Deep Learning Models for Unsupervised and Transfer Learning

by

Nitish Srivastava

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Deep Learning Models for Unsupervised and Transfer Learning

Nitish Srivastava
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2017

This thesis is a compilation of five research contributions whose goal is to do unsupervised and transfer learning by designing models that learn distributed representations using deep neural networks. First, we describe a Deep Boltzmann Machine model applied to image-text and audio-video multi-modal data. We show that the learned generative probabilistic model can jointly model both modalities and also produce good conditional distributions on each modality given the other. We use this model to infer fused high-level representations and evaluate them using retrieval and classification tasks.

Second, we propose a Boltzmann Machine based topic model for modeling bag-of-words documents. This model augments the Replicated Softmax Model with a second hidden layer of latent words without sacrificing RBM-like inference and training. We describe how this can be viewed as a beneficial modification of the otherwise rigid, complementary prior that is implicit in RBM-like models.

Third, we describe an RNN-based encoder-decoder model that learns to represent video sequences. This model is inspired by sequence-to-sequence learning for machine translation. We train an RNN encoder to come up with a representation of the input sequence that can be used to both decode the input back, and predict the future sequence. This representation is evaluated using action recognition benchmarks.

Fourth, we develop a theory of directional units and use them to construct Boltzmann Machines and Autoencoders. A directional unit is a structured, vector-valued hidden unit which represents a continuous space of features. The magnitude and direction of a directional unit represent the strength and pose of a feature within this space, respectively. Networks of these units can potentially do better coincidence detection and learn general equivariance classes. Temporal coherence based learning can be used with these units to factor out the dynamic properties of a feature, part, or object from static properties such as identity.

Last, we describe a contribution to transfer learning. We show how a deep convolutional net trained to classify among a given set of categories can transfer its knowledge to new categories even when very few labelled examples are available for the new categories.

Acknowledgements

The time I spent working on these ideas has been one of the most rewarding periods of my life. I will forever be grateful to my advisors Geoff Hinton and Russ Salakhutdinov who are not only brilliant researchers but amazing human beings. Geoff will always remain a source of inspiration. His enthusiasm, wit, and humour along with the sense of playfulness with which he approaches problems and builds insights make him an ideal advisor to have. I would also like to thank Russ for being a constant research mentor. I learned a lot from his profound understanding of research problems and am very grateful for his encouragement.

I am deeply thankful to Relu Patrascu for keeping the GPUs running smoothly without which doing this research would have been inconceivable. I had the pleasure of sharing office space with George Dahl, Navdeep Jaitly, Vlad Mnih, Shenlong Wang, Jimmy Ba, Wenjie Luo, and Tony Wu. The lively discussions around the office have been a source of ideas and amusement alike. I would especially like to thank George Dahl for his valued mentorship (both as office mate and roommate) and Navdeep and his amazing family for their warmth. I am also grateful that I had the opportunity to spend time with brilliant colleagues : Abdel-Rahman Mohammed, Alex Graves, Alex Krizhevsky, Alex Schwing, Bin Yang, Charlie Tang, Chris Maddison, Danny Tarlow, Eleni Triantafillou, Gellert Mattyus, Hang Chu, Ilya Sutskever, Ivan Vendrov, James Martens, Jamie Kiros, Jasper Snoek, Jian Yao, Justin Liang, Kaustav Kundu, Kevin Swersky, Laurent Charlin, Lisa Zhang, Lluis Castrejon, Mohammad Norouzi, Maks Volkovs, Min Bai, Namdar Homayounfar, Renjie Liao, Roger Grosse, Shikhar Sharma, Tijmen Tieleman, Yukun Zhu, Yujia Li, Ziyu Zhang, and several visiting students who made the group a stimulating environment for doing research.

I would especially like to thank Vlad Mnih and Alex Krizhevsky for their excellent code libraries (cudamat and cuda-convnet), Navdeep Jaitly for teaching me how to decode speech, and Kevin Swersky for teaching me how to ski.

I would also like to thank my committee members, Raquel Urtasun, Richard Zemel, Rob Fergus, and Roger Grosse for valuable feedback during this journey.

Most of all, I would like to thank my parents for fostering in me the love of learning and discovery, encouraging me to follow my heart, and giving me every possible privilege that a child growing up can hope to have.

Contents

Chapter 1

Introduction

One of the primary goals in the field of Computer Science is to design algorithms to solve problems computationally. While human ingenuity plays a key role in designing algorithms, there are some problems whose solutions are extremely difficult to articulate using human ingenuity alone. These problems often belong to the "AI-set", which is the set of problems that systems claiming to be intelligent ought to be able to solve. This includes problems such as finding and recognizing objects in an image, recognizing human speech, understanding natural language, and being able to move, open doors and manipulate objects. Solving these problems seems to require finding patterns in input data but specifying exactly which patterns to look for and determining how to represent information internally has been found to be an extremely onerous and cumbersome task if done by hand.

Machine learning is an approach for designing algorithms in which gathering and using experience plays a central role. This experience could be in the form of examples of correct (or incorrect) input-output behaviour, simply observing the inputs, or a more active interaction of an agent and its environment. A key premise of this approach is that statistical modeling of experience (also called data) is a scalable way of discovering the patterns and representations needed for solving AI problems. The aim is to discover models, or at least principles for model design, that can generalize to solving a wide variety of problems. Therefore, in the machine learning approach, human ingenuity is detached one step from the problem, in that it is directed towards figuring out which statistical models and what assumptions should be used to understand data, rather than being directed towards solving individual problems themselves.

Over the last 5 years, one particular design principle has been found to be extremely successful in solving a wide variety of problems. This principle is that models should learn multiple levels of distributed representations. A common embodiment of this principle is the class of models called deep neural networks which consist of interconnected neuronal units arranged in a deep hierarchy. By tuning the strengths of the connections between these units, a deep neural network can represent different functions. In fact it can represent any function arbitrarily well, given an extremely large number of units. This universal approximation property is shared by other approaches as well. However, what makes these models interesting is that many useful functions that relate to solving AI problems can be represented efficiently using relatively few neurons if they are arranged in a deep computation graph. This key insight, inspired by biological neural networks and validated on numerous real-world AI problems, has lead to an explosion of interest in such models and created the field of deep learning. Aided by

Moore's law, which has provided ever larger amounts of data and fast computation, deep learning has become the dominant approach for solving a large variety of problems in the AI-set, as evidenced by a growing list of real-world large scale deployments of these models as well as exceptional results on public benchmarks, for example, Speech Recognition (Hinton et al., 2012a), Machine Translation (Wu et al., 2016), Object Recognition and Detection (Krizhevsky et al., 2012; Szegedy et al., 2014; He et al., 2016), and Human-level Game playing (Mnih et al., 2015). This thesis contributes some deep learning models for unsupervised and transfer learning. The remainder of this chapter describes the motivation for each of these, and summarizes the contributions made in this thesis.

1.1 Unsupervised Learning

Unsupervised learning deals with the problem of finding structure in unlabeled data. Over the past few years, supervised deep learning models have achieved unprecedented success in solving hard prediction problems. In the process of doing so, these models discover distributed representations that often generalize well to data sets and tasks other than the ones they were trained on (Oquab et al., 2014; Razavian et al., 2014). However, this approach might be too constraining when it comes to learning general purpose representations that can enable a wide variety of tasks and adapt quickly to new problems without depending on human supervision. Moreover, it seems rather unnatural that we should require thousands of examples of arduously labelled data in order to understand the structure in domains such as the visual world, in which the structure seems so readily apparent. If our goal is to develop a biologically plausible representation learning scheme, then supervised learning is clearly insufficient. Therefore there are strong arguments for learning structure in data in an unsupervised way.

One of the earliest works that can be considered deep unsupervised learning is the Sigmoid Belief Network (SBN) (Neal, 1992). This paper showed how directed densely-connected graphical models can be trained using Gibbs Sampling similar to that used for Boltzmann Machines. Deep Belief Networks (Hinton et al., 2006; Hinton and Salakhutdinov, 2006), which marked the resurgence of deep learning, were designed to ameliorate the "explaining-away" problem in directed graphical models like SBNs by using complementary priors. Naturally identifiable clusters were shown to emerge in the representations learned by DBNs on multiple data domains. Vincent et al. (2008, 2010) developed a class of models called Stacked Denoising Autoencoders which also learn distributed representations of data using deep networks. Unlike DBNs, these models do not provide an explicit parameterization for the probability density that they represent but provide fast feed-forward inference of latent representations. Salakhutdinov and Hinton (2009a) proposed an efficient training algorithm for Deep Boltzmann Machines (DBMs), which are fully undirected graphical models. While DBMs have several nice properties, they have typically been applied on modestly complex datasets such as handwritten digits, faces or small images.

Chapter 2, based on the work done in Srivastava and Salakhutdinov (2012, 2014), describes a DBM model designed for multi-modal data, i.e., data consisting of different modalities such as images and text, or audio and video. The aim of this work is to test whether DBMs and algorithms for training them based on MCMC sampling can be applied to a challenging density learning problem that requires learning latent representations that can serve as a bridge across two very different modalities. We show that the DBM model is able to jointly explain both modalities and produce good conditional distributions on one given the other. This work also makes a detailed comparison to DBNs and autoencoders. Working with a challenging task, we were able to show the importance of having bottom-up and top-down information

while inferring latent representations. The inference process in DBMs, while not as straightforward as autoencoders, has several nice properties – the process of settling down to a state by repeated application of a computation graph gives the model the ability to repeatedly query its memory; sampling gives it the ability to conjecture hypotheses; and ultimately the model finds a state that fits with what it has seen before. However, training these models is hard, and becomes harder as the model gets sharper because the learning is based on Markov chain Monte-Carlo methods and the Markov chains get harder to mix.

Chapter 3, based on the work done in Srivastava et al. (2013), describes the Over-Replicated Softmax Model which is a Boltzmann Machine based topic model for modeling bag-of-words documents. The key contribution here is developing a way of modifying the rigid implicit complementary prior in RBM-like models. We extend the Replicated Softmax model (Salakhutdinov and Hinton, 2009b) by adding a second hidden layer of latent words and show that this leads to better performance with a minimal increase in training complexity.

Deep unsupervised models have also been applied to sequence learning problems. Generative probabilistic models include RTRBM (Sutskever et al., 2009) and RNN-RBM (Boulanger-Lewandowski et al., 2012). Feed-forward models such as RNN-based Sequence-to-Sequence learning models (Sutskever et al., 2014; Cho et al., 2014) have been proposed in the context of machine translation.

Chapter 4, based on the work done in Srivastava et al. (2015), describes a model that learns video representations using an RNN-based Sequence-to-Sequence framework. The key contribution here is combining two sources of training signal – predicting the future and autoencoding. This is done using two decoder RNNs, one each for future prediction and autoencoding. The model reads in a video sequence using an encoder RNN and then each decoder RNN uses the same representation (as provided by the encoder RNN) to try to predict its respective target sequence.

Chapter 5 describes unsupervised learning models constructed using Directional Units. These are vector-valued hidden units, unlike the units typically used in deep unsupervised models which are scalar-valued. Each directional unit corresponds to a space of features, as opposed to typical hidden units which correspond to a single feature. Its vector-valued activation can be decomposed as the product of a direction and a magnitude. The direction encodes which feature is present, while the magnitude encodes how strongly it is present. There are several motivations for studying these units. By associating a pose with each feature, these units can potentially make it easier for a model to look for coincidences in terms of agreement of pose. Small changes in the pose of the input can produce an equi-variant change in the pose of these feature. Therefore, networks of these units can potentially provide a general equivariance learning framework. While typical deep networks assume an instantaneous and synchronous model of computation, networks can use the phase information in directional units to model timing. Neuron firings that occur in-phase add up to produce a stronger effect than firings that happen out-of-phase. Analogously, a directional unit receiving inputs which are aligned in the same direction can produce a strong output. Extending previous work on 2-D directional units (Noest, 1987; Zemel et al., 1992, 1995b), we describe a Boltzmann machine model made of generalized N-dimensional directional units. We also propose an autoencoder model using the conditional distribution over directional units derived in the Boltzmann Machine model as a deterministic activation function. Another key contribution is temporal coherence based learning of directional unit autoencoders in which we try to maintain slowness in the magnitude of features across frames but allow the pose to model the dynamic properties of those features.

We would like to point out that there are several other recently proposed deep unsupervised learning

approaches that are not explored or discussed in this thesis. These include variational autoencoders (Kingma and Welling, 2013; Rezende et al., 2014), and their extensions such as the DRAW model (Gregor et al., 2015), Generative Adversarial Networks (Goodfellow et al., 2014) and their extensions such as LAPGAN (Denton et al., 2015), DCGAN (Radford et al., 2015), models that create their outputs one dimension at a time, such as the Pixel CNN (Oord et al., 2016b), WaveNet (Oord et al., 2016a) and Pixel Video Networks (Kalchbrenner et al., 2016). Overall, the trend in the field is towards adopting models where training and inference can be done using forward and backward passes (as opposed to more expensive MCMC based learning/inference).

1.2 Transfer Learning

Transfer learning deals with the problem of applying the knowledge extracted in the process of learning to solve one or more tasks to solve a different but related task. While unsupervised learning tries to find structure in the space of data, transfer learning tries to find structure in the space of tasks. The ability to transfer knowledge to related tasks is an important goal towards enabling strong AI, or Artificial General Intelligence (AGI), since it gives an agent the ability to adapt to a continuum of problems that it is likely to encounter.

Chapter 6, based on Srivastava and Salakhutdinov (2013), focuses on transfer learning. Here the goal is to extract knowledge gained in solving a classification task over certain categories and transfer it to new categories for which very few labelled examples are available. The proposed model learns a tree-structure over the space of categories. This structure is used to induce a prior on the weights in the last layer of a convolutional net which makes it easy to add new categories and start getting reasonable results on them with very few labelled examples.

Chapter 2

Modeling Multimodal Data with Deep Boltzmann Machines

The world around us can be perceived and represented in many different ways. For example, the task of representing a physical object can be accomplished by taking its photograph, describing it in words, scanning it with a laser or ultrasonic device, or by measuring its gravitational field. These modalities capture complementary as well as shared properties of the object. Together, this multimodal data offers a more complete and rich description of the object compared to any single modality. Other examples of multimodal data include images associated with captions and tags, videos containing visual and audio signals, and sensory perception in robotic applications which contains simultaneous inputs from visual, auditory, motor and haptic pathways. Real-world systems that rely on sensing their environments are often built with multiple and diverse sensors for redundancy, as well as to provide complementary information. Given the ubiquity and practicality of multimodal data, it is imperative to design unsupervised learning models that can work with and exploit the structure in these kinds of data.

Before we describe our model in detail, it is useful to highlight some other motivations for doing unsupervised learning with multimodal data. Different modalities can act like soft labels for each other. For example, consider bi-modal image-text data. If the same uncommon word was used in the context of several images, then there is some chance that all those images represent the same object. Conversely, if different sentences are used to describe similar looking images, then those sentences might have similar meaning. In other words, one modality might be somewhat invariant to large changes in another modality. This provides a rich learning signal that can help determine which properties should be disentangled. Moreover, the complementary information present across modalities can serve as a means to explore the world and learn new things in an unsupervised way. For example, people often caption an image to say things that may not be obvious from the image itself, such as the name of the person, place, or a particular object in the picture. By learning to associate these complementary pieces of information, a model can discover relationships. This seems like a scalable, or at least an elegant way to extract knowledge from unlabeled data.

2.1 Overview

In principle, any general-purpose unsupervised learning technique can simply operate on the concatenation of multiple modalities without bothering to deal with each input channel separately. However, each modality is often characterized by very distinct statistical properties which makes it difficult (or at least inefficient) to disregard the fact that it comes from a different input channel. Having very different statistical properties makes it much harder to discover relationships across modalities than relationships among features in the same modality. There is a lot of structure in the data but it is difficult to discover the highly non-linear relationships that exist between low-level features across different modalities. Therefore, it might be prudent to design models that try to bridge this "modality gap". The ability of deep neural networks to learn successively higher level abstract distributed representations makes them a powerful tool in this regard. In this chapter, we develop a Deep Boltzmann Machine (DBM) for multimodal data and compare it with other deep models such as deep autoencoders (DAEs) and deep belief networks (DBNs), as well as with other baselines including Multiple Kernel Learning.

Models of multimodal data can be expected to solve different kinds of problems. For a lot of applications, it is desirable to fuse multiple modalities to create a joint representation that captures the underlying concept. For example, in autonomous driving applications it might be useful to identify the concept of a vehicle or pedestrian using the sensory input as a whole. This fused and unified representation can then be communicated to the rest of the system. It is important that the representation be robust to noise and failures in individual modalities.

Other applications, such as information retrieval with cross modal queries, require remembering one modality given the others. For instance, using words or sketches to retrieve photographs. We would like a model to correlate the occurrence of the words *sunny beach* and the visual properties of an image of a sunny beach and represent them jointly, so that one is reminiscent of the other (in other words, the model assigns high probability to one conditioned on the other). It is important to note that since one modality, in general, does not completely determine the other, the conditional distribution is expected to have multiple modes.

Our proposed multimodal Deep Boltzmann Machine (DBM) model offers a unified solution to all of these problems. DBMs (Salakhutdinov and Hinton, 2009a) are undirected graphical models with bipartite connections between adjacent layers of hidden units. The key idea is to learn a joint probability density over the space of multimodal inputs. Different problems can then be formulated as conditional inference under this probability density. For example, inferring the states of the hidden units, conditioned on the data, gives us a joint representation of the multimodal data. Inferring the states of a missing data modality conditioned on the observed ones is a way to solve cross modal retrieval. For example, later in this chapter, we use a large collection of user-tagged images to learn a joint distribution over images and text $P(\mathbf{v}_{img}, \mathbf{v}_{txt}; \theta)$ where θ are the model parameters. By drawing samples from $P(\mathbf{v}_{txt}|\mathbf{v}_{img}; \theta)$ and from $P(\mathbf{v}_{img}|\mathbf{v}_{txt}; \theta)$ we can fill-in missing data, thereby doing image annotation and image retrieval respectively. Some examples of these samples are shown in Fig. 2.1.

There have been several approaches to learning from multimodal data. In particular, Huiskes et al. (2010) showed that using captions or tags, in addition to standard low-level image features significantly improves classification accuracy when using models such as Support Vector Machines (SVM) and Linear Discriminant Analysis (LDA). A similar approach of Guillaumin et al. (2010) based on the multiple kernel learning framework further demonstrated that an additional text modality can improve the accuracy of SVMs on various object recognition tasks. However, all of these approaches are discriminative by nature

Image	Given Tags	Generated Tags	Input Tags	Nearest neighbors to generated image features	
	pentax, k10d, kangarooisland, southaustralia, sa, australia, aus-traliansealion, 300mm	beach, sea, surf, strand, shore, wave, seascape, sand, ocean, waves	nature, hill, scenery, green, clouds		
	< no text >	night, lights, christmas, nightshot, nacht, nuit, notte, longexposure, noche, nocturna	flower, nature, green, flowers, petal, petals, bud		
	aheram, 0505, sarahc, moo	portrait, bw, balckandwhite, people, faces, girl, blackwhite, person, man	blue, red, art, artwork, painted, paint, artistic, surreal, gallery, bleu		
	unseulpixel, naturey crap	fall, autumn, trees, leaves, foliage, forest, woods, branches, path	bw, blackand-white, noiretblanc, bianconero, blancoynegro		

Figure 2.1: **Left:** Examples of text generated from a Deep Boltzmann Machine by sampling from $P(\mathbf{v}_{txt}|\mathbf{v}_{img};\theta)$. **Right:** Examples of images retrieved using features generated from a Deep Boltzmann Machine by sampling from $P(\mathbf{v}_{img}|\mathbf{v}_{txt};\theta)$.

and cannot make use of large amounts of unlabelled data or deal easily with missing input modalities.

On the generative side, Xing et al. (2005) used dual-wing harmoniums to build a joint model of images and text, which can be viewed as a linear Restricted Boltzmann Machine (RBM) model with Gaussian hidden units together with Gaussian and Poisson visible units. However, various data modalities will typically have very different statistical properties which makes it difficult to model them using shallow models. Most similar to our work is the recent approach of Ngiam et al. (2011) that used a deep autoencoder for speech and vision fusion. There are, however, several crucial differences. First, in this work we focus on jointly modelling very different data modalities: sparse word count vectors and real-valued dense image features. Second, we use a Deep Boltzmann Machine which is a probabilistic generative model as opposed to a feed-forward autoencoder. While both approaches have led to interesting results in several domains, using a generative model is important for applications we consider in this chapter, as it allows our model to naturally handle missing and noisy data modalities.

This chapter is organized as follows. We first give an overview of RBMs and their adaptations to handle different kinds of input data. We then describe DBMs. Next we describe multimodal DBMs and give details of their training and inference procedures. The next section describes how this model can be adapted to solve different tasks of interest. The next two sections describe experimental results on image-text and audio-video data respectively. We end with conclusions.

2.2 Background: RBMs and Their Generalizations

Restricted Boltzmann Machines (RBMs) have been used effectively in modeling distributions over binary-valued data. Boltzmann machine models and their generalizations to exponential family distributions (Welling et al., 2005) have been successfully used in many application domains. For example, the Replicated Softmax model (Salakhutdinov and Hinton, 2009b) has been shown to be effective in modeling sparse word count vectors, whereas Gaussian RBMs have been used for modeling real-valued inputs for image classification, video action recognition, and speech recognition (Lee et al., 2009a; Taylor et al., 2010; Mohamed et al., 2011). In this section we briefly review these models, as they will serve as building blocks for our multimodal model.

2.2.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (Smolensky, 1986) is an undirected graphical model with stochastic visible variables $\mathbf{v} \in \{0,1\}^D$ and stochastic hidden variables $\mathbf{h} \in \{0,1\}^F$, with each visible variable connected to each hidden variable. The model defines the following energy function $E : \{0,1\}^D \times \{0,1\}^F \rightarrow \mathbb{R}$:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} v_i h_j - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F} a_j h_j, \tag{2.1}$$

where $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}\}$ are the model parameters: W_{ij} represents the symmetric interaction term between visible unit i and hidden unit j; b_i and a_j are bias terms. The joint distribution over the visible and hidden units is defined by:

$$P(\mathbf{v}, \mathbf{h}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \quad \mathcal{Z}(\theta) = \sum_{\mathbf{v}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \tag{2.2}$$

where $\mathcal{Z}(\theta)$ is the normalizing constant. The conditional distributions over hidden $\mathbf{h}$ and visible $\mathbf{v}$ vectors factorize and can be easily derived from Eqs. 2.1, 2.2:

$$\begin{aligned} P(\mathbf{h}|\mathbf{v}; \theta) &= \prod_{j=1}^{F} p(h_j|\mathbf{v}), \quad \text{with} \quad p(h_j = 1|\mathbf{v}) = g\left(\sum_{i=1}^{D} W_{ij} v_i + a_j\right), \\ P(\mathbf{v}|\mathbf{h}; \theta) &= \prod_{i=1}^{D} p(v_i|\mathbf{h}), \quad \text{with} \quad p(v_i = 1|\mathbf{h}) = g\left(\sum_{j=1}^{F} W_{ij} h_j + b_i\right), \end{aligned}$$

where $g(x) = 1/(1+\exp(-x))$ is the logistic function. Given a set of observations $\{\mathbf{v}_n\}_{n=1}^N$, the derivative of the log-likelihood with respect to the model parameters can be obtained from Eq. 2.2:

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \log P(\mathbf{v_n}; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{data}}}\left[v_i h_j\right] - \mathbb{E}_{P_{\text{model}}}\left[v_i h_j\right],$$

where $\mathbb{E}_{P_{\text{data}}}[\cdot]$ denotes an expectation with respect to the data distribution $P_{\text{data}}(\mathbf{h}, \mathbf{v}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{\text{data}}(\mathbf{v})$, with $P_{\text{data}}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v} - \mathbf{v}_n)$ representing the empirical distribution, and $\mathbb{E}_{P_{\text{model}}}[\cdot]$

is an expectation with respect to the distribution defined by the model, as in Eq. 2.2. We will sometimes refer to $\mathbb{E}_{P_{\text{data}}}[\cdot]$ as the *data-dependent expectation*, and $\mathbb{E}_{P_{\text{model}}}[\cdot]$ as the *model's expectation*.

2.2.2 Gaussian-Bernoulli RBM

RBMs were originally developed for modeling binary vectors. Gaussian-Bernoulli RBMs (Freund and Haussler, 1994; Hinton and Salakhutdinov, 2006) are a variant that can be used for modeling real-valued vectors such as pixel intensities and filter responses. Consider modeling visible real-valued units $\mathbf{v} \in \mathbb{R}^D$, and let $\mathbf{h} \in \{0,1\}^F$ be binary stochastic hidden units. The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ of the Gaussian-Bernoulli RBM is defined as follows:

$$E(\mathbf{v}, \mathbf{h}; \theta) = \sum_{i=1}^{D} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{D}\sum_{j=1}^{F} \frac{v_i}{\sigma_i} W_{ij} h_j - \sum_{j=1}^{F} a_j h_j,$$

where $\theta = \{\mathbf{a}, \mathbf{b}, \mathbf{W}, \boldsymbol{\sigma}\}$ are the model parameters. The density that the model assigns to a visible vector $\mathbf{v}$ is given by:

$$P(\mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right), \quad \mathcal{Z}(\theta) = \int_{\mathbf{v}} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}; \theta)\right) d\mathbf{v}.$$

Similar to the standard RBMs, the conditional distributions factorize:

$$\begin{aligned} P(\mathbf{h}|\mathbf{v}; \theta) &= \prod_{j=1}^{F} p(h_j|\mathbf{v}), \quad \text{with} \quad p(h_j = 1|\mathbf{v}) = g\left(\sum_{i=1}^{D} W_{ij} \frac{v_i}{\sigma_i} + a_j\right), \\ P(\mathbf{v}|\mathbf{h}; \theta) &= \prod_{i=1}^{D} p(v_i|\mathbf{h}), \quad \text{with} \quad v_i|\mathbf{h} \sim \mathcal{N}\left(b_i + \sigma_i \sum_{j=1}^{F} W_{ij} h_j, \sigma_i^2\right), \end{aligned} \tag{2.3}$$

where $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with mean μ and variance σ^2. Observe that conditioned on the states of the hidden units (Eq. 2.3), each visible unit is modeled by a Gaussian distribution, whose mean is shifted by the weighted combination of the hidden unit activations.

Given a set of observations $\{\mathbf{v}_n\}_{n=1}^N$, the derivative of the log-likelihood with respect to the model parameters takes a very similar form when compared to binary RBMs:

$$\frac{1}{N} \sum_{n=1}^{N} \frac{\partial \log P(\mathbf{v_n}; \theta)}{\partial W_{ij}} = \mathbb{E}_{P_{\text{Data}}}\left[\frac{v_i}{\sigma_i} h_j\right] - \mathbb{E}_{P_{\text{model}}}\left[\frac{v_i}{\sigma_i} h_j\right].$$

2.2.3 Replicated Softmax Model

The Replicated Softmax Model is useful for modeling sparse count data, such as word count vectors in a document (Salakhutdinov and Hinton, 2009b). This model is a type of Restricted Boltzmann Machine in which the visible variables, that are typically binary, have been replaced by multinomial variables that can take on one of a number of different states. Each state corresponds of a different word in the dictionary. Specifically, let K be the dictionary size, M be the number of words appearing in a document, and $\mathbf{h} \in \{0,1\}^F$ be binary stochastic hidden topic features. Let $\mathbf{V}$ be a $M \times K$ observed binary matrix with $v_{ik} = 1$ iff the multinomial visible unit i takes on k^{th} value (meaning the i^{th} word in

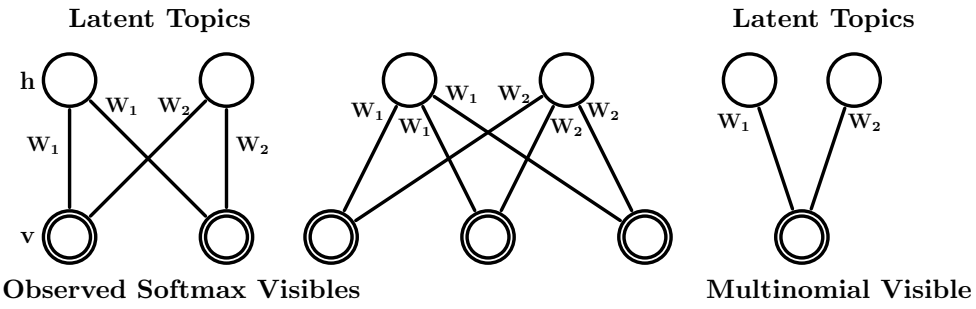


Figure 2.2: Replicated Softmax model. The top layer represents a vector $\mathbf{h}$ of stochastic, binary topic features and and the bottom layer represents softmax visible units $\mathbf{v}$. All visible units share the same set of weights, connecting them to binary hidden units. **Left:** The model for a document containing two and three words. **Right:** A different interpretation of the Replicated Softmax model, in which M softmax units with identical weights are replaced by a single multinomial unit which is sampled M times.

the document is the k^{th} dictionary word). The energy of the state $\{\mathbf{V}, \mathbf{h}\}$ can be defined as follows:

$$E(\mathbf{V}, \mathbf{h}) \ = \ -\sum_{i=1}^{M}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ijk} v_{ik} h_j - \sum_{i=1}^{M}\sum_{k=1}^{K} b_{ik} v_{ik} - \sum_{j=1}^{F} a_j h_j,$$

where $\{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$ are the model parameters: W_{ijk} is a symmetric interaction term between visible unit i that takes on value k, and hidden feature j, b_{ik} is the bias of unit i that takes on value k, and a_j is the bias of hidden feature j. The probability that the model assigns to a visible binary matrix $\mathbf{V}$ is:

$$P(\mathbf{V}, \mathbf{h}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(-E(\mathbf{V}, \mathbf{h}; \theta)\right), \quad \mathcal{Z}(\theta) = \sum_{\mathbf{V}} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{V}, \mathbf{h}; \theta)\right).$$

This defines an RBM for documents of size M. The Replicated Softmax model is a collection of such RBMs that cover all possible document sizes. All these RBMs share the same model parameters. Fig. 3.1 shows examples of RBMs for $N = 2$ and $N = 3$. Assuming that the order of the words can be ignored, all of these softmax units can share the same set of weights, connecting them to binary hidden units. In this case, the energy of the state $\{\mathbf{V}, \mathbf{h}\}$ for a document that contains M words is defined as:

$$E(\mathbf{V}, \mathbf{h}) \ = \ -\sum_{j=1}^{F}\sum_{k=1}^{K} W_{jk} \hat{v}_k h_j - \sum_{k=1}^{K} b_k \hat{v}_k - M \sum_{j=1}^{F} a_j h_j,$$

where $\hat{v}_k = \sum_{i=1}^{M} v_{ik}$ denotes the count for the k^{th} word. Observe that the bias terms of the hidden variables are scaled up by the length of the document. This scaling is important as it allows hidden units to behave sensibly when dealing with documents of different lengths. In the absence of bias scaling, the scale of the weights would get adjusted to work optimally for a typical document length. Documents longer than this would tend to saturate the units and shorter than this would lead to vague activations of the hidden units. The corresponding conditional distributions are given by:

Input	Reconstruction
chocolate, cake	cake, chocolate, sweets, dessert, cupcake, food, sugar, cream, birthday
nyc	nyc, newyork, brooklyn, queens, gothamist, manhattan, subway, streetart
dog	dog, puppy, perro, dogs, pet, filmshots, tongue, pets, nose, animal
flower, high, 花	flower, 花, high, japan, sakura, 日本, blossom, tokyo, lily, cherry
girl, rain, station, norway	norway, station, rain, girl, oslo, train, umbrella, wet, railway, weather
fun, life, children	children, fun, life, kids, child, playing, boys, kid, play, love
forest, blur	forest, blur, woods, motion, trees, movement, path, trail, green, focus
españa, agua, granada	españa, agua, spain, granada, water, andalucía, naturaleza, galicia, nieve

Table 2.1: Some examples of one-step reconstruction from the Replicated Softmax Model.

$$p(h_j = 1|\mathbf{V}) \;=\; g\left(Ma_j + \sum_{k=1}^{K} \hat{v}_k W_{jk}\right), \tag{2.4}$$

$$p(v_{ik} = 1|\mathbf{h}) \;=\; \frac{\exp\left(b_k + \sum_{j=1}^{F} h_j W_{jk}\right)}{\sum_{q=1}^{K} \exp\left(b_q + \sum_{j=1}^{F} h_j W_{jq}\right)}. \tag{2.5}$$

The learning algorithm for the Replicated Softmax model is similar to that for binary RBMs. Given a collection of N documents $\{\mathbf{V}_n\}_{n=1}^{N}$, the derivative of the log-likelihood with respect to parameters W takes the form:

$$\frac{1}{N}\sum_{n=1}^{N} \frac{\partial \log P(\mathbf{V_n})}{\partial W_{jk}} \;=\; \mathbb{E}_{P_{\text{data}}}\left[\hat{v}_k h_j\right] - \mathbb{E}_{P_{\text{model}}}\left[\hat{v}_k h_j\right].$$

The Replicated Softmax model can also be interpreted as an RBM model that uses a single visible multinomial unit with support $\{1, ..., K\}$ which is sampled M times (see Fig. 3.1, right panel).

For all of the above models, exact maximum likelihood learning is intractable because exact computation of the expectation $\mathbb{E}_{P_{\text{model}}}[\cdot]$ takes time that is exponential in $\min\{D, F\}$, i.e the number of visible or hidden units. In practice, efficient learning is performed by following an approximation to the gradient of a different objective function, called the "Contrastive Divergence" (CD) (Hinton, 2002).

One way to illustrate the working of the model is to look at one-step reconstructions of some bags of words. Table. 2.1 shows some examples. The words in the left column were given as input to the model. Then Eq. 2.4 was used to compute a distribution over hidden units. Using these probabilities as states of the units, Eq. 2.5 was used to obtain a distribution over words. The second column shows the words with the highest probability in that distribution. This model was trained using text from the MIR-Flickr data set which we use later in our experiments. We can see that the model has learned a basic understanding of which words are probable given the input words. For example, *chocolate, cake* generalizes to *sweets, desserts, food*, etc. The model often makes interesting inferences. For example, *girl, rain, station, norway* extends to *oslo, train, wet, umbrella, railway*, which are very plausible in that context. The model also captures regularities about language, discovers synonyms across multiple languages and learns about geographical relationships. This shows that the hidden units can capture these regularities and represent them as binary features.

2.3 Deep Boltzmann Machines (DBMs)

A Deep Boltzmann Machine (Salakhutdinov and Hinton, 2009a) is a network of symmetrically coupled stochastic binary units. It contains a set of visible units $\mathbf{v} \in \{0,1\}^D$, and a sequence of layers of hidden units $\mathbf{h}^{(1)} \in \{0,1\}^{F_1}$, $\mathbf{h}^{(2)} \in \{0,1\}^{F_2}$,..., $\mathbf{h}^{(L)} \in \{0,1\}^{F_L}$. There are connections only between hidden units in adjacent layers, as well as between visible and hidden units in the first hidden layer. Consider a DBM with three hidden layers[1] (i.e. $L = 3$). The energy of the joint configuration $\{\mathbf{v}, \mathbf{h}\}$ is defined as:

$$\begin{aligned} E(\mathbf{v}, \mathbf{h}; \theta) \;=\; & -\sum_{i=1}^{D}\sum_{j=1}^{F_1} W_{ij}^{(1)} v_i h_j^{(1)} - \sum_{j=1}^{F_1}\sum_{l=1}^{F_2} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} - \sum_{l=1}^{F_2}\sum_{p=1}^{F_3} W_{lp}^{(3)} h_l^{(2)} h_p^{(3)} \\ & - \sum_{i=1}^{D} b_i v_i - \sum_{j=1}^{F_1} b_j^{(1)} h_j^{(1)} - \sum_{l=1}^{F_2} b_l^{(2)} h_l^{(2)} - \sum_{p=1}^{F_3} b_p^{(3)} h_p^{(3)}, \end{aligned}$$

where $\mathbf{h} = \{\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}\}$ represent the set of hidden units, and $\theta = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{b}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \mathbf{b}^{(3)}\}$ are the model parameters, representing visible-to-hidden and hidden-to-hidden symmetric interaction terms, as well as bias terms. Biases are equivalent to weights on a connection to a unit whose state is fixed at 1. The probability that the model assigns to a visible vector $\mathbf{v}$ is given by the Boltzmann distribution:

$$P(\mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}; \theta)\right).$$

Observe that setting both $\mathbf{W}^{(2)}$=0 and $\mathbf{W}^{(3)}$=0 recovers the simpler Restricted Boltzmann Machine (RBM) model. The derivative of the log-likelihood with respect to the model parameters takes the following form:

$$\frac{\partial \log P(\mathbf{v}; \theta)}{\partial \mathbf{W}^{(1)}} \;=\; \mathbb{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{h}^{(1)\top}] - \mathbb{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{h}^{(1)\top}]. \tag{2.6}$$

The derivatives with respect to parameters $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(3)}$ take similar forms but instead involve the cross-products $\mathbf{h}^{(1)}\mathbf{h}^{(2)\top}$ and $\mathbf{h}^{(2)}\mathbf{h}^{(3)\top}$ respectively. Unlike RBMs, the conditional distribution over the states of the hidden variables conditioned on the data is no longer factorial. The exact computation of the data-dependent expectation takes time that is exponential in the number of hidden units, whereas the exact computation of the model's expectation takes time that is exponential in the number of hidden and visible units.

Deep Boltzmann Machines (DBMs) are interesting for several reasons. Firstly, like Deep Belief Networks (Hinton et al., 2006), DBMs can discover several layers of increasingly complex representations of the input, use an efficient layer-by-layer pretraining procedure (Salakhutdinov and Hinton, 2009a), can be trained on unlabelled data and can be fine-tuned for a specific task using (possibly limited) labelled data. Secondly, unlike other models with deep architectures, the approximate inference procedure for DBMs incorporates a top-down feedback in addition to the usual bottom-up pass, allowing Deep Boltz-

[1] For clarity, we use three hidden layers. Extending to models with more than three layers is straightforward.

mann Machines to better incorporate uncertainty about missing or noisy inputs. Thirdly, and perhaps most importantly, parameters of all layers can be optimized jointly by following the approximate gradient of a variational lower-bound on the likelihood objective. As we show in our experimental results, this greatly facilitates learning better generative models, particularly when modeling the multimodal data.

2.4 Multimodal Deep Boltzmann Machines

Let us first consider constructing a multimodal DBM using an image-text bi-modal DBM as our running example. Let $\mathbf{v}^m \in \mathbb{R}^D$ denote a real-valued image input and $\mathbf{v}^t \in \{1, ..., K\}$ denote an associated text input containing M words, with v_k^t denoting the count for the k^{th} word.

We start by modeling each data modality using a separate two-layer DBM. (see Fig. 2.3). Let $\mathbf{h}^{(1m)} \in \{0,1\}^{F_1^m}$ and $\mathbf{h}^{(2m)} \in \{0,1\}^{F_2^m}$ denote the two layers of hidden units. The probability that the image-specific two-layer DBM assigns to a visible vector $\mathbf{v}^m$ is given by:

$$
\begin{aligned}
P(\mathbf{v}^m;\theta^m) &= \sum_{\mathbf{h}^{(1m)},\mathbf{h}^{(2m)}} P(\mathbf{v}^m,\mathbf{h}^{(2m)},\mathbf{h}^{(1m)};\theta^m) \\
&= \frac{1}{\mathcal{Z}(\theta^m)} \sum_{\mathbf{h}^{(1m)},\mathbf{h}^{(2m)}} \exp\Bigg(-\sum_i \frac{(v_i^m - b_i^m)^2}{2\sigma_i^2} + \sum_{ij} \frac{v_i^m}{\sigma_i} W_{ij}^{(1m)} h_j^{(1m)} + \\
&\qquad \sum_{jl} W_{jl}^{(2m)} h_j^{(1m)} h_l^{(2m)} + \sum_j b_j^{(1m)} h_j^{(1m)} + \sum_l b_l^{(2m)} h_l^{(2m)} \Bigg).
\end{aligned}
$$

Note that conditioned on the states of $\mathbf{h}^{(1m)}$, the image-specific DBM uses Gaussian distribution to model the distribution over real-valued image features. Similarly, text-specific DBM uses Replicated Softmax to model the distribution over word count vectors. The corresponding probability that the text-specific two-layer DBM assigns to $\mathbf{v}^t$ is given by:

$$
\begin{aligned}
P(\mathbf{v}^t;\theta^t) &= \sum_{\mathbf{h}^{(1t)},\mathbf{h}^{(2t)}} P(\mathbf{v}^t,\mathbf{h}^{(2t)},\mathbf{h}^{(1t)};\theta^t) \\
&= \frac{1}{\mathcal{Z}(\theta^t, M)} \sum_{\mathbf{h}^{(1t)},\mathbf{h}^{(2t)}} \exp\Bigg(\sum_{jk} W_{k,j}^{(1t)} h_j^{(1t)} v_k^t + \sum_{jl} W_{jl}^{(2t)} h_j^{(1t)} h_l^{(2t)} + \\
&\qquad \sum_k b_k^t v_k^t + M \sum_j b_j^{(1t)} h_j^{(1t)} + \sum_l b_l^{(2t)} h_l^{(2t)} \Bigg),
\end{aligned}
$$

where $\mathbf{h}^{(1t)} \in \{0,1\}^{F_1^t}$, $\mathbf{h}^{(2t)} \in \{0,1\}^{F_2^t}$ represent the two layers of hidden units.

