued parameter vectors $[W_{1j}, b_{1j}], j \in [1, k]$ and $[W_{2j}, b_{2j}], j \in [1, k]$ such that:*

$$g(v) = h_1(v) - h_2(v) \tag{9.1}$$

That is, any continuous PWL function can be expressed as a difference of two convex PWL functions. The proof is given in Wang (2004).

Proposition 9.4.2. *From the* Stone-Weierstrass approximation theorem*, let C be a compact domain $C \subset \mathbb{R}^n$, $f : C \to \mathbb{R}$ be a continuous function, and $\epsilon > 0$ be any positive real number. Then there exists a continuous PWL function g, (depending*

upon ϵ), such that for all $v \in C$, $|f(v) - g(v)| < \epsilon$.

Theorem 9.4.3. Universal approximator theorem. *Any continuous function f can be approximated arbitrarily well on a compact domain $C \subset \mathbb{R}^n$ by a maxout network with two maxout hidden units.*

Sketch of Proof By Proposition 9.4.2, any continuous function can be approximated arbitrarily well (up to ϵ), by a piecewise linear function. We now note that the representation of piecewise linear functions given in Proposition 9.4.1 exactly matches a maxout network with two hidden units $h_1(v)$ and $h_2(v)$, with sufficiently large k to achieve the desired degree of approximation ϵ. Combining these, we conclude that a two hidden unit maxout network can approximate any continuous function $f(v)$ arbitrarily well on the compact domain C. In general as $\epsilon \to 0$, we have $k \to \infty$.

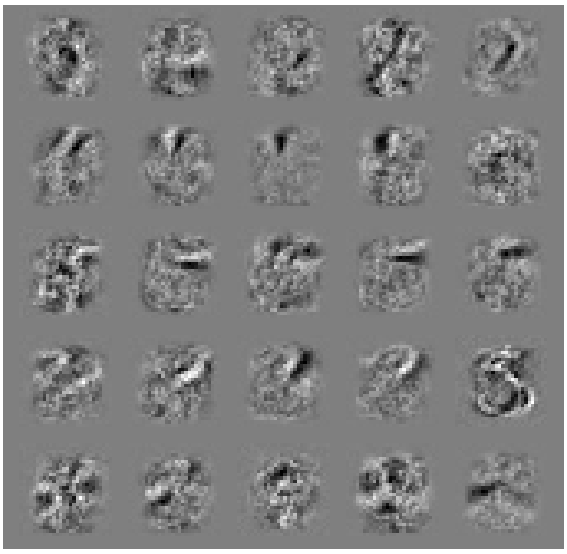


Figure 9.4: Example filters learned by a maxout MLP trained with dropout on MNIST. Each row contains the filters whose responses are pooled to form a maxout unit.

9.5 Benchmark results

We evaluated the maxout model on four benchmark datasets and set the state of the art on all of them.

Table 9.1: Test set misclassification rates for the best methods on the permutation invariant MNIST dataset. Only methods that are regularized by modeling the input distribution outperform the maxout MLP.

Method	Test error
Rectifier MLP + dropout (Srivastava, 2013)	1.05%
DBM (Salakhutdinov and Hinton, 2009)	0.95%
Maxout MLP + dropout	**0.94%**
MP-DBM (Goodfellow et al., 2013)	0.88%
Deep Convex Network (Yu and Deng, 2011)	0.83%
Manifold Tangent Classifier (Rifai et al., 2011)	0.81%
DBM + dropout (Hinton et al., 2012)	0.79%

9.5.1 MNIST

The MNIST (LeCun et al., 1998) dataset consists of 28 $\times$ 28 pixel greyscale images of handwritten digits 0-9, with 60,000 training and 10,000 test examples. For the *permutation invariant* version of the MNIST task, only methods unaware of the 2D structure of the data are permitted. For this task, we trained a model consisting of two densely connected maxout layers followed by a softmax layer. We regularized the model with dropout and by imposing a constraint on the norm of each weight vector, as in (Srebro and Shraibman, 2005). Apart from the maxout units, this is the same architecture used by Hinton et al. (2012). We selected the hyperparameters by minimizing the error on a validation set consisting of the last 10,000 training examples. To make use of the full training set, we recorded the value of the log likelihood on the first 50,000 examples at the point of minimal validation error. We then continued training on the full 60,000 example training set until the validation set log likelihood matched this number. We obtained a test set error of 0.94%, which is the best result we are aware of that does not use unsupervised pretraining. We summarize the best published results on permutation

Table 9.2: Test set misclassification rates for the best methods on the general MNIST dataset, excluding methods that augment the training data.

Method	Test error
2-layer CNN+2-layer NN (Jarrett et al., 2009)	0.53%
Stochastic pooling Zeiler and Fergus (2013a)	0.47%
Conv. maxout + dropout	**0.45%**

invariant MNIST in Table 9.1.

We also considered the MNIST dataset without the permutation invariance restriction. In this case, we used three convolutional maxout hidden layers (with spatial max pooling on top of the maxout layers) followed by a densely connected softmax layer. We were able to rapidly explore hyperparameter space thanks to the extremely fast GPU convolution library developed by Krizhevsky et al. (2012). We obtained a test set error rate of 0.45%, which sets a new state of the art in this category. (It is possible to get better results on MNIST by augmenting the dataset with transformations of the standard set of images (Ciresan et al., 2010)) A summary of the best methods on the general MNIST dataset is provided in Table 9.2.

9.5.2 CIFAR-10

The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) consists of 32×32 color images drawn from 10 classes split into 50,000 train and 10,000 test images. We preprocess the data using global contrast normalization and ZCA whitening.

We follow a similar procedure as with the MNIST dataset, with one change. On MNIST, we find the best number of training epochs in terms of validation set error, then record the training set log likelihood and continue training using the entire training set until the validation set log likelihood has reached this value. On CIFAR-10, continuing training in this fashion is infeasible because the final value of the learning rate is very small and the validation set error is very high. Training until the validation set likelihood matches the cross-validated value of the training likelihood would thus take prohibitively long. Instead, we retrain the model from

Table 9.3: Test set misclassification rates for the best methods on the CIFAR-10 dataset.

Method	Test error
Stochastic pooling Zeiler and Fergus (2013a)	15.13%
CNN + Spearmint Snoek et al. (2012)	14.98%
Conv. maxout + dropout	**11.68 %**
CNN + Spearmint + data augmentation Snoek et al. (2012)	9.50 %
Conv. maxout + dropout + data augmentation	**9.38 %**

scratch, and stop when the new likelihood matches the old one.

Our best model consists of three convolutional maxout layers, a fully connected maxout layer, and a fully connected softmax layer. Using this approach we obtain a test set error of 11.68%, which improves upon the state of the art by over two percentage points. (If we do not train on the validation set, we obtain a test set error of 13.2%, which also improves over the previous state of the art). If we additionally augment the data with translations and horizontal reflections, we obtain the absolute state of the art on this task at 9.35% error. In this case, the likelihood during the retrain never reaches the likelihood from the validation run, so we retrain for the same number of epochs as the validation run. A summary of the best CIFAR-10 methods is provided in Table 9.3.

9.5.3 CIFAR-100

The CIFAR-100 (Krizhevsky and Hinton, 2009) dataset is the same size and format as the CIFAR-10 dataset, but contains 100 classes, with only one tenth as many labeled examples per class. Due to lack of time we did not extensively cross-validate hyperparameters on CIFAR-100 but simply applied hyperparameters we found to work well on CIFAR-10. We obtained a test set error of 38.57%, which is state of the art. If we do not retrain using the entire training set, we obtain a test

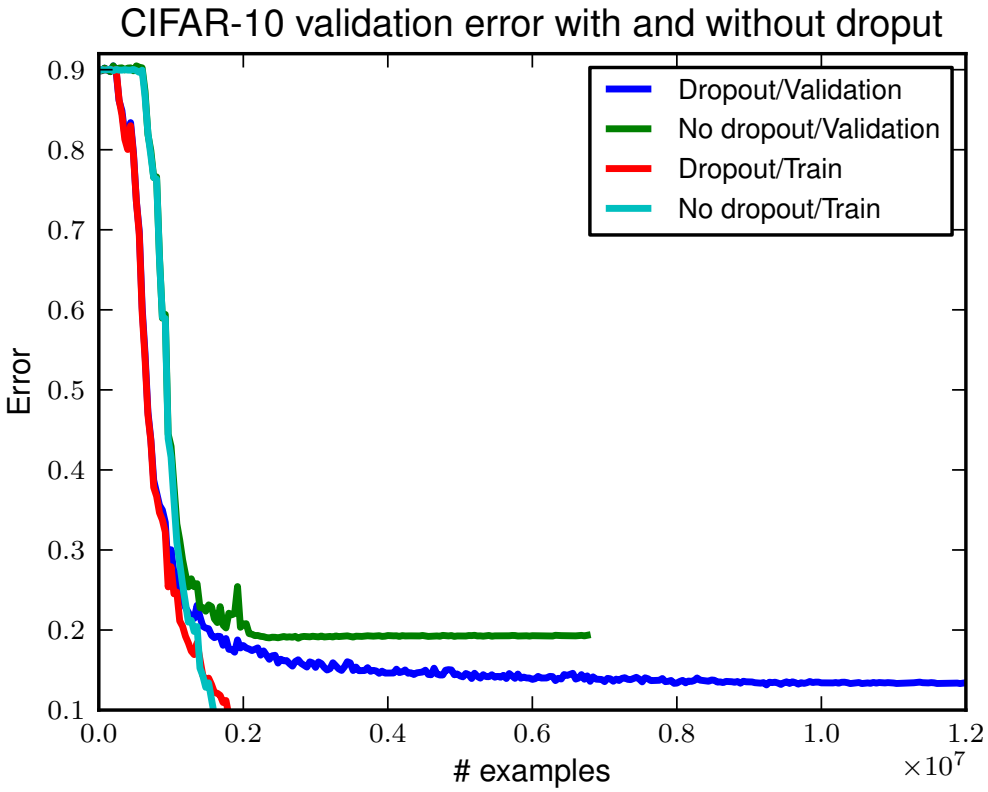


Figure 9.5: When training maxout, the improvement in validation set error that results from using dropout is dramatic. Here we find a greater than 25% reduction in our validation set error on CIFAR-10.

set error of 41.48%, which also surpasses the current state of the art. A summary of the best methods on CIFAR-100 is provided in Table 9.4.

9.5.4 Street View House Numbers

The SVHN (Netzer et al., 2011) dataset consists of color images of house numbers collected by Google Street View. The dataset comes in two formats. We consider the second format, in which each image is of size 32×32 and the task is to classify the digit in the center of the image. Additional digits may appear beside it but must be ignored. There are 73,257 digits in the training set, 26,032 digits in the test set and 531,131 additional, somewhat less difficult examples, to use as an extra training set. Following Sermanet et al. (2012), to build a validation set, we select 400 samples per class from the training set and 200 samples per class from the extra set. The remaining digits of the train and extra sets are used for training.

Table 9.4: Test set misclassification rates for the best methods on the CIFAR-100 dataset.