To form a multimodal DBM, we combine the two models by adding an additional layer on top of them. The resulting graphical model is shown in Fig. 2.3, right panel. The joint distribution over the multi-modal input, where $\mathbf{h} = \{\mathbf{h}^{(1m)}, \mathbf{h}^{(2m)}, \mathbf{h}^{(1t)}, \mathbf{h}^{(2t)}, \mathbf{h}^{(3)}\}$ denote all hidden variables, can be written

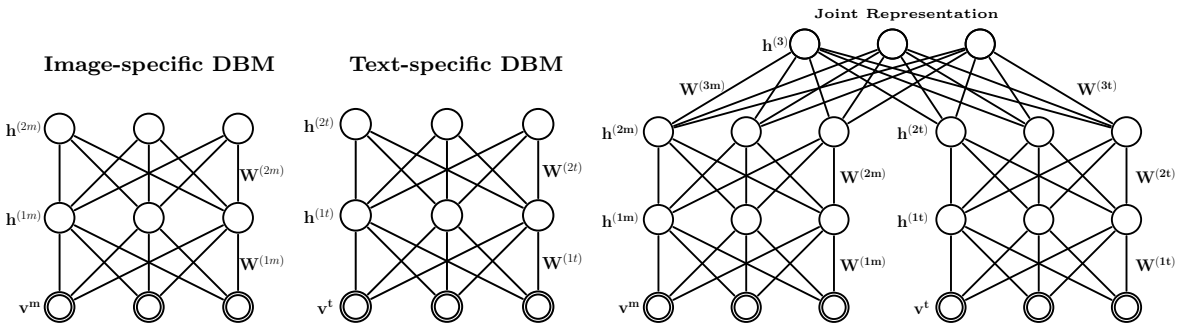


Figure 2.3: **Left:** Image-specific two-layer DBM that uses a Gaussian model to model the distribution over real-valued image features. **Middle:** Text-specific two-layer DBM that uses a Replicated Softmax model to model its distribution over the word count vectors. **Right:** A Multimodal DBM that models the joint distribution over image and text inputs. All layers but the first (bottom) layer use standard binary units.

as:

$$
\begin{aligned}
P(\mathbf{v}^m, \mathbf{v}^t; \theta) &= \sum_{\mathbf{h}^{(2m)}, \mathbf{h}^{(2t)}, \mathbf{h}^{(3)}} P(\mathbf{h}^{(2m)}, \mathbf{h}^{(2t)}, \mathbf{h}^{(3)}) \Big(\sum_{\mathbf{h}^{(1m)}} P(\mathbf{v}_m, \mathbf{h}^{(1m)} | \mathbf{h}^{(2m)}) \Big) \Big(\sum_{\mathbf{h}^{(1t)}} P(\mathbf{v}^t, \mathbf{h}^{(1t)} | \mathbf{h}^{(2t)}) \Big) \\
&= \frac{1}{\mathcal{Z}(\theta, M)} \sum_{\mathbf{h}} \exp \Bigg(\underbrace{\sum_{kj} W^{(1t)}_{kj} v^t_k h^{(1t)}_j + \sum_{jl} W^{(2t)}_{jl} h^{(1t)}_j h^{(2t)}_l + \sum_k b^t_k v^t_k + M \sum_j b^{(1t)}_j h^{(1t)}_j + \sum_l b^{(2t)}_l h^{(2t)}_l}_{\text{Replicated Softmax Text Pathway}} \\
&\underbrace{- \sum_i \frac{(v^m_i - b^m_i)^2}{2\sigma^2_i} + \sum_{ij} \frac{v^m_i}{\sigma_i} W^{(1m)}_{ij} h^{(1m)}_j + \sum_{jl} W^{(2m)}_{jl} h^{(1m)}_j h^{(2m)}_l + \sum_j b^{(1m)}_j h^{(1m)}_j + \sum_l b^{(2m)}_l h^{(2m)}_l}_{\text{Gaussian Image Pathway}} \\
&\underbrace{+ \sum_{lp} W^{(3t)} h^{(2t)}_l h^{(3)}_p + \sum_{lp} W^{(3m)} h^{(2m)}_l h^{(3)}_p + \sum_p b^{(3)}_p h^{(3)}_p}_{\text{Joint } 3^{rd} \text{ Layer}} \Bigg). \qquad (2.7)
\end{aligned}
$$

The normalizing constant depends on the number of words M in the corresponding document, since the low-level part of the text pathway contains as many softmax units as there are words in the document. Similar to the Replicated Softmax model, the multimodal DBM can be viewed as a family of different-sized DBMs that are created for documents of different lengths that share parameters.

The conditional distributions over the visible and the five sets of hidden units are given by:

$$
\begin{aligned}
p(h^{(1m)}_j = 1 | \mathbf{v}^m, \mathbf{h}^{(2m)}) &= g\left(\sum_{i=1}^{D} W^{(1m)}_{ij} \frac{v^m_i}{\sigma_i} + \sum_{l=1}^{F^m_2} W^{(2m)}_{jl} h^{(2m)}_l + b^{(1m)}_j \right), \qquad (2.8) \\
p(h^{(2m)}_l = 1 | \mathbf{h}^{(1m)}, \mathbf{h}^{(3)}) &= g\left(\sum_{j=1}^{F^m_1} W^{(2m)}_{jl} h^{(1m)}_j + \sum_{p=1}^{F_3} W^{(3m)}_{lp} h^{(3)}_p + b^{(2m)}_l \right), \\
p(h^{(1t)}_j = 1 | \mathbf{v}^t, \mathbf{h}^{(2t)}) &= g\left(\sum_{k=1}^{K} W^{(1t)}_{kj} v^t_k + \sum_{l=1}^{F^t_2} W^{(2t)}_{jl} h^{(2t)}_l + M b^{(1t)}_j \right), \\
p(h^{(2t)}_l = 1 | \mathbf{h}^{(1t)}, \mathbf{h}^{(3)}) &= g\left(\sum_{j=1}^{F^t_1} W^{(2t)}_{jl} h^{(1t)}_j + \sum_{p=1}^{F_3} W^{(3t)}_{lp} h^{(3)}_p + b^{(2t)}_l \right), \\
p(h^{(3)}_p = 1 | \mathbf{h}^{(2)}) &= g\left(\sum_{l=1}^{F^m_2} W^{(3m)}_{lp} h^{(2m)}_l + \sum_{l=1}^{F^t_2} W^{(3t)}_{lp} h^{(2t)}_l + b^{(3)}_p \right),
\end{aligned}
$$

$$p(v_{ik}^t = 1|\mathbf{h}^{(1t)}) \;=\; \frac{\exp\left(\sum_{j=1}^{F_1^t} h_j^{(1t)} W_{jk}^{(1t)} + b_k^t\right)}{\sum_{q=1}^{K} \exp\left(\sum_{j=1}^{F_1^t} h_j^{(1t)} W_{jq}^{(1t)} + b_k^t\right)}, \quad v_i^m|\mathbf{h}^{(1m)} \sim \mathcal{N}\left(\sigma_i \sum_{j=1}^{F_1^m} W_{ij}^{(1m)} h_j^{(1m)} + b_i^m, \sigma_i^2\right).$$

Extending multimodal DBMs to other data modalities requires a simple modification of the first-layer modules. For example, consider modelling video-audio bi-modal data. Unlike image-text data, video-audio data can be represented as a sequence of real-valued vector pairs. Let $\mathbf{v}^m \in \mathbb{R}^D$ denote a real-valued input from the video stream (e.g. several consecutive image frames), and $\mathbf{v}^a \in \mathbb{R}^D$ denote an associated audio input (e.g. corresponding consecutive audio frames). We can easily construct the corresponding multimodal DBM with both pathways using Gaussian RBMs as the first layer. Compared to Eq. 2.7, the the joint distribution over the multi-modal input variables, can be written as[2]:

$$P(\mathbf{v}^m, \mathbf{v}^a; \theta) = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\Bigg(\underbrace{-\sum_i \frac{(v_i^m)^2}{2\sigma_i^2} + \sum_{ij} \frac{v_i^m}{\sigma_i} W_{ij}^{(1m)} h_j^{(1m)} + \sum_{jl} W_{jl}^{(2m)} h_j^{(1m)} h_l^{(2m)}}_{\text{Gaussian Video Pathway}} \tag{2.9}$$

$$\underbrace{-\sum_i \frac{(v_i^a)^2}{2\sigma_i^2} + \sum_{ij} \frac{v_i^a}{\sigma_i} W_{ij}^{(1a)} h_j^{(1a)} + \sum_{jl} W_{jl}^{(2a)} h_j^{(1a)} h_l^{(2a)}}_{\text{Gaussian Audio Pathway}} + \underbrace{\sum_{lp} W^{(3t)} h_l^{(2t)} h_p^{(3)} + \sum_{lp} W^{(3m)} h_l^{(2m)} h_p^{(3)}}_{\text{Joint } 3^{rd} \text{ Layer}} \Bigg).$$

2.4.1 Approximate Inference and Learning

Exact maximum likelihood learning in this model is intractable, but efficient approximate learning of DBMs can be carried out by using a variational approach, where mean-field inference is used to estimate data-dependent expectations and an MCMC based stochastic approximation procedure is used to approximate the model's expected sufficient statistics.

Estimating the Data-dependent Statistics

Consider any approximating distribution $Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu})$, parameterized by a vector of parameters $\boldsymbol{\mu}$, for the posterior $P(\mathbf{h}|\mathbf{v}; \theta)$. Then the log-likelihood of the DBM model has the following variational lower bound:

$$\begin{aligned} \log P(\mathbf{v}; \theta) \;&\geq\; \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}; \theta) \log P(\mathbf{v}, \mathbf{h}; \theta) + \mathcal{H}(Q) \\ &\geq\; \log P(\mathbf{v}; \theta) - \mathrm{KL}(Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) || P(\mathbf{h}|\mathbf{v}; \theta)), \end{aligned} \tag{2.10}$$

where $\mathrm{KL}(Q||P)$ is the Kullback-–Leibler divergence between the two distributions, and $\mathcal{H}(\cdot)$ is the entropy functional. The bound becomes tight if and only if $Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) = P(\mathbf{h}|\mathbf{v}; \theta)$.

We approximate the true posterior $P(\mathbf{h}|\mathbf{v}; \theta)$, where $\mathbf{v} = \{\mathbf{v}^m, \mathbf{v}^t\}$, with a fully factorized approximating distribution over the five sets of hidden units $\{\mathbf{h}^{(1m)}, \mathbf{h}^{(2m)}, \mathbf{h}^{(1t)}, \mathbf{h}^{(2t)}, \mathbf{h}^{(3)}\}$:

[2]We omit the bias terms for the hidden layers for clarity of presentation

$$Q(\mathbf{h}|\mathbf{v};\boldsymbol{\mu}) = \left(\prod_{j=1}^{F_1^m} q(h_j^{(1m)}|\mathbf{v}) \prod_{l=1}^{F_2^m} q(h_l^{(2m)}|\mathbf{v})\right)\left(\prod_{j=1}^{F_1^t} q(h_j^{(1t)}|\mathbf{v}) \prod_{l=1}^{F_2^t} q(h_l^{(2t)}|\mathbf{v})\right) \prod_{p=1}^{F_3} q(h_p^{(3)}|\mathbf{v}), \tag{2.11}$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}^{(1m)}, \boldsymbol{\mu}^{(1t)}, \boldsymbol{\mu}^{(2m)}, \boldsymbol{\mu}^{(2t)}, \boldsymbol{\mu}^{(3)}\}$ are the mean-field parameters with $q(h_i^{(l)} = 1|\mathbf{v}) = \mu_i^{(l)}$ for $l = 1, 2, 3$.

For each training example, the variational bound of Eq. 2.10 is maximized with respect to the variational parameters $\boldsymbol{\mu}$ for fixed parameters θ. This results in the following mean-field fixed-point equations:

$$\begin{aligned}
\mu_j^{(1m)} &\leftarrow g\Big(\sum_{i=1}^{D} W_{ij}^{(1m)} \frac{v_i^m}{\sigma_i} + \sum_{l=1}^{F_2^m} W_{jl}^{(2m)} \mu_l^{(2m)}\Big), \quad \mu_l^{(2m)} \leftarrow g\Big(\sum_{j=1}^{F_1^m} W_{jl}^{(2m)} \mu_j^{(1m)} + \sum_{k=1}^{F_3} W_{lk}^{(3m)} \mu_k^{(3)}\Big), \\
\mu_j^{(1t)} &\leftarrow g\Big(\sum_{k=1}^{K} W_{kj}^{(1t)} v_k^t + \sum_{l=1}^{F_2^t} W_{jl}^{(2t)} \mu_l^{(2t)}\Big), \quad \mu_l^{(2t)} \leftarrow g\Big(\sum_{j=1}^{F_1^t} W_{jl}^{(2t)} \mu_j^{(1t)} + \sum_{k=1}^{F_3} W_{lk}^{(3t)} \mu_k^{(3)}\Big), \\
\mu_p^{(3)} &\leftarrow g\Big(\sum_{l=1}^{F_2^m} W_{lp}^{(3m)} \mu_l^{(2m)} + \sum_{l=1}^{F_2^t} W_{lp}^{(3t)} \mu_l^{(2t)}\Big),
\end{aligned} \tag{2.12}$$

where $g(x) = 1/(1 + \exp(-x))$ is the logistic function. To solve these fixed-point equations, we simply cycle through layers, updating the mean-field parameters within a single layer. The variational parameters $\boldsymbol{\mu}$ are then used to compute the data-dependent statistics in Eq. 2.6. For example,

$$\begin{aligned}
\mathbb{E}_{P_{\text{data}}}[\mathbf{v}^m \mathbf{h}^{(1m)\top}] &= \frac{1}{N}\sum_{n=1}^{N} \mathbf{v}_n^m \boldsymbol{\mu}_n^{(1m)\top} \\
\mathbb{E}_{P_{\text{data}}}[\mathbf{h}^{(1m)} \mathbf{h}^{(2m)\top}] &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{\mu}_n^{(1m)} \boldsymbol{\mu}_n^{(2m)\top},
\end{aligned}$$

where the average on the RHS is over training cases. Note the close connection between the form of the mean-field fixed point updates and the form of the conditional distribution defined by Eq. 2.8. In fact, implementing the mean-field updates requires no extra work beyond implementing the Gibbs sampler.

Estimating the Data-independent Statistics

Given the variational parameters $\boldsymbol{\mu}$, the model parameters θ are then updated to maximize the variational bound using an MCMC-based stochastic approximation (Salakhutdinov and Hinton, 2009a; Tieleman, 2008; Younes, 1998). Remember that in out setting, we are learning a whole family of different-sized DBMs that depend on the number of words, or the number of replicated softmax variables (see Eq. 2.7). Let us first assume that the text input only contains a set of M words. Learning with stochastic approximation proceeds as follows. Let θ_t and $\mathbf{x}_t = \{\mathbf{v}_t^m, \mathbf{v}^t, \mathbf{h}_t^{(1m)}, \mathbf{h}_t^{(1t)}, \mathbf{h}_t^{(2m)}, \mathbf{h}_t^{(2t)}, \mathbf{h}_t^{(3)}\}$ be the current parameters and the state. Then $\mathbf{x}_t$ and θ_t are updated sequentially as follows:

- Given $\mathbf{x}_t$, sample a new state $\mathbf{x}_{t+1}$ from the transition operator $T_{\theta_t}(\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t)$ that leaves $P(\cdot;\theta_t)$ invariant. This can be accomplished by using Gibbs sampling (see Eq. 2.8).

Algorithm 1 Learning Procedure for a Multimodal Deep Boltzmann Machine.

Require: a training set of N data vectors $\mathbf{v}_n = \{\mathbf{v}_n^m, \mathbf{v}_n^t\}$, $n = 1, ..., N$, and S, the number of persistent Markov chains. Let Λ be a diagonal $D \times D$ matrix with $\Lambda_{ii} = 1/\sigma_i$.

Randomly initialize parameter vector θ_0 and S samples: $\{\tilde{\mathbf{v}}_{0,1}, \tilde{\mathbf{h}}_{0,1}\}, ..., \{\tilde{\mathbf{v}}_{0,S}, \tilde{\mathbf{h}}_{0,S}\}$, where we define $\tilde{\mathbf{h}} = \{\tilde{\mathbf{h}}^{(1m)}, \tilde{\mathbf{h}}^{(1t)}, \tilde{\mathbf{h}}^{(2m)}, \tilde{\mathbf{h}}^{(2t)}, \tilde{\mathbf{h}}^{(3)}\}$.

for $t = 0$ to T (number of iterations) **do**

  // Variational Inference:

  for each training example $\mathbf{v}_n$, $n = 1$ to N **do**

    Run the mean-field fixed-point updates until convergence using Eq. 2.12.

    Set $\boldsymbol{\mu}_n = \boldsymbol{\mu}$.

  end for

  // Stochastic Approximation:

  for each sample $s = 1$ to S (number of persistent Markov chains) **do**

    Sample $(\tilde{\mathbf{v}}_{t+1,s}, \tilde{\mathbf{h}}_{t+1,s})$ given $(\tilde{\mathbf{v}}_{t,s}, \tilde{\mathbf{h}}_{t,s})$ by running a Gibbs sampler for one step using Eq. 2.8.

  end for

  // Parameter Update:

  // Image Pathway:

$$\mathbf{W}_{t+1}^{(1m)} = \mathbf{W}_t^{(1m)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \mathbf{v}_n^m \Lambda (\boldsymbol{\mu}_n^{(1m)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{v}}_{t+1,s}^m \Lambda (\tilde{\mathbf{h}}_{t+1,s}^{(1m)})^\top \right).$$

$$\mathbf{W}_{t+1}^{(2m)} = \mathbf{W}_t^{(2m)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \boldsymbol{\mu}_n^{(1m)} (\boldsymbol{\mu}_n^{(2m)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{h}}_{t+1,s}^{(1m)} (\tilde{\mathbf{h}}_{t+1,s}^{(2m)})^\top \right).$$

  // Text Pathway:

$$\mathbf{W}_{t+1}^{(1t)} = \mathbf{W}_t^{(1t)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \mathbf{v}_n^t (\boldsymbol{\mu}_n^{(1t)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{v}}_{t+1,s}^t (\tilde{\mathbf{h}}_{t+1,s}^{(1t)})^\top \right).$$

$$\mathbf{W}_{t+1}^{(2t)} = \mathbf{W}_t^{(2t)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \boldsymbol{\mu}_n^{(1t)} (\boldsymbol{\mu}_n^{(2t)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{h}}_{t+1,s}^{(1t)} (\tilde{\mathbf{h}}_{t+1,s}^{(2t)})^\top \right).$$

  // Joint Layer:

$$\mathbf{W}_{t+1}^{(3m)} = \mathbf{W}_t^{(3m)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \boldsymbol{\mu}_n^{(2m)} (\boldsymbol{\mu}_n^{(3)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{h}}_{t+1,s}^{(2m)} (\tilde{\mathbf{h}}_{t+1,s}^{(3)})^\top \right).$$

$$\mathbf{W}_{t+1}^{(3t)} = \mathbf{W}_t^{(3t)} + \alpha_t \left(\tfrac{1}{N} \textstyle\sum_{n=1}^{N} \boldsymbol{\mu}_n^{(2t)} (\boldsymbol{\mu}_n^{(3)})^\top - \tfrac{1}{S} \sum_{s=1}^{S} \tilde{\mathbf{h}}_{t+1,s}^{(2t)} (\tilde{\mathbf{h}}_{t+1,s}^{(3)})^\top \right).$$

  Decrease α_t.

end for

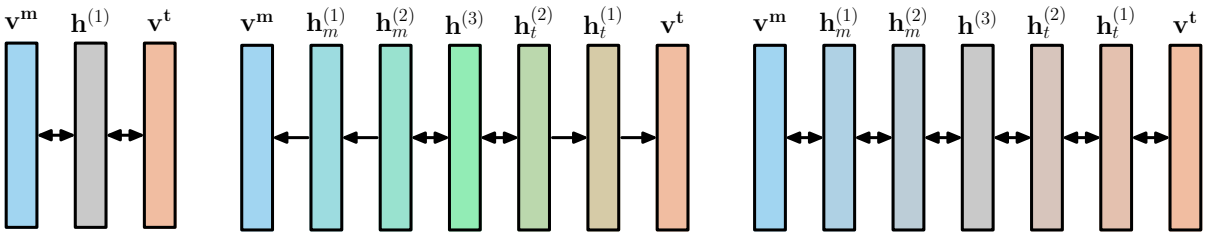
- A new parameter θ_{t+1} is then obtained by making a gradient step, where the intractable model's expectation $\mathbb{E}_{P_{\text{model}}}[\cdot]$ in the gradient is replaced by a point estimate at sample $\mathbf{x}_{t+1}$.

In practice, we typically maintain a set of S "persistent" Markov chains $X_t = \{\mathbf{x}_{t,1}, ..., \mathbf{x}_{t,S}\}$, and use an average over those particles.

The overall learning procedure for DBMs is summarized in Algorithm 1. Extensions to the variable text input is trivial. For each $m = 1, ..., M_{max}$, where M_{max} is the maximum number of words across all documents, we can create a corresponding multimodal DBM with m replicated softmax variables and shared parameters. For each model m, we simply maintain a set of S_m persistent Markov chains[3]. Learning then proceeds as discussed before.

Stochastic approximation provides asymptotic convergence guarantees and belongs to the general class of Robbins–Monro approximation algorithms (Robbins and Monro, 1951; Younes, 1998). Sufficient conditions that ensure almost sure convergence to an asymptotically stable point are given in Younes (1989, 1998); Yuille (2004). One necessary condition requires the learning rate to decrease with time, so that $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$. This condition can, for example, be satisfied simply by setting $\alpha_t = a/(b + t)$, for positive constants $a > 0$, $b > 0$. Typically, in practice, the sequence $|\theta_t|$ is bounded,

[3]Ideally, we would have each S_m be as large as computationally feasible. However, given a fixed budget for the total number of chains, we could choose S_m to be proportional to the number of documents containing m words.



(a) RBM (b) Multimodal DBN (c) Multimodal DBM

Figure 2.4: Different ways of modeling multimodal inputs.

and the Markov chain, governed by the transition kernel T_θ, is ergodic. Together with the condition on the learning rate, this ensures almost sure convergence of the stochastic approximation algorithm to an asymptotically stable point (Younes, 1998; Yuille, 2004).

Greedy Layerwise Pretraining

The learning procedure for Deep Boltzmann Machines described above can be used by initializing model parameters at random. However, the model performs much better if parameters are initialized sensibly. We therefore use a greedy layer-wise pretraining strategy by learning a stack of modified Restricted Boltzmann Machines (RBMs) (for details see Salakhutdinov and Hinton (2009a)). The pretraining procedure is quite similar to the pretraining procedure of Deep Belief Networks (Hinton et al., 2006), and it allows us to perform approximate inference by a single bottom-up pass. This fast approximate inference can also be used to initialize the mean-field, which then converges much faster than mean-field initialized at random.

2.5 Applying DBMs to Different Tasks

This section describes the salient properties of the multimodal DBM model and shows how it can be used to solve various tasks that are relevant to multimodal data, such as inferring missing modalities, obtaining a fused representation and providing an initialization for discriminative tasks.

2.5.1 Salient Features

A Multimodal DBM can be viewed as a composition of unimodal undirected pathways. Each pathway can be pretrained separately in a completely unsupervised fashion, which allows us to leverage a large supply of unlabelled data. Any number of pathways each with any number of layers could potentially be used. The type of the lower-level RBMs in each pathway could be different, accounting for different input distributions, as long as the final hidden representations at the end of each pathway are of the same type.

The intuition behind our model is as follows. Each data modality has very different statistical properties which make it difficult for a single-layer model to directly find correlations across modalities. In our model, this difference is bridged by putting layers of hidden units between the modalities. The idea is illustrated in Fig. 2.4c, which is just a different way of displaying Fig. 2.3. Compared to the simple RBM (see Fig. 2.4a), where the hidden layer $\mathbf{h}$ directly models the distribution over $\mathbf{v}^t$ and $\mathbf{v}^m$, the first layer of hidden units $\mathbf{h}^{(1m)}$ in a DBM has an easier task to perform - that of modeling the

Step 50	Step 100	Step 150	Step 200	Step 250
travel	beach	sea	water	italy
trip	ocean	beach	canada	water
vacation	waves	island	bc	sea
africa	sea	vacation	britishcolumbia	boat
earthasia	sand	travel	reflection	italia
asia	nikon	ocean	alberta	mare
men	surf	caribbean	lake	venizia
2007	rocks	tropical	quebec	acqua
india	coast	resort	ontario	ocean
tourism	shore	trip	ice	venice

Figure 2.5: Text generated by the DBM conditioned on an image by running a Gibbs sampler. Ten words with the highest probability are shown at the end of every 50 sampling steps.

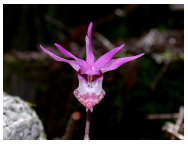
Input tags	Step 50	Step 100	Step 150	Step 200	Step 250
purple, flowers					
car, automobile					

Figure 2.6: Images retrieved by running a Gibbs sampler conditioned on the input tags. The images shown are those which are closest to the sampled image features. Samples were taken after every 50 steps.

distribution over $\mathbf{v}^m$ and $\mathbf{h}^{(2m)}$. Each layer of hidden units in the DBM contributes a small part to the overall task of modeling the distribution over $\mathbf{v}^m$ and $\mathbf{v}^t$. In the process, each layer learns successively higher-level representations and removes modality-specific correlations. Therefore, the middle layer in the network can be seen as a (relatively) "modality-free" representation of the input as opposed to the input layers which were "modality-full".

Another way of using a deep model to combine multimodal inputs is to use a Multimodal Deep Belief Network (DBN) (Fig. 2.4b) which consists of an RBM followed by a directed belief network. We emphasize that there is an important distinction between this model and the DBM model of Fig. 2.4c. In a DBN model, and related autoencoder models, the responsibility of the multimodal modeling falls entirely on the joint layer. In the DBM, on the other hand, this responsibility is spread out over the entire network. From the generative perspective, states of low-level hidden units in one pathway can influence the states of hidden units in other pathways through the higher-level layers, which is not the case for DBNs.

2.5.2 Generating Missing Modalities

The multimodal DBM can be used to generate such missing data modalities by clamping the observed modalities at the inputs and sampling the hidden modalities by running the standard Gibbs sampler.

For example, consider generating text conditioned on a given image[4] $\mathbf{v}^m$. The observed modality $\mathbf{v}^m$ is clamped at the inputs and all hidden units are initialized randomly. Alternating Gibbs sampling

[4]Generating image features conditioned on text can be done in a similar way.

is used to draw samples from $P(\mathbf{v}^t|\mathbf{v}^m)$ by updating each hidden layer given the states of the adjacent layers (see Eq. 2.8). A sample drawn from this distribution describes a multinomial distribution over the text vocabulary. This distribution can then be used to sample words. This process is illustrated for a test image in Fig. 2.5, showing the generated text after every 50 Gibbs steps. We see that not only does the sampler generate meaningful text, it shows evidence of jumping across different modes. For example, it generates *tropical*, *caribbean* and *resort* together, then moves on to *canada*, *bc*, *quebec lake*, *ice*, and then *italia*, *venizia* and *mare*. Each of these groups of words are plausible descriptions of the image. Moreover, each group is consistent within itself. This suggests that the model has been able to associate clusters of consistent descriptions with the same image. In other words, the model can capture multiple modes in the conditional distribution and access them by a Gibbs sampler.

The model can also be used to generate image features conditioned on text. Fig. 2.6 shows examples of two such runs.

2.5.3 Inferring Joint Representations

The model can also be used to generate a joint representation of data by combining multiple data modalities. For inferring the joint representation, conditioned on the observed modalities, the observed modalities are clamped and Gibbs sampling is performed to sample from $P(\mathbf{h}^{(3)}|\mathbf{v}^m, \mathbf{v}^t)$ (if both modalities are present) or from $P(\mathbf{h}^{(3)}|\mathbf{v}^m)$ (if text is missing). A faster alternative, which we adopt in our experimental results, is to use variational inference to approximate posterior $Q(\mathbf{h}^{(3)}|\mathbf{v}^m, \mathbf{v}^t)$ or $Q(\mathbf{h}^{(3)}|\mathbf{v}^m)$ (see Sec. 2.4.1). The marginals of the approximate posterior over $\mathbf{h}^{(3)}$ (variational parameters $\boldsymbol{\mu}^{(3)}$) constitute the joint representation of the inputs.

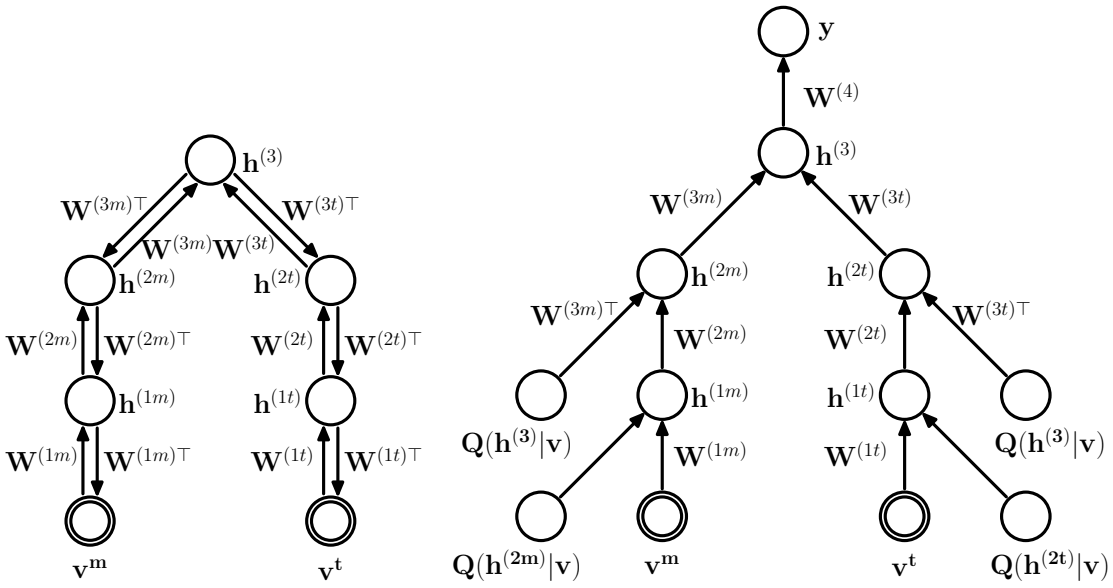
This representation can then be used to do information retrieval for multimodal or unimodal queries. Each data point in the database (whether missing some modalities or not) can be mapped to this latent space. Queries can also be mapped to this space and an appropriate distance metric can be used to retrieve results that are close to the query.

2.5.4 Discriminative Tasks

After learning, the Multimodal Deep Boltzmann Machine can be used to initialize a multilayer neural network by partially unrolling the lower layers (Salakhutdinov and Hinton, 2009a). We can then use standard backpropagation algorithm to discriminatively fine-tune the model. For each multimodal input vector $\mathbf{v} = \{\mathbf{v}^m, \mathbf{v}^t\}$, mean-field inference is used to obtain an approximate posterior distribution $Q(\mathbf{h}|\mathbf{v})$. The marginals of this approximate posterior (variational parameters $\boldsymbol{\mu}$), together with the data, can be used to create an augmented input for this multimodal deep multilayer neural network, as shown in Fig. 2.7. This augmented input is important because it helps maintain the scale of inputs that each hidden unit is expecting. For example, in Fig. 2.7a, the conditional distribution over $\mathbf{h}^{(2m)}$, as defined by the DBM model (see Eq. 2.8), takes the following form:

$$p(h_l^{(2m)} = 1|\mathbf{h}^{(1m)}, \mathbf{h}^{(3)}) \;=\; g\left(\sum_j W_{jl}^{(2m)} h_j^{(1m)} + \sum_p W_{lp}^{(3m)} h_p^{(3)} + b_l^{(2m)}\right).$$

Hence layer $\mathbf{h}^{(2m)}$ receives inputs from $\mathbf{h}^{(1m)}$ as well as $\mathbf{h}^{(3)}$. When this DBM is used to initialize a



(a) DBM (b) Unrolled DBM

Figure 2.7: After learning, the DBM model as shown in (a) is used to initialize a multilayer neural network (b), where the marginals of approximate posterior $Q(h_i = 1|\mathbf{v})$ are used as additional inputs. The network is fine-tuned by backpropagation.

feed-forward network (Fig. 2.7b), the augmented inputs $Q(\mathbf{h}^{(3)}|\mathbf{v})$ serve as a proxy for $\mathbf{h}^{(3)}$. This ensures that when the feed-forward network is fine-tuned, the hidden units in $\mathbf{h}^{(2m)}$ start off with receiving the same input as they would have received in a mean-field update during unsupervised pretraining. However, once the weights start changing during fine-tuning, the augmented inputs are no longer fixed points of the mean-field update equations and the model is free to use those inputs as it likes. The weights from $Q(\mathbf{h}^{(3)}|\mathbf{v})$ to $\mathbf{h}^{(2m)}$ are only initialized to $\mathbf{W}^{(3m)\top}$ and are not tied to the weights from $\mathbf{h}^{(2m)}$ to $\mathbf{h}^{(3)}$. This initialization scheme makes sure that the model starts fine-tuning from the same place where pretraining left off.

Note that the gradient-based fine-tuning may choose to ignore the marginals of the approximate posterior $Q(\mathbf{h}|\mathbf{v})$ by driving the corresponding weights to zero. This will result in a standard neural network, much like the neural network that is obtained from a Deep Belief Network or an autoencoder model. In practice, however, the network typically uses the entire augmented input for making predictions.

When using this model at test time, we first have to run the mean-field updates in the DBM to get the additional inputs and then use the fine-tuned feed-forward network to get the model's predictions. This creates an overhead in the running time. For all of the data sets in our experimental results, we typically used 5 mean-field updates, which was sufficient for the mean-field to settle down.

2.6 Experimental Results on Image-text data

Our first data set consists of image-text pairs. Bi-modal data of this kind exemplifies a common real-world scenario where we have some image and a few words describing that image. There is a need to build representations that fuse this information into a homogeneous space, so that each data point can be represented as a single vector. This representation would be convenient for classification and retrieval

Classes	baby, female, people, portrait	plant life, river, water	clouds, sea, sky, transport, water	animals, dog, food	clouds, sky, structures
Images					
Tags	claudia	⟨ no text ⟩	barco, pesca, boattosail, navegação	watermelon, hilarious, chihuahua, dog	colours, cores, centro, comercial, building

Figure 2.8: Some examples from the MIR-Flickr data set. Each instance in the data set is an image along with textual tags. Each image is associated with multiple classes.

problems.

2.6.1 Data Set and Feature Extraction

We used the MIR Flickr Data set (Huiskes and Lew, 2008) in our experiments. The data set consists of 1 million images retrieved from the social photography website Flickr along with their user assigned tags. An example is shown in Fig. 6.5. Among the 1 million images, 25,000 have been annotated using 24 labels including object categories such as, *bird*, *tree*, *people* and scene categories, such as *indoor*, *sky* and *night*. A stricter labeling was done for 14 of these classes where an image was annotated with a category only if that category was salient. This leads to a total of 38 classes where each image may belong to several classes. The data set also consists of an additional 975,000 unannotated images. From the 25,000 annotated images we use 10,000 images for training, 5,000 for validation and 10,000 for testing, following Huiskes et al. (2010). Mean Average Precision (MAP) is used as the performance metric. Results are averaged over 5 random splits of the 25,000 examples into train, validation and test sets.

There are more than 800,000 distinct tags in the data set. In order to keep the text representation manageable, each text input was represented using a vocabulary of the 2000 most frequent tags in the 1 million collection. After restricting to this vocabulary, the average number of tags associated with an image is 5.15 with a standard deviation of 5.13. There are 128,501 images which do not have any tags, out of which 4,551 are in the labelled 25K subset. Hence about 18% of the labelled data has images but is missing text.

Images were represented by 3857-dimensional features, that were extracted by concatenating Pyramid Histogram of Words (PHOW) features (Bosch et al., 2007), Gist (Oliva and Torralba, 2001) and MPEG-7 descriptors (EHD, HTD, CSD, CLD, SCD) (Manjunath et al., 2001). Each dimension was mean-centered and normalized to unit variance. PHOW features are bags of image words obtained by extracting dense SIFT features over multiple scales and clustering them. We used publicly available code (Vedaldi and Fulkerson, 2008; Bastan et al., 2010) for extracting these features. The extracted features are publicly available[5].

2.6.2 Model Architecture and Implementation Details

The image pathway consists of a Gaussian RBM with 3857 linear visible units and 1024 hidden units. This is followed by a layer of 1024 binary hidden units. The text pathway consists of a Replicated Softmax Model with 2000 visible units and 1024 hidden units, followed by another layer of 1024 hidden

[5] `http://www.cs.toronto.edu/~nitish/multimodal`

units. The joint layer contains 2048 hidden units. All hidden units are binary. Each Gaussian visible unit was set to have unit variance ($\sigma_i = 1$) which was kept fixed and not learned[6]. Each layer of weights was pretrained using CD-n where n was gradually increased from 1 to 20. All word count vectors were normalized so that they sum to one. This way we avoid running separate Markov chains for each document length to get the model distribution's sufficient statistics, which makes it possible to have a fast GPU implementation.

We also experimented with training a proper generative model, that is, without normalizing the data. Remember, the image-text bimodal DBM can be viewed as a family of different-sized DBMs that are created for documents of different lengths that share parameters. In this setting, we used separate MCMC chains for different sized documents. However, the results were statistically indistinct from the case when we made the simplifying assumptions. This is probably because this data set does not have a huge variance in the number of words per image (5-15 tags per image).

After training the DBM model generatively, we applied it for classification and retrieval tasks. We compared different ways of using the model for classification. The simplest method is to extract the representation at the joint hidden layer and perform 1-vs-all classification using logistic regression. We compare this to fine-tuning the model discriminatively as described in Sec. 2.5.4. We also used dropout (Hinton et al., 2012b) during fine-tuning to further improve the classification performance. For dropout, we retained each unit with probability $p = 0.8$.

2.6.3 Classification Tasks

We run two classification experiments to highlight two distinct capabilities of the proposed DBM model. In the first experiment, we train and test the model on multimodal inputs. This experiment is designed to evaluate the DBM's ability to *represent* multimodal data in a way that is useful for classification. In the second experiment, we train on multimodal inputs, but at test time we are only given images. This experiment is designed to evaluate the DBM's ability to *generate* useful text and use it as a substitute for real data.

Since examples in the data set may have multiple labels, classification accuracy is not very meaningful. Instead, we evaluate our models using Mean Average Precision (MAP) and precision at top-50 predictions (Prec@50). These are standard metrics used for multi-label classification and have been previously used to report results on this data set.

Multimodal Inputs

In out first experiment, the task is to assign labels to image-text pairs. Huiskes et al. (2010) provided baselines for this data set with Linear Discriminant Analysis (LDA) and RBF-kernel Support Vector Machines (SVMs) using the labelled 25K subset of the data. They represent the multimodal input as a concatenation of image features and word counts. Table. 2.2 shows the performance of these models. The image features did not include SIFT-based features. Therefore, to make a fair comparison, our model was first trained using the same amount of data and a similar set of features (i.e., excluding our SIFT-based features). Table. 2.2 shows that the DBM model outperforms its competitor SVM and LDA models in terms of MAP and Prec@50. The DBM achieves a MAP of 0.526, compared to 0.475 and 0.492, achieved by SVM and LDA models.

[6]We found that learning the variance made the training unstable.

Model	MAP	Prec@50
Random	0.124	0.124
SVM (Huiskes et al., 2010)	0.475	0.758
LDA (Huiskes et al., 2010)	0.492	0.754
DBM	0.526 ± 0.007	0.791 ± 0.008
DBM (using unlabelled data)	**0.585** ± 0.004	**0.836** ± 0.004

Table 2.2: Multimodal Classification Results. Mean Average Precision (MAP) and precision@50 obtained by different models. A similar set of input features is used across all models.

Model	No pretraining	DBN	DAE	DBM
Logistic regression on joint layer features	-	0.599 ± 0.004	0.600 ± 0.004	0.609 ± 0.004
Sparsity + Logistic regression	-	0.626 ± 0.003	0.628 ± 0.004	0.631 ± 0.004
Sparsity + discriminative fine-tuning	0.482 ± 0.003	0.630 ± 0.004	0.630 ± 0.003	0.634 ± 0.004
Sparsity + discriminative fine-tuning + dropout	0.575 ± 0.004	0.638 ± 0.004	0.638 ± 0.004	**0.641 ± 0.004**

Table 2.3: Comparison of MAP across different deep models. More input features were used compared to Table. 2.2. Sparsity, full discriminative fine-tuning and dropout lead to improvements across all models.

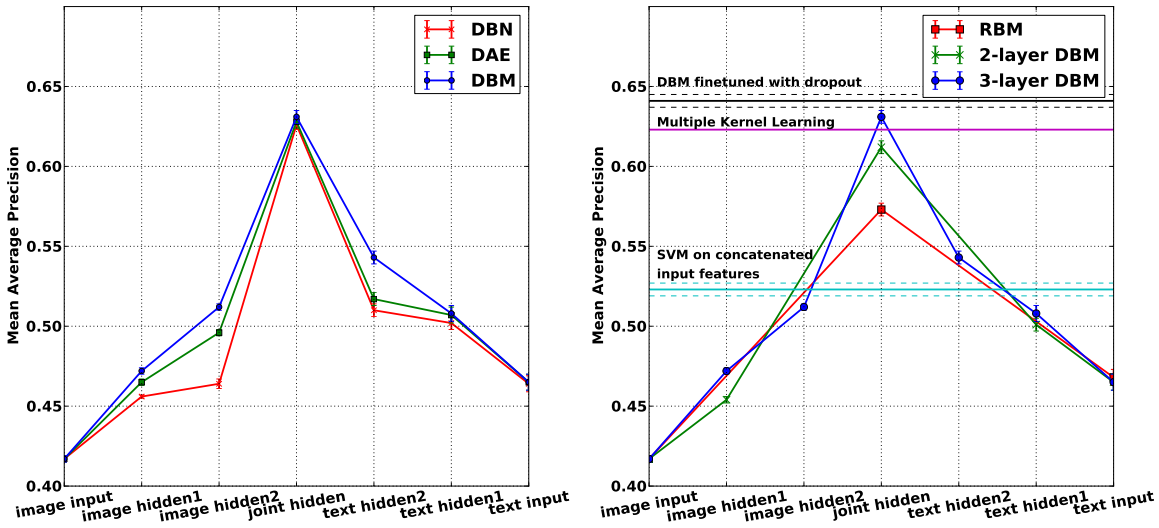
Next, we tried to see how much gain can be obtained by using the 975,000 unlabelled examples. We trained a DBM using these examples and, not surprisingly, this improved the DBM's MAP to 0.585.

Having established that DBMs outperform simple linear models, we now compare DBMs to two other deep models—Deep Belief Nets (DBNs) (Hinton et al., 2006) and Denoising Autoencoders (DAEs) (Vincent et al., 2008). We found that further improvements can be obtained by using more image features. We added PHOW features, which use dense SIFT descriptors, to learn a feature dictionary. Table. 2.3 shows results using this extended feature set. We use unlabelled data to pretrain these models. We also closely explore the benefits of full discriminative fine-tuning, regularizers that encourage sparse activations, and dropout (Hinton et al., 2012b; Srivastava, 2013).

First, we apply simple logistic regression on the high-level joint representation learned by each of the three models. As shown in Table. 2.3, the DBN and DAE obtain a MAP of 0.599 and 0.600 respectively, whereas the DBM gets 0.609. The error bars indicate that this improvement is statistically significant. The DBNs and DAEs give very similar results. Next we added a KL-sparsity regularizer (Olshausen and Field, 1996) during unsupervised pretraining of all the three models. This improved the performance across all models. In particular, the DBM achieved a MAP of 0.631. Full discriminative training further improved the DBM's MAP to 0.634. Next, we fine-tuned the model using the dropout technique proposed by Hinton et al. (2012b). Using this we achieved the a MAP of **0.641**. The DBN and DAE also produce very close results. The Multiple Kernel Learning approach proposed by Guillaumin et al. (2010) obtained a MAP of 0.623 where they used a much larger set of image features (37,152 dimensions). TagProp (Verbeek et al., 2010) obtained a MAP of 0.640 which is comparable to DBMs again using a much larger set of features.

In terms of Prec@50, the DBM achieves a score of 0.888±0.004. The DBN and DAE score 0.887±0.003 and 0.888 ± 0.004 respectively. Therefore, all the deep models do about the same in terms of this metric.

Learning a deep hierarchy of features is widely believed to be the reason why deep models have been successful in a number of machine learning tasks. In order to better understand the properties of different layers in the network, we evaluate the quality of representation at each layer of the network. We do this by measuring MAP obtained by logistic regression classifiers on the representation at different layers of the network. We choose a simple classifier so that the MAP results represent a good measure of



(a) (b)

Figure 2.9: Mean Average Precision (MAP) obtained by applying logistic regression to representations learned at different layers. **Left :** Comparison of different deep models - Deep Belief Nets, Denoising Autoencoders and Deep Boltzmann Machines. All model have the same architecture and same number of parameters. **Right:** Comparison of DBMs of different depths with SVMs and MKL models. Observe that adding depth improves performance.

the representation's discriminative ability. Fig. 2.9a compares different deep models. In all the models, MAP increases as we go from the input layer towards the joint hidden layer from either side. This shows that higher level representations become increasingly good at discovering useful features. It is interesting to note that the performance of the DBM's hidden layers increases rapidly with depth whereas that for the DBN seems to stagnate at the second layer. At the middle (joint) layer, the performance of both models increases tremendously. This behavior points to an important property of DBMs. Intuitively, the joint generative training of all the layers allows information to flow more readily between the image and text pathways. This comparison empirically verifies the observations made in Sec. 2.5.1.

Next, we investigate the effect of depth more closely on DBMs. The question that we try to answer here is how many intervening layers of hidden units should we put between the image and text modalities. It is useful to think of intervening layers as shown in Fig. 2.4c. We could just have one intervening layer, creating an RBM (image input—joint hidden layer—text input). A two-layered DBM would have 3 intervening layers (image input—image hidden 1—joint hidden—text hidden 1—text input), and so on. Fig. 2.9b compares the layer-wise performance of these models (RBM, 2-layer DBM and 3-layer DBM). The performance of other models is also indicated with horizontal lines. Comparing the performance of the joint hidden layer across the three models, we can see that having more intervening layers leads to better performance. The incremental utility of adding more layers seems to decrease.

Unimodal Inputs

In a multimodal data setting, it is very common for some data points to be missing some data modalities. For example, there may be images which do not have captions or tags. This raises interesting questions—Can we use a model that was trained on images and text, when we only have images at test time? Can this model do better than one that was trained on images alone? For multimodal DBMs the answer is

Model	MAP	Prec@50
Image LDA (Huiskes et al., 2010)	0.315	-
Image SVM (Huiskes et al., 2010)	0.375	-
Image DBN	0.463 ± 0.004	0.801 ± 0.005
Image DBM	0.469 ± 0.005	0.803 ± 0.005
Multimodal DBM (generated text)	**0.531 ± 0.005**	**0.832 ± 0.004**

Table 2.4: Unimodal Classification Results. Mean Average Precision (MAP) and precision@50 obtained by different models. A similar set of input features is used across all models.

affirmative. In this section, we describe an experiment to demonstrate this.

The task is the same as in the previous experiment. We trained a DBM using the unlabelled data and fine-tuned it for discrimination as before. The only difference is that at test time, the model was given only image inputs and used the DBM to generate and fill in missing data. This was done by mean-field updates. We also tried Gibbs sampling and found that it work just as well but with more variance.

We compare the Multimodal DBM with models that were trained using only images. We compare with baseline (RBF-kernel) SVM and LDA results, using a restricted feature set which is similar to that used in Huiskes et al. (2010). Table. 2.4 shows that the LDA and SVM models achieve a MAP of 0.315 and 0.375, respectively. A DBN trained on the similar image features improves this to 0.463. A DBM further improves this to 0.469. In both these cases, pretraining was done using images from the unlabelled set. Both models had the same number of layers and same number of hidden units in each layer. Next, we used a Multimodal DBM to infer the text input and hidden representations at each layer (using mean-field updates). At test time, these representations along with the image features were given as input to the discriminatively fine-tuned DBM. This achieved a significantly higher MAP of 0.531.

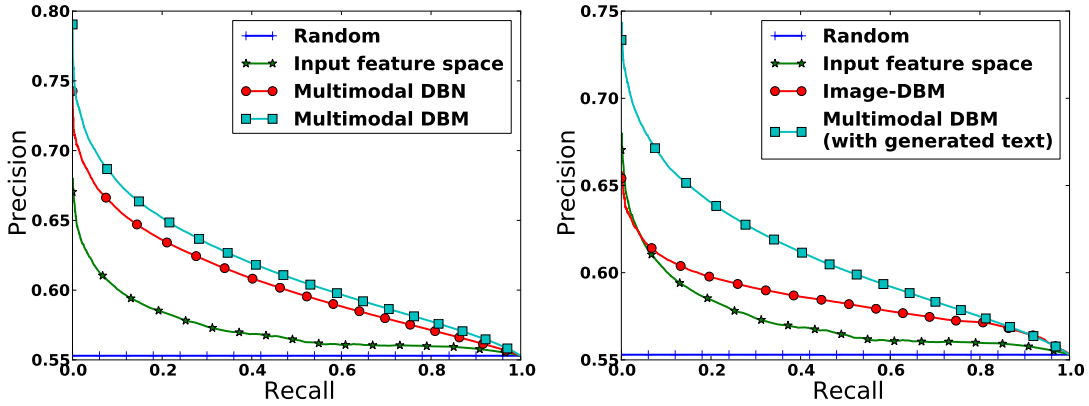
This result shows that the DBM can generate meaningful text that serves as a plausible proxy for missing data. This further suggests that *learning multimodal features helps even when some modalities are absent at test time*. The model learns much better features when it has access to multiple modalities because it is being asked to discover features that explain both modalities simultaneously. This can be interpreted as a regularization effect, where instead of the asking the model to be simple or sparse, we ask it to explain an alternative "view" of the data which lies on a very different manifold but shares essential discriminative characteristics with the original view.

2.6.4 Retrieval Tasks

The next set of experiments was designed to evaluate the quality of the learned joint representation for retrieval purposes. A database of images was created by randomly selecting 5000 image-text pairs from the test set. We also randomly selected a disjoint set of 1000 images to be used as queries. Each query contained both image and text modalities. Each data point has 38 labels. Using these, binary relevance labels were created by assuming that if any of the 38 labels overlapped between a query and a data point, then that data point is relevant to the query.

Multimodal Queries

Fig. 2.10a shows the precision-recall curves for the DBM, DBN, and DAE models (averaged over all queries). For each model, all queries and all points in the database were mapped to the joint hidden representation under that model. Cosine similarity function was used to match queries to data points.



(a) Multimodal Queries (b) Unimodal Queries

Figure 2.10: Precision-Recall curves for Retrieval Tasks.

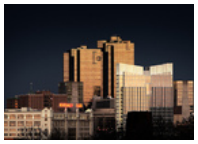
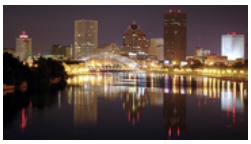
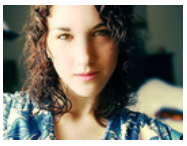
Multimodal Query	Top 4 retrieved results			
				
hongkong, causewaybay, shoppingcentre, building, mall	howell, bridge, genesee, river, rochester, downtown, building	london, uk, night, skyline, river, thames, lights, bridge	edinburgh, scotland, dusk, bank	arcoiris, fincadehierro, lluvia, sannicolas, valencia
				
me, myself, eyes, blue, hair	urban, me, abigfave, fiveflickrfavs,	trisha, mynewcamera, lake, field, girl	me, ofme, self, selfportrait	pink, prettyinpink, explored

Figure 2.11: Retrieval Results for Multimodal Queries from the DBM model.

The DBM model performs the best among the compared models achieving a MAP of 0.622. This is slightly better than the performance of the autoencoder and DBN models which achieve a MAP of 0.612 and 0.609 respectively. Fig. 2.11 shows some examples of multimodal queries and the top 4 retrieved results. Note that even though there is little overlap in terms of text, the model is able to perform well.

Unimodal Queries

The DBM model can also be used to query for images alone. Fig. 2.10b shows the precision-recall curves for the DBM model along with other unimodal models. Each model received the same set of test image queries as input. The joint hidden representation was inferred keeping the text input layer unclamped. Using this representation, the DBM model was able to achieve far better results than any unimodal method (MAP of 0.614 as compared to 0.587 for an Image-DBM and 0.578 for an Image-DBN).

Image				
Generated Tags	water, glass, beer, bottle, drink, wine, bubbles, splash, drops, drop	portrait, women, army, soldier, mother, postcard, soldiers	nikon, d200, nikkor, d50, tamron, d300, d90, f28, sb600, d60	obama, barackobama, election, politics, president, hope, change

Figure 2.12: Examples where the DBM does not work well.

2.6.5 When Does the Model Not Work?

In this section, we analyze the DBM model to understand when it fails to work and what exactly goes wrong. Fig. 2.12 shows some examples where the model fails to generate meaningful text. To diagnose the problem, we looked at the Markov Chains that lead to these results. By visual observation, it was clear that some of these chains got stuck in a region of space and never came out. This happened often when the text sampler reached the space of frequently occurring tags, such as those which refer to camera brands or lens specifications. These tags occur across all kinds of images and seem to take up a huge probability mass under the model independent of the image.

2.7 Experimental Results with Video-Audio Data

We next demonstrate our approach on video-audio bimodal data. We use data sets that consist of videos of lip movements along with the sound recordings of the words being spoken. This setting has been previously explored in the context of deep multimodal learning by Ngiam et al. (2011) using sparse denoising autoencoders.

2.7.1 Preprocessing and Data Sets

We represent the auditory information using 40 dimensional log-filter banks along with temporal derivatives to create a 120-d frame for 20 ms speech windows with a stride of 10 ms. Similar to Ngiam et al. (2011) we extract 60×80 mouth regions from the video using a simple object detector (Dalal and Triggs, 2005). The detections were cleaned by median filtering. The extracted mouth regions were compressed to 32 dimensions with PCA. Temporal derivatives were then added to create a 96 dimensional representation for each frame. We combined several data sets in this experiment.

CUAVE (Patterson et al., 2002): This data set consists of 36 speakers speaking the digits 0 to 9. The data set has each speaker speaking with different facial orientations (front and sideways) and speaking speeds. We exclude the sideways oriented portions of the data set for simplicity. We use half the speakers for testing and the other half for training.

AVLetters (Matthews et al., 2002): This data set consists of 10 speakers speaking letters A-Z three times each. This data set does not come with raw audio and was used for unsupervised pretraining of the video pathway. We treat this data set as if it were missing audio and evaluate the DBM's ability of fill in the missing data and use it for classification.

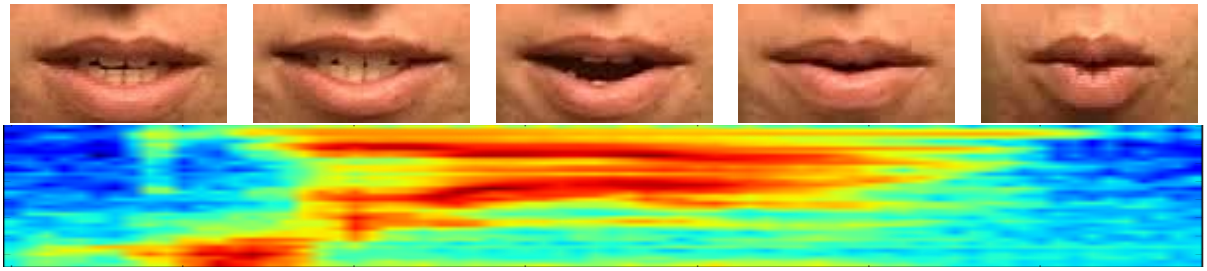


Figure 2.13: An example of audio-video data extracted from the CUAVE data set.

AVLetters 2 (Cox et al., 2008): This data set consists of high resolution recordings from 5 speakers speaking letters A-Z. The videos were down-sampled. This data set was used for unsupervised training of the entire model.

TIMIT (Fisher et al., 1986): This data set consists of recordings from 680 speakers covering 8 major dialects of American English reading ten phonetically-rich sentences in a controlled environment. We used this for the unsupervised pretraining of the auditory pathway.

In addition to these, Ngiam et al. (2011) use the Stanford Data Set which consists of 23 speakers speaking the letters A-Z and digits 0-9. However, this data set is not publicly available yet and we were unable to use it. Since all of the above data sets differ in terms of video recording environments, we use PCA in the hope of ameliorating some of these differences. In all the experiments, all available data was used for unsupervised pretraining. Fig. 2.13 shows an example of the data.

2.7.2 Model Description

A Multimodal DBM was trained with 4 consecutive image frames and 10 consecutive audio frames since they roughly correspond to same amount of time. Both pathways used Gaussian RBMs as the first layer, as defined in Eq. 2.9. The auditory pathway consisted of 1200 input units followed by 2 layers of 1024 hidden units. The visual pathway had 384 input units followed by 2 layers of 1024 hidden units. The joint layer had 2048 hidden units.

The task was to label each utterance with the digit or letter that was being uttered. Different utterances had different lengths. We obtained a fixed length representation by applying average pooling on the features obtained from the joint layer. In addition, we divided each utterance into 3 equal splits and average pooled the features over those separately. These 4 sets of pooled features were concatenated to form the multimodal representation of the input. We then used a linear SVM to classify based on these representations. This is the same as the method used in Ngiam et al. (2011). No discriminative fine-tuning of the DBM was performed.

2.7.3 Classification Results

We report the results of two classification experiments. In the first experiment, we classify utterances from the CUAVE data set into 10 digit classes. We use the DBM to extract features using both video and audio inputs. We compare this to a DBN, DAE (Ngiam et al., 2011) and various other methods. The results are shown in Table. 2.5. Linear SVM on concatenated video and audio features serves as a baseline, which achieves 63.5% accuracy. A video-only RBM achieves 65.4%, which can be improved to 67.2% with a 3-layer DBN and to 67.8% with a 3-layer DBM. The denoising autoencoder achieves an even better performance of 68.7%. Ngiam et al. (2011) showed that adding audio features seems to

Method	Classification accuracy %
Concatenated video and audio features	63.5
Video RBM (Ngiam et al., 2011)	65.4 ± 0.6
Multimodal DAE (Ngiam et al., 2011)	66.7
Multimodal DBN	67.2 ± 0.9
Video DBM	67.8 ± 1.1
Video DAE (Ngiam et al., 2011)	68.7 ± 1.8
Multimodal DBM	**69.0 ± 1.5**
Discrete Cosine Transform (Gurban and Thiran, 2009)	64
Active Appearance Model (AAM) (Papandreou et al., 2007)	75.7
Fused Holistic + Patch (Lucey and Sridharan, 2006)	77.08
Visemic AAM (Papandreou et al., 2009)	83

Table 2.5: Classification results on the CUAVE data set.

Method	Classification accuracy
Video features (Ngiam et al., 2011)	46.2
Video RBM (Ngiam et al., 2011)	54.2 ± 3.3
Multimodal DAE (Ngiam et al., 2011)	59.2
Video DBM	61.8 ± 2.5
Multimodal DBN	63.2 ± 2.1
Video DAE (Ngiam et al., 2011)	64.4 ± 2.4
Multimodal DBM	**64.7 ± 2.5**
Multiscale Spatial Analysis (Matthews et al., 2002)	44.6
Local Binary Pattern (Zhao et al., 2009)	58.85

Table 2.6: Classification results on the AVLetters data set.

hurt the performance of DAEs, reducing it down to 66.7%. The Multimodal DBM does not suffer from adding audio features and improves the performance slightly to 69.0%. However, this is not a significant improvement over the Video DAE. Note that the DBM was trained on less data compared to the Video DAE of Ngiam et al. (2011). The Multimodal DBM does improve significantly on the performance of the Video DBM trained on the same amount of data.

The performance of the deep models is much worse than that of Active Appearance Models (Papandreou et al., 2007, 2009) and Patch-based methods (Lucey and Sridharan, 2006). However, these models use a different train-test split and specialized image preprocessing techniques that are specifically designed for visual speech recognition tasks.

In our second experiment, we try to classify utterances from the AVLetters data set into 26 letter classes. In this case the audio input is considered missing and we use the DBM to infer the joint hidden representation keeping the audio input unclamped. We do the same for a DBN as well as compare to DAEs and other methods. Table. 2.6 shows the results.

The baseline model which uses the preprocessed video features achieves 46.2% accuracy. An RBM on the same features achieves 54.2%, whereas a 3-layer DBM gets 61.8% and a DAE gets 64.4%. However, the Multimodal DAE again suffers from adding audio features at test time compared to a Video DAE, getting an accuracy of 59.2%. The Multimodal DBM, on the other hand, improves over the Video DBM and gets 64.7%, essentially matching the performance of the Video DAE (even though it used less data).

These experiments show that the Multimodal DBM model can effectively combine features across modalities. It consistently shows improvements over training on unimodal data, even when only unimodal inputs are given at test time.

2.8 Conclusion

In this chapter, we developed a Deep Boltzmann Machine model for learning representations of multimodal data. We described how this model can be used to solve multiple tasks of interest. We evaluated the model's ability to *fuse* multiple data modalities into a unified representation by using the learned representation for retrieval and for solving discriminative tasks. In addition, we evaluated the model's ability to *generate* a missing modality conditioned on other modalities by trying to do unimodal retrieval. We discovered that multimodal training is a useful *regularizer* in that models trained on multimodal data (but given unimodal data at test time) outperform models trained only on unimodal data. This experimentally validates that trying to explain diverse modalities simulateneously aids generalization. Our model achieves strong classification results on the bi-modal MIR-Flickr data set and performs well on CUAVE and AVLetters video-audio data sets, demonstrating the usefulness of this approach.

Since the publication of this work, significant progress has been made in deep learning and it is pertinent to put this work in the context of the current state of research in this field. Firstly, in this work we started off by representing images using hand-designed feature descriptors. It has now been convincingly shown that it is much better to use learned convolutional weights to work with images. Features learned by convolutional neural nets trained on large supervised datasets can be substituted for the hand-designed ones used in this work. Moreover, the Boltzmann machine itself can be made convolutional, making it practical to work directly with high-resolution images. This approach has been applied for modeling shapes (Eslami et al., 2014). Stacks of convolutional RBMs have also been used to model images (Lee et al., 2009b). Thus, the idea of having convolutional weights can be applied to Boltzmann Machines as well.

Secondly, unsupervised learning models have been proposed that are easier to train than Boltzmann Machines (because they do not rely on mixing Markov chains), but still try to retain some elements of top-down and bottom-up computation. A good example is the Multi-Prediction Boltzmann Machine (Goodfellow et al., 2013a) in which the computation graph used for alternating Gibbs sampling in unrolled and the model is trained like a denoising auto-encoder with noise being injected at multiple places within the unrolled network. Ladder Networks (Valpola, 2014; Rasmus et al., 2015) can also be interpreted in terms of combining bottom-up and top-down influences. In this case, the model is similar to a denoising auto-encoder except that any hidden state in the decoder is obtained by combining top-down inference (from the higher-level hidden state) and bottom-up inference (from the corresponding encoder layer).

Lastly, other unsupervised techniques based on Generative Adversarial Networks (Goodfellow et al., 2014) and variational autoencoders (Kingma and Welling, 2013; Rezende et al., 2014) have been extremely successful at modeling complex data distribution such as raw pixels, sound, and videos. While these models have shown impressive results, it is not obvious how they encorporate properties that come naturally with Boltzmann Machines – such as the ability to repeatedly query its associative memory, jump to different modes, and search across multiple hypotheses to explain the data. Therefore, Boltzmann Machines are arguably interesting models and present potentially fruitful avenues for future research, especially along the lines of adding more structure in them (both in the connectivity of the units and the form of the energy function) and applying them to problems where exploring a space of hypotheses is crucial.

Chapter 3

Over-Replicated Softmax Model

Text documents are the primary means of communicating and storing information across the world, be it in the form of news stories, emails, webpages, tweets, or research articles. Therefore, representing the information content of a document in a form that is suitable for solving problems such as classification or similarity-based retrieval is an important task. The aim of topic modeling is to create such representations by discovering the latent topic structure in collections of documents.

3.1 Overview

In this chapter, we introduce a Boltzmann Machine based topic model, which we call the Over-Replicated Softmax Model, that is suitable for extracting a distributed semantic representation of a document expressed as a bag-of-words vector. This model is an extension of the Replicated Softmax Model (Salakhutdinov and Hinton, 2009b). It can be described as a two hidden layer Boltzmann Machine, where the first hidden layer consists of standard binary hidden units, but the second hidden layer consists of softmax units which correspond to latent words. Since the visible layer consists of observed words (represented as replicated softmaxes), having these latent words adds even more replicated softmaxes to the model. Hence, the name Over-Replicated Softmax Model.

A common approach to topic modeling is to build a generative probabilistic model of the bag of words in a document. Directed graphical models, such as Latent Dirichlet Allocation (LDA), CTM, and H-LDA have been extensively used for this (Blei et al., 2003, 2010; Mimno and McCallum, 2008). Non-parametric extensions of these models have also been quite successful (Teh et al., 2006; Blei, 2012; Griffiths and Steyvers, 2004). Even though exact inference in these models is hard, efficient inference schemes, including stochastic variational inference, online inference, and collapsed Gibbs have been developed that make it feasible to train and use these methods (Teh et al., 2008; Wang and Blei, 2009; Canini et al., 2009). Another approach is to use undirected graphical models such as the Replicated Softmax model (Salakhutdinov and Hinton, 2009b). In this model, inferring latent topic representations is exact and efficient. However, training is still hard and often requires careful hyperparameter selection. These models typically perform better than LDA in terms of both the log probability they assign to unseen data and their document retrieval and document classification accuracy. Neural network based approaches, such as Neural Autoregressive Density Estimators (DocNADE) (Larochelle and Lauly, 2012), have been to shown to outperform the Replicated Softmax model.

The Replicated Softmax model is a family of Restricted Boltzmann Machines (RBMs) with shared parameters. An important feature of RBMs is that they solve the "explaining-way" problem of directed graphical models by having a complementary prior over hidden units. However, this implicit prior may not be the optimal prior to use and having some degree of flexibility in defining the prior may be advantageous. One way of adding this additional degree of flexibility, while still avoiding the explaining-away problem, is to learn a two hidden layer Deep Boltzmann Machine (DBM). This model adds another layer of hidden units on top of the first hidden layer with bi-partite, undirected connections. The new connections come with a new set of weights. However, this additional implicit prior comes at the cost of more expensive training and inference. Therefore, we have the following two extremes: On one hand, RBMs can be efficiently trained (e.g. using Contrastive Divergence), inferring the state of the hidden units is exact, but the model defines a rigid, implicit prior. On the other hand, a two hidden layer DBM defines a more flexible prior over the hidden representations, but training and performing inference in a DBM model is considerably harder.

In this chapter, we try to find middle ground between these extremes and build a model that tries to combine the best elements of both. We introduce a two hidden layer DBM model, which we call the Over-Replicated Softmax model. This model is easy to train, has fast approximate inference and still retains some degree of flexibility towards manipulating the prior. Our experiments show that this flexibility is enough to improve on the performance of the standard Replicated Softmax model, both as a generative model and as a feature extractor. The model also outperforms LDA and DocNADE in terms of classification and retrieval tasks.

Before we describe our model, we briefly review the Replicated Softmax model which is a stepping stone towards the proposed Over-Replicated Softmax model.

3.2 Replicated Softmax Model

This model has been described in detail in the previous chapter (Sec. 2.2.3). The description here will be brief and mainly serve to introduce the notation needed to describe the Over-Replicated Softmax model. Let K be the dictionary size, N be the number of words appearing in a document, and $\mathbf{h} \in \{0,1\}^F$ be binary stochastic hidden topic features. Let $\mathbf{V}$ be a $N \times K$ observed binary matrix with $v_{ik} = 1$ if visible unit i takes on the k^{th} value. We define the energy of the state $\{\mathbf{V}, \mathbf{h}\}$ as :

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}; \boldsymbol{\theta}) &= -\sum_{i=1}^{N}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ijk} h_j v_{ik} \\ &\quad -\sum_{i=1}^{N}\sum_{k=1}^{K} v_{ik} b_{ik} - N\sum_{j=1}^{F} h_j a_j, \end{aligned} \tag{3.1}$$

where $\boldsymbol{\theta} = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$ are the model parameters; W_{ijk} is a symmetric interaction term between visible unit i that takes on value k, and hidden feature j, b_{ik} is the bias of unit i that takes on value k, and a_j is the bias of hidden feature j. The probability that the model assigns to a visible binary matrix $\mathbf{V}$ is:

$$\begin{aligned} P(\mathbf{V}; \boldsymbol{\theta}) &= \frac{1}{\mathcal{Z}(\boldsymbol{\theta}, N)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{V}, \mathbf{h}; \boldsymbol{\theta})\right) \\ \mathcal{Z}(\boldsymbol{\theta}, N) &= \sum_{\mathbf{V}'}\sum_{\mathbf{h}'} \exp\left(-E(\mathbf{V}', \mathbf{h}'; \boldsymbol{\theta})\right), \end{aligned} \tag{3.2}$$

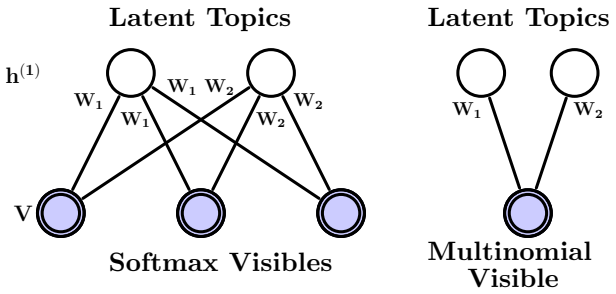


Figure 3.1: The Replicated Softmax model. The top layer represents a vector $\mathbf{h}$ of stochastic, binary topic features and the bottom layer represents softmax visible units $\mathbf{V}$. All visible units share the same set of weights, connecting them to binary hidden units. **Left:** The model for a document containing $N = 3$ words. **Right:** A different interpretation of the Replicated Softmax model, in which N softmax units with identical weights are replaced by a single multinomial unit which is sampled N times.

where $\mathcal{Z}(\boldsymbol{\theta}, N)$ is known as the partition function, or normalizing constant. This defines an RBM for documents of size N. The Replicated Softmax model is a collection of such RBMs that cover all possible document sizes. All these RBMs share the same model parameters. Fig. 3.1 shows an example of an RBM for $N = 3$. Since the order of the words can be ignored, all of these softmax units can share the same set of weights, connecting them to binary hidden units. In this case, the energy of the state $\{\mathbf{V}, \mathbf{h}\}$ for a document that contains N words is defined as:

$$E(\mathbf{V}, \mathbf{h}) \;=\; -\sum_{j=1}^{F}\sum_{k=1}^{K} W_{jk} h_j \hat{v}_k - \sum_{k=1}^{K} \hat{v}_k b_k - N \sum_{j=1}^{F} h_j a_j,$$

where $\hat{v}_k = \sum_{i=1}^{N} v_i^k$ denotes the count for the k^{th} word. The conditional distributions are given by softmax and logistic functions:

$$P(h_j^{(1)} = 1) \;=\; \sigma\Big(\sum_{k=1}^{K} W_{jk} \hat{v}_k + N a_j\Big), \tag{3.3}$$

$$P(v_{ik} = 1) \;=\; \frac{\exp\Big(\sum_{j=1}^{F} W_{jk} h_j^{(1)} + b_k\Big)}{\sum_{k'=1}^{K} \exp\Big(\sum_{j=1}^{F} W_{jk'} h_j^{(1)} + b_{k'}\Big)}. \tag{3.4}$$

3.3 Over-Replicated Softmax Model

The Over-Replicated Softmax model is a family of two hidden layer Deep Boltzmann Machines (DBM). Let us consider constructing a Boltzmann Machine with two hidden layers for a document containing N words, as shown in Fig. 3.2. The visible layer $\mathbf{V}$ consists of N softmax units. These units are connected to a binary hidden layer $\mathbf{h}^{(1)}$ with shared weights, exactly like in the Replicated Softmax model in Fig. 3.1. The second hidden layer consists of M softmax units represented by $\mathbf{H}^{(2)}$. Similar to $\mathbf{V}$, $\mathbf{H}^{(2)}$ is an $M \times K$ binary matrix with $h_{mk}^{(2)} = 1$ if the m-th hidden softmax unit takes on the k-th value.

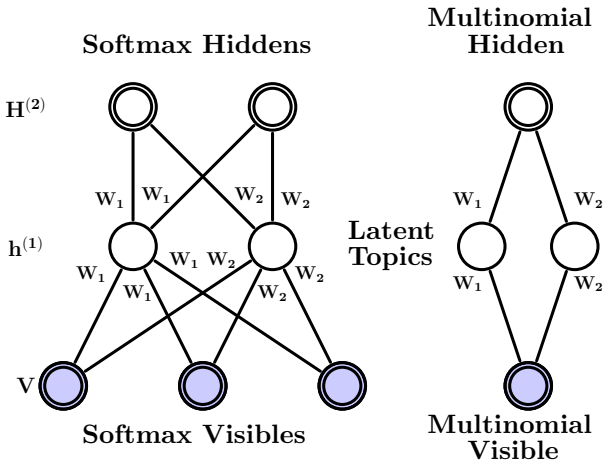


Figure 3.2: The Over-Replicated Softmax model. The bottom layer represents softmax visible units $\mathbf{V}$. The middle layer represents binary latent topics $\mathbf{h}^{(1)}$. The top layer represents softmax hidden units $\mathbf{H}^{(\mathbf{2})}$. All visible and hidden softmax units share the same set of weights, connecting them to binary hidden units. **Left:** The model for a document containing $N = 3$ words with $M = 2$ softmax hidden units. **Right:** A different interpretation of the model, in which N softmax units with identical weights are replaced by a single multinomial unit which is sampled N times and the M softmax hidden units are replaced by a multinomial unit sampled M times.

The energy of the joint configuration $\{\mathbf{V}, \mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}\}$ is defined as:

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}; \boldsymbol{\theta}) = &- \sum_{i=1}^{N}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{ijk}^{(1)} h_j^{(1)} v_{ik} \\ &- \sum_{i'=1}^{M}\sum_{j=1}^{F}\sum_{k=1}^{K} W_{i'jk}^{(2)} h_j^{(1)} h_{i'k}^{(2)} - \sum_{i=1}^{N}\sum_{k=1}^{K} v_{ik} b_{ik}^{(1)} \\ &- (M+N)\sum_{j=1}^{F} h_j^{(1)} a_j - \sum_{i=1}^{M}\sum_{k=1}^{K} h_{ik}^{(2)} b_{ik}^{(2)} \end{aligned} \tag{3.5}$$

where $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{a}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}\}$ are the model parameters. Similar to the Replicated Softmax model, we create a separate document-specific DBM with as many visible softmax units as there are words in the document. We also fix the number M of the second-layer softmax units across all documents. All of the first layer softmax units share the same set of weights (which amounts to ignoring the word order and treating the document as a bag of words). Moreover, we also tie the first and second layer weights. Thus we have $W_{ijk}^{(1)} = W_{i'jk}^{(2)} = W_{jk}$ and $b_{ik}^{(1)} = b_{i'k}^{(2)} = b_k$. Compared to the standard Replicated Softmax model, this model has more replicated softmaxes (hence the name "Over-Replicated"). Unlike the visible softmaxes, these additional softmaxes are unobserved and constitute a second hidden layer[1]. The energy can be simplified to:

$$\begin{aligned} E(\mathbf{V}, \mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}; \boldsymbol{\theta}) = &- \sum_{j=1}^{F}\sum_{k=1}^{K} W_{jk} h_j^{(1)} \left(\hat{v}_k + \hat{h}_k^{(2)}\right) \\ &- \sum_{k=1}^{K} \left(\hat{v}_k + \hat{h}_k^{(2)}\right) b_k - (M+N)\sum_{j=1}^{F} h_j^{(1)} a_j \end{aligned} \tag{3.6}$$

where $\hat{v}_k = \sum_{i=1}^{N} v_{ik}$ denotes the count for the k^{th} word in the input and $\hat{h}_k^{(2)} = \sum_{i=1}^{M} h_{ik}^{(2)}$ denotes the

[1]This model can also be seen as a Dual-Wing Harmonium (Xing et al., 2005) in which one wing is unclamped.

count for the k^{th} "latent" word in the second hidden layer. The joint probability distribution is defined as:

$$P(\mathbf{V}, \mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}; \boldsymbol{\theta}) = \frac{\exp\left(-E(\mathbf{V}, \mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}; \boldsymbol{\theta})\right)}{\mathcal{Z}(\boldsymbol{\theta}, N)},$$

Note that the normalizing constant depends on the number of words N in the corresponding document, since the model contains as many visible softmax units as there are words in the document. So the model can be viewed as a family of different-sized DBMs that are created for documents of different lengths, but with a fixed-sized second-layer.

A pleasing property of the Over-Replicated Softmax model is that it has exactly the same number of trainable parameters as the Replicated Softmax model. However, the model's marginal distribution over $\mathbf{V}$ is different, as the second hidden layer provides an additional implicit prior. The model's prior over the latent topics $\mathbf{h}^{(1)}$ can be viewed as the geometric mean of the two probability distributions: one defined by an RBM composed of $\mathbf{V}$ and $\mathbf{h}^{(1)}$, and the other defined by an RBM composed of $\mathbf{h}^{(1)}$ and $\mathbf{H}^{(\mathbf{2})}$:[2]

$$P(\mathbf{h}^{(1)}; \boldsymbol{\theta}) = \frac{1}{\mathcal{Z}(\boldsymbol{\theta}, N)} \underbrace{\left(\sum_{\mathbf{v}} \exp\left(\sum_{j=1}^{F}\sum_{k=1}^{K} W_{jk}\hat{v}_k h_j^{(1)}\right)\right)}_{\text{RBM with } \mathbf{h}^{(1)} \text{ and } \mathbf{v}}$$
$$\underbrace{\left(\sum_{\mathbf{H}^{(\mathbf{2})}} \exp\left(\sum_{j=1}^{F}\sum_{k=1}^{K} W_{jk}\hat{h}_k^{(2)} h_j^{(1)}\right)\right)}_{\text{RBM with } \mathbf{h}^{(1)} \text{ and } \mathbf{H}^{(\mathbf{2})}}. \tag{3.7}$$

Observe that $\sum_{k=1}^{K} \hat{v}_k = N$ and $\sum_{k=1}^{K} \hat{h}_k^{(2)} = M$, so the strength of this prior can be varied by changing the number M of second-layer softmax units. For example, if $M = N$, then the model's marginal distribution over $\mathbf{h}^{(1)}$, defined in Eq. 3.7, is given by the product of two distributions each of which comes from the same number of words. In this DBM, the second-layer performs roughly 1/2 of the modeling work compared to the first layer (Salakhutdinov and Hinton, 2012). Hence, for documents containing few words ($N \ll M$) the prior over hidden topics $\mathbf{h}^{(1)}$ will be dominated by the second-layer, whereas for long documents ($N \gg M$) the effect of having a second-layer will diminish. As we show in our experimental results, having this additional flexibility in terms of defining an implicit prior over $\mathbf{h}^{(1)}$ significantly improves model performance, particularly for small and medium-sized documents.