Method	Test error
Learned pooling (Malinowski and Fritz, 2013)	43.71%
Stochastic poolingZeiler and Fergus (2013a)	42.51%
Conv. maxout + dropout	**38.57%**

Table 9.5: Test set misclassification rates for the best methods on the SVHN dataset.

Method	Test error
Sermanet et al. (2012)	4.90%
Stochastic pooling Zeiler and Fergus (2013a)	2.80 %
Rectifiers + dropout Srivastava (2013)	2.78 %
Rectifiers + dropout + synthetic translation Srivastava (2013)	2.68 %
Conv. maxout + dropout	**2.47 %**

For SVHN, we did not train on the validation set at all. We used it only to find the best hyperparameters. We applied local contrast normalization preprocessing the same way as Zeiler and Fergus (2013a). Otherwise, we followed the same approach as on MNIST. Our best model consists of three convolutional maxout hidden layers and a densely connected maxout layer followed by a densely connected softmax layer. We obtained a test set error rate of 2.47%, which sets the state of the art. A summary of comparable methods is provided in Table 9.5.

9.6 Comparison to rectifiers

One obvious question about our results is whether we obtained them by improved preprocessing or larger models, rather than by the use of maxout. For MNIST we used no preprocessing, and for SVHN, we use the same preprocessing

as Zeiler and Fergus (2013a). However on the CIFAR datasets we did use a new form of preprocessing. We therefore compare maxout to rectifiers run with the same processing and a variety of model sizes on this dataset.

By running a large cross-validation experiment (see Fig. 9.6) we found that maxout offers a clear improvement over rectifiers. We also found that our preprocessing and size of models improves rectifiers and dropout beyond the previous state of the art result. Cross-channel pooling is a method for reducing the size of state and number of parameters needed to have a given number of filters in the model. Performance seems to correlate well with the number of filters for maxout but with the number of output units for rectifiers–i.e, rectifier units do not benefit much from cross-channel pooling. Rectifier units do best without cross-channel pooling but with the same number of filters, meaning that the size of the state and the number of parameters must be about k times higher for rectifiers to obtain generalization performance approaching that of maxout.

9.7 Model averaging

Having demonstrated that maxout networks are effective models, we now analyze the reasons for their success. We first identify reasons that maxout is highly compatible with dropout's approximate model averaging technique.

The intuitive justification for averaging sub-models by dividing the weights by 2 given by (Hinton et al., 2012) is that this does exact model averaging for a single layer model, softmax regression. To this characterization, we add the observation that the model averaging remains exact if the model is extended to multiple linear layers. While this has the same representational power as a single layer, the expression of the weights as a product of several matrices could have a different inductive bias. More importantly, it indicates that dropout does exact model averaging in deeper architectures provided that they are locally linear among the space of inputs to each layer that are visited by applying different dropout masks.

We argue that dropout training encourages maxout units to have large linear regions around inputs that appear in the training data. Because each sub-model must make a good prediction of the output, each unit should learn to have roughly

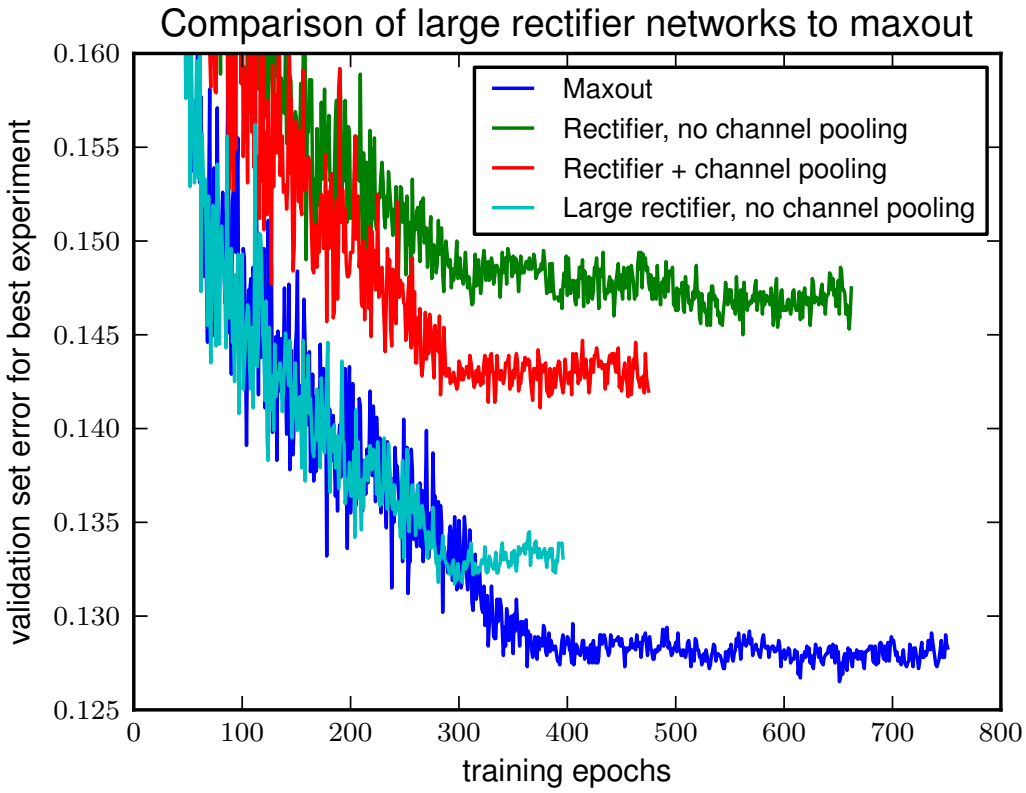


Figure 9.6: We cross-validated the momentum and learning rate for four architectures of model: 1) Medium-sized maxout network. 2) Rectifier network with cross-channel pooling, and exactly the same number of parameters and units as the maxout network. 3) Rectifier network without cross-channel pooling, and the same number of units as the maxout network (thus fewer parameters). 4) Rectifier network without cross-channel pooling, but with k times as many units as the maxout network. Because making layer i have k times more outputs increases the number of inputs to layer $i+1$, this network has roughly k times more parameters than the maxout network, and requires significantly more memory and runtime. We sampled 10 learning rate and momentum schedules and random seeds for dropout, then ran each configuration for all 4 architectures. Each curve terminates after failing to improve the validation error in the last 100 epochs.

the same activation regardless of which inputs are dropped. In a maxout network with arbitrarily selected parameters, varying the dropout mask will often move the effective inputs far enough to escape the local region surrounding the clean inputs in which the hidden units are linear, i.e., changing the dropout mask could frequently change which piece of the piecewise function an input is mapped to. Maxout *trained with dropout* may have the identity of the maximal filter in each unit change relatively rarely as the dropout mask changes. Networks of linear operations and $\max(\cdot)$ may learn to exploit dropout's approximate model averaging technique well.

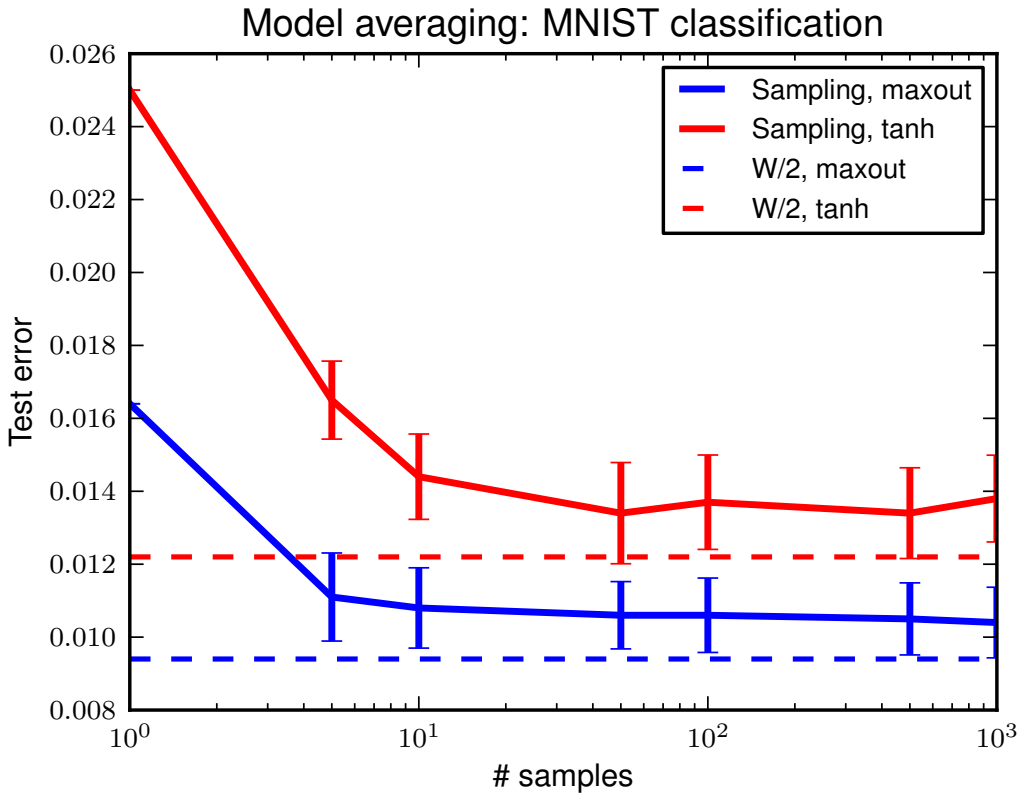


Figure 9.7: The error rate of the prediction obtained by sampling several sub-models and taking the geometric mean of their predictions approaches the error rate of the prediction made by dividing the weights by 2. However, the divided weights still obtain the best test error, suggesting that dropout is a good approximation to averaging over a very large number of models. Note that the correspondence is more clear in the case of maxout.

Many popular activation functions have significant curvature nearly everywhere. These observations suggest that the approximate model averaging of dropout will not be as accurate for networks incorporating such activation functions. To test this, we compared the best maxout model trained on MNIST with dropout to a hyperbolic tangent network trained on MNIST with dropout. We sampled several subsets of each model and compared the geometric mean of these sampled models' predictions to the prediction made using the dropout technique of dividing the weights by 2. We found evidence that dropout is indeed performing model averaging, even in multilayer networks, and that it is more accurate in the case of maxout. See Fig. 9.7 and Fig. 9.8 for details.

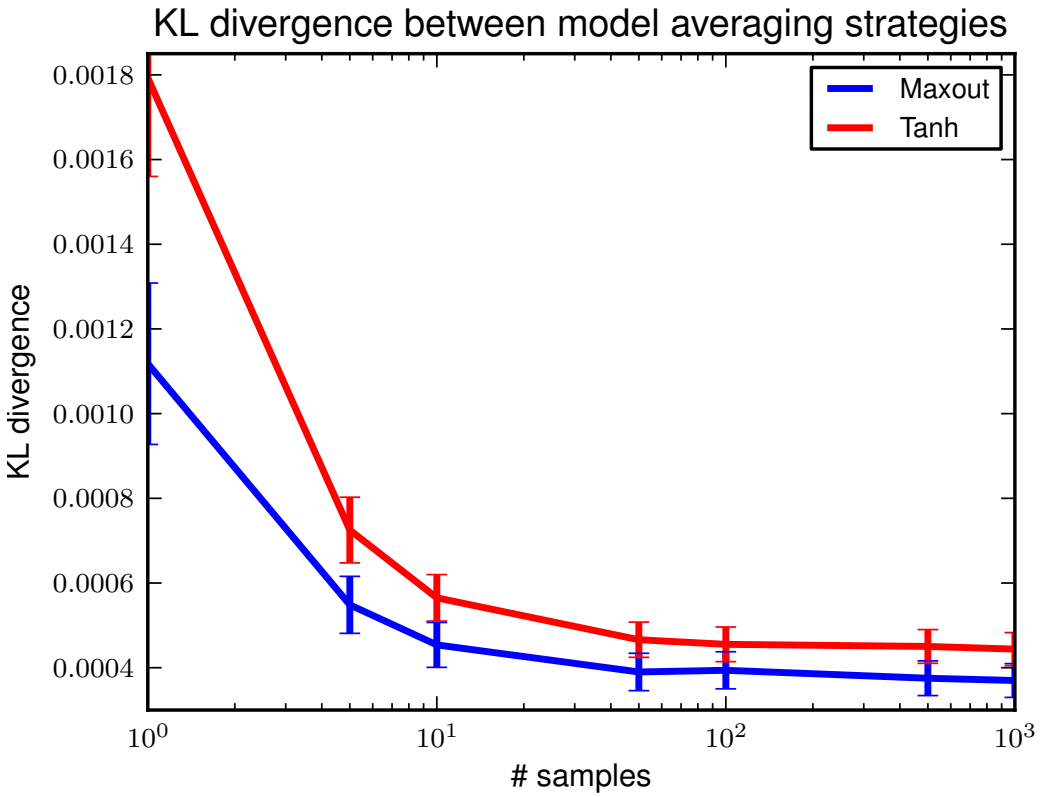


Figure 9.8: The KL divergence between the distribution predicted using the dropout technique of dividing the weights by 2 and the distribution obtained by taking the geometric mean of the predictions of several sampled models decreases as the number of samples increases. This suggests that dropout does indeed do model averaging, even for deep networks. The approximation is more accurate for maxout units than for tanh units.

9.8 Optimization

The second key reason that maxout performs well is that it improves the bagging style training phase of dropout. Note that the arguments in section 9.7 motivating the use of maxout also apply equally to rectified linear units (Salinas and Abbott, 1996; Hahnloser, 1998; Glorot et al., 2011). The only difference between maxout and max pooling over a set of rectified linear units is that maxout does not include a 0 in the max. Superficially, this seems to be a small difference, but we find that including this constant 0 is very harmful to optimization in the context of dropout. For instance, on MNIST our best validation set error with an MLP is 1.04%. If we include a 0 in the max, this rises to over 1.2%. We argue that, when trained with dropout, maxout is easier to optimize than rectified linear units with cross-channel

pooling.

9.8.1 Optimization experiments

To verify that maxout yields better optimization performance than max pooled rectified linear units when training with dropout, we carried out two experiments. First, we stressed the optimization capabilities of the training algorithm by training a small (two hidden convolutional layers with $k = 2$ and sixteen kernels) model on the large (600,000 example) SVHN dataset. When training with rectifier units the training error gets stuck at 7.3%. If we train instead with maxout units, we obtain 5.1% training error. As another optimization stress test, we tried training very deep and narrow models on MNIST, and found that maxout copes better with increasing depth than pooled rectifiers. See Fig. 9.9 for details.

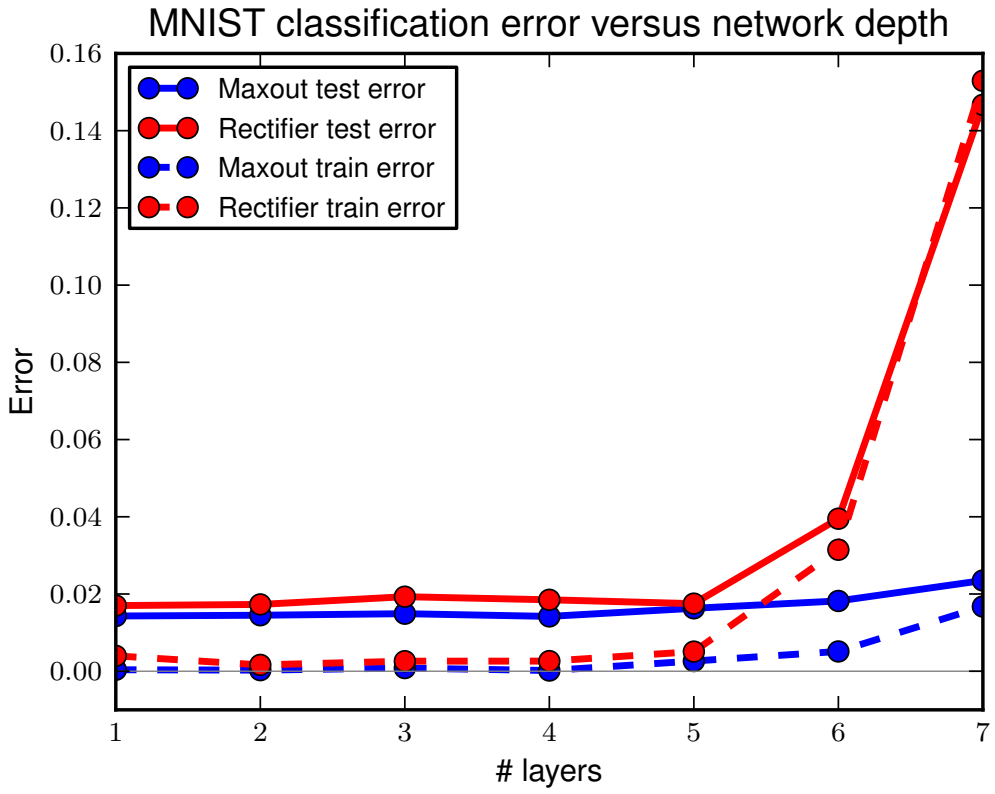


Figure 9.9: We trained a series of models with increasing depth on MNIST. Each layer contains only 80 units (k=5) to make fitting the training set difficult. Maxout optimization degrades gracefully with depth but pooled rectifier units worsen noticeably at 6 layers and dramatically at 7.

9.8.2 Saturation

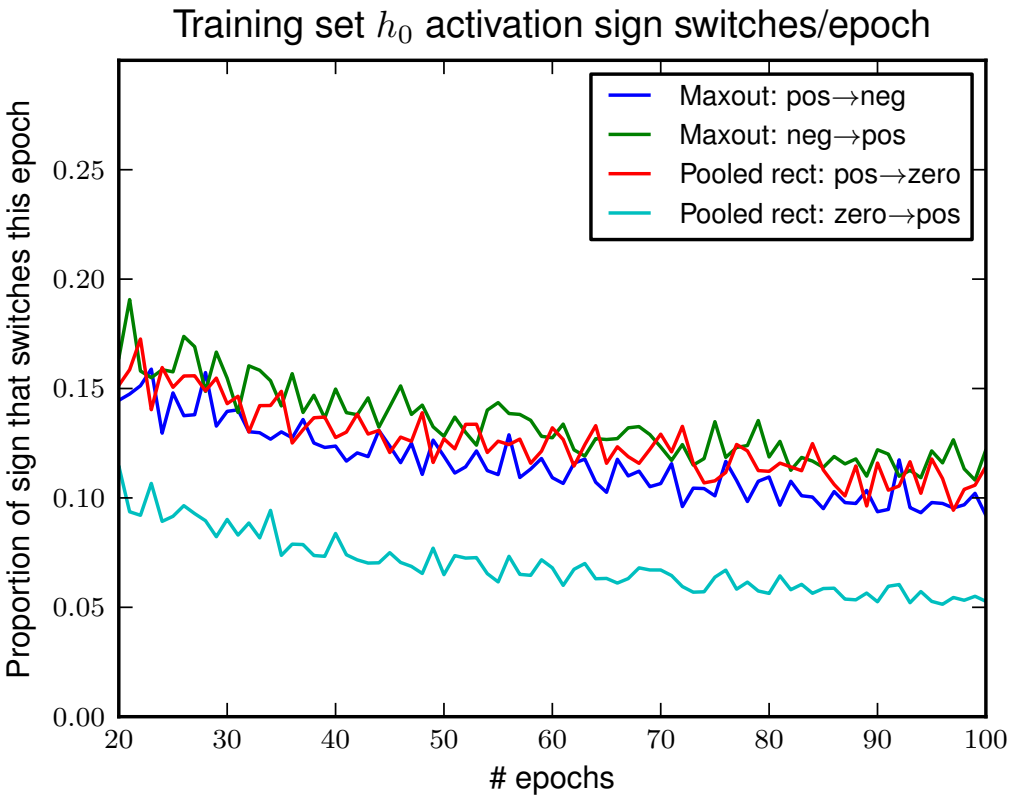


Figure 9.10: During dropout training, rectifier units transition from positive to 0 activation more frequently than they make the opposite transition, resulting a preponderence of 0 activations. Maxout units freely move between positive and negative signs at roughly equal rates.

Optimization proceeds very differently when using dropout than when using ordinary stochastic gradient descent. SGD usually works best with a small learning rate that results in a smoothly decreasing objective function, while dropout works best with a large learning rate, resulting in a constantly fluctuating objective function. Dropout rapidly explores many different directions and rejects the ones that worsen performance, while SGD moves slowly and steadily in the most promising direction. We find empirically that these different operating regimes result in different outcomes for rectifier units. When training with SGD, we find that the rectifier units saturate at 0 less than 5% of the time. When training with dropout, we initialize the units to sature rarely but training gradually increases their saturation rate to 60%. Because the 0 in the $\max(0, z)$ activation function is a constant, this blocks the gradient from flowing through the unit. In the absence of

gradient through the unit, it is difficult for training to change this unit to become active again. *Maxout does not suffer from this problem because gradient always flows through every maxout unit*–even when a maxout unit is 0, this 0 is a function of the parameters and may be adjusted Units that take on negative activations may be steered to become positive again later. Fig. 9.10 illustrates how active rectifier units become inactive at a greater rate than inactive units become active when training with dropout, but maxout units, which are always active, transition between positive and negative activations at about equal rates in each direction. We hypothesize that the high proportion of zeros and the difficulty of escaping them impairs the optimization performance of rectifiers relative to maxout.

To test this hypothesis, we trained two MLPs on MNIST, both with two hidden layers and 1200 filters per layer pooled in groups of 5. When we include a constant 0 in the max pooling, the resulting trained model fails to make use of 17.6% of the filters in the second layer and 39.2% of the filters in the second layer. A small minority of the filters usually took on the maximal value in the pool, and the rest of the time the maximal value was a constant 0. Maxout, on the other hand, used all but 2 of the 2400 filters in the network. Each filter in each maxout unit in the network was maximal for some training example. All filters had been utilised and tuned.