3.3.1 Learning

Let $\mathbf{h} = \{\mathbf{h}^{(1)}, \mathbf{H}^{(\mathbf{2})}\}$ be the set of hidden units in the two-layer DBM. Given a collection of L documents $\{\mathbf{V}\}_{l=1}^{L}$, the derivative of the log-likelihood with respect to model parameters W takes the form:

$$\frac{1}{L}\sum_{l=1}^{L} \frac{\partial \log P(\mathbf{V_l}; \boldsymbol{\theta})}{\partial W_{jk}} = \mathbb{E}_{P_{\text{data}}}\left[(\hat{v}_k + \hat{h}_k^{(2)}) h_j^{(1)}\right] - \mathbb{E}_{P_{\text{Model}}}\left[(\hat{v}_k + \hat{h}_k^{(2)}) h_j^{(1)}\right],$$

[2] We omit the bias terms for clarity of presentation.

where $\mathbb{E}_{P_{\text{data}}}[\cdot]$ denotes an expectation with respect to the data distribution $P_{\text{data}}(\mathbf{h}, \mathbf{V}) = P(\mathbf{h}|\mathbf{V}; \boldsymbol{\theta})P_{\text{data}}(\mathbf{V})$, with $P_{\text{data}}(\mathbf{V}) = \frac{1}{L}\sum_l \delta(\mathbf{V} - \mathbf{V}_l)$ representing the empirical distribution, and $\mathbb{E}_{P_{\text{Model}}}[\cdot]$ is an expectation with respect to the distribution defined by the model. Similar to the Replicated Softmax model, exact maximum likelihood learning is intractable, but approximate learning can be performed using a variational approach (Salakhutdinov and Hinton, 2009a). We use mean-field inference to estimate data-dependent expectations and an MCMC based stochastic approximation procedure to approximate the model's expected sufficient statistics.

Consider any approximating distribution $Q(\mathbf{h}|\mathbf{V}; \boldsymbol{\mu})$, parameterized by a vector of parameters $\boldsymbol{\mu}$, for the posterior $P(\mathbf{h}|\mathbf{V}; \boldsymbol{\theta})$. Then the log-likelihood of the DBM model has the following variational lower bound:

$$\log P(\mathbf{V}; \boldsymbol{\theta}) \ \geq \ \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{V}; \boldsymbol{\mu}) \log P(\mathbf{V}, \mathbf{h}; \boldsymbol{\theta}) + \mathcal{H}(Q),$$

where $\mathcal{H}(\cdot)$ is the entropy functional. The bound becomes tight if and only if $Q(\mathbf{h}|\mathbf{V}; \boldsymbol{\mu}) = P(\mathbf{h}|\mathbf{V}; \boldsymbol{\theta})$.

For simplicity and speed, we approximate the true posterior $P(\mathbf{h}|\mathbf{V}; \boldsymbol{\theta})$ with a fully factorized approximating distribution over the two sets of hidden units, which corresponds to the so-called mean-field approximation:

$$Q^{MF}(\mathbf{h}|\mathbf{V}; \boldsymbol{\mu}) = \prod_{j=1}^{F} q(h_j^{(1)}|\mathbf{V}) \prod_{i=1}^{M} q(h_i^{(2)}|\mathbf{V}), \tag{3.8}$$

where $\boldsymbol{\mu} = \{\boldsymbol{\mu}^{(1)}, \boldsymbol{\mu}^{(2)}\}$ are the mean-field parameters with $q(h_j^{(1)} = 1) = \mu_j^{(1)}$ and $q(h_{ik}^{(2)} = 1) = \mu_k^{(2)}$, $\forall i \in \{1, \ldots, M\}$, s.t. $\sum_{k=1}^{K} \mu_k^{(2)} = 1$. Note that due to the shared weights across all of the hidden softmaxes, $q(h_{ik}^{(2)})$ does not dependent on i. In this case, the variational lower bound takes a particularly simple form:

$$\begin{aligned}\log P(\mathbf{V}; \boldsymbol{\psi}) &\geq \sum_{\mathbf{h}} Q^{MF}(\mathbf{h}|\mathbf{V}; \boldsymbol{\mu}) \log P(\mathbf{V}, \mathbf{h}; \boldsymbol{\psi}) + \mathcal{H}(Q^{MF}) \\ &\geq \left(\hat{\mathbf{v}}^\top + M\boldsymbol{\mu}^{(2)\top}\right) \mathbf{W}\boldsymbol{\mu}^{(1)} - \log \mathcal{Z}(\boldsymbol{\theta}, N) + \mathcal{H}(Q^{MF}),\end{aligned}$$

where $\hat{\mathbf{v}}$ is a $K \times 1$ vector, with its k^{th} element $\hat{v}_k$ containing the count for the k^{th} word. Since $\sum_{k=1}^{K} \hat{v}_k = N$ and $\sum_{k=1}^{K} \mu_k^{(2)} = 1$, the first term in the bound linearly combines the effect of the data (which scales as N) with the prior (which scales as M). For each training example, we maximize this lower bound with respect to the variational parameters $\boldsymbol{\mu}$ for fixed parameters $\boldsymbol{\psi}$, which results in the mean-field fixed-point equations:

$$\mu_j^{(1)} \ \leftarrow \ \sigma\Bigg(\sum_{k=1}^{K} W_{jk}\left(\hat{v}_k + M\mu_k^{(2)}\right)\Bigg), \tag{3.9}$$

$$\mu_k^{(2)} \ \leftarrow \ \frac{\exp\left(\sum_{j=1}^{F} W_{jk}\mu_j^{(1)}\right)}{\sum_{k'=1}^{K} \exp\left(\sum_{j=1}^{F} W_{jk'}\mu_j^{(1)}\right)}, \tag{3.10}$$

where $\sigma(x) = 1/(1 + \exp(-x))$ is the logistic function. To solve these fixed-point equations, we simply cycle through layers, updating the mean-field parameters within a single layer.

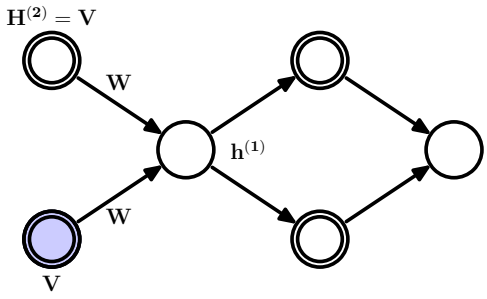


Figure 3.3: Pretraining a two-layer Boltzmann Machine using one-step contrastive divergence. The second hidden softmax layer is initialized to be the same as the observed data. The units in the first hidden layer have stochastic binary states, but the reconstructions of both the visible and second hidden layer use probabilities, so both reconstructions are identical.

Given the variational parameters $\boldsymbol{\mu}$, the model parameters $\boldsymbol{\psi}$ are then updated to maximize the variational bound using an MCMC-based stochastic approximation (Salakhutdinov and Hinton, 2009a; Tieleman, 2008; Younes, 1998). Let $\boldsymbol{\psi}_t$ and $\mathbf{x}_t = \{\mathbf{V}_t, \mathbf{h}^{(1)}{}_t, \mathbf{h}^{(2)}{}_t\}$ be the current parameters and the state. Then $\mathbf{x}_t$ and $\boldsymbol{\psi}_t$ are updated sequentially as follows: given $\mathbf{x}_t$, sample a new state $\mathbf{x}_{t+1}$ using alternating Gibbs sampling.

A new parameter $\boldsymbol{\psi}_{t+1}$ is then obtained by making a gradient step, where the intractable model's expectation $\mathbb{E}_{P_{\text{model}}}[\cdot]$ in the gradient is replaced by a point estimate at sample $\mathbf{x}_{t+1}$.

In practice, to deal with variable document lengths, we take a minibatch of data and run one Markov chain for each training case for a few steps. To update the model parameters, we use an average over those chains. Similar to Contrastive Divergence learning, in order to provide a good starting point for the sampling, we initialize each chain at $\hat{\mathbf{h}}^{(1)}$ by sampling from the mean-field approximation to the posterior $q(\mathbf{h}^{(1)}|\mathbf{V})$.

3.3.2 An Efficient Pretraining Algorithm

The proper training procedure for the DBM model described above is quite slow. This makes it very important to pretrain the model so that the model parameters start off in a nice region of space. Fortunately, due to parameter sharing between the visible and hidden softmax units, there exists an efficient pretraining method.

Consider a DBM with N observed and M hidden softmax units. Let us first assume that the number of hidden softmaxes M is the same as the number of words N in a given document. If we were given the initial state vector $\mathbf{H}^{(\mathbf{2})}$, we could train this DBM using one-step contrastive divergence with mean-field reconstructions of both the states of the visible and the hidden softmax units, as shown in Fig. 3.3. Since we are not given the initial state, one option is to set $\mathbf{H}^{(\mathbf{2})}$ to be equal to the data $\mathbf{V}$. Provided we use mean-field reconstructions for both the visible and second-layer hidden units, one-step contrastive divergence is then exactly the same as training a Replicated Softmax RBM with only one hidden layer but with bottom-up weights that are twice the top-down weights.

To pretrain a DBM with different number of visible and hidden softmaxes ($M \neq N$), we train an RBM with the bottom-up weights scaled by a factor of $1 + \frac{M}{N}$. In other words, in place of using $\mathbf{W}$ to

compute the conditional probability of the hidden units (see Eq. 3.3), we use $(1 + \frac{M}{N})\mathbf{W}$:

$$P(h_j^{(1)} = 1|\mathbf{V}) = \sigma\Big((1 + \frac{M}{N})\sum_{k=1}^{K} v_k W_{kj}\Big). \tag{3.11}$$

The conditional probability of the observed softmax units remains the same as in Eq. 3.4. This procedure is equivalent to training an RBM with $N + M$ observed visible units with each of the M extra units set to be the empirical word distribution in the document, i.e.. for $i \in \{N + 1, \ldots, N + M\}$,

$$v_{ik} = \frac{\sum_{j=1}^{N} v_{jk}}{\sum_{j=1}^{N}\sum_{k'=1}^{K} v_{jk'}}$$

Thus the M extra units are not 1-of-K, but represent distributions over the K words[3].

This way of pretraining the Over-Replicated Softmax DBMs with tied weights will not in general maximize the likelihood of the weights. However, in practice it produces models that reconstruct the training data well and serve as a good starting point for generative fine-tuning of the two-layer model.

3.3.3 Inference

The posterior distribution $P(\mathbf{h}^{(1)}|\mathbf{V})$ represents the latent topic structure of the observed document. Conditioned on the document, these activation probabilities can be inferred using the mean-field approximation used to infer data-dependent statistics during training.

A fast alternative to the mean-field posterior is to multiply the visible to hidden weights by a factor of $1 + \frac{M}{N}$ and approximate the true posterior with a single matrix multiply, using Eq. 3.11. Setting $M = 0$ recovers the proper posterior inference step for the standard Replicated Softmax model. This simple scaling operation leads to significant improvements. The results reported for retrieval and classification experiments used the fast pretraining and fast inference methods.

3.3.4 Controlling the Strength of the Prior

The number of hidden softmaxes M affects the strength of the additional prior. The value of M can be chosen using a validation set. Since the value of M is fixed for all Over-Replicated DBMs, the effect of the prior will be less for documents containing many words. This is particularly easy to see in Eq. 3.11. As N becomes large, the scaling factor approaches 1, diminishing the part of implicit prior coming from the M hidden softmax units. Thus the value of M can be chosen based on the distribution of lengths of documents in the corpus.

3.4 Experiments

In this section, we evaluate the Over-Replicated Softmax model both as a generative model and as a feature extraction method for retrieval and classification. Two datasets are used - 20 Newsgroups and Reuters Corpus Volume I (RCV1-v2).

[3]Note that when $M = N$, we recover the setting of having the bottom-up weights being twice the top-down weights.

3.4.1 Description of datasets

The 20 Newsgroups dataset consists of 18,845 posts taken from the Usenet newsgroup collection. Each post belongs to exactly one newsgroup. Following the preprocessing in Salakhutdinov and Hinton (2009b); Larochelle and Lauly (2012), the data was partitioned chronologically into 11,314 training and 7,531 test articles. After removing stopwords and stemming, the 2000 most frequent words in the training set were used to represent the documents.

The Reuters RCV1-v2 contains 804,414 newswire articles. There are 103 topics which form a tree hierarchy. Thus documents typically have multiple labels. The data was randomly split into 794,414 training and 10,000 test cases. The available data was already preprocessed by removing common stopwords and stemming. We use a vocabulary of the 10,000 most frequent words in the training dataset.

3.4.2 Training details

The Over-Replicated Softmax model was first pretrained with Contrastive Divergence using the weight scaling technique described in Sec. 3.3.2. Minibatches of size 128 were used. A validation set was held out from the training set for hyperparameter selection (1,000 cases for 20 newsgroups and 10,000 for RCV1-v2). The value of M and number of hidden units were chosen over a coarse grid using the validation set. Typically, $M = 100$ performed well on both datasets. Increasing the number of hidden units lead to better performance on retrieval and classification tasks, until serious overfitting became a problem around 1000 hidden units. For perplexity, 128 hidden units worked quite well and having too many units made the estimates of the partition function obtained using AIS unstable. Starting with CD-1, the number of Gibbs steps was stepped up by one after every 10,000 weight updates till CD-20. Weight decay was used to prevent overfitting. Additionally, in order to encourage sparsity in the hidden units, KL-sparsity regularization was used. We decayed the learning rate as $\frac{\epsilon_0}{1+t/T}$, with $T = 10,000$ updates. This approximate training was sufficient to give good results on retrieval and classification tasks. However, to obtain good perplexity results, the model was trained properly using the method described in Sec. 3.3.1. Using 5 steps for mean-field inference and 20 for Gibbs sampling was found to be sufficient. This additional training gave improvements in terms of perplexity but the improvement on classification and retrieval tasks was not statistically significant.

We also implemented the standard Replicated Softmax model. The training procedure was the same as the pretraining process for the Over-Replicated Softmax model. Both the models were implemented on GPUs. Pretraining took 3-4 hours for the 2-layered Boltzmann Machines (depending on M) and the proper training took 10-12 hours. The DocNADE model was run using the publicly available code[4]. We used default settings for all hyperparameters, except the learning rates which were tuned separately for each hidden layer size and data set.

3.4.3 Perplexity

We compare the Over-Replicated Softmax model with the Replicated Softmax model in terms of perplexity. Computing perplexities involves computing the partition functions for these models. We used Annealed Importance Sampling (Neal, 2001) for doing this. In order to get reliable estimates, we ran 128 Markov chains for each document length. The average test perplexity per word was computed as

[4] `http://www.dmi.usherb.ca/~larocheh/code/DocNADE.zip`

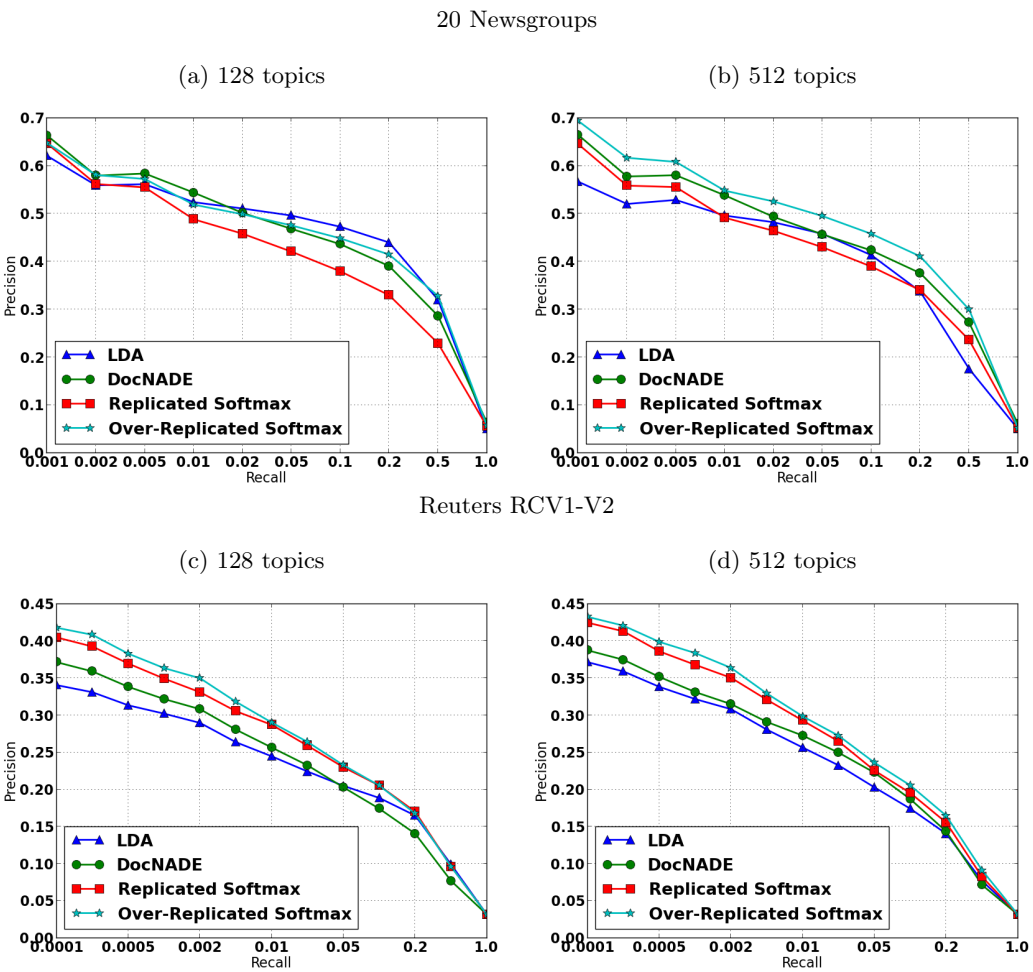


Figure 3.4: Comparison of Precision-Recall curves for document retrieval. All Over-Replicated Softmax models use $M = 100$ latent words.

$\exp\left(-1/L\sum_{l=1}^{L} 1/N_l \log p(\mathbf{v}_l)\right)$, where N_l is the number of words in document l. Table. 3.1 shows the perplexity averaged over $L = 1000$ randomly chosen test cases for each data set. Each of the models has 128 latent topics. Table. 3.1 shows that the Over-Replicated Softmax model assigns slightly lower perplexity to the test data compared to the Replicated Softmax model. For the Reuters data set the perplexity decreases from 1081 to 1060, and for 20 Newsgroups, it decreases from 965 to 958. Though the decrease is small, it is statistically significant since the standard deviation was typically ± 2 over 10 random choices of 1000 test cases. Increasing the value of M increases the strength of the prior, which leads to further improvements in perplexities. Note that the estimate of the log probability for 2-layered Boltzmann Machines is a lower bound on the actual log probability. So the perplexities we show are upper bounds and the actual perplexities may be lower (provided the estimate of the partition function is close to the actual value).

3.4.4 Document Retrieval

In order to do retrieval, we represent each document $\mathbf{V}$ as the conditional posterior distribution $P(\mathbf{h}^{(1)}|\mathbf{V})$. This can be done exactly for the Replicated Softmax and DocNADE models. For two-layered Boltzmann

	20 News	**Reuters**
Training set size	11,072	794,414
Test set size	7,052	10,000
Vocabulary size	2,000	10,000
Avg Document Length	51.8	94.6
Perplexities		
Unigram	1335	2208
Replicated Softmax	965	1081
Over-Rep. Softmax ($M = 50$)	961	1076
Over-Rep. Softmax ($M = 100$)	**958**	**1060**

Table 3.1: Comparison of the average test perplexity per word. All models use 128 topics.

Machines, we extract this representation using the fast approximate inference as described in Sec. 3.3.3. Performing more accurate inference using the mean-field approximation method did not lead to statistically different results. For the LDA, we used 1000 Gibbs sweeps per test document in order to get an approximate posterior over the topics.

Documents in the training set (including the validation set) were used as a database. The test set was used as queries. For each query, documents in the database were ranked using cosine distance as the similarity metric. The retrieval task was performed separately for each label and the results were averaged. Fig. 3.4 compares the precision-recall curves. As shown by Fig. 3.4, the Over-Replicated Softmax DBM outperforms other models on both datasets, particularly when retrieving the top few documents.

To find the source of improvement, we analyzed the effect of document length of retrieval performance. Fig. 3.5 plots the average precision obtained for query documents arranged in order of increasing length. We found that the Over-Replicated Softmax model gives large gains on documents with small numbers of words, confirming that the implicit prior imposed using a fixed value of M has a stronger effect on short documents. As shown in Fig. 3.5, DocNADE and Replicated Softmax models often do not do well for documents with few words. On the other hand, the Over-Replicated softmax model performs significantly better for short documents. In most document collections, the length of documents obeys a power law distribution. For example, in the 20 newsgroups dataset 50% of the documents have fewer than 35 words (Fig. 3.5c). This makes it very important to do well on short documents. The Over-Replicated Softmax model achieves this goal.

3.4.5 Document Classification

In this set of experiments, we evaluate the learned representations from the Over-Replicated Softmax model for the purpose of document classification. Since the objective is to evaluate the quality of the representation, simple linear classifiers were used. Multinomial logistic regression with a cross entropy loss function was used for the 20 newsgroups data set. The evaluation metric was classification accuracy. For the Reuters dataset, we used independent logistic regressions for each label since it is a multi-label classification problem. The evaluation metric was Mean Average Precision.

Table. 3.2 shows the results of these experiments. The Over-Replicated Softmax model performs significantly better than the standard Replicated Softmax model and LDA across different network sizes on both datasets. For the 20 newsgroups dataset using 512 topics, LDA gets 64.2% accuracy. Replicated Softmax (67.7%) and DocNADE (68.4%) improve upon this. The Over-Replicated Softmax

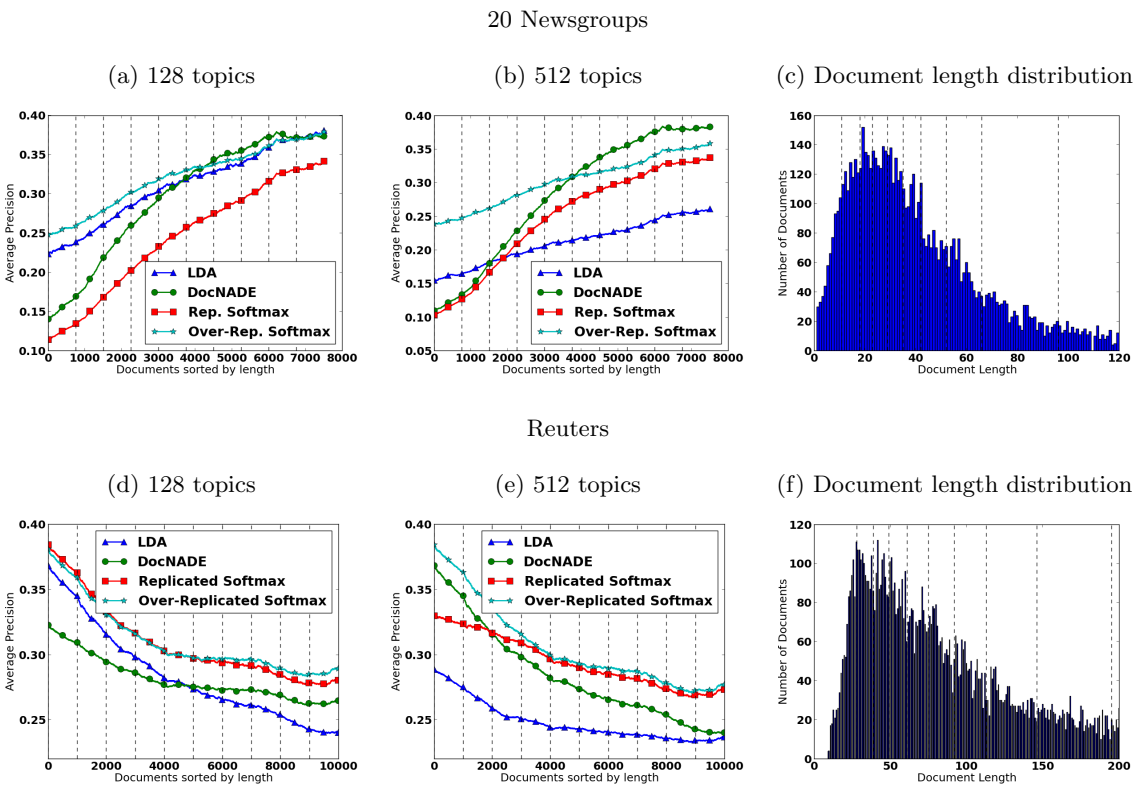


Figure 3.5: Effect of document size on retrieval performance for different topic models. The x-axis in Figures (a), (b), (d), (e) represents test documents arranged in increasing order of their length. The y-axis shows the average precision obtained by querying that document. The plots were smoothed to make the general trend visible. Figures (c) and (f) show the histogram of document lengths for the respective datasets. The dashed vertical lines denote 10-percentile boundaries. **Top :** Average Precision on the 20 Newsgroups dataset. **Bottom :** Mean Average Precision on the Reuters dataset. The Over-Replicated Softmax models performs significantly better for documents with few words. The adjoining histograms in each row show that such documents occur quite frequently in both data sets.

model further improves the result to 69.4%. The difference is larger for the Reuters dataset. In terms of Mean Average Precision (MAP), the Over-Replicated Softmax model achieves 0.453 which is a very significant improvement upon DocNADE (0.427) and Replicated Softmax (0.421).

We further examined the source of improvement by analyzing the effect of document length on the classification performance. Similar to retrieval, we found that the Over-Replicated Softmax model performs well on short documents. For long documents, the performance of the different models was similar.

3.5 Conclusion

The Over-Replicated Softmax model described in this chapter is an effective way of defining a flexible prior over the latent topic features of an RBM. This model causes no increase in the number of trainable parameters and only a minor increase in training algorithm complexity. Deep Boltzmann Machines are typically slow to train. However, our fast approximate training method makes it possible to train the model with CD, just like an RBM. The features extracted from documents using the Over-Replicated

Model	20 News		Reuters	
	128	**512**	**128**	**512**
LDA	65.7	64.2	0.304	0.351
DocNADE	**67.0**	68.4	0.388	0.417
Replicated Softmax	65.9	67.7	0.390	0.421
Over-Rep. Softmax	66.8	**69.1**	**0.401**	**0.453**

Table 3.2: Comparison of Classification accuracy on 20 Newsgroups dataset and Mean Average Precision on Reuters RCV1-v2.

Softmax model perform better than features from the standard Replicated Softmax and LDA models and are comparable to DocNADE across different network sizes.

While the number of hidden softmax units M, controlling the strength of the prior, was chosen once and fixed across all DBMs, it is possible to have M depend on N. One option is to set $M = cN$, $c > 0$. In this case, for documents of all lengths, the second-layer would perform perform $c/c+1$ of the modeling work compared to the first layer. Another alternative is to set $M = N_{max} - N$, where N_{max} is the maximum allowed length of all documents. In this case, our DBM model will always have the same number of replicated softmax units $N_{max} = N + M$, hence the same architecture and a single partition function. Given a document of length N, the remaining $N_{max} - N$ words can be treated as missing. All of these variations improve upon the standard Replicated Softmax model, LDA, and DocNADE models.

This chapter models documents as bags of words. Making this choice is clearly suboptimal because a lot of useful information is contained in the sequence of words. This information is immediately lost in any bag of words model. In recent years, progress has been made in training Recurrent Neural Networks (RNNs) that make use of the sequence of words to learn representations of sentences, for example, Skip-thought vectors (Kiros et al., 2015). The ability of RNNs to model sentences is also readily apparent in the context of neural machine translation models (Cho et al., 2014; Sutskever et al., 2014). The current direction of research in modeling documents is towards making use of these powerful sequence learning models.

Chapter 4

Unsupervised Learning of Visual Representations Using LSTMs

Videos are an abundant and rich source of visual information and can be seen as a window into the physics of the world we live in. They show us examples of what constitutes objects, how objects move against backgrounds, what happens when cameras move and how things get occluded. However, videos when represented as a temporal sequence of pixel intensities lie in an extremely high-dimensional space where distinct factors of variation such as illumination, pose, view point, camera motion, object motion, object identity and object attributes are heavily entangled. We would like to discover computations which can disentangle these factors to produce a representation that is equivariant (or invariant) to subsets of these factors. The hope is that such a representation would provide a generic and expressive interface between low-level visual inputs and high-level AI. This would make it easy to solve tasks in the AI-set which require access to visual stimuli.

4.1 Overview

Supervised learning has been extremely successful in learning good visual representations that not only produce good results at the task they are trained for (Krizhevsky et al., 2012; Szegedy et al., 2014; He et al., 2016), but also transfer well to other tasks and datasets. Therefore, it is natural to extend the same approach to learning video representations. This has led to research in 3D convolutional nets (Ji et al., 2013; Tran et al., 2014), different temporal fusion strategies (Karpathy et al., 2014) and exploring different ways of presenting visual information to convolutional nets (Simonyan and Zisserman, 2014a). However, videos are much higher dimensional entities compared to single images. Therefore, it becomes increasingly difficult to do credit assignment and learn long range structure, unless we collect much more labelled data or do a lot of feature engineering (for example computing the right kinds of flow features) to keep the dimensionality low. The costly work of collecting more labelled data and the tedious work of doing more clever engineering can go a long way in solving particular problems, but this is ultimately unsatisfying as a machine learning solution. This highlights the need for using unsupervised learning to find and represent structure in videos. Moreover, videos have a lot of structure in them (spatial and temporal regularities) which makes them particularly well suited as a domain for building unsupervised learning models.

Recently, recurrent neural networks using the Long Short Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, 1997) have been used successfully to perform various supervised sequence learning tasks, such as speech recognition (Graves and Jaitly, 2014), machine translation (Sutskever et al., 2014; Cho et al., 2014), and caption generation for images (Vinyals et al., 2014). They have also been applied on videos for recognizing actions and generating natural language descriptions (Donahue et al., 2014). A general sequence to sequence learning framework was described by Sutskever et al. (2014) in which a recurrent network is used to encode a sequence into a fixed length representation, and then another recurrent network is used to decode a sequence out of that representation. In this work, we apply and extend this framework to learn representations of sequences of images.

When designing any unsupervised learning model, it is crucial to have the right inductive biases and choose the right objective function so that the learning signal points the model towards learning useful features. The key inductive bias in the LSTM Encoder-Decoder framework is that the same operation must be applied at each time step to propagate information to the next step. This enforces the fact that the physics of the world remains the same, irrespective of input. The same physics acting on any state, at any time, must produce the next state. Our model works as follows. The Encoder LSTM runs through a sequence of frames to come up with a representation. This representation is then decoded through another LSTM to produce a target sequence. We consider different choices of the target sequence. One choice is to predict the same sequence as the input. The motivation is similar to that of autoencoders – we wish to capture all that is needed to reproduce the input but at the same time go through the inductive biases imposed by the model. Another option is to predict the future frames. Here the motivation is to learn a representation that extracts all that is needed to extrapolate the motion and appearance beyond what has been observed. These two natural choices can also be combined. In this case, there are two decoder LSTMs – one that decodes the representation into the input sequence and another that decodes the same representation to predict the future.

The inputs to the model can, in principle, be any representation of individual video frames. However, for the purposes of this work, we limit our attention to two kinds of inputs. The first is image patches. For this we use natural image patches as well as a dataset of moving MNIST digits. The second is high-level "percepts" extracted by applying a convolutional net trained on ImageNet. These percepts are the states of last (and/or second-to-last) layers of rectified linear hidden states from a convolutional neural net model.

In order to evaluate the learned representations we qualitatively analyze the reconstructions and predictions made by the model. For a more quantitative evaluation, we use these LSTMs as initializations for the supervised task of action recognition. If the unsupervised learning model comes up with useful representations then the classifier should be able to perform better, especially when there are only a few labelled examples. We find that this is indeed the case.

4.2 Related Work

The first approaches to learning representations of videos in an unsupervised way were based on ICA (van Hateren and Ruderman, 1998; Hurri and Hyvärinen, 2003). Le et al. (2011) approached this problem using multiple layers of Independent Subspace Analysis modules. Generative models for understanding transformations between pairs of consecutive images are also well studied (Memisevic, 2013; Memisevic and Hinton, 2010; Susskind et al., 2011). This work was extended recently by Michalski et al. (2014) to

model longer sequences.

Ranzato et al. (2014) proposed a generative model for videos. The model uses a recurrent neural network to predict the next frame or interpolate between frames. In this work, the authors highlight the importance of choosing the right loss function. It is argued that squared loss in input space is not the right objective because it does not respond well to small distortions in input space. The proposed solution is to quantize image patches into a large dictionary and train the model to predict the identity of the target patch. This does solve some of the problems of squared loss but it introduces an arbitrary dictionary size into the picture and altogether removes the idea of patches being similar or dissimilar to one other. Designing an appropriate loss function that respects our notion of visual similarity is a very hard problem (in a sense, almost as hard as the modeling problem we want to solve in the first place). Therefore, in this chapter, we use the simple squared loss objective function as a starting point and focus on designing an encoder-decoder RNN architecture that can be used with any loss function.

4.3 Model Description

In this section, we describe several variants of our LSTM Encoder-Decoder model. The basic unit of our network is the LSTM cell block. Our implementation of LSTMs follows closely the one discussed by Graves (2013).

4.3.1 Long Short Term Memory

In this section we briefly describe the LSTM unit which is the basic building block of our model. The unit is shown in Fig. 4.1 (reproduced from Graves (2013)). Each LSTM unit has a cell which has a state

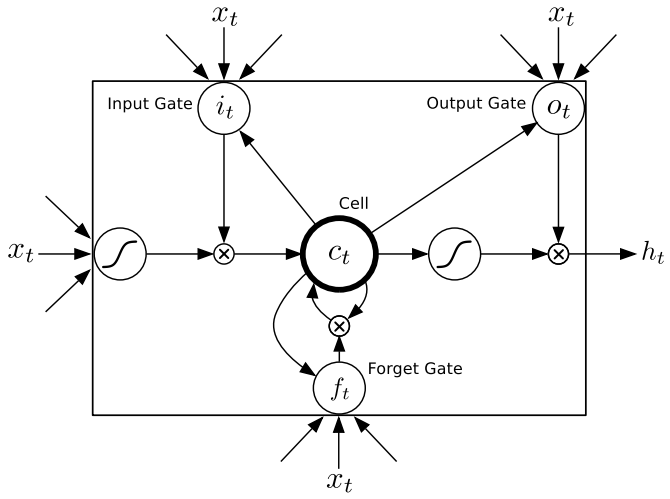


Figure 4.1: LSTM unit

c_t at time t. This cell can be thought of as a memory unit. Access to this memory unit for reading or modifying it is controlled through sigmoidal gates – input gate i_t, forget gate f_t and output gate o_t. The LSTM unit operates as follows. At each time step it receives inputs from two external sources at each of the four terminals (the three gates and the input). The first source is the current frame $\mathbf{x}_t$. The second

source is the previous hidden states of all LSTM units in the same layer $\mathbf{h}_{t-1}$. Additionally, each gate has an internal source, the cell state c_{t-1} of its cell block. The links between a cell and its own gates are called *peephole* connections. The inputs coming from different sources get added up, along with a bias. The gates are activated by passing their total input through the logistic function. The total input at the input terminal is passed through the tanh non-linearity. The resulting activation is multiplied by the activation of the input gate. This is then added to the cell state after multiplying the cell state by the forget gate's activation f_t. The final output from the LSTM unit h_t is computed by multiplying the output gate's activation o_t with the updated cell state passed through a tanh non-linearity. These updates are summarized for a layer of LSTM units as follows

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + W_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i\right), \\
\mathbf{f}_t &= \sigma\left(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + W_{cf}\mathbf{c}_{t-1} + \mathbf{b}_f\right), \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh\left(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c\right), \\
\mathbf{o}_t &= \sigma\left(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + W_{co}\mathbf{c}_t + \mathbf{b}_o\right), \\
\mathbf{h}_t &= \mathbf{o}_t \tanh(\mathbf{c}_t).
\end{aligned}
$$

Note that all $W_{c\bullet}$ matrices are diagonal, whereas the rest are dense. The key advantage of using an LSTM unit over a traditional neuron in an RNN is that the cell state in an LSTM unit *sums* activities over time. Since derivatives distribute over sums, the error derivatives don't vanish quickly as they get sent back into time. This makes it easy to do credit assignment over long sequences and discover long-range features.

4.3.2 LSTM Autoencoder Model

In this section, we describe a model that uses Recurrent Neural Nets (RNNs) made of LSTM units to do unsupervised learning. The model consists of two RNNs – the encoder LSTM and the decoder LSTM as shown in Fig. 4.2. The input to the model is a sequence of vectors (image patches or features). The encoder LSTM reads in this sequence. After the last input has been read, the decoder LSTM takes over and outputs a prediction for the target sequence. The target sequence is same as the input sequence, but in reverse order. Reversing the target sequence makes the optimization easier because the model can get off the ground by looking at low range correlations. This is also inspired by how lists are represented in LISP. The encoder can be seen as creating a list by applying the `cons` function on the previously constructed list and the new input. The decoder essentially unrolls this list, with the hidden to output weights extracting the element at the top of the list (`car` function) and the hidden to hidden weights extracting the rest of the list (`cdr` function). Therefore, the first element out is the last element in.

The decoder can be of two kinds – conditional or unconditioned. A conditional decoder receives the last generated output frame as input, i.e., the dotted input in Fig. 4.2 is present. An unconditioned decoder does not receive that input. This is discussed in more detail in Sec. 4.3.4. Fig. 4.2 shows a single layer LSTM Autoencoder. The architecture can be extend to multiple layers by stacking LSTMs on top of each other.

Why should this learn good features?

The state of the encoder LSTM after the last input has been read is the representation of the input video.

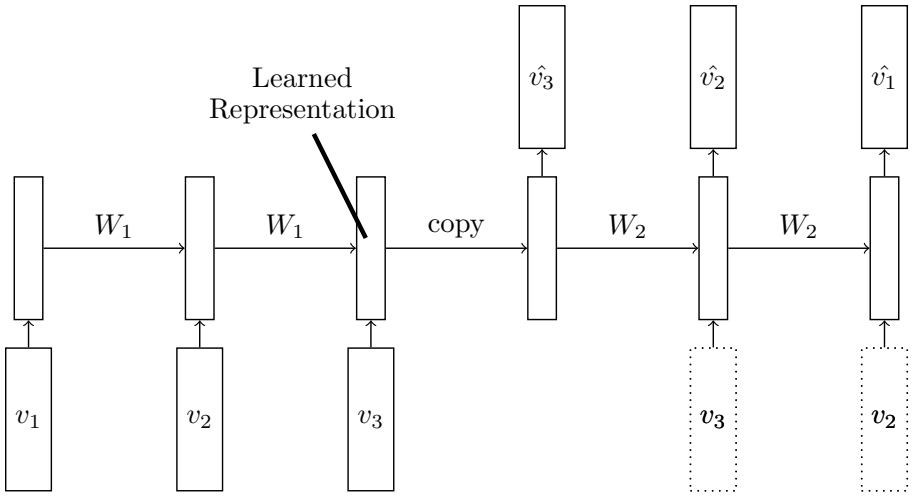


Figure 4.2: LSTM Autoencoder Model

The decoder LSTM is being asked to reconstruct back the input sequence from this representation. In order to do so, the representation must retain information about the appearance of the objects and the background as well as the motion contained in the video. However, an important question for any autoencoder-style model is what prevents it from learning an identity mapping and effectively copying the input to the output. In that case all the information about the input would still be present but the representation will be no better than the input. There are two factors that control this behaviour. First, the fact that there are only a fixed number of hidden units makes it unlikely that the model can learn trivial mappings for arbitrary length input sequences. Second, the same LSTM operation is used to decode the representation recursively. This means that the same dynamics must be applied on the representation at any stage of decoding. This further prevents the model from learning an identity mapping.

4.3.3 LSTM Future Predictor Model

Another natural unsupervised learning task for sequences is predicting the future. This is the approach used in language models for modeling sequences of words. The design of the Future Predictor Model is same as that of the Autoencoder Model, except that the decoder LSTM in this case predicts frames of the video that come after the input sequence (Fig. 4.3). Ranzato et al. (2014) use a similar model but predict only the next frame at each time step. This model, on the other hand, predicts a long sequence into the future. Here again we can consider two variants of the decoder – conditional and unconditioned.