9.8.3 Lower layer gradients and bagging

To behave differently from SGD, dropout requires the gradient to change noticeably as the choice of which units to drop changes. If the gradient is approximately constant with respect to the dropout mask, then dropout simplifies to SGD training. We tested the hypothesis that rectifier networks suffer from diminished gradient flow to the lower layers of the network by monitoring the variance with respect to dropout masks for fixed data during training of two different MLPs on MNIST. The variance of the gradient on the output weights was 1.4 times larger for maxout on an average training step, while the variance on the gradient of the first layer weights was 3.4 times larger for maxout than for rectifiers. Combined with our previous result showing that maxout allows training deeper networks, this greater variance suggests that maxout better propagates varying information downward to the lower layers and helps dropout training to better resemble bagging for the

lower-layer parameters. Rectifier networks, with more of their gradient lost to saturation, presumably cause dropout training to resemble regular SGD toward the bottom of the network.

9.9 Conclusion

We have proposed a new activation function called maxout that is particularly well suited for training with dropout, and for which we have proven a universal approximation theorem. We have shown empirical evidence that dropout attains a good approximation to model averaging in deep models. We have shown that maxout exploits this model averaging behavior because the approximation is more accurate for maxout units than for tanh units. We have demonstrated that optimization behaves very differently in the context of dropout than in the pure SGD case. By designing the maxout gradient to avoid pitfalls such as failing to use many of a model's filters, we are able to train deeper networks than is possible using rectifier units. We have also shown that maxout propagates variations in the gradient due to different choices of dropout masks to the lowest layers of a network, ensuring that every parameter in the model can enjoy the full benefit of dropout and more faithfully emulate bagging training. The state of the art performance of our approach on five different benchmark tasks motivates the design of further models that are explicitly intended to perform well when combined with inexpensive approximations to model averaging.

Acknowledgements

The authors would like to thank the developers of Theano (Bergstra et al., 2010; Bastien et al., 2012), in particular Frédéric Bastien and Pascal Lamblin for their assistance with infrastructure development and performance optimization. We would also like to thank Yann Dauphin for helpful discussions.

10 Prologue to Fourth Article

10.1 Article Details

Multi-digit number recognition from Street View imagery using deep convolutional neural networks. Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet (2014). In *International Conference on Learning Representations*.

Personal Contribution. Because this article was written in the context of an internship with the Street Smart team at Google, my personal contribution to this article was less than to the other articles used in this thesis. The rest of the Street Smart team already had the basic idea of doing transcription with a single neural net before I arrived, and ran the experiments necessary for scientific publication after I left. My main contribution was to clearly define the equations needed for maximum likelihood training and MAP inference in the output sequence model, to write the code for those features, and to configure the system for good performance on the internal, private dataset. I did not perform any of the experiments designed to explain the factors driving the success of the system, nor did I do any of the work on the publicly available dataset.

10.2 Context

Prior to this work, no results had been published for the full sequence transcription task on the Street View House Numbers dataset. Previous work, such as that presented in chapter 9, had exclusively focused on the isolated digit recognition task. Viable technologies already existed for sequence transcription, but were mostly evaluated on other datasets. These technologies were not based on learning to perform localization and segmentation in a single neural network, but

rather employed hand-designed probabilistic models that sometimes used neural networks to implement individual conditional probability distributions within the model. See (Alsharif and Pineau, 2013) for an example of a recent state of the art system employing this approach.

10.3 Contributions

The contribution of this paper is twofold. First, we demonstrate a solution to the problem of short sequence transcription for geocoding. Second, in doing so, we have shown that deep neural networks can learn to perform complicated localization and segmentation tasks rather than simply recognition tasks.

10.4 Recent Developments

This paper is very recent, and will in fact not be presented until after the writing of this thesis. There are therefore no more recent developments to report.

11 Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

11.1 Introduction

Recognizing multi-digit numbers in photographs captured at street level is an important component of modern-day map making. A classic example of a corpus of such street level photographs is Google's Street View imagery comprised of hundreds of millions of geo-located 360 degree panoramic images. The ability to automatically transcribe an address number from a geo-located patch of pixels and associate the transcribed number with a known street address helps pinpoint, with a high degree of accuracy, the location of the building it represents.

More broadly, recognizing numbers in photographs is a problem of interest to the optical character recognition community. While OCR on constrained domains like document processing is well studied, arbitrary multi-character text recognition in photographs is still highly challenging. This difficulty arises due to the wide variability in the visual appearance of text in the wild on account of a large range of fonts, colors, styles, orientations, and character arrangements. The recognition problem is further complicated by environmental factors such as lighting, shadows, specularities, and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs.

In this paper, we focus on recognizing multi-digit numbers from Street View panoramas. While this reduces the space of characters that need to be recognized, the complexities listed above still apply to this sub-domain. Due to these complexities, traditional approaches to solve this problem typically separate out the localization, segmentation, and recognition steps.

In this paper we propose a unified approach that integrates these three steps via the use of a deep convolutional neural network that operates directly on the image pixels. This model is configured with multiple hidden layers (our best configuration had eleven layers, but our experiments suggest deeper architectures may

obtain better accuracy, with diminishing returns), all with feedforward connections. We employ DistBelief to implement these large-scale deep neural networks. The key contributions of this paper are: (a) a unified model to localize, segment, and recognize multi-digit numbers from street level photographs (b) a new kind of output layer, providing a conditional probabilistic model of sequences (c) empirical results that show this model performing best with a deep architecture (d) reaching human level performance at specific operating thresholds.

We have evaluated this approach on the publicly available Street View House Numbers (SVHN) dataset and achieve over 96% accuracy in recognizing street numbers. We show that on a per-digit recognition task, we improve upon the state-of-the-art and achieve 97.84% accuracy. We also evaluated this approach on an even more challenging dataset generated from Street View imagery containing several tens of millions of street number annotations and achieve over 90% accuracy. Our evaluations further indicate that at specific operating thresholds, the performance of the proposed system is comparable to that of human operators. To date, our system has helped us extract close to 100 million street numbers from Street View imagery worldwide.

The rest of the paper is organized as follows: Section 11.2 explores past work on deep neural networks and on Photo-OCR. Sections 11.3 and 11.4 list the problem definition and describe the proposed method. Section 11.5 describes the experimental set up and results. Key takeaway ideas are discussed in Section 11.6.

11.2 Related work

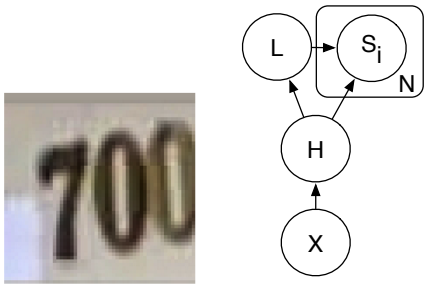
Convolutional neural networks (Fukushima, 1980; LeCun et al., 1998) are neural networks with sets of neurons having tied parameters. Like most neural networks, they contain several filtering layers with each layer applying an affine transformation to the vector input followed by an elementwise non-linearity. In the case of convolutional networks, the affine transformation can be implemented as a discrete convolution rather than a fully general matrix multiplication. This makes convolutional networks computationally efficient, allowing them to scale to large images. It also builds equivariance to translation into the model (in other words, if the image is shifted by one pixel to the right, then the output of the convolution is

also shifted one pixel to the right; the two representations vary equally with translation). Image-based convolutional networks typically use a *pooling layer* which summarizes the activations of many adjacent filters with a single response. Such pooling layers may summarize the activations of groups of units with a function such as their maximum, mean, or L2 norm. These pooling layers help the network be robust to small translations of the input.

Increases in the availability of computational resources, increases in the size of available training sets, and algorithmic advances such as the use of piecewise linear units (Jarrett et al., 2009; Glorot et al., 2011; Goodfellow et al., 2013) and dropout training (Hinton et al., 2012) have resulted in many recent successes using deep convolutional neural networks. Krizhevsky et al. (2012) obtained dramatic improvements in the state of the art in object recognition. Zeiler and Fergus (2013b) later improved upon these results.

On huge datasets, such as those used at Google, overfitting is not an issue, and increasing the size of the network increases both training and testing accuracy. To this end, Dean et al. (2012) developed DistBelief, a scalable implementation of deep neural networks, which includes support for convolutional networks. We use this infrastructure as the basis for the experiments in this paper.

Convolutional neural networks have previously been used mostly for applications such as recognition of single objects in the input image. In some cases they have been used as components of systems that solve more complicated tasks. Girshick et al. (2013) use convolutional neural networks as feature extractors for a system that performs object detection and localization. However, the system as a whole is larger than the neural network portion trained with backprop, and has special code for handling much of the mechanics such as proposing candidate object regions. Szegedy et al. (2013) showed that a neural network could learn to output a heatmap that could be post-processed to solve the object localization problem. In our work, we take a similar approach, but with less post-processing and with the additional requirement that the output be an ordered sequence rather than an unordered list of detected objects. Alsharif and Pineau (2013) use convolutional maxout networks (Goodfellow et al., 2013) to provide many of the conditional probability distributions used in a larger model using HMMs to transcribe text from images. In this work, we propose to solve similar tasks involving localization and segmentation, but we propose to perform the entire task completely within the



a) b)

Figure 11.1: a) An example input image to be transcribed. The correct output for this image is "700". b) The graphical model structure of our sequence transcription model, depicted using plate notation (Buntine, 1994) to represent the multiple S_i. Note that the relationship between X and H is deterministic. The edges going from L to S_i are optional, but help draw attention to the fact that our definition of $P(\mathbf{S} \mid X)$ does not query S_i for $i > L$.

learned convolutional network. In our approach, there is no need for a separate component of the system to propose candidate segmentations or provide a higher level model of the image.

11.3 Problem description

Street number transcription is a special kind of sequence recognition. Given an image, the task is to identify the number in the image. See an example in Fig. 11.1a. The number to be identified is a sequence of digits, $\mathbf{s} = s_1, s_2, \ldots, s_n$. When determining the *accuracy* of a digit transcriber, we compute the proportion of the input images for which the length n of the sequence and every element s_i of the sequence is predicted correctly. There is no "partial credit" for getting individual digits of the sequence correct. This is because for the purpose of making a map, a building can only be found on the map from its address if the whole street number was transcribed correctly.

For the purpose of building a map, it is extremely important to have at least human level accuracy. Users of maps find it very time consuming and frustrating to be led to the wrong location, so it is essential to minimize the amount of incorrect transcriptions entered into the map. It is, however, acceptable not to transcribe every input image. Because each street number may have been photographed many

times, it is still quite likely that the proportion of buildings we can place on the map is greater than the proportion of images we can transcribe. We therefore advocate evaluating this task based on the *coverage* at certain levels of accuracy, rather than evaluating only the total degree of accuracy of the system. To evaluate coverage, the system must return a confidence value, such as the probability of the most likely prediction being correct. Transcriptions below some confidence threshold can then be discarded. The coverage is defined to be the proportion of inputs that are not discarded. The coverage at a certain specific accuracy level is the coverage that results when the confidence threshold is chosen to achieve that desired accuracy level. For map-making purposes, we are primarily interested in coverage at 98% accuracy or better, since this roughly corresponds to human accuracy.

Using confidence thresholding allows us to improve maps incrementally over time–if we develop a system with poor accuracy overall but good accuracy at some threshold, we can make a map with partial coverage, then improve the coverage when we get a more accurate transcription system in the future. We can also use confidence thresholding to do as much of the work as possible via the automated system and do the rest using more expensive means such as hiring human operators to transcribe the remaining difficult inputs.