Why should this learn good features?

In order to predict the next few frames correctly, the model needs information about which objects and background are present and how they are moving so that the motion can be extrapolated. The hidden state coming out from the encoder will try to capture this information. Therefore, this state can be seen as a representation of the input sequence.

4.3.4 Conditional Decoder

For each of these two models, we can consider two possibilities - one in which the decoder LSTM is conditioned on the last generated frame and the other in which it is not. In the experimental section,

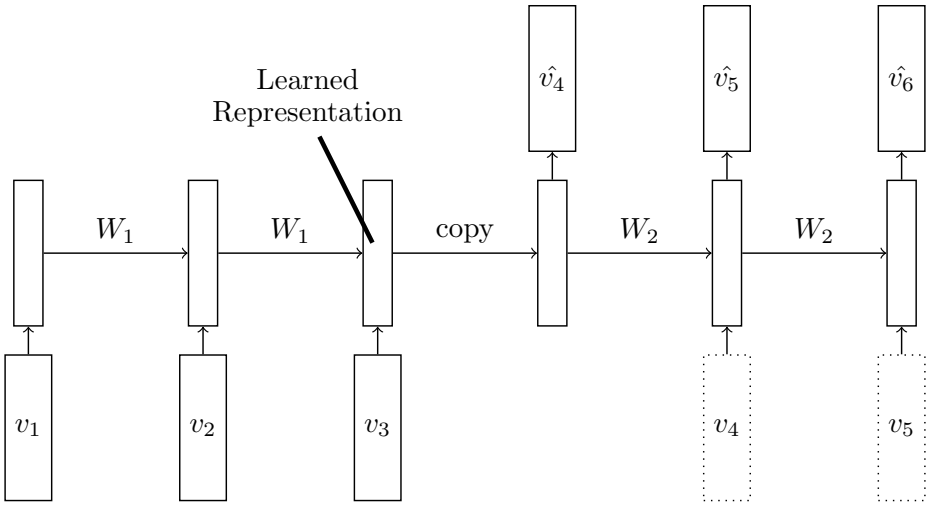


Figure 4.3: LSTM Future Predictor Model

we explore these choices quantitatively. Here we briefly discuss arguments for and against a conditional decoder. A strong argument in favour of using a conditional decoder is that it allows the decoder to model multiple modes in the target sequence distribution. Without that, we would end up averaging the multiple modes in the low-level input space. However, this is an issue only if we expect multiple modes in the target sequence distribution. For the LSTM Autoencoder, there is only one correct target and hence a unimodal target distribution. But for the LSTM Future Predictor there is a possibility of multiple targets given an input because even if we assume a deterministic universe, everything needed to predict the future will not necessarily be observed in the input.

There is also an argument against using a conditional decoder from the optimization point-of-view. There are strong short-range correlations in video data, for example, most of the content of a frame is same as the previous one. If the decoder was given access to the last few frames while generating a particular frame at training time, it would find it easy to pick up on these correlations. There would only be a very small gradient that tries to fix up the extremely subtle errors that require long term knowledge about the input sequence. In an unconditioned decoder, this input is removed and the model is forced to look for information deep inside the encoder.

4.3.5 A Composite Model

The two tasks – reconstructing the input and predicting the future can be combined to create a composite model as shown in Fig. 4.4. Here the encoder LSTM is asked to come up with a state from which we can *both* predict the next few frames as well as reconstruct the input.

This composite model tries to overcome the shortcomings that each model suffers on its own. A high-capacity autoencoder would suffer from the tendency to learn trivial representations that just memorize the inputs. However, this memorization is not useful at all for predicting the future. Therefore, the composite model cannot just memorize information. On the other hand, the future predictor suffers form the tendency to store information only about the last few frames since those are most important for predicting the future, i.e., in order to predict v_t, the frames $\{v_{t-1}, \ldots, v_{t-k}\}$ are much more important than v_0, for some small value of k. Therefore the representation at the end of the encoder will have

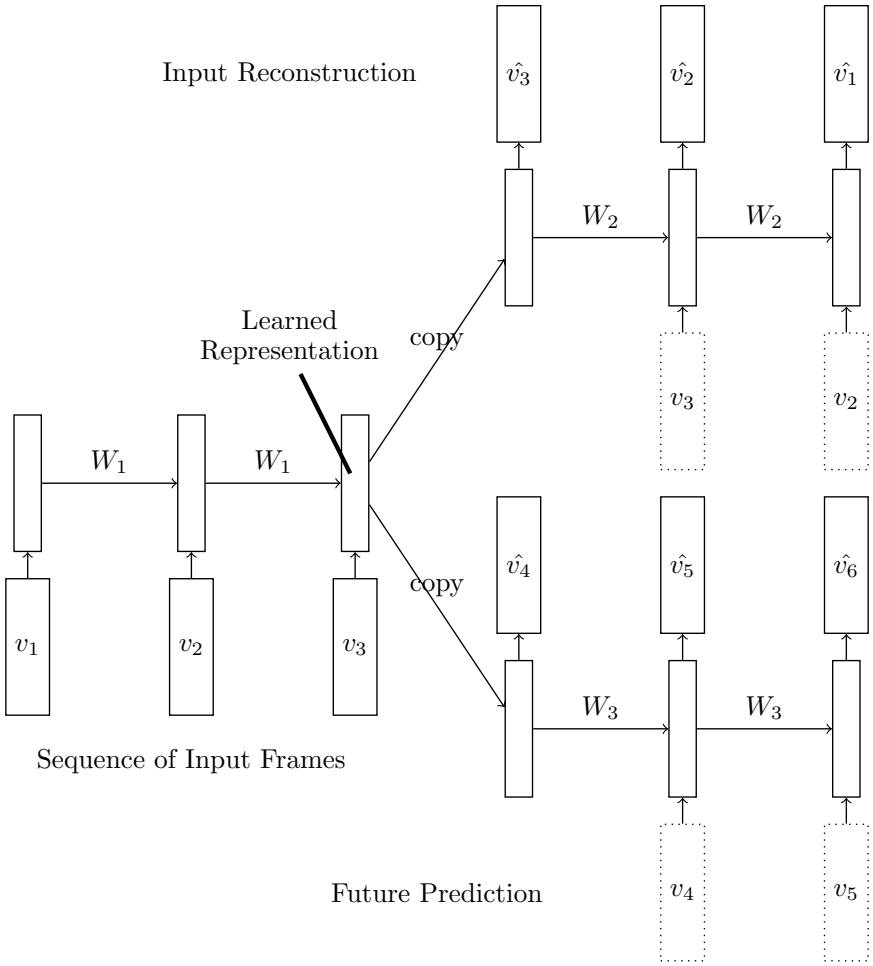


Figure 4.4: The Composite Model: The LSTM predicts the future as well as the input sequence.

forgotten about a large part of the input. But if we ask the model to also predict *all* of the input sequence, then it cannot just pay attention to the last few frames.

4.4 Experiments

We design experiments to accomplish the following objectives:

- Get a qualitative understanding of what the LSTM learns to do.
- Measure the benefit of initializing networks for supervised learning tasks with the weights found by unsupervised learning, especially with very few training examples.
- Compare the different proposed models - Autoencoder, Future Predictor and Composite models and their conditional variants.
- Compare with state-of-the-art action recognition benchmarks.

4.4.1 Datasets

We use the UCF-101 and HMDB-51 datasets for supervised tasks. The UCF-101 dataset (Soomro et al., 2012) contains 13,320 videos with an average length of 6.2 seconds belonging to 101 different action categories. The dataset has 3 standard train/test splits with the training set containing around 9,500 videos in each split (the rest are test). The HMDB-51 dataset (Kuehne et al., 2011) contains 5100 videos belonging to 51 different action categories. Mean length of the videos is 3.2 seconds. This also has 3 train/test splits with 3570 videos in the training set and rest in test.

To train the unsupervised models, we used a subset of the Sports-1M dataset (Karpathy et al., 2014), that contains 1 million YouTube clips. Even though this dataset is labelled for actions, we did not do any supervised experiments on it because of logistical constraints with working with such a huge dataset. We instead collected 300 hours of video by randomly sampling 10 second clips from the dataset. It is possible to collect better samples if instead of choosing randomly, we extracted videos where a lot of motion is happening and where there are no shot boundaries. However, we did not do so in the spirit of unsupervised learning, and because we did not want to introduce any unnatural bias in the samples. We also used the supervised datasets (UCF-101 and HMDB-51) for unsupervised training. However, we found that using them did not give any significant advantage over just using the YouTube videos.

We extracted percepts using the convolutional neural net model of Simonyan and Zisserman (2014b). The videos have a resolution of 240×320 and were sampled at almost 30 frames per second. We took the central 224×224 patch from each frame and ran it through the convnet. This gave us the RGB percepts. Additionally, for UCF-101, we computed flow percepts by extracting flows using the Brox method and training the temporal stream convolutional network as described by Simonyan and Zisserman (2014a). We found that the fc6 features worked better than fc7 for single frame classification using both RGB and flow percepts. Therefore, we used the 4096-dimensional fc6 layer as the input representation of our data. Besides these percepts, we also trained the proposed models on 32×32 patches of pixels.

All models were trained using backprop on a single NVIDIA Titan GPU. A two layer 2048 unit Composite model that predicts 13 frames and reconstructs 16 frames took 18-20 hours to converge on 300 hours of percepts. We initialized weights by sampling from a uniform distribution whose scale was set to 1/sqrt(fan-in). Biases at all the gates were initialized to zero. Peep-hole connections were initialized to zero. The supervised classifiers trained on 16 frames took 5-15 minutes to converge. The code can be found at `https://github.com/emansim/unsupervised-videos`.

4.4.2 Visualization and Qualitative Analysis

The aim of this set of experiments to visualize the properties of the proposed models.

Experiments on MNIST

We first trained our models on a dataset of moving MNIST digits. In this dataset, each video was 20 frames long and consisted of two digits moving inside a 64×64 patch. The digits were chosen randomly from the training set and placed initially at random locations inside the patch. Each digit was assigned a velocity whose direction was chosen uniformly randomly on a unit circle and whose magnitude was also chosen uniformly at random over a fixed range. The digits bounced-off the edges of the 64×64 frame and overlapped if they were at the same location. The reason for working with this dataset is that it is infinite in size and can be generated quickly on the fly. This makes it possible to explore the model without expensive disk accesses or overfitting issues. It also has interesting behaviours due to occlusions

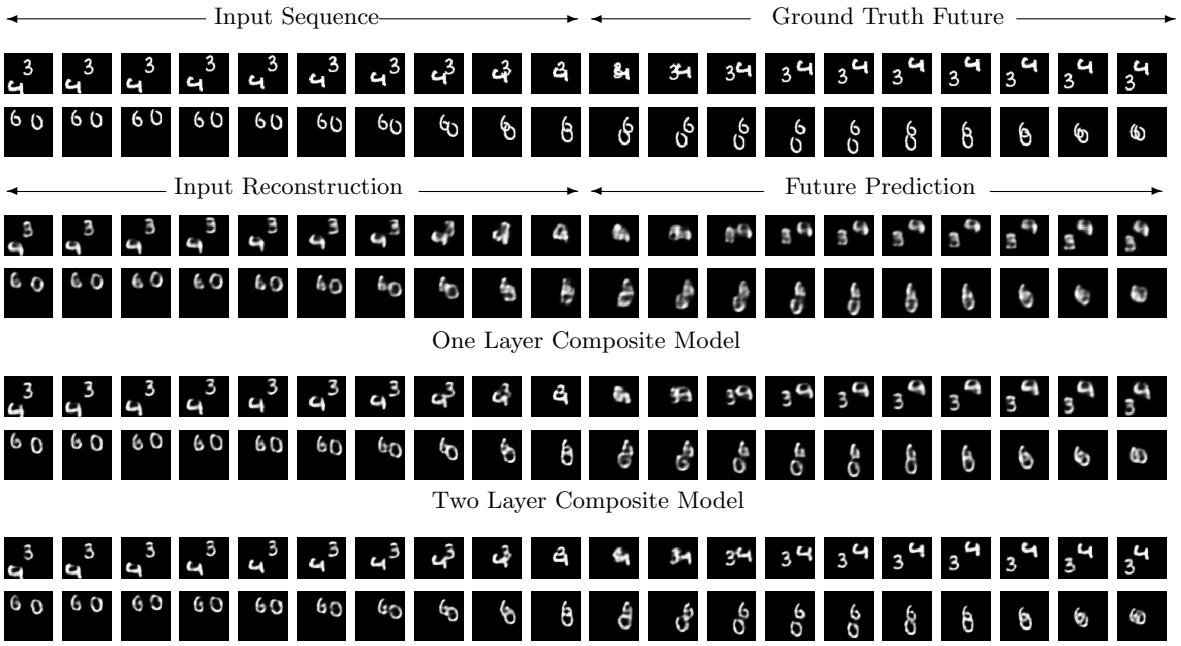


Figure 4.5: Reconstruction and future prediction obtained from the Composite Model on a dataset of moving MNIST digits.

and the dynamics of bouncing off the walls.

We first trained a single layer Composite Model. Each LSTM had 2048 units. The encoder took 10 frames as input. The decoder tried to reconstruct these 10 frames and the future predictor attempted to predict the next 10 frames. We used logistic output units with a cross entropy loss function. Fig. 4.5 shows two examples of running this model. The true sequences are shown in the first two rows. The next two rows show the reconstruction and future prediction from the one layer Composite Model. It is interesting to note that the model figures out how to separate superimposed digits and can model them even as they pass through each other. This shows some evidence of *disentangling* the two independent factors of variation in this sequence. The model can also correctly predict the motion after bouncing off the walls. In order to see if adding depth helps, we trained a two layer Composite Model, with each layer having 2048 units. We can see that adding depth helps the model make better predictions. Next, we changed the future predictor by making it conditional. We can see that this model makes sharper predictions.

Experiments on Natural Image Patches

Next, we tried to see if our models can also work with natural image patches. For this, we trained the models on sequences of 32×32 natural image patches extracted from the UCF-101 dataset. In this case, we used linear output units and the squared error loss function. The input was 16 frames and the model was asked to reconstruct the 16 frames and predict the future 13 frames. Fig. 4.6 shows the results obtained from a two layer Composite model with 2048 units. We found that the reconstructions and the predictions are both very blurry. We then trained a bigger model with 4096 units. The outputs from this model are also shown in Fig. 4.6. We can see that the reconstructions get much sharper.

Generalization over time scales

In this experiment, we test if the model can work at time scales that are different than what it was

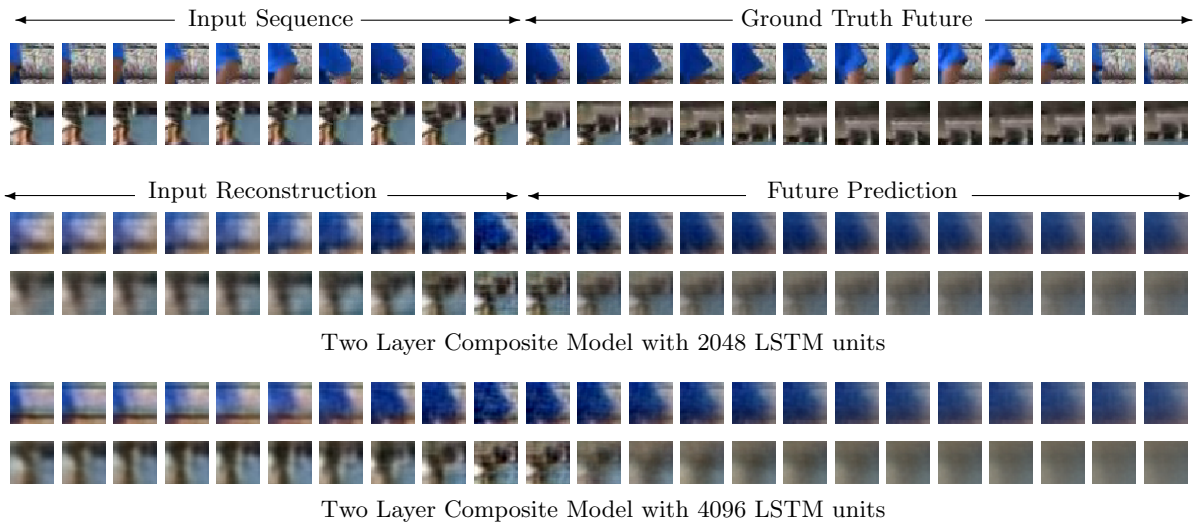
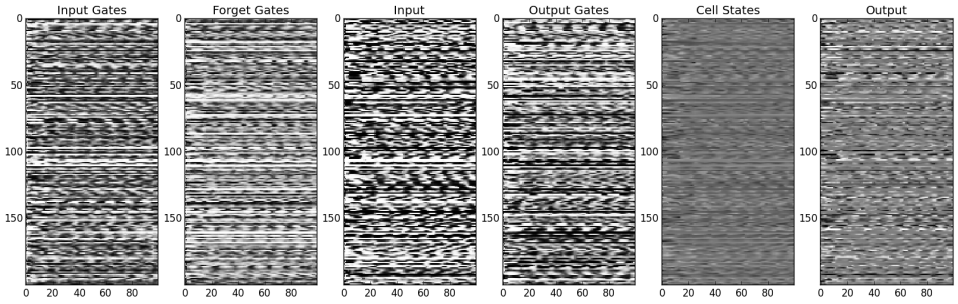


Figure 4.6: Reconstruction and future prediction obtained from the Composite Model on a dataset of natural image patches. The first two rows show ground truth sequences. The model takes 16 frames as inputs. Only the last 10 frames of the input sequence are shown here. The next 13 frames are the ground truth future. In the rows that follow, we show the reconstructed and predicted frames for two instances of the model.

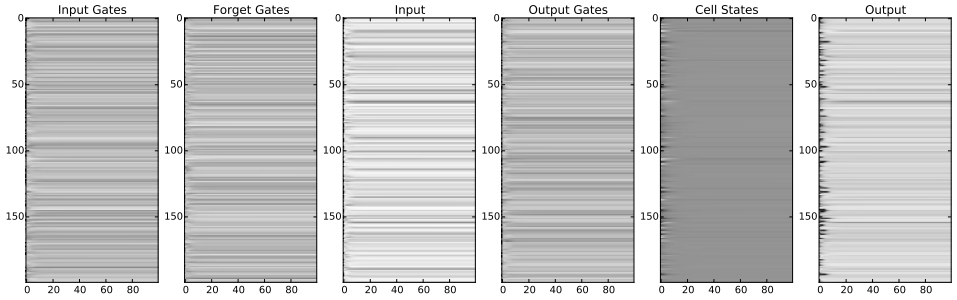
trained on. We take a one hidden layer unconditioned Composite Model trained on moving MNIST digits. The model has 2048 LSTM units and looks at a 64×64 input. It was trained on input sequences of 10 frames to reconstruct those 10 frames as well as predict 10 frames into the future. In order to test if the future predictor is able to generalize beyond 10 frames, we let the model run for 100 steps into the future. Fig. 4.7a shows the pattern of activity in the LSTM units of the future predictor pathway for a randomly chosen test input. It shows the activity at each of the three sigmoidal gates (input, forget, output), the input (after the tanh non-linearity, before being multiplied by the input gate), the cell state and the final output (after being multiplied by the output gate). Even though the units are ordered randomly along the vertical axis, we can see that the dynamics has a periodic quality to it. The model is able to generate persistent motion for long periods of time. In terms of reconstruction, the model only outputs blobs after the first 15 frames, but the motion is relatively well preserved. More results, including long range future predictions over hundreds of time steps can see been at `http://www.cs.toronto.edu/~nitish/unsupervised_video`. To show that setting up a periodic behaviour is not trivial, Fig. 4.7b shows the activity from a randomly initialized future predictor. Here, the LSTM state quickly converges and the outputs blur completely.

Out-of-domain Inputs

Next, we test this model's ability to deal with out-of-domain inputs. For this, we test the model on sequences of one and three moving digits. The model was trained on sequences of two moving digits, so it has never seen inputs with just one digit or three digits. Fig. 4.8 shows the reconstruction and future prediction results. For one moving digit, we can see that the model can do a good job but it really tries to hallucinate a second digit overlapping with the first one. The second digit shows up towards the end of the future reconstruction. For three digits, the model merges digits into blobs. However, it does well at getting the overall motion right. This highlights a key drawback of modeling entire frames of input in a single pass. In order to model videos with variable number of objects, we perhaps need models that not only have an attention mechanism in place, but can also learn to execute themselves a variable number of times and do variable amounts of computation.



(a) Trained Future Predictor



(b) Randomly Initialized Future Predictor

Figure 4.7: Pattern of activity in 200 randomly chosen LSTM units in the Future Predictor of a 1 layer (unconditioned) Composite Model trained on moving MNIST digits. The vertical axis corresponds to different LSTM units. The horizontal axis is time. The model was only trained to predict the next 10 frames, but here we let it run to predict the next 100 frames. **Top**: The dynamics has a periodic quality which does not die out. **Bottom** : The pattern of activity, if the trained weights in the future predictor are replaced by random weights. The dynamics quickly dies out.

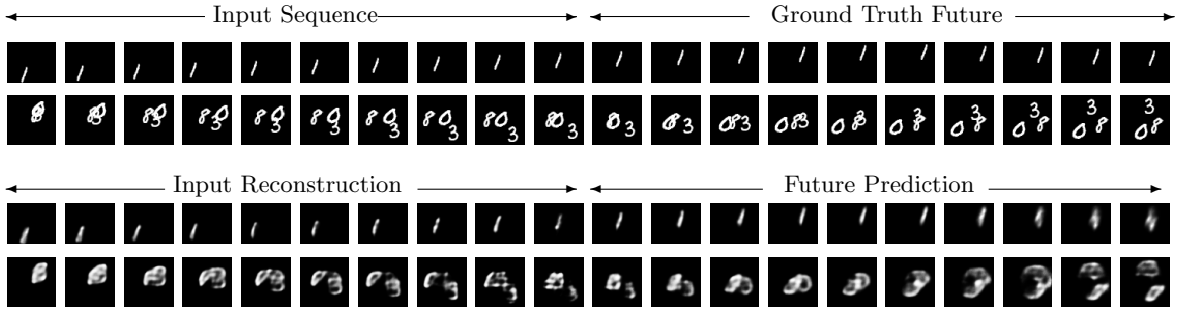
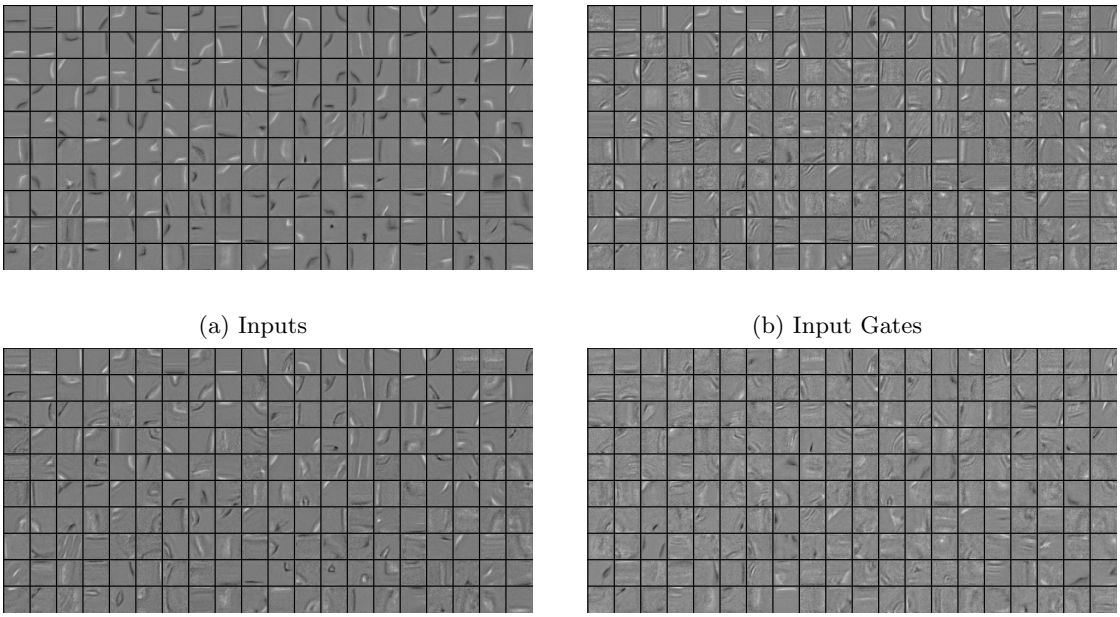


Figure 4.8: Out-of-domain runs. Reconstruction and Future prediction for test sequences of one and three moving digits. The model was trained on sequences of two moving digits.



(a) Inputs (b) Input Gates

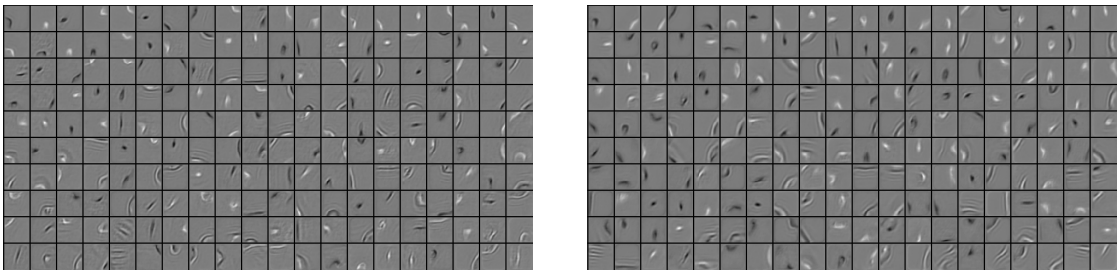
(c) Forget Gates (d) Output Gates

Figure 4.9: Input features from a Composite Model trained on moving MNIST digits. In an LSTM, each input frame is connected to four sets of units - the input, the input gate, forget gate and output gate. These figures show the top-200 features ordered by L_2 norm of the input features. The features in corresponding locations belong to the same LSTM unit.

Visualizing Features

Next, we visualize the features learned by this model. Fig. 4.9 shows the weights that connect each input frame to the encoder LSTM. There are four sets of weights. One set of weights connects the frame to the input units. There are three other sets, one corresponding to each of the three gates (input, forget and output). Each weight has a size of 64×64. A lot of features look like thin strips. Others look like higher frequency strips. It is conceivable that the high frequency features help in encoding the direction and velocity of motion.

Fig. 4.10 shows the output features from the two LSTM decoders of a Composite Model. These correspond to the weights connecting the LSTM output units to the output layer. They appear to be somewhat qualitatively different from the input features shown in Fig. 4.9. There are many more output features that are local blobs, whereas those are rare in the input features. In the output features, the ones that do look like strips are much shorter than those in the input features. One way to interpret this is the following. The model needs to know about motion (which direction and how fast things are moving) from the input. This requires *precise* information about location (thin strips) and velocity (high frequency strips). But when it is generating the output, the model wants to hedge its bets so that it does not suffer a huge loss for predicting things sharply at the wrong place. This could explain why the output features have somewhat bigger blobs. The relative shortness of the strips in the output features can be explained by the fact that in the inputs, it does not hurt to have a longer feature than what is needed to detect a location because information is coarse-coded through multiple features. But in the output, the model may not want to put down a feature that is bigger than any digit because other units will have to conspire to correct for it.



(a) Input Reconstruction (b) Future Prediction

Figure 4.10: Output features from the two decoder LSTMs of a Composite Model trained on moving MNIST digits. These figures show the top-200 features ordered by L_2 norm.

4.4.3 Action Recognition on UCF-101/HMDB-51

The aim of this set of experiments is to see if the features learned by unsupervised learning can help improve performance on supervised tasks.

We trained a two layer Composite Model with 2048 hidden units with no conditioning on either decoders. The model was trained on percepts extracted from 300 hours of YouTube data. The model was trained to autoencode 16 frames and predict the next 13 frames. We initialize an LSTM classifier with the weights learned by the encoder LSTM from this model. The classifier is shown in Fig. 4.11. The output from each LSTM in the second layer goes into a softmax classifier that makes a prediction about the action being performed at each time step. Since only one action is being performed in each video in the datasets we consider, the target is the same at each time step. At test time, the predictions made at each time step are averaged. To get a prediction for the entire video, we average the predictions from all 16 frame blocks in the video with a stride of 8 frames. Using a smaller stride did not improve results.

The baseline for comparing these models is an identical LSTM classifier but with randomly initialized weights. All classifiers used dropout regularization, where we dropped activations as they were communicated across layers but not through time within the same LSTM as proposed in Zaremba et al. (2014). We emphasize that this is a very strong baseline and does significantly better than just using single frames. Using dropout was crucial in order to train good baseline models especially with very few training examples.

Fig. 4.12 compares three models - single frame classifier (logistic regression), baseline LSTM classifier and the LSTM classifier initialized with weights from the Composite Model as the number of labelled videos per class is varied. Note that having one labelled video means having many labelled 16 frame blocks. We can see that for the case of very few training examples, unsupervised learning gives a substantial improvement. For example, for UCF-101, the performance improves from 29.6% to 34.3% when training on only one labelled video. As the size of the labelled dataset grows, the improvement becomes smaller. Even for the full UCF-101 dataset we still get a considerable improvement from 74.5% to 75.8%. On HMDB-51, the improvement is from 42.8% to 44.0% for the full dataset (70 videos per class) and 14.4% to 19.1% for one video per class. Although, the improvement in classification by using unsupervised learning was not as big as we expected, we still managed to yield an additional improvement over a strong baseline. We discuss some avenues for improvements later.

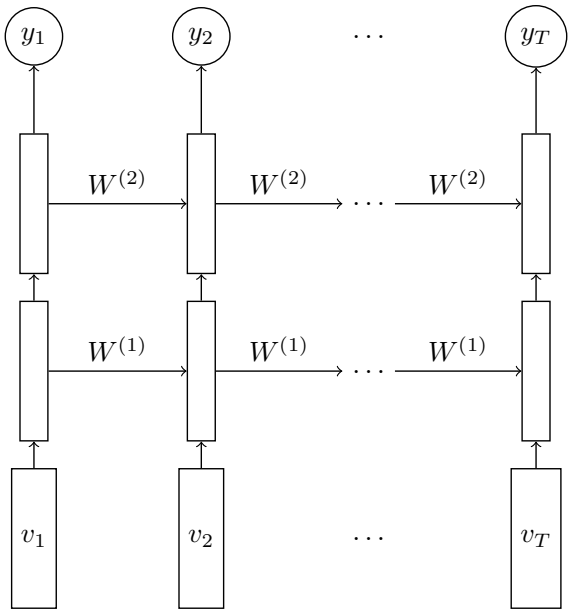
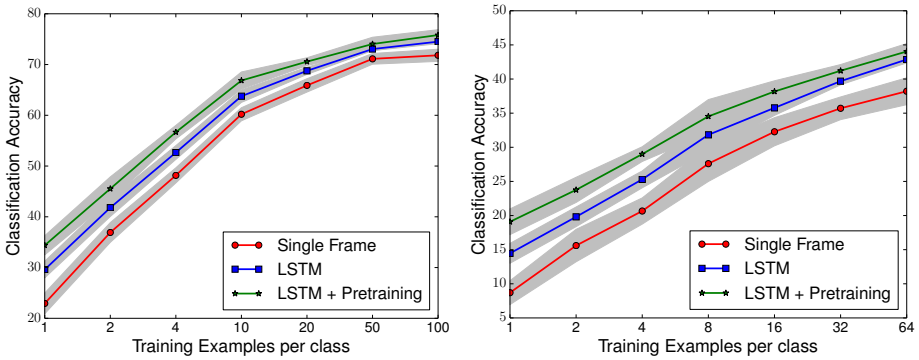


Figure 4.11: LSTM Classifier.



(a) UCF-101 RGB (b) HMDB-51 RGB

Figure 4.12: Effect of pretraining on action recognition with change in the size of the labelled training set. The error bars are over 10 different samples of training sets.

We further ran similar experiments on the optical flow percepts extracted from the UCF-101 dataset. A temporal stream convolutional net, similar to the one proposed by Simonyan and Zisserman (2014b), was trained on single frame optical flows as well as on stacks of 10 optical flows. This gave an accuracy of 72.2% and 77.5% respectively. Here again, our models took 16 frames as input, reconstructed them and predicted 13 frames into the future. LSTMs with 128 hidden units improved the accuracy by 2.1% to 74.3% for the single frame case. Bigger LSTMs did not improve results. By pretraining the LSTM, we were able to further improve the classification to 74.9% (± 0.1). For stacks of 10 frames we improved very slightly to 77.7%. These results are summarized in Table. 4.1.

4.4.4 Comparison of Different Model Variants

The aim of this set of experiments is to compare the different variants of the model proposed in this chapter. Since it is always possible to get lower reconstruction error by copying the inputs, we cannot use input reconstruction error as a measure of how good a model is doing. However, we can use the

Model	UCF-101 RGB	UCF-101 1- frame flow	HMDB-51 RGB
Single Frame	72.2	72.2	40.1
LSTM classifier	74.5	74.3	42.8
Composite LSTM Model + Finetuning	**75.8**	**74.9**	**44.1**

Table 4.1: Summary of Results on Action Recognition.

Model	Cross Entropy on MNIST	Squared loss on image patches
Future Predictor	350.2	225.2
Composite Model	344.9	210.7
Conditional Future Predictor	343.5	221.3
Composite Model with Conditional Future Predictor	341.2	208.1

Table 4.2: Future prediction results on MNIST and image patches. All models use 2 layers of LSTMs.

error in predicting the future as a reasonable measure of how good the model is doing. Besides, we can use the performance on supervised tasks as a proxy for how good the unsupervised model is doing. In this section, we present results from these two analyses.

Future prediction results are summarized in Table. 4.2. For MNIST we compute the cross entropy of the predictions with respect to the ground truth, both of which are 64×64 patches. For natural image patches, we compute the squared loss. We see that the Composite Model always does a better job of predicting the future compared to the Future Predictor. This indicates that having the autoencoder along with the future predictor to force the model to remember more about the inputs actually helps predict the future better. Next, we can compare each model with its conditional variant. Here, we find that the conditional models perform better, as was also noted in Fig. 4.5.

Next, we compare the models using performance on a supervised task. Table. 4.3 shows the performance on action recognition achieved by finetuning different unsupervised learning models. Besides running the experiments on the full UCF-101 and HMDB-51 datasets, we also ran the experiments on small subsets of these to better highlight the case where we have very few training examples. We find that all unsupervised models improve over the baseline LSTM which is itself well-regularized by using dropout. The Autoencoder model seems to perform consistently better than the Future Predictor. The Composite model which combines the two does better than either one alone. Conditioning on the generated inputs does not seem to give a clear advantage over not doing so. The Composite Model with a conditional future predictor works the best, although its performance is almost same as that of the Composite Model.

4.4.5 Comparison with Other Action Recognition Benchmarks

Finally, we compare our models to the state-of-the-art action recognition results. The performance is summarized in Table. 4.4. The table is divided into three sets. The first set compares models that use only RGB data (single or multiple frames). The second set compares models that use explicitly computed flow features only. Models in the third set use both.

On RGB data, our model performs at par with the best deep models. It performs 3% better than

Method	UCF-101 small	UCF-101	HMDB-51 small	HMDB-51
Baseline LSTM	63.7	74.5	25.3	42.8
Autoencoder	66.2	75.1	28.6	44.0
Future Predictor	64.9	74.9	27.3	43.1
Conditional Autoencoder	65.8	74.8	27.9	43.1
Conditional Future Predictor	65.1	74.9	27.4	43.4
Composite Model	67.0	**75.8**	29.1	**44.1**
Composite Model with Conditional Future Predictor	**67.1**	**75.8**	**29.2**	44.0

Table 4.3: Comparison of different unsupervised pretraining methods. UCF-101 small is a subset containing 10 videos per class. HMDB-51 small contains 4 videos per class.

Method	UCF-101	HMDB-51
Spatial Convolutional Net (Simonyan and Zisserman, 2014a)	73.0	40.5
C3D (Tran et al., 2014)	72.3	-
C3D + fc6 (Tran et al., 2014)	**76.4**	-
LRCN (Donahue et al., 2014)	71.1	-
Composite LSTM Model	75.8	44.0
Temporal Convolutional Net (Simonyan and Zisserman, 2014a)	**83.7**	54.6
LRCN (Donahue et al., 2014)	77.0	-
Composite LSTM Model	77.7	-
LRCN Donahue et al. (2014)	82.9	-
Two-stream Convolutional Net Simonyan and Zisserman (2014a)	88.0	59.4
Multi-skip feature stacking Lan et al. (2014)	**89.1**	**65.1**
Composite LSTM Model	84.3	-

Table 4.4: Comparison with state-of-the-art action recognition models.

the LRCN model that also used LSTMs on top of convnet features[1]. Our model performs better than C3D features that use a 3D convolutional net. However, when the C3D features are concatenated with fc6 percepts, they do slightly better than our model.

The improvement for flow features over using a randomly initialized LSTM network is quite small. We believe this is at least partly due to the fact that the flow percepts already capture a lot of the motion information that the LSTM would otherwise discover.

When we combine predictions from the RGB and flow models, we obtain 84.3% accuracy on UCF-101. We believe further improvements can be made by running the model over different patch locations and mirroring the patches. Also, our model can be applied deeper inside the convnet instead of just at the top-level.

4.5 Conclusions

We proposed models based on LSTMs that can learn good video representations. We compared them and analyzed their properties through visualizations. Moreover, we managed to get an improvement on supervised tasks. The best performing model was the Composite Model that combined an autoencoder and a future predictor. Conditioning on generated outputs did not have a significant impact on the performance for supervised tasks, however it made the future predictions look slightly better. The

[1]However, the improvement is only partially from unsupervised learning, since we used a better convnet model.

model was able to persistently generate motion well beyond the time scales it was trained for. However, it lost the precise object features rapidly after the training time scale. The features at the input and output layers were found to have some interesting properties.

To further get improvements for supervised tasks, we believe that the model can be extended by applying it convolutionally across patches of the video and stacking multiple layers of such models. Applying this model in the lower layers of a convolutional net could help extract motion information that would otherwise be lost across max-pooling layers.

Chapter 5

Unsupervised Learning with Directional-Unit Networks

Unsupervised learning models often aim to learn representations of data in terms of the states of latent variables. In the context of neural networks, these latent variables (or hidden unit activations) have traditionally been binary or real-valued. While there has been a lot of work done in building different neural network architectures and optimizing different loss functions, neural activations have typically been thought of as scalar-valued quantities. In this work, we explore the use of vector-valued hidden units, which we call directional units. Any N-dimensional vector-valued hidden unit is ultimately a collection of N scalar-valued ones. However, these units will be activated and interpreted as a group. Therefore, it is apt to think of them as constituting one entity. In this chapter, we will build several unsupervised learning models using such units including Boltzmann Machines and autoencoders, and discuss the intuition behind their design. We will also present a qualitative analysis of what these networks learn to represent.

5.1 Motivation

Before we describe the models in detail, it is useful to highlight the motivations for having such hidden units and illustrate this with an example.

5.1.1 Representing a Space of Features and Equivariance Within it

In a typical neural net, each neuron can be thought of as a detector for a *particular feature*, and its activation is a measure of how strongly that feature is detected. On the other hand, in the case of directional units, each unit will be associated with a *space of features*. Its activation $\mathbf{h} \in \mathbb{R}^N$ will be thought of as a product of a unit norm direction $\hat{\mathbf{h}}$ and a non-negative scalar magnitude $||\mathbf{h}||$. The direction will tell us which feature (within the feature space) is present and the magnitude will be a measure of how strongly that feature is present. In other words, each directional unit has a coordinate frame associated with it. This coordinate frame is laid out on the surface of a unit sphere in $\mathbb{R}^N$. Each point in this coordinate frame corresponds to a "pose" that features in this space can take. For example, we can think of a directional unit that detects edges in a region of the visual field and encodes the

precise position of the edge within this region through its direction. Examples of such units are shown in Fig. 5.1. If the edge moves slightly, the pose rotates accordingly. Therefore, small changes in the position or orientation of the edge lead to an equivariant change in the pose of the feature. However the magnitude of the feature is invariant to the precise location of the feature and only indicates its presence. In this manner, directional units can concisely represent a space of features. They can change their pose smoothly in response to change in the pose of their input. A network of directional units can, in a sense, be "locked" in terms of relative poses. Any global change in the input's pose can be simply transferred equivariantly throughout the network. The potential to provide a general equivariance learning framework is a strong motivation for this work.

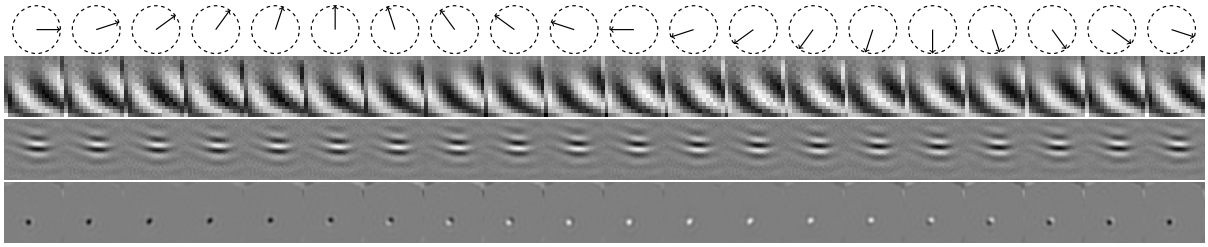


Figure 5.1: Examples of 2D-Directional units. The first row indicates the pose of the feature as a direction. Each subsequent row corresponds to a different directional unit and shows the features which have that pose.

5.1.2 Coincidence Detection

Another motivation for having directional units is that they can be made sensitive to coincidences. If the weight matrix connecting one directional unit to another is orthogonal, then this weight matrix essentially encodes the relative pose transformation between the two units. In other words, if the two units correspond to a part and the whole of an object respectively, then the weight matrix tells us how to transform the pose of the part to get the pose of the whole (and vice-versa). In a feed-forward operation (Fig. 5.2), the whole would receive relative judgements of its pose from all of its parts and will add them up. If all the poses agree, the resulting vector sum will have a large magnitude and its direction will be the agreed upon pose of the whole. This makes sense because if all the parts agree then the whole must be present and confidently so. If the poses disagree, then the resulting sum will have a smaller magnitude, reflecting that the evidence does not strongly support the existence of the whole.

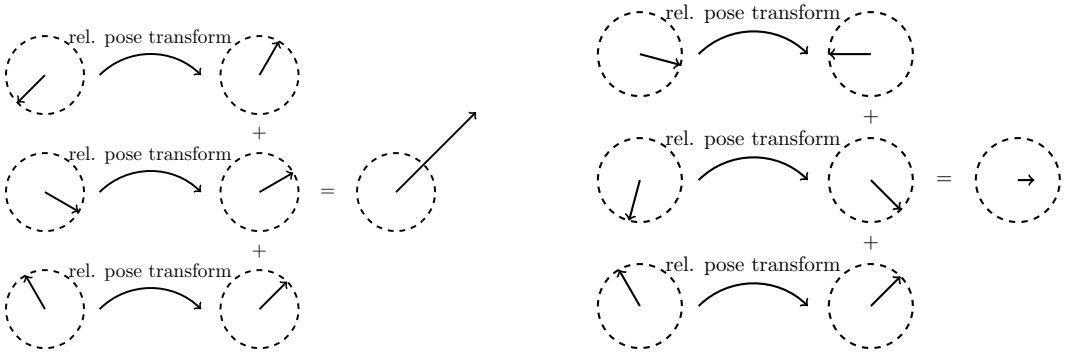


Figure 5.2: Directional units can look for agreement of pose.

5.1.3 Taking the Timing of Neuron Firing into Account

One aspect of biological neural networks that has been not been borrowed into popular artificial neural networks is the fact that neurons operate in a continuous time setting. If two spikes incident on a neuron arrive at roughly the same time, they are likely to add up to produce a bigger membrane potential compared to the case when their arrival is distant in time. This could make the difference between the receptor neuron firing or not. While a lot of work has been done in modeling spiking neurons, these models have not yet produced the same kind of impressive results as simpler models that assume all computations happen synchronously. One way to model the effects of timing, without actually performing asynchronous computations, is to associate each activation with a phase in addition to a magnitude, as done in 2-D directional units. Inputs arriving with the same phase will lead to an activation with a bigger magnitude than inputs with different phase which will tend to cancel each other. While differently timed spikes cannot cancel each other in the same way that inputs to a 2-D directional unit can, directional units help capture at least some aspects of timing. Moreover, while biological neurons are limited to one time axis, N-dimensional directional units (for $N > 2$) can generalize the notion of asynchrony in time to multiple "time" axes.

5.2 Related Work

Directional units, especially 2-D ones, have been well studied. 2-D directional units can be represented as complex-valued units. Noest (1987) introduced Phasor neural networks which were made out of units that had a phase and magnitude. These units were applied to model associative memories (Noest, 1988). Eckhorn et al. (1988) studied neurons as oscillators endowed with a phase. Baldi and Meir (1990) used coupled oscillators to model textures. Gislén et al. (1992) proposed Rotor Neurons whose states lie on the surface of a D dimensional sphere and used them to study the dynamics of electric charges moving on such a sphere. Lengyel and Dayan (2007) described a memory model for the hippocampus using neurons with phase. Work has also been done to model neural synchrony (Uhlhaas et al., 2009) using complex valued hidden units (Reichert and Serre, 2013). Zemel et al. (1992, 1995a) proposed a Boltzmann Machine for 2-D directional units, which was the first inspiration for our work and shares a lot of the same motivations as described above. Our work builds upon this by generalizing to N-dimensional directional units, combining directional units of different dimensionalities, interfacing real-valued and binary units with directional units, using directional units for building feed-forward networks and using temporal coherence for training these networks. Another major influence was the work of Cadieu and Olshausen (2012) which proposed a sparse coding based model with complex-valued coefficients and weights. Here temporal coherence was exploited by enforcing that amplitude and time derivatives of phase must change slowly with time. In our framework, the same idea can be applied to feed-forward networks made of directional units. The advantage over sparse coding is that these networks can be made deep, all layers can be trained together with backprop and inference can be done by a feed-forward computation.

5.3 Directional-Unit Boltzmann Machines

In this section we develop a Boltzmann Machine model which uses directional units. This will be a generalization of the DUBM model proposed by Zemel et al. (1992).

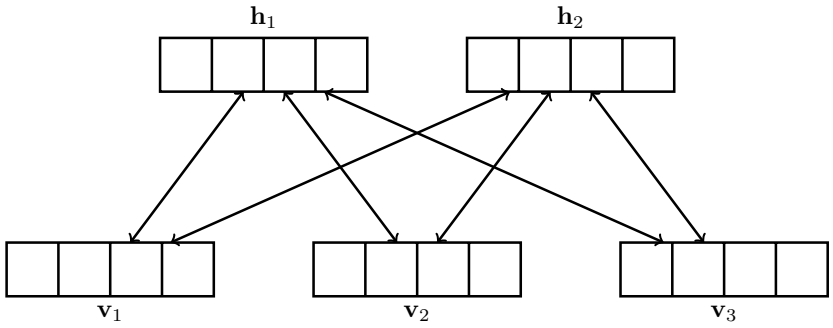


Figure 5.3: A Directional RBM with $N = 4$, $I = 3$ and $J = 2$.

5.3.1 Model Description

First, we will build a Restricted Boltzmann Machine model where both visible and hidden units are directional and have the same dimensionality. Suppose we have visible units $V = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_I)$ and hidden units $H = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_J)$. Each $\mathbf{v}_i$ and $\mathbf{h}_j$ is an N-dimensional vector of unit norm (Fig. 5.3). We can define the following energy function and probability distribution

$$E(V, H; \theta) = -\sum_{i=1}^{I}\sum_{j=1}^{J} \mathbf{v}_i^\top \mathbf{W}_{ij} \mathbf{h}_j$$
$$P(V, H; \theta) = \frac{1}{Z(\theta)} e^{-E(V,H)},$$

where $\theta = \{\mathbf{W}_{ij} | 1 \leq i \leq I, 1 \leq j \leq J\}$ is the set of all model parameters and $Z(\theta)$ is the normalizing constant. Each $\mathbf{W}_{ij}$ is an $N \times N$ scaled orthogonal matrix, that is, $\mathbf{W}_{ij} = c_{ij}\boldsymbol{\mathcal{W}}_{ij}$, where c_{ij} is a non-negative scalar and $\boldsymbol{\mathcal{W}}_{ij}^\top = \boldsymbol{\mathcal{W}}_{ij}^{-1}$. The contribution from any pair of directional units $\mathbf{v}_i$ and $\mathbf{h}_j$ to the energy is

$$E_{ij} = -c_{ij}\mathbf{v}_i^\top \boldsymbol{\mathcal{W}}_{ij} \mathbf{h}_j.$$

Intuitively, this energy function says that the desired configuration of the two directional units $\mathbf{v}_i$ and $\mathbf{h}_j$ is such that when $\mathbf{h}_j$ is transformed by $\boldsymbol{\mathcal{W}}_{ij}$, it should be parallel to $\mathbf{v}_i$. Conversely, if $\mathbf{v}_i$ is transformed by $\boldsymbol{\mathcal{W}}_{ij}^\top$, it should be parallel to $\mathbf{h}_j$. This interaction is illustrated in Fig. 5.4. Note that it is important that $\boldsymbol{\mathcal{W}}_{ij}^\top$ be equal to $\boldsymbol{\mathcal{W}}_{ij}^{-1}$, otherwise the direction that $\mathbf{h}_j$ wants $\mathbf{v}_i$ to be in, would not return the favour to $\mathbf{h}_j$. In other words, we would like the preferences that $\mathbf{v}_i$ and $\mathbf{h}_j$ express for each other to be mutually reinforcing. This is a key property that binds the dimensions of a directional unit together and makes the unit function as one entity, as opposed to a collection of N scalar units. It is helpful to recall the similar behaviour in standard binary Boltzmann Machines. In these models, individual scalar-valued units express mutually reinforcing preferences for each other, for example, consider two binary units v_i and h_j connected with a weight w_{ij}. If v_i is 1 and w_{ij} is positive, then it wants h_j to be 1. In return, if h_j is 1 it wants v_i to be 1. The orthogonality constraint is the multi-dimensional generalization of this property. In the DUBM model proposed by Zemel et al. (1992), the states and weights were represented as complex numbers. In our formalism, this can be seen as the 2×2 weight matrix $\boldsymbol{\mathcal{W}}_{ij}$ being constrained to be a rotation matrix. The orthogonal constraint that we propose is a weaker constraint (since orthogonal matrices can represent rotations, reflections and combinations

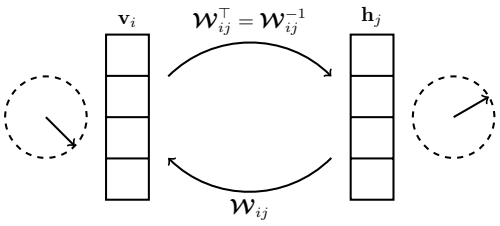


Figure 5.4: The interaction between two directional units.

thereof). We chose to do so because orthogonality constraints satisfy the mutual reinforcing requirement, are easier to extend to higher dimensional spaces, and can be extended to connecting differently sized directional units.

The non-negative scalar c_{ij} encodes the strength of this connection. If c_{ij} is large, then the two units have a strong mutual preference for their relative poses. If it is small (close to zero), then the two units are more or less independent.

With this intuition regarding pair-wise interactions in mind, we now zoom-out and look at the how a particular directional unit interacts with all the other units that are connected to it (Fig. 5.5).

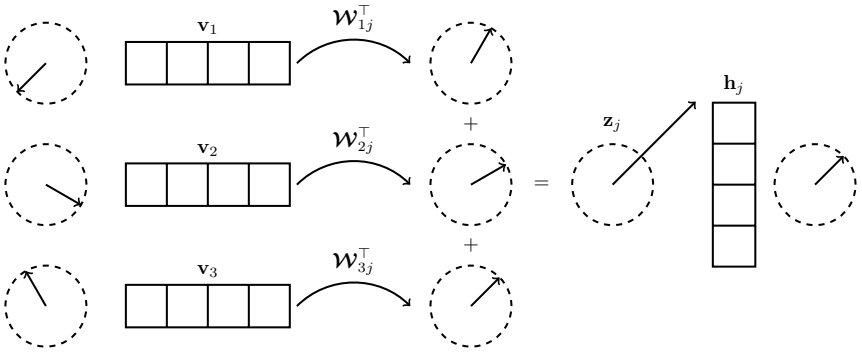


Figure 5.5: The interaction between directional unit $\mathbf{h}_j$ and all the visible units.

The contribution to the total energy coming from hidden unit $\mathbf{h}_j$ is

$$E_j = -\left(\sum_i \mathbf{v}_i^\top \mathbf{W}_{ij}\right)\mathbf{h}_j = -\mathbf{z}_j^\top \mathbf{h}_j,$$

where $\mathbf{z}_j = \sum_i \mathbf{W}_{ij}^\top \mathbf{v}_i$ is the total "input" coming into hidden unit j from all the visibles. The conditional distribution on the state of any hidden unit j given this total input is

$$P(\mathbf{h}_j|V;\theta) = \frac{e^{-E_j}}{\int_{||\mathbf{h}'_j||=1} e^{-E'_j} d\mathbf{h}'_j} = \frac{e^{\mathbf{z}_j^\top \mathbf{h}_j}}{\int_{||\mathbf{h}'_j||=1} e^{\mathbf{z}_j^\top \mathbf{h}'_j} d\mathbf{h}'_j} = \frac{1}{C_N(\mathbf{z}_j)} e^{\mathbf{z}_j^\top \mathbf{h}_j} \tag{5.1}$$

where the normalizing constant $C_N(\mathbf{z}_j)$ is an integral over the domain of $\mathbf{h}_j$ which is a unit sphere in $\mathbb{R}^N$. This distribution says that the probability that directional unit j takes a value $\mathbf{h}_j$ is proportional to $\exp(\mathbf{z}_j^\top \mathbf{h}_j)$. This implies that directions $\mathbf{h}_j$ that align with $\mathbf{z}_j$ will have a high probability density. If $||\mathbf{z}_j||$ is large, the distribution on $\mathbf{h}_j$ will be be sharply peaked around the direction of $\mathbf{z}_j$. This distribution is called the von Mises-Fisher distribution and has several nice properties, for example, it is the maximum

entropy distribution over directions given a mean and variance (analogous to Gaussian distributions in $\mathbb{R}^N$).

For the rest of this discussion, we will omit the subscript j, since we are describing the distribution at one hidden unit only. Let $\mathbf{z} = \kappa\boldsymbol{\mu}$, where κ is the magnitude of $\mathbf{z}$ and $\boldsymbol{\mu}$ is unit norm. Then Eq. 5.1 can be written as

$$P(\mathbf{h}|V;\theta) = \frac{1}{C_N(\kappa)} e^{\kappa\boldsymbol{\mu}^\top \mathbf{h}}. \tag{5.2}$$

We will see later (Eq. 5.3) that the normalizing constant $C_N(\mathbf{z})$ depends only on κ and not on the direction $\boldsymbol{\mu}$. Fig. 5.6a shows what this probability density function looks like for different values of κ. We can see that large values of κ imply a more peaked distribution. An intuitive interpretation of this distribution is that from the perspective of a hidden unit, each visible unit is sending it a preference for what state that visible unit would like it to take. The total input is a weighted sum of all preferences. If all the visible units agree, then the magnitude of the total input, κ, will be large and the distribution of the hidden state will be sharply peaked around the agreed upon direction $\boldsymbol{\mu}$. If there is disagreement, then the magnitude will be smaller, leading to higher variance in the state of the hidden unit. Therefore, *disagreement about the state of a hidden unit causes an increase in its uncertainty.* This is an important property of directional units – that disagreements lead to, not only a small expected magnitude, but also a large degree of uncertainty. In the extreme case, if the input is zero, there is complete disagreement and the distribution becomes uniform.

In order to force a directional unit to look for coincidences among the incoming preferences, it is important to make sure that no single preference can overwhelm the directional unit. Therefore, we constrain the values of c_{ij} to be less than a fixed upper bound c which is set empirically. This ensures that a high value of κ can be achieved only when multiple pieces of evidence agree.

The normalization constant in Eq. 5.1 can be written as

$$C_N(\mathbf{z}) = \int_{||\mathbf{h}||=1} e^{\kappa\boldsymbol{\mu}^\top \mathbf{h}} d\mathbf{h} = \frac{(2\pi)^{N/2}}{\kappa^{N/2-1}} I_{N/2-1}(\kappa) \equiv C_N(\kappa). \tag{5.3}$$

Here I_p is the modified Bessel function of the first kind and order p. We find that $C_N(\kappa\boldsymbol{\mu})$ depends only on κ and will be denoted as $C_N(\kappa)$. Under this distribution, the expected value of $\mathbf{h}$ is

$$E[\mathbf{h}|V;\theta] = E[\mathbf{h}|\kappa\boldsymbol{\mu}] = \int_{||\mathbf{h}||=1} \mathbf{h} \frac{1}{C_N(\kappa)} e^{\kappa\boldsymbol{\mu}^\top \mathbf{h}} d\mathbf{h} = \boldsymbol{\mu} \frac{I_{N/2}(\kappa)}{I_{N/2-1}(\kappa)} = \boldsymbol{\mu} S_N(\kappa) \tag{5.4}$$

A derivation for Eqs. 5.3, 5.4 can be found in Appendix A.2. The expected value of a hidden unit is the normalized total input $\boldsymbol{\mu}$, scaled by a function $S_N(\kappa)$ which depends on the magnitude of the total input κ. This scaling function is the ratio of modified Bessel functions of the first kind. Intuitively Eq. 5.4 can be seen as an activation function. It takes $\mathbf{z}$ as input, decomposes it into a direction $\boldsymbol{\mu}$ and magnitude κ, and outputs the direction scaled by a factor that depends on the magnitude. $S_N(\kappa)$ is a ratio of modified Bessel functions which are individually non-trivial to compute. However, their ratio can be computed quickly (Appendix A.3). The scaling factor has the property that for all $N \geq 2$, $S_N(0) = 0$ and $S_N(\kappa) \to 1$ as $\kappa \to \infty$. It is monotonically increasing. Fig. 5.6b shows the behaviour of this scaling function for different values of N. As the magnitude of the input increases, the scaling factor tends to one. This can also be seen through Fig. 5.6a where as κ increases, the variance shrinks and expected

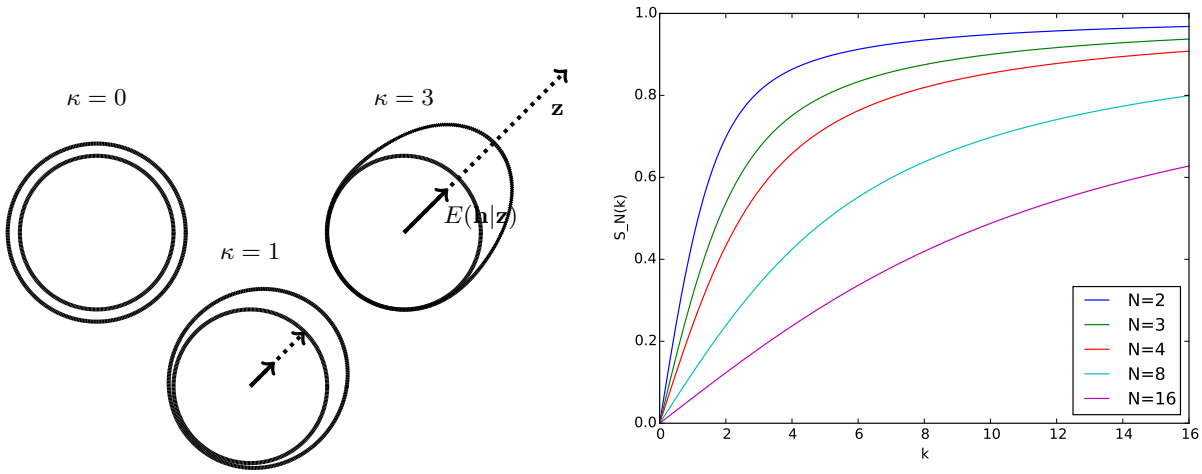


Figure 5.6: **Left** Conditional distribution on the state of a 2-D directional unit. The total input coming into the unit is $\mathbf{z}$ and $\kappa = ||\mathbf{z}||$. As κ increases the distribution gets more concentrated. **Right** Ratio of Bessel functions activation $S_N(\kappa)$.

value of $\mathbf{h}$ tends to $\boldsymbol{\mu}$. For this to happen, the scaling function must tend to 1. When κ is close to zero, the distribution is close to uniform and the expected value is close to 0. Therefore, the scaling function must tend to 0. Other functions which have this property include $\frac{\kappa}{1+\kappa}$ or $\frac{\kappa^2}{1+\kappa^2}$, which may be considered as replacements, especially for the feed-forward models that we develop later. However, it is not clear what kind of energy function they would correspond to.

5.3.2 Connecting Directional Units of Different Dimensionalities

The dimensionality of a directional unit limits the size and complexity of the space of features that can be represented by that unit. It is conceivable that we may want to construct networks where more complex high-dimensional directional units can interact with simpler low-dimensional ones, the intuition being that simple features (such as edges) have few degrees of freedom and fewer dimensions are needed to capture their pose, compared to more complex features which require more degrees of freedom.

The model described above can be extended to include directional units of different dimensionalities. Suppose we have directional units $\mathbf{v}_i \in \mathbb{R}^N$ and $\mathbf{h}_j \in \mathbb{R}^M$, where $N < M$. Let $\mathbf{W}_{ij}$ be an $N \times M$ weight matrix connecting these units. We can use the same energy function as before, but replace the orthogonality constraint on the weights by a semi-orthogonality constraint, i.e., $\mathbf{W}_{ij} = c_{ij}\boldsymbol{\mathcal{W}}_{ij}$ where $\boldsymbol{\mathcal{W}}_{ij}\boldsymbol{\mathcal{W}}_{ij}^\top = I_{N\times N}$. Since $N < M$, we cannot hope to fully recover the direction of $\mathbf{h}_j$ by transforming it using $\mathbf{W}_{ij}$ and transforming back using $\mathbf{W}_{ij}^\top$. However, this semi-orthogonality constraint ensures that at least the component of $\mathbf{h}_j$ that lies in the M-dimensional subspace spanned by $\mathbf{W}_{ij}$ can be recovered. In other words, when $\mathbf{v}_i$ is transformed using $\mathbf{W}_{ij}^\top$ into an M-dimensional vector, it can only span a subspace of dimensionality N within the larger space that it lives in. This means that the low-dimensional unit can only specify its preference over this subspace of the high dimensional unit. Conversely, when $\mathbf{h}_j$ is transformed using $\mathbf{W}_{ij}$ to an N-dimensional space, it will lose any component of its pose that is orthogonal to the subspace spanned by $\mathbf{W}_{ij}$. This interaction is illustrated in Fig. 5.7. This falls in line with the intuition for connecting such units together. Essentially, the higher dimensional

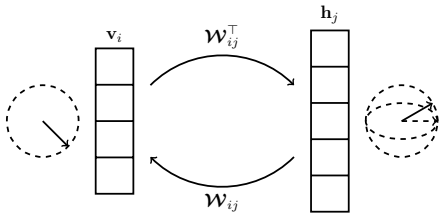


Figure 5.7: The interaction between two directional units of different dimensionalities. The unit with smaller dimensionality states preferences in a sub-space of the unit with higher dimensionality.

unit will take on a pose such that its projections into the lower dimensional subspaces simultaneously agree with the poses of all those units. Any lower dimensional unit cannot propose the complete pose of the higher dimensional one since it doesn't have enough degrees of freedom. But it can state a preference as far as it own subspace is concerned. In this way, each low-dimensional directional unit models a simple feature and the network's weights figure out how a more complex directional unit can be formed out of them.

5.3.3 Connecting Directional Units with Non-Directional units

Direction-valued hidden units can be used with binary and real-valued visible units as well. Suppose we have real-valued visible units $\mathbf{v} = (v_1, v_2, \ldots, v_N) \in \mathbb{R}^N$ and directional hidden units $H = (\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_J)$, where each $\mathbf{h}_j$ is an M-dimensional vector with unit norm. If we wish to model the real-valued units as Gaussian, we can define the energy function

$$E(\mathbf{v}, H; \theta) = \sum_{i=1}^{N} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{N}\sum_{j=1}^{J} \frac{v_i - b_i}{\sigma_i} \mathbf{w}_{ij}^\top \mathbf{h}_j \tag{5.5}$$

where $\mathbf{w}_{ij}$ is an M-dimensional vector of weights connecting visible unit v_i with hidden unit $\mathbf{h}_j$; σ_i and b_i are the mean and spread of the Gaussian at each visible unit, and θ is the set of all these model parameters. Let $\mathbf{W}_j = \mathbf{w}_{\bullet j}$ be an $N \times M$ weight matrix whose i-th row is $\mathbf{w}_{ij}^\top$. This matrix connects the visible units to hidden unit j. We impose a semi-orthogonality constraint, i.e., $\mathbf{W}_j = c_j \boldsymbol{\mathcal{W}}_j$ where $\boldsymbol{\mathcal{W}}_j^\top \boldsymbol{\mathcal{W}}_j = I_{M \times M}$ (assuming $M < N$). The reason for doing so is similar to that in the previous section – we want the preference expressed by each directional unit towards $\mathbf{v}$ to in turn reinforce its own state.

The conditional distribution on the visibles given the hiddens is factorial $P(\mathbf{v}|H;\theta) = \prod_i P(v_i|H;\theta)$ where

$$P(v_i|H;\theta) = \mathcal{N}\left(b_i + \sigma_i \sum_j \mathbf{w}_{ij}^\top \mathbf{h}_j, \sigma_i^2\right).$$

The conditional distribution on the hiddens given the visibles is von Mises-Fisher

$$P(\mathbf{h}_j|V;\theta) = \frac{1}{C_N(\kappa_j)} e^{\mathbf{z}_j^\top \mathbf{h}_j}, \quad \text{where } \mathbf{z}_j = \sum_i \frac{v_i}{\sigma_i} \mathbf{w}_{ij} \text{ and } \kappa_j = ||\mathbf{z}_j||.$$

Similarly, Bernoulli-Directional Unit RBMs can also be defined using the energy function

$$E(\mathbf{v}, H; \theta) = -\sum_{i=1}^{N} v_i b_i - \sum_{i=1}^{N}\sum_{j=1}^{J} v_i \mathbf{w}_{ij}^\top \mathbf{h}_j.$$

The conditionals are

$$P(v_i = 1|H;\theta) = \sigma(b_i + \sum_j \mathbf{w}_{ij}^\top \mathbf{h}_j), \text{ where } \sigma(x) = 1/(1+e^{-x})$$

$$P(\mathbf{h}_j|V;\theta) = \frac{1}{C_N(\kappa_j)} e^{\mathbf{z}_j^\top \mathbf{h}_j}, \quad \text{where } \mathbf{z}_j = \sum_i v_i \mathbf{w}_{ij} \text{ and } \kappa_j = ||\mathbf{z}_j||.$$

In this chapter, we will use these models to train on binary and real-valued image pixels.

5.3.4 Inducing Sparsity in Directional Units

Sparsity is a natural prior to presume in many representation learning schemes because it implies that among the features that we wish to learn, only a few should be necessary to explain any given instance of the data. In standard neural networks this is imposed by encouraging activations of neurons to be zero. By definition, directional units take on values on the unit sphere. So it does not make sense for them to be zero. However, they can be zero in expectation. According to Eq. 5.4, the expected value of a directional unit will be small if the length κ of its total input vector is small. To impose a prior that each directional unit ought to be 'off" most of the time, we place a sparse prior on κ. Following previous work done in the context of sparse coding (Olshausen and Field, 1996; Cadieu and Olshausen, 2012) we use a Cauchy distribution

$$P(\kappa) \propto \left(1 + \left(\frac{\kappa}{\sigma}\right)^2\right)^{-1},$$

where σ is the scale associated with this distribution. This heavy-tailed prior, peaked at zero, essentially says that we expect κ to be small most of the time, but when it does end up big, it should not be penalized too much for trying to grow even more.

5.3.5 Contrastive Divergence Learning

We have described several Restricted Boltzmann Machine models that use directional units. They differ in terms of the domain of their random variables. However, the form of the energy function in each of them is essentially the same as that of standard RBMs. Therefore, the learning algorithms for standard RBMs can be directly used.

Contrastive Divergence (CD) (Hinton, 2002; Hinton et al., 2006) and its variants (Tieleman, 2008; Tieleman and Hinton, 2009) have been widely used for training RBMs. CD learning is an approximation to maximum likelihood learning. Maximum likelihood learning requires computing the difference of sufficient statistics on the data and model distribution. For RBMs, computing the statistics on the data distribution is tractable, but that on the model distribution is not. In CD learning, MCMC sampling is used to approximate the statistics on the model distribution. This requires being able to sample from the conditional distributions on the visibles and hiddens. CD learning has been discussed in previous chapters in the context of training RBMs and DBMs where binary and real-valued units were involved. Therefore, the only new thing we need in order to train the models introduced in this chapter, is the

ability to deal with directional units by computing their expected value under and drawing samples from the N-dimensional von Mises-Fisher distribution for any given value of $N \geq 2$. Eq. 5.4 specifies how to compute the expected value of a directional unit under this distribution and a detailed derivation can be found in Appendix A.2. For sampling, we use the fast rejection sampling algorithm from Wood (1994). The sampling algorithm is described in Appendix A.4 and can be implemented efficiently on GPUs.

5.3.6 Preserving Orthogonality

In the models that we described, we imposed orthogonality (or semi-orthogonality) constraints on weight matrices involving directional units. Such constraints are often encountered in machine learning problems such as ICA (Hyvärinen and Oja, 2000), PCA (Jolliffe, 1986, 1995) and its variants such as Sparse PCA (Zou et al., 2004), and various tensor decomposition problems (Anandkumar et al., 2014; Hsu and Kakade, 2013). The set of orthonormal matrices $\mathbb{N}^{N\times M} \subset \mathbb{R}^{N\times M}$ has been well studied (Grassmann, 1862; Stiefel, 1935). This set constitutes a Stiefel manifold which is Riemannian under the canonical inner-product $d(A, B) = trA^\top B$. This structure can be used to formulate optimization schemes which follow a path on this manifold.

Suppose we are minimizing an objective function $L(W)$ under the constraint $W^\top W = I$, where $W \in \mathbb{R}^{N\times M}$ is a matrix with $M \leq N$. Starting with a W that satisfies the constraint, taking a step of size η along the gradient descent direction G will move it to $W \leftarrow W - \eta G$ which, in general, will not satisfy the constraint. A commonly used method to impose orthogonality is to project W to a "nearest" orthogonal matrix, bringing it back to the Stiefel manifold. If "nearest" is in the sense of the Frobenius norm, then the projection can be computed by taking the SVD of $W = U\Sigma V^\top$ and replacing the singular values by in Σ by 1. Therefore $W' = UV^\top$ is the required projection. Alternatively, this can be computed as $W' = W(W^\top W)^{-\frac{1}{2}}$. However, these methods require computing eigen value decompositions or computing inverses which may cause instabilities and are non-trivial to accelerate on GPU hardware. In our experiments, we used an iterative scheme proposed by Björck and Bowie (1971) that relies only on computing dot products and works well when the matrix being projected is not too far from being orthogonal (this is the case for us because at every step, we have an orthogonal W and we only make small changes to it). In this iterative method, we run the update

$$W \leftarrow W + \frac{1}{2}W(I - W^\top W), \tag{5.6}$$

until $||I - W^\top W||$ is less than some pre-specified tolerance. It is easy to see that orthogonal matrices are fixed points for these updates.

Other techniques have been proposed which try to follow the geodesics on the Stiefel manifold (Edelman et al., 1999; Nishimori, 1999; Fiori, 2001). In these approaches the optimization takes an η-size step along the geodesic of the manifold, instead of doing so in Euclidean space and then projecting back. The hope is that this will respect the structure of the domain over which we are optimizing, leading to faster convergence. This closely relates to the intuition behind natural gradient approaches (Amari, 1998). Given the weights W_t, gradient G_t and learning rate η at time step t, the update rule is

$$P_t \equiv G_t W_t^\top - W_t G_t^\top$$
$$W_{t+1} = \exp(-\eta P_t) W_t$$

where $\exp(\bullet)$ denotes the matrix exponential $\exp(X) = I + X + X^2/2 + \ldots$ where X is a square matrix. Computing the exponential accurately is somewhat expensive. Therefore, we approximate $\exp(X) \approx I + X$ which is reasonable if $X = -\eta P_t$ is a small gradient step. Therefore, the update becomes

$$\begin{aligned} W_{t+1} &= (I - \eta P_t) W_t \\ &= W_t - \eta \left(G_t W_t^\top - W_t G_t^\top \right) W_t \\ &= W_t - \eta \left(G_t W_t^\top W_t - W_t G_t^\top W_t \right) \\ &= W_t - \eta \left(G_t - W_t G_t^\top W_t \right) \end{aligned} \tag{5.7}$$

because $W_t^\top W_t = I$. Comparing Eq. 5.7 to the standard update $W_{t+1} = W_t - \eta G_t$, it is easy to see that the new update rule is essentially modifying the gradient G_t by a correction term. It can be shown that this modified gradient lies in a plane that is tangent to the Stiefel manifold at W_t. In our experiments we found that this simple first order correction improves convergence. After making an update using this rule, we run iterative orthogonalization using Eq. 5.6 which typically converges in 1-2 steps. Note that the proposed models involve lots of relatively small orthogonal matrices (one for each pair of directional units). Therefore, it is efficient to run the computations required for this optimization using batched matrix-matrix dot products on GPUs. To summarize, the total overhead cost for preserving orthogonality can be kept fairly low by using iterative orthogonalization (Eq. 5.6) and corrected gradients (Eq. 5.7).

5.3.7 Qualitative Results

In this section, we visualize the features learned by RBMs that involve directional units. We train these models on affNIST (Tieleman, 2014) which is a dataset created by applying affine transforms to MNIST digits and placing them on a larger canvas (40×40).

Fig. 5.8 shows some features that were learned by a Binary-Directional unit RBM. The features are visualized as follows. Each directional unit represents a space of features specified by a 1600×2 weight matrix. Let the two columns of this matrix be $\mathbf{w}_1$ and $\mathbf{w}_2$. These two orthogonal vectors span the space of features represented by this directional unit. We visualize this space in two ways. One way is to look at these basis vectors. The first two rows in Fig. 5.8 show the basis vectors. The next two rows show an alternate visualization in terms of an amplitude and phase. To compute this visualization, we take each pair of weights w_{1i} and w_{2i} and represent them as an amplitude a_i and phase θ_i such that $w_{1i} = a_i \cos\theta_i$ and $w_{2i} = a_i \sin\theta_i$. The third and fourth rows show the amplitude and phase respectively. The region where the amplitude is high corresponds to the region of the input where the directional unit is active. From the phase image, we can see the pixels that need to be activated in phase to make this directional unit fire. The learned features correspond to edges of different sizes at different locations of the visual field.

Fig. 5.9 shows the features learned by a 3-D directional unit on the same dataset. Fig. 5.10 shows an alternative visualization of a 3D-directional unit. In this visualization, it can be seen that the pose of this 3-D directional unit has learnt to capture different orientations and locations of an edge in a localized region.

We found that sparsity regularization (Sec. 5.3.4) is very important to make these models work in

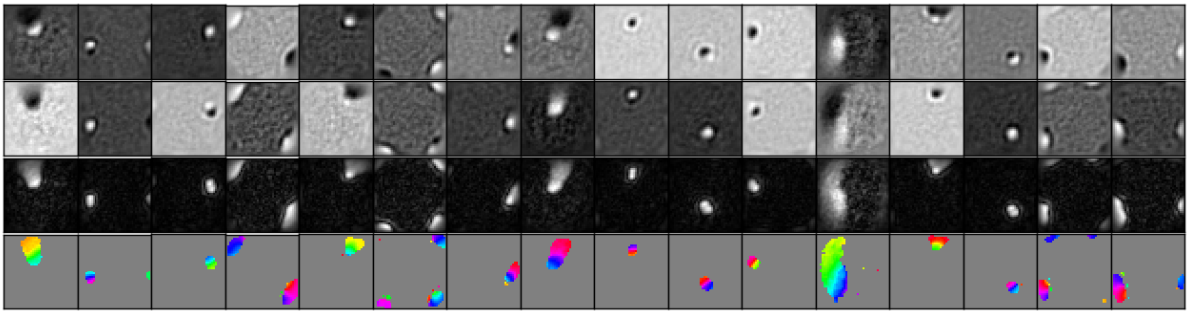


Figure 5.8: Features from a 2D Directional unit RBM on affNIST. Each column represents one directional unit. The first two rows are the two basis functions that connect the input to first and second dimensions of the directional unit. The next two rows show an alternate visualization of the same feature in terms of amplitude and phase. The third row is the amplitude of this feature. The bright region is where this feature will be active. The fourth row is the phase of the feature shown using a circular color-map. Regions where the amplitude is small have been grayed out.

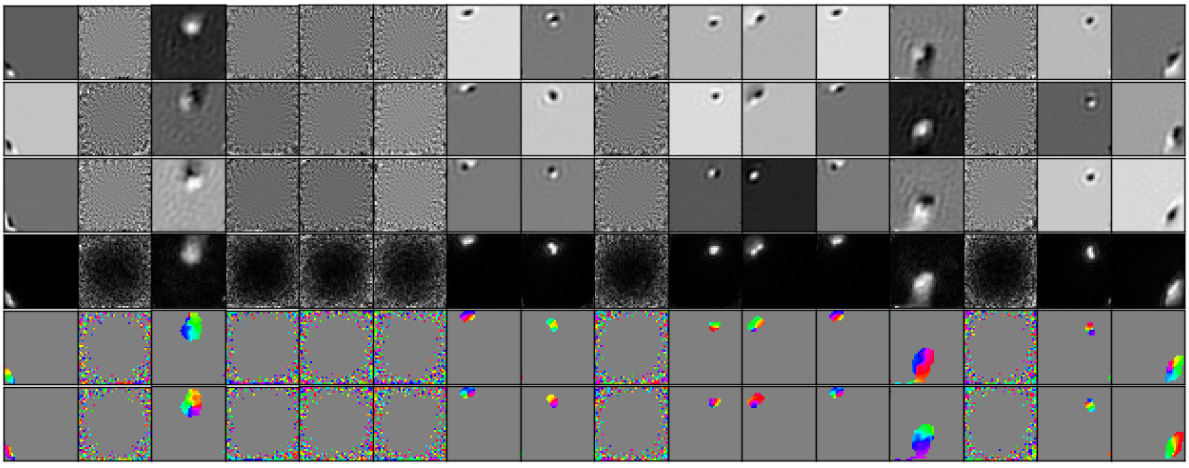


Figure 5.9: Features from a 3D Directional unit RBM on affNIST. The first three rows represent the three basis vectors. The next three represent the amplitude and phases. Note that some hidden units (for example, column 2) have failed to learn reasonable features.