One special property of the street number transcription problem is that the sequences are of bounded length. Very few street numbers contain more than five digits, so we can use models that assume the sequence length n is at most some constant N, with $N = 5$ for this work. Systems that make such an assumption should be able to identify whenever this assumption is violated and refuse to return a transcription so that the few street numbers of length greater than N are not incorrectly added to the map after being transcribed as being length N. (Alternately, one can return the most likely sequence of length N, and because the probability of that transcription being correct is low, the default confidence thresholding mechanism will usually reject such transcriptions without needing special code for handling the excess length case)

11.4 Methods

Our basic approach is to train a probabilistic model of sequences given images. Let $\mathbf{S}$ represent the output sequence and X represent the input image. Our goal is then to learn a model of $P(\mathbf{S} \mid X)$ by maximizing $\log P(\mathbf{S} \mid X)$ on the training set.

To model $\mathbf{S}$, we define $\mathbf{S}$ as a collection of N random variables $S_1, \ldots, S_N$ representing the elements of the sequence and an additional random variable L representing the length of the sequence. We assume that the identities of the separate digits are independent from each other, so that the probability of a specific sequence $\mathbf{s} = s_1, \ldots, s_n$ is given by

$$P(\mathbf{S} = \mathbf{s}|X) = P(L = n \mid X)\Pi_{i=1}^{n} P(S_i = s_i \mid X).$$

This model can be extended to detect when our assumption that the sequence has length at most N is violated. To allow for detecting this case, we simply add an additional value of L that represents this outcome.

Each of the variables above is discrete, and when applied to the street number transcription problem, each has a small number of possible values: L has only 7 values (0, ..., 5, and "more than 5"), and each of the digit variables has 10 possible values. This means it is feasible to represent each of them with a softmax classifier that receives as input features extracted from X by a convolutional neural network. We can represent these features as a random variable H whose value is deterministic given X. In this model, $P(\mathbf{S} \mid X) = P(\mathbf{S} \mid H)$. See Fig. 11.1b for a graphical model depiction of the network structure.

To train the model, one can maximize $\log P(\mathbf{S} \mid X)$ on the training set using a generic method like stochastic gradient descent. Each of the softmax models (the model for L and each S_i) can use exactly the same backprop learning rule as when training an isolated softmax layer, except that a digit classifier softmax model backprops nothing on examples for which that digit is not present.

At test time, we predict

$$\mathbf{s} = (l, s_1, \ldots, s_l) = \text{argmax}_{L,S_1,\ldots,S_L} \log P(S \mid X).$$

This argmax can be computed in linear time. The argmax for each character can be computed independently. We then incrementally add up the log probabilities for each character. For each length l, the complete log probability is given by this running sum of character log probabilities, plus $\log P(l \mid x)$. The total runtime is

thus $O(N)$.

We preprocess by subtracting the mean of each image. We do not use any whitening (Hyvärinen et al., 2001), local contrast normalization (Sermanet et al., 2012), etc.

11.5 Experiments

In this section we present our experimental results. First, we describe our state of the art results on the public Street View House Numbers dataset in section 11.5.1. Next, we describe the performance of this system on our more challenging, larger but internal version of the dataset in section 11.5.2. We then present some experiments analyzing the performance of the system in section 11.5.3.

11.5.1 Public Street View House Numbers dataset

The Street View House Numbers (SVHN) dataset (Netzer et al., 2011) is a dataset of about 200k street numbers, along with bounding boxes for individual digits, giving about 600k digits total. To our knowledge, all previously published work cropped individual digits and tried to recognize those. We instead take original images containing multiple digits, and focus on recognizing them all simultaneously.

We preprocess the dataset in the following way – first we find the small rectangular bounding box that will contain individual character bounding boxes. We then expand this bounding box by 30% in both the x and the y direction, crop the image to that bounding box and resize the crop to 64×64 pixels. We then crop a 54×54 pixel image from a random location within the 64×64 pixel image. This means we generated several randomly shifted versions of each training example, in order to increase the size of the dataset. Without this data augmentation, we lose about half a percentage point of accuracy. Because of the differing number of characters in the image, this introduces considerable scale variability – for a single digit street number, the digit fills the whole box, meanwhile a 5 digit street number will have to be shrunk considerably in order to fit.

Our best model obtained a sequence transcription accuracy of 96.03%. This is not accurate enough to use for adding street numbers to geographic location databases for placement on maps. However, using confidence thresholding we obtain 95.64% coverage at 98% accuracy. Since 98% accuracy is the performance of human operators, these transcriptions are acceptable to include in a map. We encourage researchers who work on this dataset in the future to publish coverage at 98% accuracy as well as the standard accuracy measure. Our system achieves a character-level accuracy of 97.84%. This is slightly better than the previous state of the art for a single network on the individual character task of 97.53% (Goodfellow et al., 2013).

Training this model took approximately six days using 10 replicas in DistBelief. The exact training time varies for each of the performance measures reported above–we picked the best stopping point for each performance measure separately, using a validation set.

Our best architecture consists of eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. All connections are feedforward and go from one layer to the next (no skip connections). The first hidden layer contains maxout units (Goodfellow et al., 2013) (with three filters per unit) while the others contain rectifier units (Jarrett et al., 2009; Glorot et al., 2011). The number of units at each spatial location in each layer is [48, 64, 128, 160] for the first four layers and 192 for all other locally connected layers. The fully connected layers contain 3,072 units each. Each convolutional layer includes max pooling and subtractive normalization. The max pooling window size is 2×2. The stride alternates between 2 and 1 at each layer, so that half of the layers don't reduce the spatial size of the representation. All convolutions use zero padding on the input to preserve representation size. The subtractive normalization operates on 3x3 windows and preserves representation size. All convolution kernels were of size 5×5. We trained with dropout applied to all hidden layers but not the input.

11.5.2 Internal Street View data

Internally, we have a dataset with tens of millions of transcribed street numbers. However, on this dataset, there are no ground truth bounding boxes available. We use an automated method (beyond the scope of this paper) to estimate the centroid

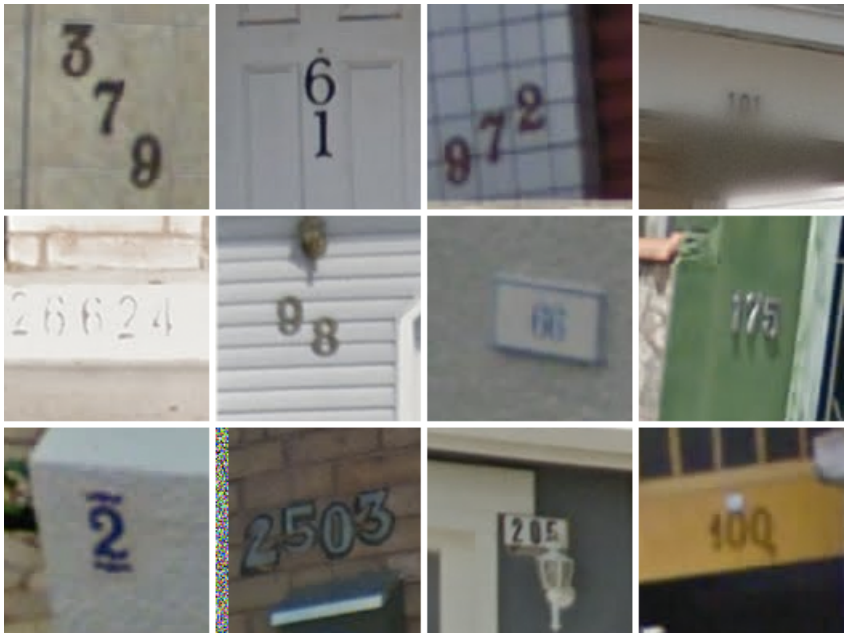


Figure 11.2: Difficult but correctly transcribed examples from the internal street numbers dataset. Some of the challenges in this dataset include diagonal or vertical layouts, incorrectly applied blurring from license plate detection pipelines, shadows and other occlusions.

of each house number, then crop to a 128 $\times$ 128 pixel region surrounding the house number. We do not rescale the image because we do not know the extent of the house number. This means the network must be robust to a wider variation of scales than our public SVHN network. On this dataset, the network must also localize the house number, rather than merely localizing the digits within each house number. Also, because the training set is larger in this setting, we did not need augment the data with random translations.

This dataset is more difficult because it comes from more countries (more than 12), has street numbers with non-digit characters and the quality of the ground truth is lower. See Fig. 11.2 for some examples of difficult inputs from this dataset that our system was able to transcribe correctly, and Fig. 11.3 for some examples of difficult inputs that were considered errors.

We obtained an overall sequence transcription accuracy of 91% on this more challenging dataset. Using confidence thresholding, we were able to obtain a coverage of 83% with 99% accuracy, or 89% coverage at 98% accuracy. On this task, due to the larger amount of training data, we did not see significant overfitting like we saw in SVHN so we did not use dropout. Dropout tends to increase training

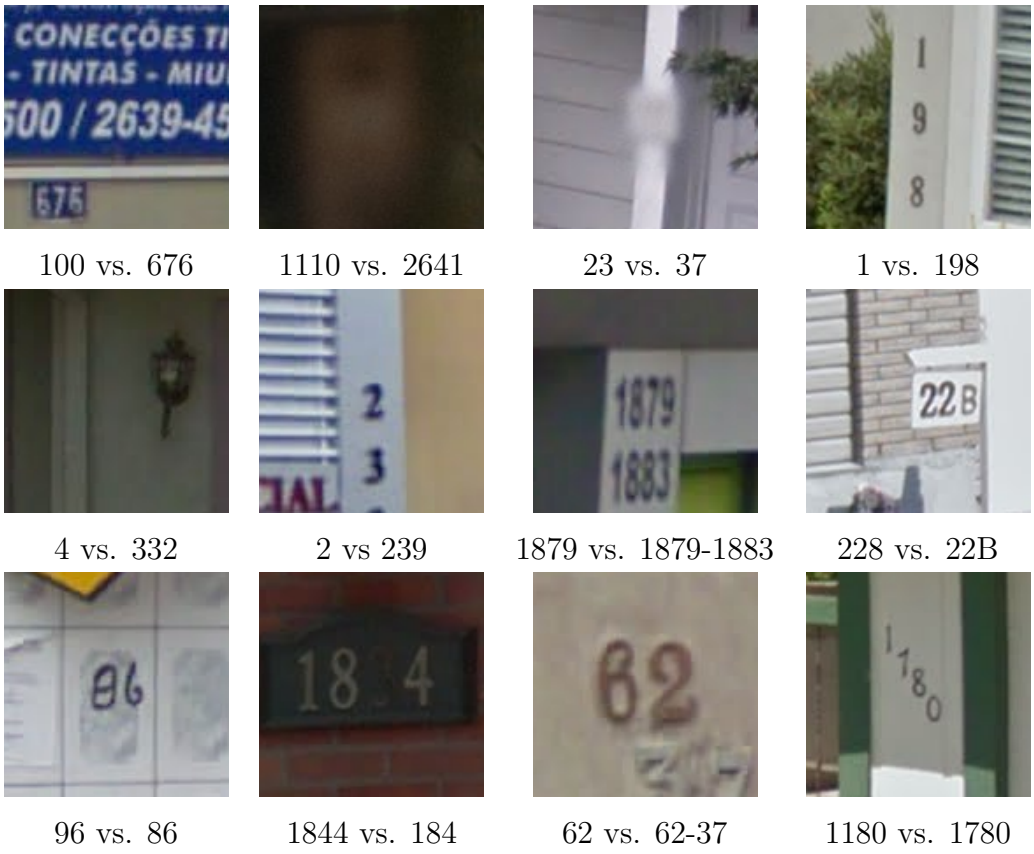


Figure 11.3: Examples of incorrectly transcribed street numbers from the large internal dataset (transcription vs. ground truth). Note that for some of these, the "ground truth" is also incorrect. The ground truth labels in this dataset are quite noisy, as is common in real world settings. Some reasons for the ground truth errors in this dataset include: 1. The data was repurposed from an existing indexing pipeline where operators manually entered street numbers they saw. It was impractical to use the same size of images as the humans saw, so heuristics were used to create smaller crops. Sometimes the resulting crop omits some digits. 2. Some examples are fundamentally ambiguous, for instance street numbers including non-digit characters, or having multiple street numbers in same image which humans transcribed as a single number with an arbitrary separator like "," or "-".

time, and our largest models are already very costly to train. We also did not use maxout units. All hidden units were rectifiers (Jarrett et al., 2009; Glorot et al., 2011). Our best architecture for this dataset is similar to the best architecture for the public dataset, except we use only five convolutional layers rather than eight. (We have not tried using eight convolutional layers on this dataset; eight layers may obtain slightly better results but the version of the network with five convolutional layers performed accurately enough to meet our business objectives) The locally connected layers have 128 units per spatial location, while the fully connected layers have 4096 units per layer.

11.5.3 Performance analysis

In this section we explore the reasons for the unprecedented success of our neural network architecture for a complicated task involving localization and segmentation rather than just recognition. We hypothesize that for such a complicated task, depth is crucial to achieve an efficient representation of the task. State of the art recognition networks for images of cropped and centered digits or objects may have between two to four convolutional layers followed by one or two densely connected hidden layers and the classification layers (Goodfellow et al., 2013). In this work we used several more convolutional layers. We hypothesize that the depth was crucial to our success. This is most likely because the earlier layers can solve the localization and segmentation tasks, and prepare a representation that has already been segmented so that later layers can focus on just recognition. Moreover, we hypothesize that such deep networks have very high representational capacity, and thus need a large amount of data to train successfully. Prior to our successful demonstration of this system, it would have been reasonable to expect that factors other than just depth would be necessary to achieve good performance on these tasks. For example, it could have been possible that a sufficiently deep network would be too difficult to optimize. In Fig. 11.4, we present the results of an experiment that confirms our hypothesis that depth is necessary for good performance on this task.

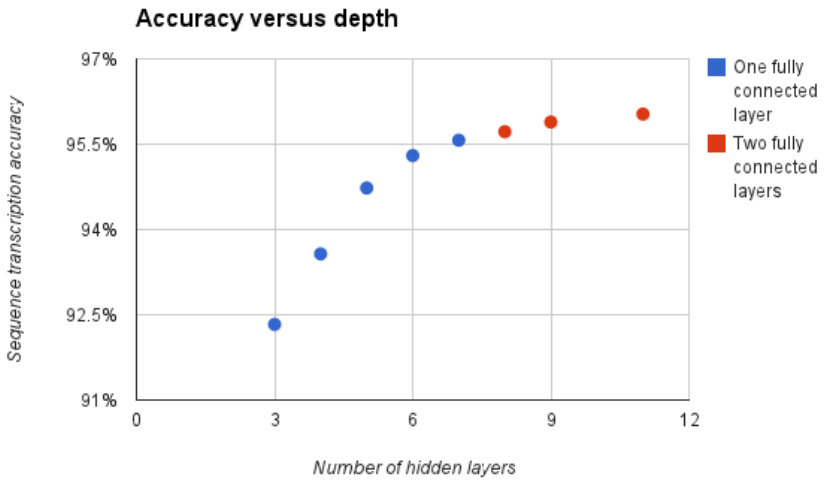


Figure 11.4: Performance analysis experiments on the public SVHN dataset show that fairly deep architectures are needed to obtain good performance on the sequence transcription task.

11.5.4 Application to Geocoding

The motivation for the development of this model was to decrease the cost of geocoding as well as scale it worldwide and keep up with change in the world. The model has now reached a high enough quality level that we can automate the extraction of street numbers on Street View images. Also, even if the model can be considered quite large, it is still efficient.

We can for example transcribe all the views we have of street numbers in France in less than an hour using our Google infrastructure. Most of the cost actually comes from the detection stage that locates the street numbers in the large Street View images. Worldwide, we automatically detected and transcribed close to 100 million physical street numbers at operator level accuracy. Having this new dataset significantly increased the geocoding quality of Google Maps in several countries especially the ones that did not already have other sources of good geocoding. In Fig. 11.5, you can see some automatically extracted street numbers from Street View imagery captured in South Africa.

11.6 Discussion

We believe with this model we have solved OCR for short sequences for many applications. On our particular task, we believe that now the biggest gain we could easily get is to increase the quality of the training set itself as well as increasing its size for general OCR transcription.

One caveat to our results with this architecture is that they rest heavily on the assumption that the sequence is of bounded length, with a reasonably small maximum length N. For unbounded N, our method is not directly applicable, and for large N our method is unlikely to scale well. Each separate digit classifier requires its own separate weight matrix. For long sequences this could incur too high of a memory cost. When using DistBelief, memory is not much of an issue (just use more machines) but statistical efficiency is likely to become problematic. Another problem with long sequences is the cost function itself. It's also possible that, due to longer sequences having more digit probabilities multiplied together, a model of longer sequences could have trouble with systematic underestimation of the sequence length.

One possible solution could be to train a model that outputs one "word" (N character sequence) at a time and then slide it over the entire image followed by a simple decoding. Some early experiments in this direction have been promising.

Perhaps our most interesting finding is that neural networks can learn to perform complicated tasks such as simultaneous localization and segmentation of ordered sequences of objects. This approach of using a single neural network as an entire end-to-end system could be applicable to other problems, such as general text transcription or speech recognition.

Acknowledgments

We would like to thank Ilya Sutskever and Samy Bengio for helpful discussions. We would also like to thank the entire operation team in India that did the labeling effort and without whom this research would not have been possible.

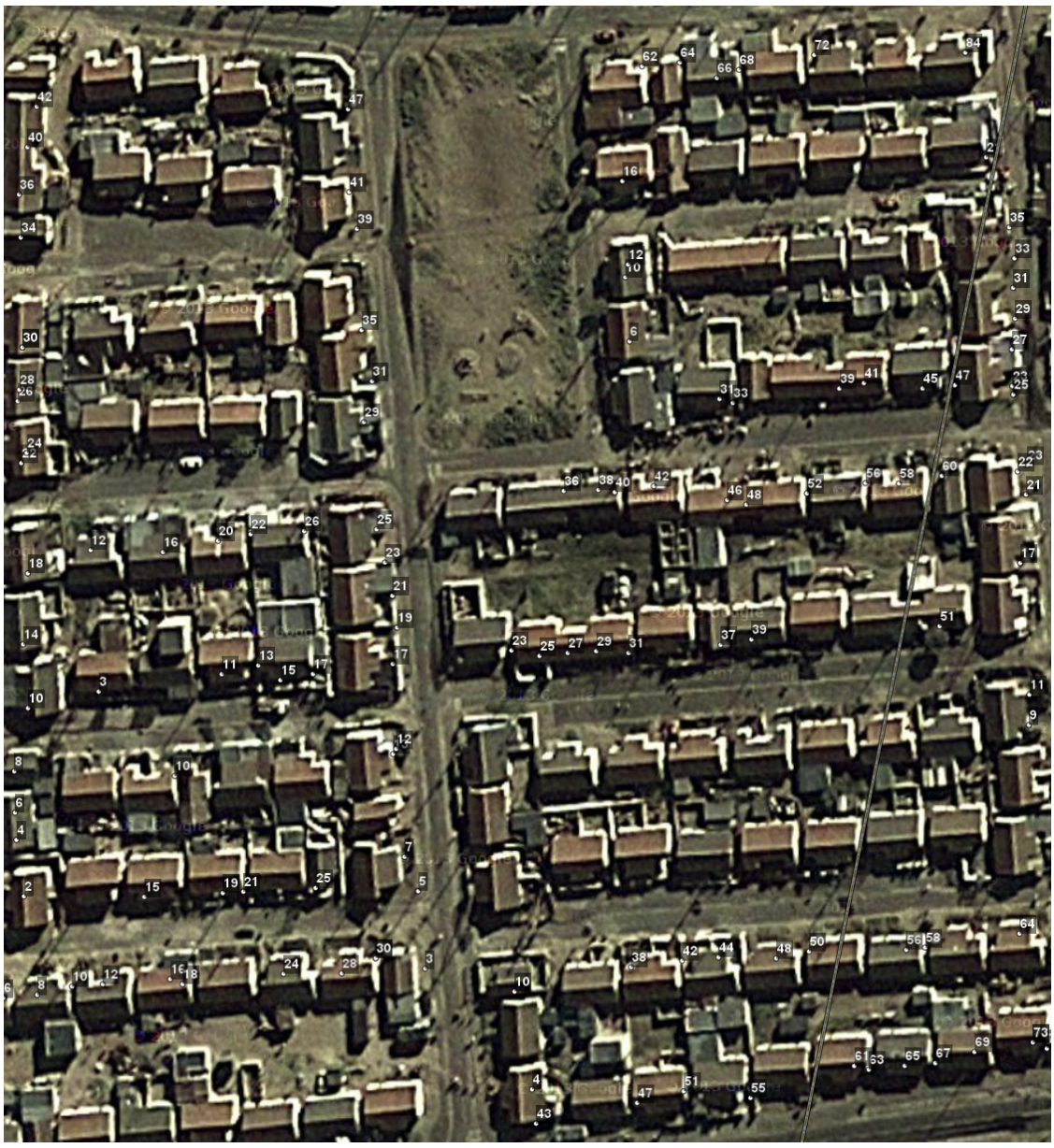


Figure 11.5: Automatically extracted street numbers from Street View imagery captured in South Africa.

.

A Example transcription network inference

In this appendix we provide a detailed example of how to run inference in a trained network to transcribe a house number. The purpose of this appendix is to remove any ambiguity from the more general description in the main text.

Transcription begins by computing the distribution over the sequence $\mathbf{S}$ given an image $\mathbf{X}$. See Fig. A.1 for details of how this computation is performed.

To commit to a single specific sequence transcription, we need to compute $\text{argmax}_{\mathbf{s}} P(\mathbf{S} = \mathbf{s} \mid \mathbf{H})$. It is easiest to do this in log scale, to avoid multiplying together many small numbers, since such multiplication can result in numerical underflow. i.e., in practice we actually compute $\text{argmax}_{\mathbf{s}} \log P(\mathbf{S} = \mathbf{s} \mid \mathbf{H})$.