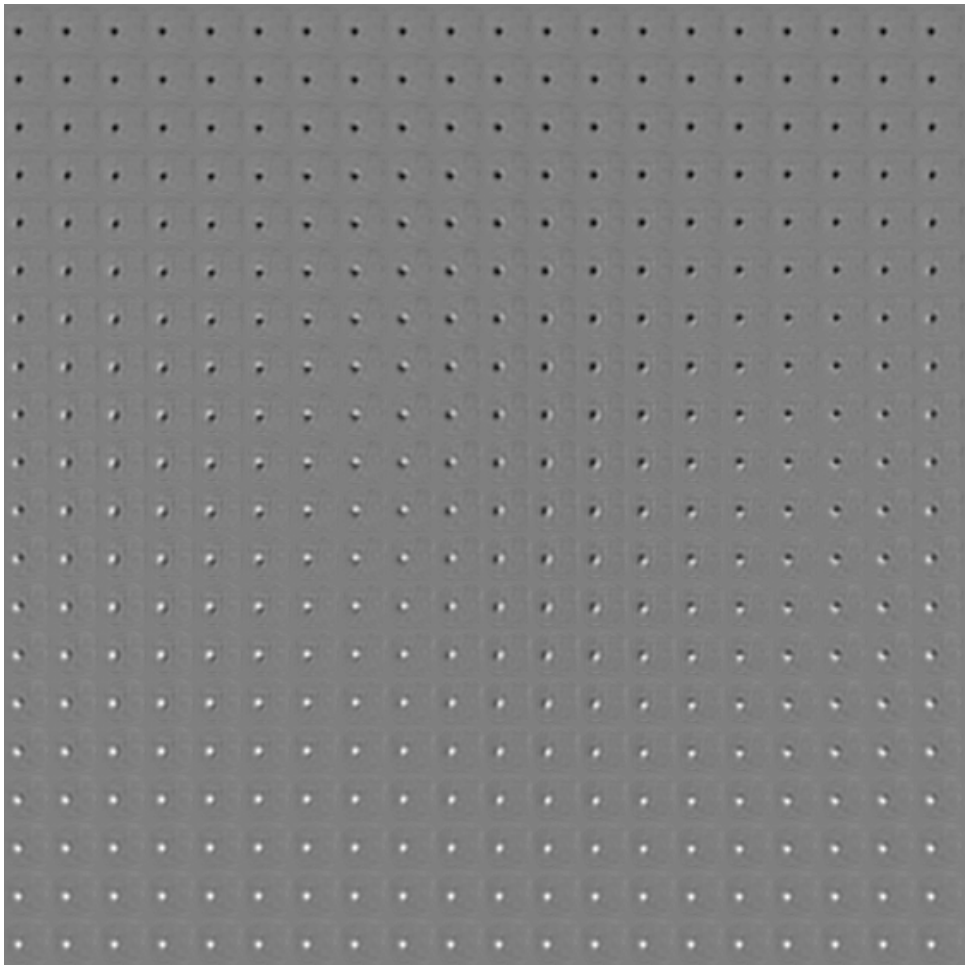


Figure 5.10: An example of a 3-D directional unit on MNIST. A 3D directional unit lies on 2-D manifold (the surface of a 3-D sphere). This figure is a visualization of this manifold. Going along the horizontal axis corresponds to walking along a circle of latitude on a 3-D sphere. Going from the top to bottom corresponds to moving from pole to pole. This unit has learned to represent edges in different orientations and positions within a local region.

practice.

5.4 Directional Unit Autoencoders

Directional units can be used in feed-forward networks, and in particular for making auto-encoders. This section describes how to feed forward and back propagate through directional units, and gives some qualitative results and analysis of the features learned by these networks.

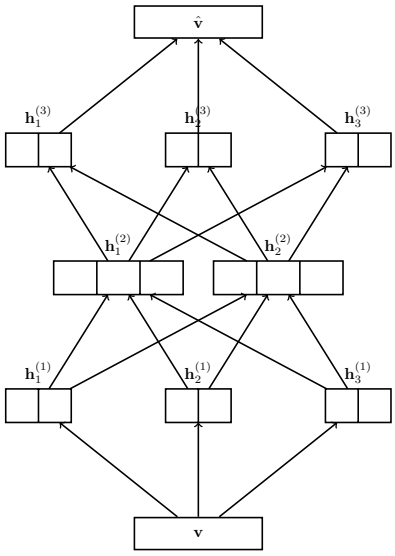


Figure 5.11: An example of a Directional unit Autoencoder.

5.4.1 Feed-forward Operation

In the feed-forward operation, the activation function for each directional unit is the expected value of that unit under the conditional von Mises-Fisher distribution which is given by Eq. 5.4. This is analogous to the case for binary unit RBMs and sigmoid autoencoders, where the expectation under the conditional Bernoulli distribution from binary RBMs, the sigmoid function, becomes a deterministic activation function in the autoencoder. Essentially, if an N-dimensional direction hidden unit receives a total input $\mathbf{z} = \kappa\boldsymbol{\mu}$, then the output from that unit is

$$\mathbf{y} = \boldsymbol{\mu} S_N(\kappa),$$

where $\kappa = ||\mathbf{z}||$ and $\boldsymbol{\mu}$ is the unit vector along the direction of $\mathbf{z}$. While $\boldsymbol{\mu}$ represents what feature is present, the scaling function $S_N(\kappa)$ represents how strongly it is present. Units that receive small magnitude inputs get suppressed by the scaling function. Units which receive a large magnitude input, output a normalized version of their inputs. Naively, computing the value of the scaling function would require computing the values of modified Bessel functions $I_{N/2}(\kappa)$ and $I_{N/2-1}(\kappa)$. However, there is a continued fraction expression for $S_N(\kappa)$ which makes it easy to approximate it very efficiently (Appendix A.3).

5.4.2 Backpropagating Through Directional Units

The feed-forward operation is a normalization followed by scaling. Therefore, the output is a smooth function of the input which means that the function is straight-forward to backpropagate through. The only non-trivial operation in the backpropagation is the derivative of the scaling function. Fortunately, there is a simple expression for it

$$\frac{\partial S_N(\kappa)}{\partial \kappa} = 1 - S_N(\kappa)^2 - \frac{N-1}{\kappa} S_N(\kappa).$$

Therefore, no expensive computations are required to backpropagate through the scaling function, if we are willing to spend some memory to store the values of $S_N(\kappa)$ computed during the forward pass. As $\kappa \to 0$, $\frac{\partial S_N(\kappa)}{\partial \kappa} \to \frac{1}{N}$, and as $\kappa \to \infty$, the derivative goes to zero.

5.4.3 Qualitative Results

Using these forward and backward computations, we can construct deep neural networks composed of directional units. For example, consider an autoencoder as shown in Fig. 5.11 for autoencoding image patches. It consists of real-valued input units, a hidden layer of directional units, followed by the real-valued output layer. Fig. 5.12 shows the features obtained by training this network on affNIST. The features are of different types. Some features learn to be localized edges of different frequencies, others are more global, covering the entire field of view.

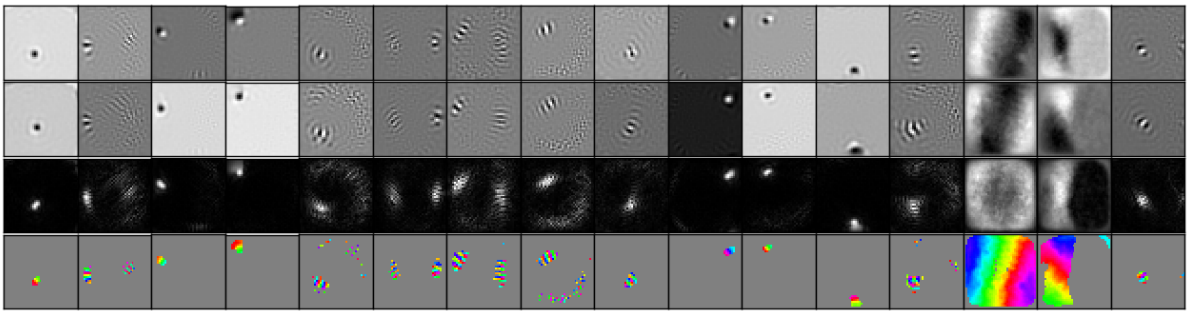


Figure 5.12: Features from a 2D Directional unit Autoencoder on affNIST.

5.5 Learning Directional Unit Autoencoders using Temporal Coherence

One of the motivations for using directional units is that they make it easy to succinctly represent feature spaces and associate a pose with each detected feature. As seen in previous sections, we are able to learn such feature spaces by training directional unit Boltzmann Machines and Autoencoders on static images. However, an even more powerful unsupervised learning signal can be obtained by training on videos and exploiting temporal coherence.

5.5.1 Motivation

Coherence across time is a prominent feature of visual sensory data. Things in the world change locally and slowly, if at all. This is a fundamental artifact of living in a low energy, low temperature physical system with localized and embodied objects. Consider an object moving across a visual frame. High-level properties of an object, for example object identity, do not vary at all with time. The object's pose varies slowly and smoothly. However low-level features, for example the state of a particular pixel or the response of a linear filter at some patch vary at a much faster rate, especially in the presence of background clutter and camera motion.

The use of temporal coherence as a signal for unsupervised learning is based on the idea that since we want to obtain high-level representations of objects, we ought to be looking for features that vary slowly (Hinton, 1989). Of course, in addition to varying slowly, these features must also explain the data, or have some entropy or variance to prevent collapse. Slow Feature Analysis (Wiskott and Sejnowski, 2002) is one popular approach. Neural network models exploiting slowness have also been proposed (Mobahi et al., 2009; Wang and Gupta, 2015). These models try to optimize some objective function, discriminative or generative, and impose an additional regularization that the neuron activations across adjacent frames must not be too different (typically in terms of L_1 or L_2 norm).

However, directional units allow us to make a finer judgement about what exactly we want to be slowly varying. Instead of saying that the activation of each unit should be the same across frames, we can ask for only the *length* of each directional unit to be the same, leaving the *direction* of the unit free to change as needed to model the data. This imposes the prior belief that while the presence of a feature must change slowly, its pose can change rapidly. In other words, the content of an image should change slowly but not the dynamic properties associated with that content. Therefore, instead of uniformly suppressing all change, we are selectively suppressing the change that models presence of a feature but not the one that models its pose. Optimizing under this prior should encourage the direction of any unit to try to explain the rapid changes in the data. Over a large dataset of videos, each directional unit will witness its input exhibiting pose changes in a variety of ways and must learn to model these using its direction, because at the next time step it will not be allowed to change its level of activation much but is free to change direction. This idea has previously been used in the excellent work of Cadieu and Olshausen (2012) which describes a two-layer sparse coding model with complex-valued coefficients and weights. In this model, the first layer is learned by asking the coefficients to model the data under the prior that amplitudes of the coefficients be slowly changing and sparse. This first layer sparse coder is similar to a one hidden layer RBM or autoencoder, where the hidden layer is made of 2-D directional units. However, the 2-D directional units in our model differ from the complex-valued coefficients in that after applying the activation function, the length of 2-D directional units is bounded by 1, whereas the complex coefficients are unconstrained. Also our model learns a coupled encoder-decoder model so that inference can be done in a feed-forward way. Our model is also easy to extend to deeper network architectures.

5.5.2 Model Description

Fig. 5.13 shows an autoencoder model augmented with a slowness prior. At each directional unit j, we add an additional cost that the lengths of the inputs to these units κ_j be close together across time, i.e,

$$L_{\text{slow}} = \sum_j \left(\kappa_j^{(t)} - \kappa_j^{(t+1)}\right)^2$$

or alternatively close in an L_1 sense,

$$L_{\text{slow}} = \sum_j |\kappa_j^{(t)} - \kappa_j^{(t+1)}|.$$

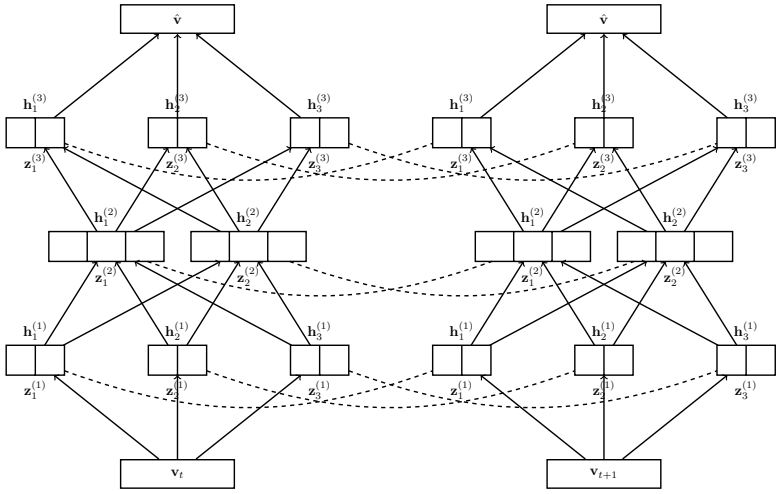


Figure 5.13: A Directional unit Autoencoder with temporal coherence loss. The units joined by a dashed line are encouraged to have the same input length.

The total loss function being minimized is then the sum of the reconstruction, sparsity and the slowness costs, weighted by their relative strengths

$$L = L_{\text{rec}} + \lambda_1 L_{\text{sparse}} + \lambda_2 L_{\text{slow}}.$$

5.5.3 Qualitative Results

We trained temporal coherence based autoencoders on videos of affNIST digits and natural videos from Cadieu and Olshausen (2012). Fig. 5.14 shows the features learned on affNIST. We can see that these features try to represent global rotation and shifts, as well as very localized translations.

Fig. 5.15 shows the features learned on 32×32 natural image patches. These features show that the model learns edges at different frequencies, orientations and locations.

5.6 Conclusion

In this chapter, we described a formalism for using directional units in different kinds of neural networks. Based on the preliminary qualitative experiments, we can conclude that these models are able to learn

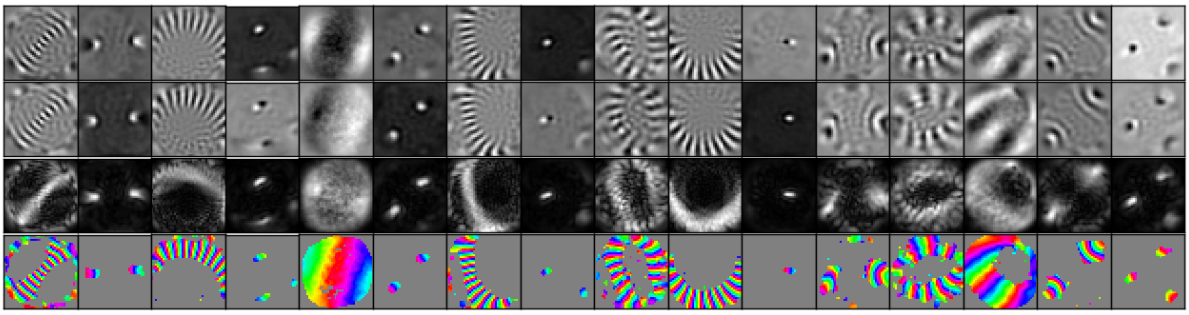


Figure 5.14: Features from a 2D Directional unit Autoencoder on affNIST trained with temporal coherence.

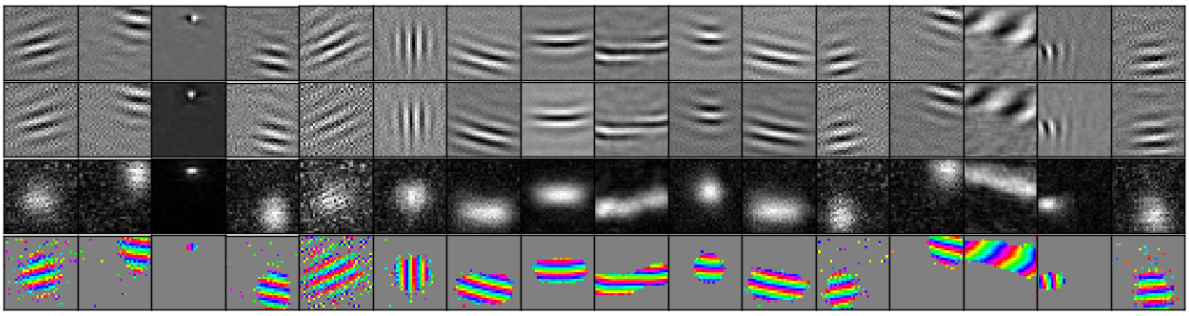


Figure 5.15: Features from a 2D Directional unit Autoencoder on natural video sequences trained with temporal coherence.

meaningful features. The fact that they can be equivariant to small changes in the pose of features, while at the same time being invariant to the overall presence of those features is encouraging. A lot more work is needed to make these models work in practice for real-world problems. However, these units provide an interesting alternative way of learning distributed representation. They could be especially useful in learning representations of optical flow because they are highly sensitive to low-level pose information.

This work also helped identify some interesting properties of the proposed models. In our formulation of directional unit Boltzmann Machines, directional units have no "off" switch. In other words, any sample state drawn from the Boltzmann Machine distribution will have all directional units present with unit magnitude. The absence of a unit is indicated by a high variance in the pose of that unit, as opposed to the unit taking any particular value. Therefore, *in expectation*, a directional unit can be turned off, but any sample will not contain this information. This leads to some interesting properties. One property is that these models are, in a sense, more entropy-driven than free-energy driven. One way to see this is that a directional unit being off is a high entropy situation (since the unit is free to take any value on the unit sphere), whereas in a standard binary Boltzmann Machine, a unit being off is a low entropy situation. So these models treat being off in very different ways. A directional unit is off when changing its state does not affect the energy, whereas in a standard RBM, a hidden unit is off when changing its state to "on" would have a big detrimental impact on the energy. Therefore, the preference to be off is guided by higher entropy in the case of directional units as opposed to being guided by lower energy, which is the case for binary RBMs. This property may have some desirable consequences, for example, it could encourage mixing of Markov chains more readily since they are unlikely to get stuck with a lot of directional units flailing about. It may also have undesirable consequences such as making

it harder to get reliable gradient estimates from small mini-batches. We have not been able to leverage this property to our advantage yet but we hope to do so in future. Mean-field formulations are less influenced by this, especially the feed-forward models discussed in this work.

One way to recover the property that samples from the model should indicate the presence or absence of features, we can add a binary gating unit with each directional unit. If the gate is off, then the corresponding directional unit does not participate in the energy function. This idea has been previously used in Hinton et al. (1999) in the context of building a hierarchical community of experts. This is a promising direction for future work with directional units.

In the current formulation, a directional unit represents the pose of a feature as a point on a unit sphere in N-dimensions. This restricts the space of variations of a feature to those that are topologically consistent with a unit sphere. In particular, this may exclude certain transformations that involve scaling. Therefore networks of directional units need to find clever ways to model these features. This is another shortcoming of our model that we hope to address in future.

Another fruitful direction for future work is using these units to do internal routing of information. If directional units can indeed discover agreements in pose, then it should be possible to use this assignment of parts to wholes to send information about the part only to its parent and preclude it from interfering in any other whole. This could not only help build more interpretable representations (with tangible parts and wholes), but also reduce the clutter and confusion caused by each unit trying to explain things that have already been explained, or using evidence that has already been used for a competing hypothesis. This imposes the inductive bias that there can be only one interpretation of the data at a time. This is a highly pronounced effect in biological cognition but something that has not explicitly been imposed in modern neural networks.

Chapter 6

Transfer Learning with Tree-based Priors

Transfer learning is the problem of extracting knowledge in the process of solving one or more tasks and applying it to a different but related task. The ability to transfer knowledge to new tasks can truly empower AI systems to become intelligent in a general sense. Therefore, this is an important problem to solve towards the goal of building an Artificial General Intelligence (AGI). The previous chapters of this thesis dealt with unsupervised learning, where the goal was to learn the structure in the space of *input data*. On the other hand, this chapter will deal with transfer learning, where the goal is to learn the structure in the space of *tasks*. This structure embodies the relationship and level of relatedness between different tasks. Figuring out how to represent this structure and how to make use of it to transfer knowledge is a fertile area of research. In this chapter, we focus on classification tasks and propose a method of learning this structure in the form of a tree over classes.

6.1 Overview

Learning classifiers that generalize well is a hard problem when only few training examples are available. For example, if we had only 5 images of a cheetah, it would be hard to train a classifier to be good at distinguishing cheetahs against hundreds of other classes, working off pixels alone. Any powerful machine learning model is likely to overfit the few examples. This chapter is based on the idea that performance can be improved using the natural structure inherent in the set of classes. For example, we know that cheetahs are related to tigers, lions, jaguars, and leopards. Having labeled examples from these related classes should make the task of learning from 5 cheetah examples much easier. Knowing this class structure should allow us to borrow "knowledge" from relevant classes so that only the distinctive features specific to cheetahs need to be learned. At the very least, the model should confuse cheetahs with these animals rather than with completely unrelated classes, such as cars or lamps.

Finding relatedness of tasks is a hard problem. This is because in order to find which tasks are related, we first need to gain knowledge about what the tasks are – i.e., have a good model for each one of them. But in order to learn a good model for a task (given only few examples), we need to transfer knowledge from related tasks, which requires knowing which classes are related. This creates a cyclic dependency. One way to circumvent it is to use an external knowledge source, such as a human, to

specify the class structure by hand. Another way to resolve this dependency is to iteratively learn a model of the what the tasks are and what relationships exist between them, using one to improve the other. In this chapter, we follow this bootstrapping approach.

We propose a way of learning class structure and classifier parameters in the context of deep neural networks. The aim is to improve classification accuracy for classes which have few examples. Deep neural networks trained discriminatively with back propagation have been shown to achieve state-of-the-art performance on difficult classification problems where a large amount of labeled data is available (Krizhevsky et al., 2012; Hinton et al., 2012a). The case of smaller amounts of data or datasets which contain rare classes has been relatively less studied. To address this shortcoming, our model augments neural networks with a tree-based prior over the last layer of weights. We structure the prior so that related classes share the same prior. This shared prior captures the features that are common across all members of any particular superclass. Therefore, a class with few examples, for which the model would otherwise be unable to learn good features, can now have access to good features just by virtue of belonging to the superclass.

Learning a hierarchical structure over classes has been extensively studied in the machine learning, statistics, and vision communities. A large class of models based on hierarchical Bayesian models have been used for transfer learning (Shahbaba and Neal, 2007; Evgeniou and Pontil, 2004; Bart et al., 2008; Daumé, 2009; Fei-Fei et al., 2006). The hierarchical topic model for image features of Bart et al. (2008) can discover visual taxonomies in an unsupervised fashion from large datasets but was not designed for rapid learning of new categories. Fei-Fei et al. (2006) also developed a hierarchical Bayesian model for visual categories, with a prior on the parameters of new categories that was induced from other categories. However, their approach is not well-suited as a generic approach to transfer learning because they learned a single prior shared across all categories. A number of models based on hierarchical Dirichlet processes have also been used for transfer learning (Xue et al., 2007; Salakhutdinov et al., 2011a). However, almost all of the the above-mentioned models are generative by nature. These models typically resort to Markov Chain Monte-Carlo (MCMC) approaches for inference, that are hard to scale to large datasets. Furthermore, they tend to perform worse than discriminative approaches, particularly as the number of labeled examples increases.

A large class of discriminative models that enable discovering and sharing information among related classes (Kim and Xing, 2010; Zweig and Weinshall, 2013; Kang et al., 2011) has also been used for transfer learning. Most similar to our work is the model proposed by Salakhutdinov et al. (2011b) which defined a generative prior over classifier parameters and a prior over tree structures to identify relevant categories. However, this work focused on a very specific object detection task and used an SVM model with pre-defined HOG features as its input. In this chapter, we demonstrate our method on two different deep architectures (1) convolutional nets with pixels as input and single-label softmax outputs and (2) fully connected nets pretrained using deep Boltzmann machines with image features and text tokens as input, and multi-label logistic units as output. Our model improves performance over strong baselines in both cases. Overall, our model learns low-level features, high-level features – as well as a hierarchy over classes – in an end-to-end way.

6.2 Model Description

Let $\mathcal{X} = \{\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^N\}$ be a set of N data points and $\mathcal{Y} = \{\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^N\}$ be the set of corresponding labels, where each label $\mathbf{y}^i$ is a K dimensional vector of targets. These targets could be binary, one-of-K, or real-valued. In our setting, it is useful to think of each $\mathbf{x}^i$ as an image and $\mathbf{y}^i$ as a one-of-K encoding of the label. The model is a multi-layer neural network (see Fig. 6.1a). Let $\mathbf{w}$ denote the set of all parameters of this network (weights and biases for all the layers), excluding the top-level weights, which we denote separately as $\boldsymbol{\beta} \in \mathbb{R}^{D \times K}$. Here D represents the number of hidden units in the last hidden layer. The conditional distribution over $\mathcal{Y}$ can be expressed as

$$P(\mathcal{Y}|\mathcal{X}) = \int_{\mathbf{w},\boldsymbol{\beta}} P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) P(\mathbf{w}) P(\boldsymbol{\beta}) d\mathbf{w} d\boldsymbol{\beta}. \tag{6.1}$$

In general, this integral is intractable, and we typically resort to MAP estimation to determine the values of the model parameters $\mathbf{w}$ and β that maximize

$$\log P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) + \log P(\mathbf{w}) + \log P(\boldsymbol{\beta}).$$

Here, $\log P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta)$ is the log-likelihood function and the other terms are priors over the model's parameters. A typical choice of prior is a Gaussian distribution with diagonal covariance:

$$\boldsymbol{\beta}_k \sim \mathcal{N}\left(0, \frac{1}{\lambda} I_D\right), \forall k \in \{1, \ldots, K\}.$$

Here $\boldsymbol{\beta}_k \in \mathbb{R}^D$ denotes the classifier parameters for class k. Note that this prior assumes that each $\boldsymbol{\beta}_k$ is independent of all other $\boldsymbol{\beta}_i$'s. In other words, a-priori, the weights for label k are not related to any other label's weights. This is a reasonable assumption when nothing is known about the labels. It works quite well for most applications with large number of labeled examples per class. However, if we know that the classes are related to one another, priors which respect these relationships may be more suitable. Such priors would be crucial for classes that only have a handful of training examples, since the effect of the prior would be more pronounced. In this work, we focus on developing such a prior.

6.2.1 Learning With a Fixed Tree Hierarchy

Let us first assume that the classes have been organized into a fixed tree hierarchy which is available to us. We will relax this assumption later when we learn this hierarchy by placing a prior over tree structures and find the maximum a-posteriori solution that respects both the prior and the likelihood. For now, we focus on evaluating the likelihood. For ease of exposition, consider a two-level hierarchy[1], as shown in Fig. 6.1b. There are K leaf nodes corresponding to the K classes. They are connected to S super-classes which group together similar basic-level classes. Each leaf node k is associated with a weight vector $\boldsymbol{\beta}_k \in \mathbb{R}^D$. Each super-class node s is associated with a vector $\boldsymbol{\theta}_s \in \mathbb{R}^D$, $s = 1, ..., S$. We define the following generative model for $\boldsymbol{\beta}$

$$\boldsymbol{\theta}_s \sim \mathcal{N}\left(\mathbf{0}, \frac{1}{\lambda_1} I_D\right), \quad \boldsymbol{\beta}_k \sim \mathcal{N}\left(\boldsymbol{\theta}_{\text{parent}(k)}, \frac{1}{\lambda_2} I_D\right). \tag{6.2}$$

[1]The model can be easily generalized to deeper hierarchies.

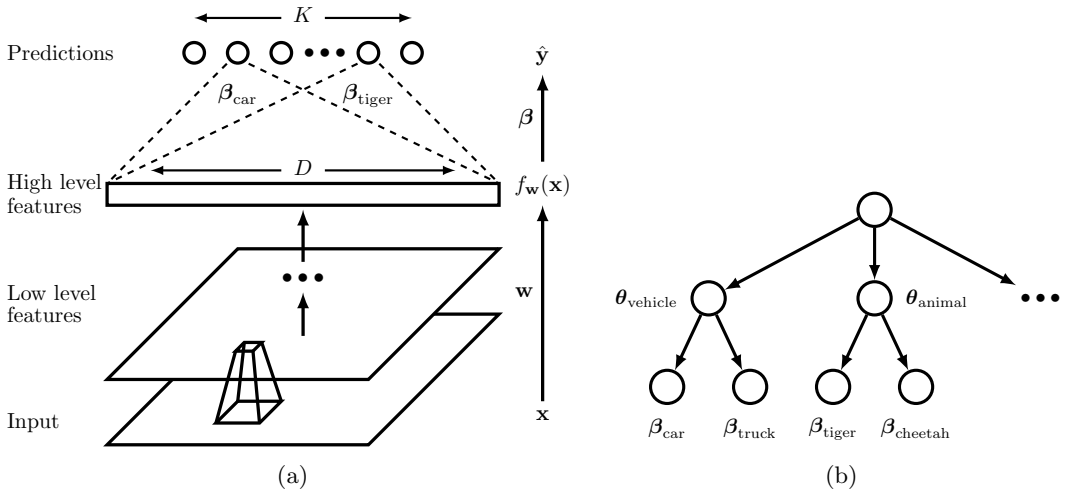


Figure 6.1: Our model: A deep neural network with priors over the classification parameters. The priors are derived from a hierarchy over classes.

This prior expresses relationships between classes. For example, it asserts that $\boldsymbol{\beta}_{\text{car}}$ and $\boldsymbol{\beta}_{\text{truck}}$ are both deviations from $\boldsymbol{\theta}_{\text{vehicle}}$. Similarly, $\boldsymbol{\beta}_{\text{cat}}$ and $\boldsymbol{\beta}_{\text{dog}}$ are deviations from $\boldsymbol{\theta}_{\text{animal}}$. Eq. 6.1 can now be re-written to include $\boldsymbol{\theta}$ as follows

$$P(\mathcal{Y}|\mathcal{X}) = \int_{\mathbf{w},\boldsymbol{\beta},\boldsymbol{\theta}} P(\mathcal{Y}|\mathcal{X},\mathbf{w},\beta)P(\mathbf{w})P(\boldsymbol{\beta}|\boldsymbol{\theta})P(\boldsymbol{\theta})d\mathbf{w}d\boldsymbol{\beta}d\boldsymbol{\theta}. \tag{6.3}$$

We can perform MAP inference to determine the values of $\{\mathbf{w}, \boldsymbol{\beta}, \boldsymbol{\theta}\}$ that maximize

$$\log P(\mathcal{Y}|\mathcal{X},\mathbf{w},\beta) + \log P(\mathbf{w}) + \log P(\boldsymbol{\beta}|\boldsymbol{\theta}) + \log P(\boldsymbol{\theta}).$$

In terms of a loss function, we wish to minimize

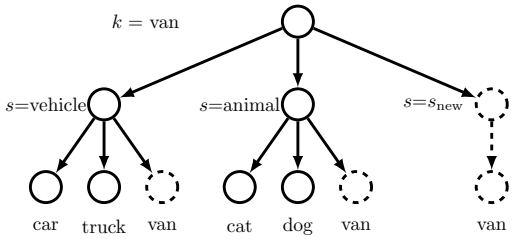
$$\begin{aligned} L(\mathbf{w},\boldsymbol{\beta},\boldsymbol{\theta}) &= -\log P(\mathcal{Y}|\mathcal{X},\mathbf{w},\boldsymbol{\beta}) - \log P(\mathbf{w}) - \log P(\boldsymbol{\beta}|\boldsymbol{\theta}) - \log P(\boldsymbol{\theta}) \\ &= -\log P(\mathcal{Y}|\mathcal{X},\mathbf{w},\boldsymbol{\beta}) + \frac{\lambda^2}{2}||\mathbf{w}||^2 + \frac{\lambda_2}{2}\sum_{k=1}^{K}||\boldsymbol{\beta}_k - \boldsymbol{\theta}_{\text{parent}(k)}||^2 + \frac{\lambda_1}{2}||\boldsymbol{\theta}||^2. \end{aligned} \tag{6.4}$$

Note that by setting $\boldsymbol{\theta}$ to zero, this loss function recovers our standard loss function. The choice of normal distributions in Eq. 6.2 leads to a nice property that maximization over $\boldsymbol{\theta}$, given $\boldsymbol{\beta}$ can be done in closed form. It just amounts to taking a (scaled) average of all $\boldsymbol{\beta}_k$'s which are children of $\boldsymbol{\theta}_s$. Let $C_s = \{k|\text{parent}(k) = s\}$, then

$$\boldsymbol{\theta}_s^* = \frac{1}{|C_s| + \lambda_1/\lambda_2}\sum_{k \in C_s} .\boldsymbol{\beta}_k \tag{6.5}$$

Therefore, the loss function in Eq. 6.4 can be optimized by iteratively performing the following two steps. In the first step, we maximize over $\mathbf{w}$ and $\boldsymbol{\beta}$ keeping $\boldsymbol{\theta}$ fixed. This can be done using standard stochastic gradient descent (SGD). Then, we maximize over $\boldsymbol{\theta}$ keeping $\boldsymbol{\beta}$ fixed. This can be done in closed form using Eq. 6.5. In practical terms, the second step is almost instantaneous and only needs to be performed after every T gradient descent steps, where T is around 10-100. Therefore, learning is

Require: $\mathcal{X}, \mathcal{Y}$, classes K, superclasses S, initial $\mathbf{z}$, L, M.
Initialize $\mathbf{w}, \beta$.
repeat
  // Optimize $\mathbf{w}, \beta$ with fixed $\mathbf{z}$.
  $\mathbf{w}, \boldsymbol{\beta} \leftarrow$ SGD $(\mathcal{X}, \mathcal{Y}, \mathbf{w}, \boldsymbol{\beta}, \mathbf{z})$ for L steps.
  // Optimize $\mathbf{z}, \beta$ with fixed $\mathbf{w}$.
  RandomPermute(K)
  for k in K **do**
    for s in $S \cup \{s_{\text{new}}\}$ **do**
      $z_k \leftarrow s$
      $\boldsymbol{\beta}^s \leftarrow$ SGD $(f_{\mathbf{w}}(\mathcal{X}), \mathcal{Y}, \boldsymbol{\beta}, \mathbf{z})$ for M steps.
    end for
    $s' \leftarrow$ ChooseBestSuperclass$(\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots)$
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}^{s'}$, $z_k \leftarrow s'$, $S \leftarrow S \cup \{s'\}$
  end for
until convergence



Algorithm 2: Procedure for learning the tree.

almost identical to standard gradient descent. It allows us to exploit the structure over labels at a very nominal cost in terms of computational time.

6.2.2 Learning the Tree Hierarchy

So far we have assumed that our model is given a fixed tree hierarchy. Now, we show how the tree structure can be learned during training. Let $\mathbf{z}$ be a K-length vector that specifies the tree structure, that is, $z_k = s$ indicates that class k is a child of super-class s. We place a non-parametric Chinese Restaurant Process (CRP) prior over $\mathbf{z}$. This prior $P(\mathbf{z})$ gives the model the flexibility to have any number of superclasses but provides a way to control the tendency to create such superclasses. Suppose we have a partition of a set of classes into K superclasses. Now a new class arrives. We need to decide which superclass should this incoming class be associated with, or if a new superclass should be created. Under the CRP prior, the probability of adding it to superclass s is $\frac{c^s}{K+\gamma}$ where c^s is the number of children of superclass s. The probability of creating a new superclass is $\frac{\gamma}{K+\gamma}$. In essence, it prefers to add a new node to an existing large superclass instead of spawning a new one. The strength of this preference is controlled by γ. Equipped with the CRP prior over $\mathbf{z}$, the conditional over $\mathcal{Y}$ takes the following form

$$P(\mathcal{Y}|\mathcal{X}) = \sum_{\mathbf{z}} \left(\int_{\mathbf{w},\boldsymbol{\beta},\boldsymbol{\theta}} P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) P(\mathbf{w}) P(\boldsymbol{\beta}|\boldsymbol{\theta}, \mathbf{z}) P(\boldsymbol{\theta}) d\mathbf{w} d\boldsymbol{\beta} d\boldsymbol{\theta} \right) P(\mathbf{z}). \tag{6.6}$$

MAP inference in this model leads to the following optimization problem

$$\max_{\mathbf{w},\boldsymbol{\beta},\boldsymbol{\theta},\mathbf{z}} \log P(\mathcal{Y}|\mathcal{X}, \mathbf{w}, \beta) + \log P(\mathbf{w}) + \log P(\boldsymbol{\beta}|\boldsymbol{\theta}, \mathbf{z}) + \log P(\boldsymbol{\theta}) + \log P(\mathbf{z}).$$

Maximization over $\mathbf{z}$ is problematic because the domain of $\mathbf{z}$ is a huge discrete set. Fortunately, this can be approximated using a simple and parallelizable search procedure. We first initialize the tree sensibly. This can be done by hand or by extracting a semantic tree from WordNet (Miller, 1995). Let the number of superclasses in the tree be S. We optimize over $\{\mathbf{w}, \boldsymbol{\beta}, \boldsymbol{\theta}\}$ for L steps using this tree. Then, a leaf node is picked uniformly at random from the tree and $S + 1$ tree proposals are generated

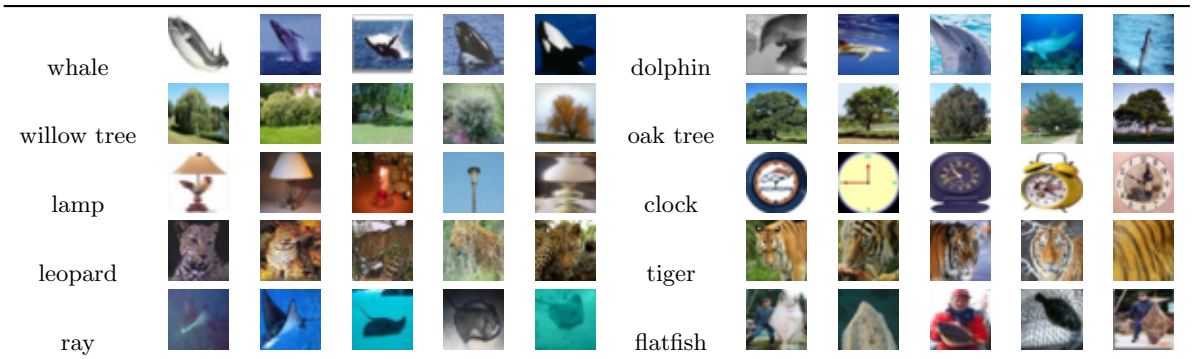


Figure 6.2: Examples from CIFAR-100. Five randomly chosen examples from 8 of the 100 classes are shown. Classes in each row belong to the same superclass.

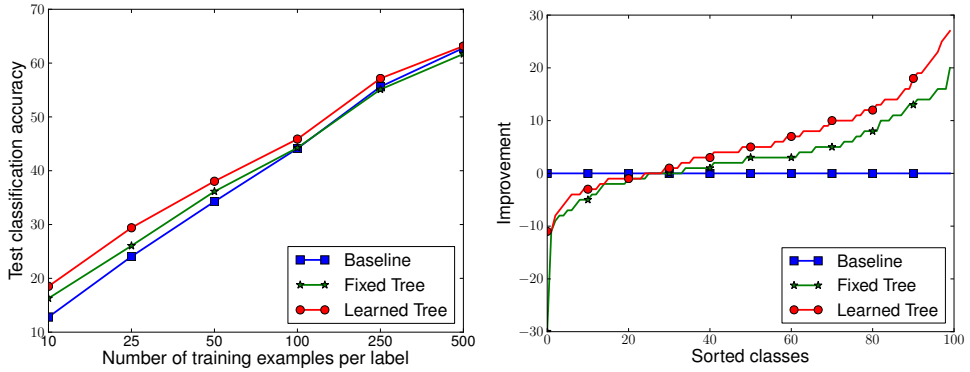
as follows. S proposals are generated by attaching this leaf node to each of the S superclasses. One additional proposal is generated by creating a new super-class and attaching the label to it. This process is shown in Algorithm 2. We then re-estimate $\{\boldsymbol{\beta}, \boldsymbol{\theta}\}$ for each of these $S+1$ trees for a few steps. Note that each of the $S+1$ optimization problems can be performed independently, in parallel. The best tree is then picked using a validation set. This process is repeated by picking another node and again trying all possible locations for it. After each node has been picked once and potentially repositioned, we take the resulting tree and go back to optimizing $\mathbf{w}, \boldsymbol{\beta}$ using this newly learned tree in place of the given tree. If the position of any class in the tree did not change during a full pass through all the classes, the hierarchy discovery is said to have converged. When training this model on CIFAR-100, this amounts to interrupting the stochastic gradient descent after every 10,000 steps to find a better tree. The amount of time spent in learning this tree is a small fraction of the total time (about 5%).

6.3 Experiments on CIFAR-100

The CIFAR-100 dataset (Krizhevsky, 2009) consists of 32×32 color images belonging to 100 classes. These classes are divided into 20 groups of 5 each. For example, the superclass **fish** contains *aquarium fish*, *flatfish*, *ray*, *shark* and *trout*; and superclass **flowers** contains *orchids*, *poppies*, *roses*, *sunflowers* and *tulips*. Some examples from this dataset are shown in Fig. 6.2. We chose this dataset because it has a large number of classes with a few examples in each, making it ideal for demonstrating the utility of transfer learning. There are only 600 examples of each class of which 500 are in the training set and 100 in the test set. We preprocessed the images by doing global contrast normalization followed by ZCA whitening.

6.3.1 Model Architecture and Training Details

We used a convolutional neural network with 3 convolutional hidden layers followed by 2 fully connected hidden layers. All hidden units used a rectified linear activation function. Each convolutional layer was followed by a max-pooling layer. Dropout (Hinton et al., 2012b) was applied to all the layers of the network with the probability of retaining a hidden unit being $p = (0.9, 0.75, 0.75, 0.5, 0.5, 0.5)$ for the different layers of the network (going from input to convolutional layers to fully connected layers). Max-norm regularization (Hinton et al., 2012b) was used for weights in both convolutional and fully connected layers. The initial tree was chosen based on the superclass structure given in the data set.