Note that $\log \text{softmax}(\mathbf{z})$ can be computed efficiently and with numerical stability with the formula $\log \text{softmax}(\mathbf{z})_i = z_i - \sum_j \exp(z_j)$. It is best to compute the log probabilities using this stable approach, rather than first computing the probabilities and then taking their logarithm. The latter approach is unstable; it can incorrectly yield $-\infty$ for small probabilities.

Suppose that we have all of our output probabilities computed, and that they are the following (these are idealized example values, not actual values from the model):

	$L = 0$	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 5$	$L > 5$
$P(L)$	.002	.002	.002	.9	.09	.002	.002
$\log P(L)$	-6.2146	-6.2146	-6.2146	-0.10536	-2.4079	-6.2146	-6.2146

	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$	$i=9$
$P(S_1=i)$	.00125	.9	.00125	.00125	.00125	.00125	.00125	.1	.00125	.00125
$\log P(S_1=i)$	-6.6846	-0.10536	-6.6846	-6.6846	-6.6846	-6.6846	-6.6846	-2.4079	-6.6846	-6.6846
$P(S_2=i)$	.00125	.00125	.00125	.00125	.00125	.00125	.00125	.9	.00125	.1
$\log P(S_2=i)$	-6.6846	-6.6846	-6.6846	-6.6846	-6.6846	-6.6846	-6.6846	-0.10536	-6.6846	-2.4079
$P(S_3=i)$	.00125	.00125	.00125	.00125	.00125	.9	.1	.00125	.00125	.00125
$\log P(S_3=i)$	-6.6846	-6.6846	-6.6846	-6.6846	-6.6846	-0.10536	-2.4079	-6.6846	-6.6846	-6.6846
$P(S_4=i)$	.08889	.2	.08889	.08889	.08889	.08889	.08889	.08889	.08889	.08889
$\log P(S_4=i)$	-2.4204	-1.6094	-2.4204	-2.4204	-2.4204	-2.4204	-2.4204	-2.4204	-2.4204	-2.4204
$P(S_5=i)$	.1	.1	.1	.1	.1	.1	.1	.1	.1	.1
$\log P(S_5=i)$	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026	-2.3026

Refer to the example input image in Fig. A.1 to understand these probabilities. The correct length is 3. Our distribution over L accurately reflects this, though we do think there is a reasonable possibility that L is 4–maybe the edge of the door looks like a fourth digit. The correct transcription is 175, and we do assign these digits the highest probability, but also assign significant probability to the first digit being a 7, the second being a 9, or the third being a 6. There is no fourth digit, but if we parse the edge of the door as being a digit, there is some chance of it being a 1. Our distribution over the fifth digit is totally uniform since there is no fifth digit.

Our independence assumptions mean that when we compute the most likely sequence, the choice of which digit appears in each position doesn't affect our choice of which digit appears in the other positions. We can thus pick the most likely digit in each position separately, leaving us with this table:

j	$\text{argmax}_{s_j} \log P(S_j = s_j)$	$\max_{s_j} \log P(S_j = s_j)$
1	1	-0.10536
2	7	-0.10536
3	5	-0.10536
4	1	-1.6094
5	0	-2.3026

Finally, we can complete the maximization by explicitly calculating the probability of all seven possible sequence lengths:

L	Prediction	$\log P(S_1, \ldots S_L)$	$\log P(\mathbf{S})$
0		0.	-6.2146
1	1	-0.1054	-7.2686
2	17	-0.2107	-8.3226
3	175	-0.3161	-0.42144
4	1751	-1.9255	-4.3334
5	17510	-4.2281	-10.443
> 5	17510...	-4.2281	-10.443

Here the third column is just a cumulative sum over $\log P(S_L)$ so it can be computed in linear time. Likewise, the fourth column is just computed by adding the third column to our existing $\log P(L)$ table. It is not even necessary to keep this final table in memory, we can just use a for loop that generates it one element at a time and remembers the maximal element.

The correct transcription, 175, obtains the maximal log probability of -0.42144, and the model outputs this correct transcription.

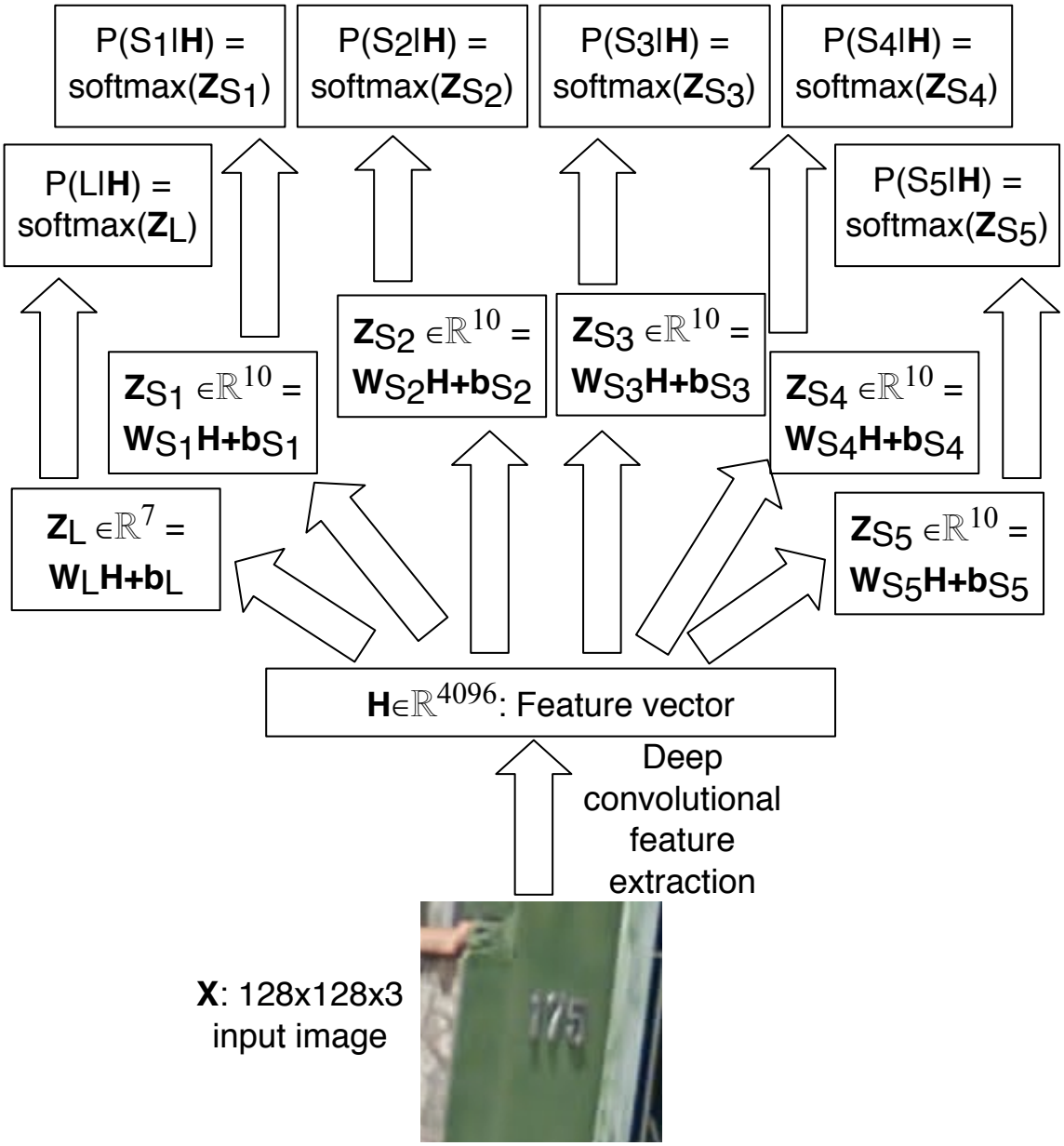


Figure A.1: Details of the computational graph we used to transcribe house numbers. In this diagram, we show how we compute the parameters of $P(\mathbf{S} \mid \mathbf{X})$, where X is the input image and $\mathbf{S}$ is the sequence of numbers depicted by the image. We first extract a set of features $\mathbf{H}$ from $\mathbf{X}$ using a convolutional network with a fully connected final layer. Note that only one such feature vector is extracted for the entire image. We do not use an HMM that models features explicitly extracted at separate locations. Because the final layer of the convolutional feature extractor is fully connected and has no weight sharing, we have not explicitly engineered any concept of spatial location into this representation. The network must learn its own means of representing spatial location in $\mathbf{H}$. Six separate softmax classifiers are then connected to this feature vector $\mathbf{H}$, i.e., each softmax classifier forms a response by making an affine transformation of $\mathbf{H}$ and normalizing this response with the softmax function. One of these classifiers provides the distribution over the sequence length $P(L \mid \mathbf{H})$, while the others provide the distribution over each of the members of the sequence, $P(S_1 \mid \mathbf{H}), \ldots, P(S_5 \mid \mathbf{H})$.

Bibliography

Alsharif, O. and J. Pineau (2013). End-to-end text recognition with hybrid HMM maxout models. Technical report, arXiv:1310.1811.

Arnold, L. and Y. Ollivier (2012, December). Layer-wise learning of deep generative models. Technical report, arXiv:1212.1524.

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. (2009). 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., P. Lamblin, D. Popovici, and H. Larochelle (2007). 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., E. Thibodeau-Laufer, and J. Yosinski (2013). Deep generative stochastic networks trainable by backprop. Technical Report arXiv:1306.1091, Universite de Montreal.

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010, June). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

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

Blackwell, D. (1947). Conditional Expectation and Unbiased Sequential Estimation. *Ann.Math.Statist. 18*, 105–110.

bo Duan, K. and S. S. Keerthi (2005). Which is the best multiclass SVM method? an empirical study. In *Proceedings of the Sixth International Workshop on Multiple Classifier Systems*, pp. 278–285.

Brakel, P., D. Stroobandt, and B. Schrauwen (2013). Training energy-based models for time-series imputation. *Journal of Machine Learning Research 14*, 2771–2797.

Breiman, L. (1994). Bagging predictors. *Machine Learning 24*(2), 123–140.

Bryson, A. E., W. F. Denham, and S. E. Dreyfus (1963). Optimal programming problems with inequality constraints. *AIAA journal 1*(11), 2544–2550.

Buntine, W. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research 2*, 159–225.

Byrd, R. H., P. Lu, J. Nocedal, and C. Zhu (1995, September). A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput. 16*(5), 1190–1208.

Cai, M., Y. Shi, and J. Liu (2013). Deep maxout neural networks for speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pp. 291–296. IEEE.

Ciresan, D. C., U. Meier, L. M. Gambardella, and J. Schmidhuber (2010). Deep big simple neural nets for handwritten digit recognition. *Neural Computation 22*, 1–14.

Coates, A., B. Huval, T. Wang, D. Wu, B. Catanzaro, and N. Andrew (2013, May). Deep learning with cots hpc systems. In S. Dasgupta and D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Volume 28, pp. 1337–1345. JMLR Workshop and Conference Proceedings.

Coates, A., H. Lee, and A. Y. Ng (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*.

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

Cortes, C. and V. Vapnik (1995). Support vector networks. *Machine Learning 20*, 273–297.

Courville, A., J. Bergstra, and Y. Bengio (2011a). A Spike and Slab Restricted Boltzmann Machine. In *Proceedings of The Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'11)*.

Courville, A., J. Bergstra, and Y. Bengio (2011b, June). Unsupervised models of images by spike-and-slab RBMs. In *Proceedings of theTwenty-eight International Conference on Machine Learning (ICML'11)*.

Cover, T. (2006). *Elements of Information Theory*. Wiley-Interscience.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems 2*, 303–314.

Dean, J., G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng (2012). Large scale distributed deep networks. In *NIPS'2012*.

Deng, L., M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton (2010). Binary coding of speech spectrograms using a deep auto-encoder. In *Interspeech 2010*, Makuhari, Chiba, Japan.

Desjardins, G., A. C. Courville, and Y. Bengio (2012). On training deep Boltzmann machines. *CoRR abs/1203.4416*.

Douglas, S., S.-I. Amari, and S.-Y. Kung (1999). On gradient adaptation with unit-norm constraints.

Drucker, H., C. J. Burges, L. Kaufman, C. J. C, B. L. Kaufman, A. Smola, and V. Vapnik (1996). Support vector regression machines.

Fukunaga, K. and L. Hostetler (1975, January). The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on 21*(1), 32–40.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 193–202.

Garrigues, P. and B. Olshausen (2008). Learning horizontal connections in a sparse coding model of natural images. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pp. 505–512. Cambridge, MA: MIT Press.

Girshick, R., J. Donahue, T. Darrell, and J. Malik (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. Technical report, arXiv:1311.2524.

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

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., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (2014). Multi-digit number recognition from Street View imagery using deep convolutional neural networks. In *International Conference on Learning Representations*.

Goodfellow, I. J., A. Courville, and Y. Bengio (2013, August). Scaling up spike-and-slab models for unsupervised feature learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*(8), 1902–1914.

Goodfellow, I. J., D. Erhan, P.-L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio (2013). Challenges in representation learning: A report on three machine learning contests.

Goodfellow, I. J., M. Mirza, A. Courville, and Y. Bengio (2013, December). Multi-prediction deep Boltzmann machines. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. Nips Foundation (http://books.nips.cc).

Goodfellow, I. J., D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien, and Y. Bengio (2013). Pylearn2: a machine learning research library. *arXiv preprint arXiv:1308.4214*.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. In S. Dasgupta and D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*, pp. 1319–1327. ACM.

Hahnloser, R. H. R. (1998). On the piecewise analysis of networks of linear threshold neurons. *Neural Networks 11*(4), 691–697.

Heckerman, D., D. M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research 1*, 49–75.

Hinton, G. E. (2000). Training products of experts by minimizing contrastive divergence. Technical Report GCNU TR 2000-004, Gatsby Unit, University College London.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation 14*, 1771–1800.

Hinton, G. E. (2010). A practical guide to training restricted Boltzmann machines. Technical Report UTML TR 2010-003, Department of Computer Science, University of Toronto.

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

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*, 359–366.

Hyvärinen, A., J. Hurri, and P. O. Hoyer (2009). *Natural Image Statistics: A probabilistic approach to early computational vision.* Springer-Verlag.

Hyvärinen, A., J. Karhunen, and E. Oja (2001). *Independent Component Analysis*. Wiley-Interscience.

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.

Jia, Y. and C. Huang (2011). Beyond spatial pyramids: Receptive field learning for pooled image features. *NIPS*2011 Workshop on Deep Learning and Unsupervised Feature Learning*.

Kavukcuoglu, K., P. Sermanet, Y.-L. Boureau, K. Gregor, M. Mathieu, and Y. LeCun (2010). Learning convolutional feature hierarchies for visual recognition. In *NIPS'10*.

Keerthi, S. S., D. Decoste, and T. Joachims (2005). A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research 6*, 2005.

Kindermann, R. (1980). *Markov Random Fields and Their Applications (Contemporary Mathematics ; V. 1)*. American Mathematical Society.

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Kolmogorov, A. (1953). *Unbiased Estimates:*. American Mathematical Society translations. American Mathematical Society.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

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)*.

Kullback, S. and R. A. Leibler (1951). On information and sufficiency. *Annals of Mathematical Statistics 22*, 49–86.

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, Q. V., M. Ranzato, R. Salakhutdinov, A. Ng, and J. Tenenbaum (2011). *NIPS Workshop on Challenges in Learning Hierarchical Models: Transfer Learning and Optimization*. https://sites.google.com/site/nips2011workshop.

Le Roux, N. and Y. Bengio (2008, June). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation 20*(6), 1631–1649.

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

LeCun, Y., F.-J. Huang, and L. Bottou (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'04)*, Volume 2, Los Alamitos, CA, USA, pp. 97–104. IEEE Computer Society.

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 (Qc), Canada: ACM.

Long, P. M. and R. A. Servedio (2010). Restricted Boltzmann machines are hard to approximately evaluate or simulate. In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*.

Lücke, J. and A.-S. Sheikh (2011). A closed-form EM algorithm for sparse coding. arXiv:1105.2493.

Malinowski, M. and M. Fritz (2013). Learnable pooling regions for image classification. In *International Conference on Learning Representations: Workshop track*.

Martens, J. (2010, June). Deep learning via Hessian-free optimization. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pp. 735–742. ACM.

McClelland, J. L., D. E. Rumelhart, and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 2. Cambridge: MIT Press.

Miao, Y., F. Metze, and S. Rawat (2013). Deep maxout networks for low-resource speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pp. 398–403. IEEE.

Mitchell, T. J. and J. J. Beauchamp (1988). Bayesian variable selection in linear regression. *J. Amer. Statistical Assoc. 83*(404), 1023–1032.

Mitchell, T. M. (1997). *Machine Learning*. New York: McGraw-Hill.

Mohamed, S., K. Heller, and Z. Ghahramani (2012). Bayesian and L1 approaches to sparse unsupervised learning. In *ICML'2012*.

Montavon, G. and K.-R. Müller (2012). Learning feature hierarchies with centered deep Boltzmann machines. *CoRR abs/1203.4416*.

Neal, R. and G. Hinton (1999). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in Graphical Models*. Cambridge, MA: MIT Press.

Neal, R. M. (2001, April). Annealed importance sampling. *Statistics and Computing 11*(2), 125–139.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). Reading digits in natural images with unsupervised feature learning. Deep Learning and Unsupervised Feature Learning Workshop, NIPS.

Newey, W. and D. McFadden (1994). Large sample estimation and hypothesis testing. *Handbook of Econometrics 4*, 2111–2245.

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

Olshausen, B. A. and D. J. Field (1997, December). Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Research 37*, 3311–3325.

Pearl, J. (1985, August). Bayesian networks: A model of self-activated memory for evidential reasoning. In *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine*, pp. 329–334.

Pearlmutter, B. (1994). Fast exact multiplication by the Hessian. *Neural Computation 6*(1), 147–160.

Raina, R., A. Battle, H. Lee, B. Packer, and A. Y. Ng (2007). Self-taught learning: transfer learning from unlabeled data. In Z. Ghahramani (Ed.), *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pp. 759–766. ACM.

Ranzato, M. and G. H. Hinton (2010). Modeling pixel means and covariances using factorized third-order Boltzmann machines. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'10)*, pp. 2551–2558. IEEE Press.

Rao, C. R. (1973). *Linear Statistical Inference and its Applications* (2nd ed.). New York: J. Wiley and Sons.

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)*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986a). Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing*, Volume 1, Chapter 8, pp. 318–362. Cambridge: MIT Press.

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

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

Salinas, E. and L. F. Abbott (1996, October). A model of multiplicative neural responses in parietal cortex. *Proc Natl Acad Sci U S A 93*(21), 11956–11961.

Saul, L. K. and M. I. Jordan (1996). Exploiting tractable substructures in intractable networks. In D. Touretzky, M. Mozer, and M. Hasselmo (Eds.), *Advances in Neural Information Processing Systems 8 (NIPS'95)*. MIT Press, Cambridge, MA.

Schraudolph, N. N. (2002). Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation 14*(7), 1723–1738.

Sermanet, P., S. Chintala, and Y. LeCun (2012). Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*.

Smirnov, E. (2013). North atlantic right whale call detection with convolutional neural networks.

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing*, Volume 1, Chapter 6, pp. 194–281. Cambridge: MIT Press.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*.

Srebro, N. and A. Shraibman (2005). Rank, trace-norm and max-norm. In *Proceedings of the 18th Annual Conference on Learning Theory*, pp. 545–560. Springer-Verlag.

Srivastava, N. (2013). Improving neural networks with dropout. Master's thesis, U. Toronto.

Steinhaus, H. (1957). Sur la division des corps matériels en parties. In *Bull. Acad. Polon. Sci.*, pp. 801–804.

Stinchcombe, M. and H. White (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation function. In *International Joint Conference on Neural Networks (IJCNN)*, Washington DC, pp. 613–617. IEEE.

Stoyanov, V., A. Ropson, and J. Eisner (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *AISTATS'2011*.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. In *ICML*.

Szegedy, C., A. Toshev, and D. Erhan (2013). Deep neural networks for object detection. In *NIPS'2013*.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 1064–1071. ACM.

Titsias, M. K. and M. Lázaro-Gredilla (2011). Spike and slab variational inference for multi-task and multiple kernel learning. In *NIPS'2011*.

Titterington, D., A. Smith, and U. Makov (1985). *Statistical Analysis of Finite Mixture Distributions*. Wiley, New York.

Uria, B., I. Murray, and H. Larochelle (2013). A deep and tractable density estimator. Technical Report arXiv:1310.1757.

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

Vapnik, V. N. and A. Y. Chervonenkis (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications 16*, 264–280.

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In *ICML 2008*.

Wang, S. (2004). General constructive representations for continuous piecewise-linear functions. *IEEE Trans. Circuits Systems 51*(9), 1889–1896.

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

Warde-Farley, D., I. J. Goodfellow, P. Lamblin, G. Desjardins, F. Bastien, and Y. Bengio (2011). pylearn2. http://deeplearning.net/software/pylearn2.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Ph. D. thesis, Harvard University.

Xie, J., B. Xu, and Z. Chuang (2013). Horizontal and vertical ensemble with deep representation for classification. Workshop on Challenges in Representation Learning, ICML.

Younes, L. (1998). On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates. In *Stochastics and Stochastics Models*, pp. 177–228.

Yu, D. and L. Deng (2011). Deep convex net: A scalable architecture for speech pattern classification. In *INTERSPEECH*, pp. 2285–2288.

Yu, K., Y. Lin, and J. Lafferty (2011). Learning image representations from the pixel level via hierarchical sparse coding. In *CVPR'2011*.

Zeiler, M., G. Taylor, and R. Fergus (2011). Adaptive deconvolutional networks for mid and high level feature learning. In *ICML*.

Zeiler, M. D. and R. Fergus (2013a). Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations*.

Zeiler, M. D. and R. Fergus (2013b). Visualizing and understanding convolutional networks. Technical Report Arxiv 1311.2901.

Zhou, M., H. Chen, J. W. Paisley, L. Ren, G. Sapiro, and L. Carin (2009). Non-parametric Bayesian dictionary learning for sparse image representations. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 2295–2303.