(a) (b)

Figure 6.3: Classification results on CIFAR-100. **Left**: Test set classification accuracy for different number of training examples per class. **Right**: Improvement over the baseline when trained on 10 examples per class. The learned tree models were initialized at the given tree.

We learned a tree using Algorithm 2 with $L = 10,000$ and $M = 100$. The initial and final learned trees are shown in Appendix B.1.2.

6.3.2 Experiments with Few Examples per Class

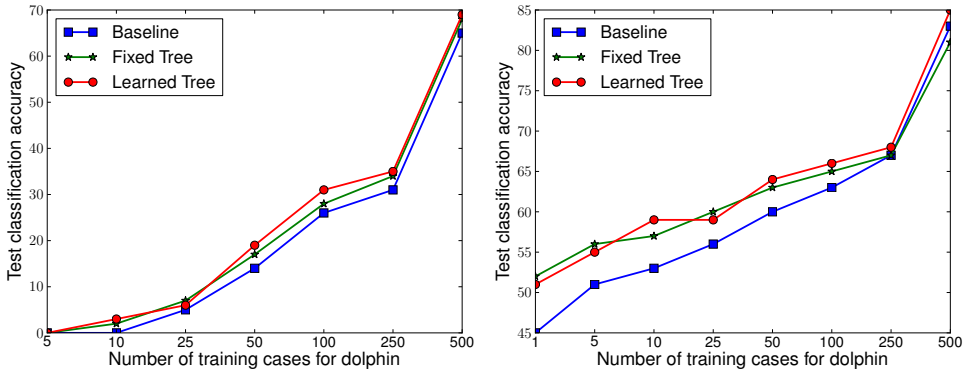
In our first set of experiments, we worked in a scenario where each class has very few examples. The aim was to assess whether the proposed model allows related classes to borrow information from each other. For a baseline, we used a standard convolutional neural network with the same architecture as our model. This is an extremely strong baseline and already achieved excellent results, outperforming all previously reported results on this dataset as shown in Table. 6.1. We created 5 subsets of the data by randomly choosing 10, 25, 50, 100 and 250 examples per class, and trained four models on each subset. The first was the baseline. The second was our model using the given tree structure (100 classes grouped into 20 superclasses) which was kept fixed during training. The third and fourth were our models with a learned tree structure. The third one was initialized with a random tree and the fourth with the given tree. The random tree was constructed by drawing a sample from the CRP prior and randomly assigning classes to leaf nodes.

The test performance of these models is compared in Fig. 6.3a. We observe that if the number of examples per class is small, the fixed tree model already provides significant improvement over the baseline. The improvement diminishes as the number of examples increases and eventually the performance falls below the baseline (61.7% vs 62.8%). However, the learned tree model does better. Even with 10 examples per class, it gets an accuracy of 18.52% compared to the baseline model's 12.81% or the fixed tree model's 16.29%. Thus the model can get almost a 50% relative improvement when few examples are available. As the number of examples increases, the relative improvement decreases. However, even for 500 examples per class, the learned tree model improves upon the baseline, achieving a classification accuracy of 63.15%. Note that initializing the model with a random tree decreases model performance, as shown in Table. 6.1.

Next, we analyzed the learned tree model to find the source of the improvements. We took the model trained on 10 examples per class and looked at the classification accuracy separately for each class. The aim was to find which classes gain or suffer the most. Fig. 6.3b shows the improvement obtained by

Method	Test Accuracy %
Conv Net + max pooling	56.62 ± 0.03
Conv Net + stochastic pooling (Zeiler and Fergus, 2013)	57.49
Conv Net + maxout (Goodfellow et al., 2013b)	61.43
Conv Net + max pooling + dropout (Baseline)	62.80 ± 0.08
Baseline + fixed tree	61.70 ± 0.06
Baseline + learned tree (Initialized randomly)	61.20 ± 0.35
Baseline + learned tree (Initialized from given tree)	**63.15 ± 0.15**

Table 6.1: Classification results on CIFAR-100. All models were trained on the full training set.



(a) (b)

Figure 6.4: Results on CIFAR-100 with few examples for the *dolphin* class. **Left**: Test set classification accuracy for different number of examples. **Right**: Accuracy when classifying a *dolphin* as *whale* or *shark* is also considered correct.

different classes over the baseline, where the classes are sorted by the value of the improvement over the baseline. Observe that about 70 classes benefit in different degrees from learning a hierarchy for parameter sharing, whereas about 30 classes perform worse as a result of transfer. For the learned tree model, the classes which improve most are *willow tree* (+26%) and *orchid* (+25%). The classes which lose most from the transfer are *ray* (-10%) and *lamp* (-10%).

We hypothesize that the reason why certain classes gain a lot is that they are very similar to other classes within their superclass and thus stand to gain a lot by transferring knowledge. For example, the superclass for *willow tree* contains other trees, such as *maple tree* and *oak tree*. However, *ray* belongs to superclass *fish* which contains more typical examples of fish that are very dissimilar in appearance. With the fixed tree, such transfer hurts performance (*ray* did worse by -29%). However, when the tree was learned, this class split away from the *fish* superclass to join a new superclass and did not suffer as much. Similarly, *lamp* was under *household electrical devices* along with *keyboard* and *clock*. Putting different kinds of electrical devices under one superclass makes semantic sense but does not help for visual recognition tasks. This highlights a key limitation of hierarchies based on semantic knowledge and advocates the need to learn the hierarchy so that it becomes relevant to the task at hand. The full learned tree can be seen in Appendix B.1.2 Table. B.2.

Classes	baby, female, people, portrait	plant life, river, water	clouds, sea, sky, transport, water	animals, dog, food	clouds, sky, structures
Images					
Tags	claudia	⟨ no text ⟩	barco, pesca, boattosail, navegação	watermelon, dog, hilarious, chihuahua	colors, cores, centro, commercial, building

Figure 6.5: Some examples from the MIR-Flickr dataset. Each instance in the dataset is an image along with textual tags. Each image has multiple classes.

6.3.3 Experiments with Few Examples for One Class

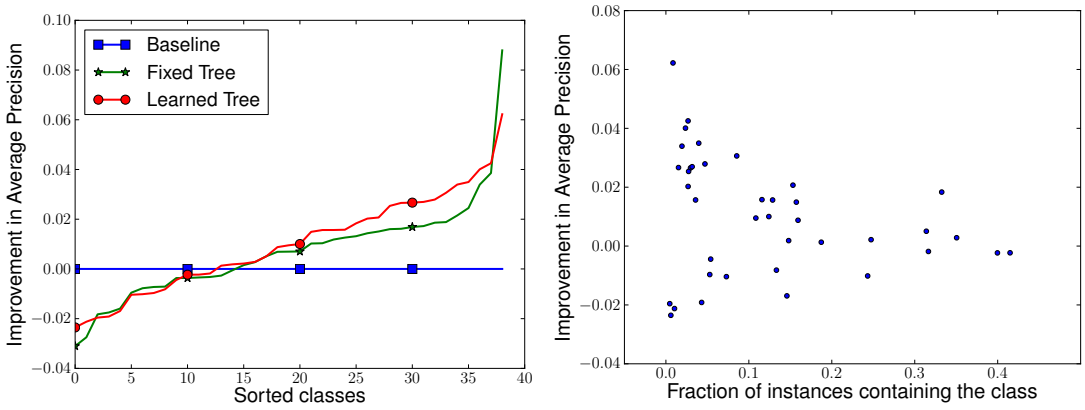
In this set of experiments, we worked in a scenario where there are lots of examples for different classes, but only few examples of one particular class. The aim was to see whether the model transfers information from other classes that it has learned to this "rare" class. We constructed training sets by randomly drawing either 5, 10, 25, 50, 100, 250 or 500 examples from the *dolphin* class and all 500 training examples for the other 99 classes. We trained the baseline, fixed tree and learned tree models with each of these datasets. The objective was kept the same as before and no special attention was paid to the *dolphin* class. Fig. 6.4a shows the test accuracy for correctly predicting the *dolphin* class. We see that transfer learning helped tremendously. For example, with 10 cases, the baseline gets 0% accuracy whereas the transfer learning model can get around 3%. Even for 250 cases, the learned tree model gives significant improvements (31% to 34%). We repeated this experiment for classes other than *dolphin* as well and found similar improvements. The details for those can be found in Appendix B.3.

In addition to performing well on the class with few examples, we would also want any errors to be sensible. To check if this was indeed the case, we evaluated the performance of the above models treating the classification of *dolphin* as *shark* or *whale* to also be correct, since we believe these to be reasonable mistakes. Fig. 6.4b shows the classification accuracy under this assumption for different models. Observe that the transfer learning methods provide significant improvements over the baseline. Even when we have just 1 example for *dolphin*, the accuracy jumps from 45% to 52%.

6.4 Experiments on MIR Flickr

The Multimedia Information Retrieval Flickr Data set (Huiskes and Lew, 2008) consists of 1 million images collected from the social photography website Flickr along with their user assigned tags. Among the 1 million images, 25,000 have been annotated using 38 labels. These labels include object categories such as, *bird*, *tree*, *people*, as well as scene categories, such as *indoor*, *sky* and *night*. Each image has multiple labels. Some examples are shown in Fig. 6.5.

This dataset is different from CIFAR-100 in many ways. In the CIFAR-100 dataset, our model was trained using image pixels as input and each image belonged to only one class. MIR-Flickr is a multimodal dataset for which we used standard computer vision image features and word counts as inputs. The CIFAR-100 models used a multi-layer convolutional network, whereas for this dataset we use a fully connected neural network initialized by unrolling a Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009a). Moreover, this dataset offers a more natural class distribution where some classes occur more often than others. For example, *sky* occurs in over 30% of the instances, whereas *baby* occurs



(a) Class-wise improvement (b) Improvement vs. number of examples

Figure 6.6: Results on MIR Flickr. **Left**: Improvement in Average Precision over the baseline for different methods. **Right**: Improvement of the learned tree model over the baseline for different classes along with the fraction of test cases which contain that class. Each dot corresponds to a class. Classes with few examples (towards the left of plot) usually get significant improvements.

in fewer than 0.4%. We also used 975,000 unlabeled images for unsupervised training of the DBM. We use the publicly available features and train-test splits from Srivastava and Salakhutdinov (2012).

6.4.1 Model Architecture and Training Details

In order to make our results directly comparable to Srivastava and Salakhutdinov (2012), we used the same network architecture as described therein. The authors of the dataset (Huiskes et al., 2010) provided a high-level categorization of the classes which we use to create an initial tree. This tree structure and the one learned by our model are shown in Appendix B.2.2. We used Algorithm 2 with $L = 500$ and $M = 100$.

6.4.2 Classification Results

For a baseline, we used a Multimodal DBM model after finetuning it discriminatively with dropout. This model already achieves state-of-the-art results, making it a very strong baseline. The results of the experiment are summarized in Table. 6.2. The baseline achieved a MAP of 0.641, whereas our model with a fixed tree improved this to 0.648. Learning the tree structure further pushed this up to 0.651. For this dataset, the learned tree was not significantly different from the given tree. Therefore, we expected the improvement from learning the tree to be marginal. However, the improvement over the baseline was significant, showing that transferring information between related classes helped.

Looking closely at the source of gains, we found that similar to CIFAR-100, some classes gain and others lose as shown in Fig. 6.6a. It is encouraging to note that classes which occur rarely in the dataset improve the most. This can be seen in Fig. 6.6b which plots the improvements of the learned tree model over the baseline against the fraction of test instances that contain that class. For example, the average precision for *baby* which occurs in only 0.4% of the test cases improves from 0.173 (baseline) to 0.205 (learned tree). This class borrows from *people* and *portrait* both of which occur very frequently. The performance on *sky* which occurs in 31% of the test cases stays the same.

Method	**MAP**
Logistic regression on Multimodal DBM (Srivastava and Salakhutdinov, 2012)	0.609
Multiple Kernel Learning SVMs (Guillaumin et al., 2010)	0.623
TagProp (Verbeek et al., 2010)	0.640
Multimodal DBM + finetuning + dropout (Baseline)	0.641 ± 0.004
Baseline + fixed tree	0.648 ± 0.004
Baseline + learned tree (initialized from given tree)	**0.651 ± 0.005**

Table 6.2: Mean Average Precision obtained by different models on the MIR-Flickr data set.

6.5 Conclusion

We proposed a model that augments standard neural networks with tree-based priors over the classification parameters. These priors follow the hierarchical structure over classes and enable the model to transfer knowledge from related classes. We also proposed a way of learning the hierarchical structure. Experiments show that the model achieves excellent results on two challenging datasets.

Chapter 7

Conclusions and Future Work

We conclude by summarizing our contributions, pointing out the strengths and weaknesses of the proposed models, exploring avenues for future research and highlighting the relevant research developments that have taken place since these contributions were published.

7.1 Deep Boltzmann Machines

The first two chapters of this thesis focus on applying and extending Deep Boltzmann Machines. From the results obtained on modeling multimodal data, we can conclude that DBMs can model the joint distribution over very different modalities. We can also conclude that Contrastive Divergence, which is only an approximation to the correct maximum likelihood training objective, is sufficient to produce decent generative performance (as evidenced by retrieval results on unimodal data) even in this relatively complex domain. However, multiple CD steps are needed during training to make this happen. An important take-away message is that multimodal training can serve as a regularizer, in the sense that even when unimodal data is available at test time, models that were trained to explain multiple data modalities simultaneously were able to perform better that models that were just trained to explain unimodal data.

This work also brought to focus several difficulties in training these models which constitute the major drawbacks. Learning is quite slow since in relies on mixing of Markov chains to draw good samples from the model distribution. As the model learns and becomes sharper, this mixing becomes harder. These problems can be ameliorated to some extent using fast weights (Tieleman and Hinton, 2009) but are still an impediment to making the training work reliably on new problems and datasets. Several new approaches to unsupervised learning are being studied, including variational autoencoders (VAEs)(Kingma and Welling, 2013; Rezende et al., 2014), generative adversarial networks (GANs)(Goodfellow et al., 2014; Denton et al., 2015; Radford et al., 2015), models that generate data sequentially (Gregor et al., 2015), and even ones that generate data one dimension at a time (Oord et al., 2016b; Kalchbrenner et al., 2016; Oord et al., 2016a). In contrast to MCMC-based learning used for training DBMs, all these approaches use objective functions that allow for backpropagation-based learning. This is a major advantage when training very deep networks as backpropagation sends information throughout the network more effectively compared to the methods used in this work for training DBMs where information flow is more diffuse. One way to see this effect is to note that if the highest layer of a DBM refuses to partici-

pate in the modeling process by becoming "dead" (i.e., assuming a constant state as a result of having large magnitude biases), then the rest of the model can still assign a decent log-prob to the data, with this highest layer's contribution being just a bias to the second-highest layer. In this case, an N-layer DBM essentially reduces to an $N-1$ layer one. Therefore, the loss of an entire layer of weights is not catastrophic for the model. On the other hand, the complete loss of a layer of weights in a feed-forward model (such as an autoencoder, VAE or GAN) would be extremely detrimental since it would disconnect the latent variables from the data. Therefore, feed-forward models *must* make each additional layer do something sensible, whereas this is not a requirement for DBMs, only a preference. This makes it hard to train very deep Boltzmann Machines [1]. Another major drawback of DBMs is that the true objective function (log-likelihood of the data) cannot be computed efficiently which makes it hard to do day-to-day development. Relying on a pseudo-objective such as one-step reconstruction during model development and hyper-parameter tuning can often lead to sub-optimal generative performance.

In spite of the drawbacks, Deep Boltzmann Machines have a number of properties that make them interesting to study and improve. The stochastic inference procedure in DBMs can be seen as a search process that allows the model to explore different ways of explaining the data potentially making big jumps across modes. The ability of the model to repeatedly query its memory and test out different explanations seems like a powerful tool to have. Since only local information is needed to do learning (without the need for a global backprop), these models can easily incorporate non-differentiable computational units. Most work on DBMs until now has focused on rather simple kinds of interactions between units and very generic connection graphs. These models have the advantage of being general-purpose but also the disadvantage of not imposing enough structure a-priori that could have helped solve the problem. In future work, structured hidden units with well-designed interactions can be used to create models which would impose more informative inductive biases. One such attempt is the work on directional units described in this thesis which tries to make it easy to discover coincidences based on agreement in pose. Incorporating more domain-specific structures such as local connectivity for images and different measures of smoothness for video sequences are also likely to be useful. DBMs can be used to search over more interesting hypothesis spaces, such as, routing the information about parts to their respective wholes, thereby creating a parse tree for an image.

7.2 Learning Sequence Representations

The main conclusion from the work on learning sequence representations using LSTM encoder-decoder networks is that training the model to perform both future prediction and autoencoding simultaneously helps it learn better representations than using only one of those objectives. We showed that these models are capable of generating complex temporal patterns over extended time periods. While the deterministic version of this model can do a reasonable job of predicting the future for simple sequences such as moving MNIST digits, it fails to produce good predictions on natural videos where stochasticity is important to model multiple modes in the output distribution. Recent work in generating sequences using different ways of injecting stochasticity has led to much better video generation results (Kalchbrenner et al., 2016; Lotter et al., 2016; Walker et al., 2016; Xue et al., 2016). While our work used mean-squared error as the loss function, subsequent work has explored using learned similarity functions (Mathieu et al., 2015).

[1] However, it should be noted that Boltzmann Machines are deep in a different sense – doing inference in them involves running iterative updates. This property has potentially many benefits and mimicking this in a feed-forward model would require a lot of unrolling, implying a large effective depth per DBM layer.

Video modeling has also been done by conditioning on physical interactions (Finn et al., 2016; Chiappa et al., 2017) and using dynamically changing weights (Brabandere et al., 2016).

A lot of future work remains to be done in doing unsupervised learning from videos. Models based on GANs have already shown that it is possible to generate photorealistic images (Zhang et al., 2016). The GAN framework presents a way to solve one of the major issues with generating images (and videos) which is coming up with a good loss function that can operate on pixels. The blurry prediction problem created by using MSE loss (as observed in our work) can be avoided with this approach. In future, it would be natural to start using adversarial learning to train discriminators and use them to compare generated and true data. With the loss function issue under control, it would be easier to focus on developing ways of injecting stochasticity in the models so that multi-modal conditional distributions over the future can be well represented. Most current models are relatively unconstrained in terms of what kind of distributed representation they learn (as long as they can generate good-looking data). It is not readily apparent how to use these representations to build a general purpose vision system that can solve new problems of interest quickly. Therefore, an important direction of research is finding more interpretable representations that enable the full 3D reconstruction of an environment given a video. With autonomous driving becoming a major industry, such models are likely to receive a lot of focus.

7.3 Directional units

The work on directional units explores a promising framework for representation learning. We derive a Boltzmann Machine formalism for describing networks of N-dimensional directional units, propose a way of connecting such units to non-directional units and to directional units of different dimensionalities. We also describe how to use these units in feed-forward networks. Moreover, we propose a temporal coherence based learning algorithm that explicitly uses the direction of a directional unit to model dynamic properties of a feature while using the length (or amplitude) of the feature to model its presence. The preliminary qualitative results demonstrate that the models are able to learn low-level representations which encode small equivariances over a space of features.

This exploration also discovered some key issues with these models. One of the properties of the proposed Boltzmann Machine model is that the directional units do not have an "off switch". While their *expected* magnitude can be zero, any sample drawn from the model will have all directional units present with a magnitude of one. This leads to an interesting property that these models are more entropy-driven than free-energy driven, as discussed in more detail in the chapter on directional units. One way to augment these models to avoid this problem is by having each directional unit be accompanied by a binary gating unit which can act like the off switch. Turning this gate off effectively removes the directional unit from the model. Preliminary results with these models are encouraging and we plan to develop these further in future.

Another concern with these models is that a directional unit can model a space of features only if that space is topologically consistent with the surface of a sphere. Therefore, networks of directional units must find very clever ways to model certain transformations, such as those which involve scaling. Augmenting each directional unit with a rectified linear unit could be one way to add a positive unbounded scalar which could enlarge the space of features being represented in a useful way. In general, we believe that a lot of interesting work remains to be done in designing more structured hidden units that can help induce desirable properties, for example, sensitivity to coincidences.

One promising direction for future work is using directional units to do dynamic routing. The main hypothesis here is that if directional units can find agreement in pose, then they should be able to figure out which "whole" is the parent responsible for each "part". The pose of this part should then be routed only to the corresponding parent. This would impose the inductive bias that the same part cannot belong to two different parents. In other words, at a time, there can only be one interpretation of the data. Current state-of-the-art vision models do not explicitly impose this belief even though this is significantly evident in human vision. Routing data to relevant processing units effectively removes data that has already been explained from influencing all the other decision making going on in the network. This could potentially lead to more interpretable high-level and intermediate representations.

7.4 Transfer Learning

Finally, the work on transfer learning using tree-based priors describes a way of transferring knowledge gained in the process of learning a deep neural net classifier over a given number of categories, to new categories which have few labelled examples. We show how the structure over classes can be learned using a standard prior over tree structures combined with a discriminative likelihood function.

Our work explored a relatively simple transfer learning problem because it involved transferring knowledge across known discrete output categories. One of the most important directions for future work in transfer learning is in the context of Reinforcement Learning (RL) where we would like to create an agent capable of transferring knowledge (such as intermediate goal states) across different tasks. In this case, the aim is to discover sub-tasks and their relationships, when these sub-tasks are not known a-priori. This would enable the agent to be intelligent in a general sense, where it can learn to do new tasks with very few demonstrations or trials by breaking down new tasks into familiar sub-tasks.

7.5 Overall Conclusions

To summarize, this thesis describes some methods for doing unsupervised and transfer learning using deep networks. We explore a wide variety of models – undirected generative models such as Boltzmann Machines, feed-forward sequence learning models using recurrent nets, representation learning models that use directional units and a simple example of transfer learning in the context of deep neural net classifiers. While these contributions are diverse and disparate, they share the common design principle of using learned distributed representations. This representation learning paradigm is likely to be a powerful tool along the path to understanding and implementing intelligence.

Appendices

Appendix A

Directional Units

This supplementary material includes derivations of properties of von Mises-Fisher distributions which have been borrowed from the excellent work of Sra (2007).

A.1 Polar Coordinates

A vector $\mathbf{x} \in \mathbb{R}^n$ represented in Cartesian coordinates as $(x_1, x_2, \ldots, x_n)$ can be written in polar coordinates as $(r, \theta_1, \theta_2, \ldots, \theta_{n-1})$ where

$$\begin{aligned} x_1 &= r\cos\theta_1 \\ x_2 &= r\sin\theta_1\cos\theta_2 \\ x_3 &= r\sin\theta_1\sin\theta_2\cos\theta_3 \\ &\ldots \\ x_{n-1} &= r\sin\theta_1\sin\theta_2\ldots\cos\theta_{n-1} \\ x_n &= r\sin\theta_1\sin\theta_2\ldots\sin\theta_{n-1} \end{aligned}$$

where $r \in [0, \infty)$, $\theta_i \in [0, \pi) \forall i \leq n-2$, $\theta_{n-1} \in [0, 2\pi)$. A differential element of space in Cartesian coordinates $d\mathbf{x} = |\det \mathbf{J}| dr d\boldsymbol{\theta}$ where

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_1}{\partial r} & \frac{\partial x_1}{\partial \theta_1} & \ldots & \frac{\partial x_1}{\partial \theta_{n-1}} \\ \frac{\partial x_2}{\partial r} & \frac{\partial x_2}{\partial \theta_1} & \ldots & \frac{\partial x_2}{\partial \theta_{n-1}} \\ \frac{\partial x_3}{\partial r} & \vdots & \vdots & \frac{\partial x_3}{\partial \theta_{n-1}} \\ \vdots & \ldots & \ldots & \vdots \\ \frac{\partial x_n}{\partial r} & \frac{\partial x_n}{\partial \theta_1} & \ldots & \frac{\partial x_n}{\partial \theta_{n-1}} \end{bmatrix}$$

It can be shown that

$$|\det \mathbf{J}| = r^{n-1} \prod_{j=2}^{n} \sin^{n-j} \theta_{j-1} \tag{A.1}$$

A.2 Normalization Constant and Expected Value under the von Mises-Fisher Distribution

Given a vector $\mathbf{z} \in \mathbb{R}^n$, the von Mises-Fisher distribution over unit vectors $\mathbf{x}$ is given by

$$p(\mathbf{x}) = \frac{1}{C(\mathbf{z})} e^{\mathbf{z}^\top \mathbf{x}}.$$

In order to derive an expression for the normalization constant, we would like to integrate

$$\int_{||\mathbf{x}||=1} e^{\mathbf{z}^\top \mathbf{x}} d\mathbf{x}.$$

We can write the vector $\mathbf{z}$ as $\kappa\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ has unit norm. It turns out that the normalization constant depends only on κ and can be written as

$$C_n(\kappa) = \int_{||\mathbf{x}||=1} e^{\kappa \boldsymbol{\mu}^\top \mathbf{x}} d\mathbf{x}$$

Now we can make a coordinate transform $\mathbf{y} = \mathbf{T}\mathbf{x}$ where $\mathbf{T}$ is an orthonormal matrix such that the first row of $\mathbf{T}$ is $\boldsymbol{\mu}^\top$ (rotate the coordinate system so that the first axis aligns with $\boldsymbol{\mu}$).

$$C_n(\kappa) = \int_{||\mathbf{T}^{-1}\mathbf{y}||=1} e^{\kappa \boldsymbol{\mu}^\top \mathbf{T}^{-1}\mathbf{y}} |\det\, \mathbf{T}| d\mathbf{y}$$

Since $\mathbf{T}$ is orthonormal, $\det\, \mathbf{T} = 1$. Also $\mathbf{T}^{-1} = \mathbf{T}^\top$. Therefore, $\boldsymbol{\mu}^\top \mathbf{T}^{-1} = \boldsymbol{\mu}^\top \mathbf{T}^\top = (1, 0, \ldots, 0)$ because the first column of $\mathbf{T}^\top$ is $\boldsymbol{\mu}$ and the rest are orthogonal to $\boldsymbol{\mu}$. The set of vectors $\mathbf{T}^{-1}\mathbf{y}$ is the same as the set of all $\mathbf{y}$'s since $\mathbf{y}$ goes over the entire surface of a unit sphere and $\mathbf{T}$ is orthonormal. Therefore, we get

$$C_n(\kappa) = \int_{||\mathbf{y}||=1} e^{\kappa (1,0,\ldots,0)^\top \mathbf{y}} d\mathbf{y}.$$

Writing $\mathbf{y}$ in polar coordinates with $r = 1$ and using Eq. A.1,

$$C_n(\kappa) = \int_0^\pi e^{\kappa \cos\theta_1} \sin^{n-2}\theta_1 d\theta_1 \prod_{j=3}^{n-1} \int_0^\pi \sin^{n-j}\theta_{j-1} d\theta_{j-1} \int_0^{2\pi} d\theta_{n-1}.$$

Now we use the following identities

$$\int_o^\pi e^{\kappa\cos\theta} \sin^{n-2}\theta d\theta = \Gamma\left(\frac{n-1}{2}\right)\Gamma\left(\frac{1}{2}\right)\left(\frac{2}{\kappa}\right)^{n/2-1} I_{n/2-1}(\kappa)$$
$$\int_o^\pi \sin^n\theta d\theta = \frac{\sqrt{\pi}\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}+1\right)}.$$

This gives,

$$C_n(\kappa) = \frac{(2\pi)^{n/2}}{\kappa^{n/2-1}} I_{n/2-1}(\kappa).$$

Expectation of $\mathbf{h}$ can be computed as the following integral

$$\begin{aligned}
E[\mathbf{h}|\kappa,\boldsymbol{\mu}] &= \int_{||\mathbf{h}||=1} \mathbf{h}P(\mathbf{h}|\kappa,\boldsymbol{\mu})d\mathbf{h} \\
&= \frac{1}{C_n(\kappa)} \int_0^\pi \cos\theta_1 e^{\kappa\cos\theta_1} \sin^{n-2}\theta_1 d\theta_1 \prod_{j=3}^{n-1} \int_0^\pi \sin^{n-j}\theta_{j-1} d\theta_{j-1} \int_0^{2\pi} d\theta_{n-1} \\
&= \frac{\int_0^\pi \cos\theta_1 e^{\kappa\cos\theta_1} \sin^{n-2}\theta_1 d\theta_1}{\int_0^\pi e^{\kappa\cos\theta_1} \sin^{n-2}\theta_1 d\theta_1}
\end{aligned}$$

Let

$$\begin{aligned}
L_n(\kappa) &= \int_0^\pi e^{\kappa\cos\theta} \sin^{n-2}\theta d\theta \\
M_n(\kappa) &= \int_0^\pi \cos\theta e^{\kappa\cos\theta} \sin^{n-2}\theta d\theta
\end{aligned}$$

Then integrating by parts,

$$\begin{aligned}
M_n(\kappa) &= \int_0^\pi \cos\theta e^{\kappa\cos\theta} \sin^{n-2}\theta d\theta \\
&= \sin\theta e^{\kappa\cos\theta} \sin^{n-2}\theta \Big|_0^\pi - \int_0^\pi \sin\theta \left(e^{\kappa\cos\theta} \left((n-2)\sin^{n-3}\theta\cos\theta - \kappa\sin^{n-1}\theta \right) \right) d\theta \\
&= 0 - (n-2)\int_0^\pi \cos\theta e^{\kappa\cos\theta} \sin^{n-2}\theta d\theta + \kappa \int_0^\pi e^{\kappa\cos\theta} \sin^n\theta d\theta \\
\implies M_n(\kappa) &= \kappa L_{n+2}(\kappa) - (n-2)M_n(\kappa) \\
\implies M_n(\kappa) &= \frac{\kappa}{n-1} L_{n+2}(\kappa)
\end{aligned}$$

Therefore,

$$E[\mathbf{h}|\kappa,\boldsymbol{\mu}] = \frac{M_n(\kappa)}{L_n(\kappa)} = \frac{\kappa}{n-1}\frac{L_{n+2}(\kappa)}{L_n(\kappa)}$$

Again using the identity

$$L_n(\kappa) = \Gamma\left(\frac{n-1}{2}\right)\Gamma\left(\frac{1}{2}\right)\left(\frac{2}{\kappa}\right)^{n/2-1} I_{n/2-1}(\kappa)$$

and $\Gamma(n+1) = n\Gamma(n)$, we get

$$E[\mathbf{h}|\kappa,\boldsymbol{\mu}] = \frac{\kappa}{n-1}\frac{L_{n+2}(\kappa)}{L_n(\kappa)} = \frac{\kappa}{n-1}\frac{\Gamma\left(\frac{n+1}{2}\right)\Gamma\left(\frac{1}{2}\right)\left(\frac{2}{\kappa}\right)^{n/2} I_{n/2}(\kappa)}{\Gamma\left(\frac{n-1}{2}\right)\Gamma\left(\frac{1}{2}\right)\left(\frac{2}{\kappa}\right)^{n/2-1} I_{n/2-1}(\kappa)} = \frac{I_{n/2}(\kappa)}{I_{n/2-1}(\kappa)}$$

A.3 Computing ratio of modified Bessel Functions

Computing the expected value of a hidden unit requires computing the ratio of successive modified Bessel functions : $S_N(\kappa) = I_{N/2}(\kappa)/I_{N/2-1}(\kappa)$. Numerical libraries usually provide methods for computing modified Bessel functions, for example `scipy.special.iv` in Python. Even CUDA provides routines for computing I_1 and I_0 on GPUs (`cyl_bessel_i1f` and `cyl_bessel_i0f`). However, the values of $I_s(\kappa)$

tend to blow up very quickly with κ, for example, $I_0(50) = 2.9 \times 10^{20}, I_0(100) = 1.07 \times 10^{42}$. This can sometimes test the limits of single precision computation. Fortunately, we only need to compute the *ratio* of modified Bessel functions which can be done without computing the functions themselves. There is a continued fraction expression for the ratio.

$$\frac{I_{s+1}(\kappa)}{I_s(\kappa)} = \cfrac{1}{\frac{2(s+1)}{\kappa} + \cfrac{1}{\frac{2(s+2)}{\kappa} + \cfrac{1}{\frac{2(s+3)}{\kappa} + \frac{1}{\cdots}}}}$$

This can be unrolled to some finite length to approximate the ratio. We unroll for 1000 steps in our implementation. This approximation is very easy to implement on a GPU.

A.4 Sampling from von Mises-Fisher Distributions

For direction units, the conditional distribution is von Mises-Fisher. Wood (1994) describe an algorithm for sampling from the von Mises-Fisher distribution using rejection sampling. Suppose we want to draw a sample from the VMF-distribution on a sphere in N-dimensions which has its mode $\boldsymbol{\mu}$ and concentration parameter κ. We will first see how we can draw a sample from a distribution with mode $(1, 0, \ldots, 0)^\top$ (We will then rotate the sample so that the distribution has mode $\boldsymbol{\mu}$). Ulrich (1984) showed that

$$X \sim \frac{1}{C_N(\kappa)} e^{\kappa (1,0,\ldots,0)^\top \mathbf{x}}$$

if and only if $X^\top = (w, \sqrt{1-w^2}V^\top)$, where V is drawn from the uniform distribution on the surface of a sphere in $N-1$-dimensions and w is a scalar in the range $[-1, 1]$ drawn from the distribution

$$f(w) = \frac{1}{C} \left(1 - w^2\right)^{(n-3)/2} e^{\kappa w}$$

where $C = \sqrt{\pi} \left(\frac{\kappa}{2}\right)^{n/2-1} I_{n/2-1}(\kappa) \Gamma\left(\frac{n-1}{2}\right)$. Samples can be drawn from this distribution using the envelope function

$$e(x, b) = \frac{1}{D} \left(1 - w^2\right)^{(n-3)/2} \left((1+b) - (1-b)x\right)^{-(m-1)}$$

where $D = \Gamma\left(\frac{n-1}{2}\right)^2 b^{-(n-1)/2} (2\Gamma(n-1))^{-1}$ and b is a parameter whose value can be optimized. This envelope distribution is easy to sample from, by transforming samples from the Beta distribution Beta $((n-1/2), (n-1)/2)$. The optimal value of b is found by maximizing the acceptance ratio for the given values of κ and n. The complete sampling algorithm is described in Algorithm 3. See Wood (1994) for more details[1]. The acceptance ratio for this algorithm is very high. The envelope is a close fit to the actual distribution. The acceptance ratio increases with n and decreases with κ. The worst situation is $n = 2$ and $\kappa \to \infty$. It can be shown that even in this worst case, the acceptance ratio cannot be lower than 66%. Therefore, a few trials are sufficient to get a sample.

[1] The algorithm in Ulrich (1984) has a bug, which has been corrected in Wood (1994)

Algorithm 3 Sampling from von Mises-Fisher distribution (Wood, 1994)

Require: κ, $\boldsymbol{\mu}$, n
  $\alpha \leftarrow (n-1)/2$
  $b \leftarrow (-\kappa + \sqrt{\kappa^2 + \alpha^2})/\alpha$
  $x \leftarrow (1-b)/(1+b)$
  $c \leftarrow \kappa x + 2\alpha \log(1-x^2)$
  repeat
    $\theta \sim U(-\pi/2, \pi/2)$, $r \sim U(0,1)$, $u \sim U(0,1)$
    $z \leftarrow \frac{1}{2} + \frac{\sin\theta}{2}(1 - r^{1/(\alpha-0.5)})$ ▷ Sample from Beta (α, α).
    $w \leftarrow (1-(1+b)z)/(1-(1-b)z)$ ▷ Transform the sample.
  until $\kappa w + 2\alpha \log(1-xw) - c \geq \log(u)$
  $V \sim \mathcal{N}(0, I_{n-1})$
  $V \leftarrow V - \langle \boldsymbol{\mu}, V \rangle \boldsymbol{\mu}$
  $X \leftarrow w\boldsymbol{\mu} + \sqrt{1-w^2}\frac{V}{||V||}$
  return X

Appendix B

Transfer Learning With Tree-based Priors

B.1 Experimental Details for CIFAR-100

In this section we describe the experimental setup for CIFAR-100, including details of the architecture and the structures of the initial and learned trees.

B.1.1 Architecture and training details

We choose the architecture that worked best for the baseline model using a validation set. All the experiments reported in the paper for this dataset use this network architecture. It consists of three convolutional layers followed by 2 fully connected layers. Each convolutional layer is followed by a max-pooling layer. The convolutional layers have 96, 128 and 256 filters respectively. Each convolutional layer has a 5×5 receptive field applied with a stride of 1 pixel. Each max pooling layer pools 3×3 regions at strides of 2 pixels. The two fully connected hidden layers having 2048 units each. All units use the rectified linear activation function. Dropout was applied to all the layers of the network with the probability of retaining the unit being $p = (0.9, 0.75, 0.75, 0.5, 0.5, 0.5)$ for the different layers of the network (going from input to convolutional layers to fully connected layers). In addition, the max-norm constraint with $c = 4$ was used for all the weights.

Training the model on very small subsets of the data poses multiple problems. One of them is holding out a validation set. When training with 10 examples of class, we want to mimic a situation when we just have these 10 examples, and not a separate large validation set which has many more examples of this class. Therefore, we held out 30% of training data as validation, even when only 10 training examples are present. In order to remove noise, we repeated the experiment multiple times, selecting 7 random training cases and 3 random validation cases. Once we used the validation set to determine the hyperparameters, we combined it with the training set and trained the model down to the training set cross entropy that was obtained when the validation set was kept separate. This allows us to use the full training set which is crucial in very small dataset regimes.

B.1.2 Initial and Learned Trees

The authors of the CIFAR-100 data set (Krizhevsky, 2009) divide the set of 100 classes into 20 superclasses. These classes are shown in Table. B.1. Using this tree as an initialization, we used our model to learn the tree. The learned tree is shown in Table. B.2. This tree was learned with 50 examples per class. The superclasses do not have names since they were learned. We can see that the tree makes some sensible moves. For example, it creates a superclass for *shark*, *dolphin* and *whale*. It creates a superclass for worm-like creatures such as *caterpillar*, *worm* and *snake*. However this class also includes *snail* which is harder to explain. It includes *television* along with other furniture items, removing it from the house hold electrical devices superclass. However, some choices do not seem good. For example, *clock* is put together with *bicycle* and *motorcycle*. The only similarity between them is presence of circular objects.

Superclass	**Classes**
aquatic mammals	dolphin, whale, seal, otter, beaver
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchid, poppy, rose, sunflower, tulip
food containers	bottle, bowl, can, cup, plate
fruit and vegetables	apple, mushroom, orange, pear, sweet pepper
household electrical devices	clock, keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium sized mammals	fox, porcupine, possum, raccoon, skunk
non insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple tree, oak tree, palm tree, pine tree, willow tree
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn mower, rocket, streetcar, tank, tractor

Table B.1: Fixed tree hierarchy for the CIFAR-100 dataset.

B.2 Experimental Details for MIR Flickr

In this section we describe the experimental setup for MIR Flickr, including details of the architecture and the structures of the initial and learned trees.

B.2.1 Architecture and training details

The model was initialized by unrolling a multimodal DBM. The DBM consisted of two pathways. The image pathway had 3857 input units, followed by 2 hidden layers of 1024 hidden units. The text pathway had 2000 input units (corresponding the top 2000 most frequent tags in the dataset), followed by 2 hidden layers of 1024 hidden units each. These two pathways were combined at the joint layer which had 2048 hidden units. All hidden units were logistic. The DBM was trained using publicly available features and code. We additionally finetuned the entire model using dropout and used that as our baseline. During

Superclass	Classes
superclass 1	dolphin, whale, shark
superclass 2	aquarium fish, trout, flatfish
superclass 3	orchid, poppy, rose, sunflower, tulip, butterfly
superclass 4	bottle, bowl, can, cup, plate
superclass 5	apple, mushroom, orange, pear, sweet pepper
superclass 6	keyboard, telephone
superclass 7	bed, chair, couch, table, wardrobe, television
superclass 8	bee, beetle, cockroach, lobster
superclass 9	bear, leopard, lion, tiger, wolf
superclass 10	castle, house, skyscraper, train
superclass 11	cloud, forest, mountain, plain, sea
superclass 12	camel, cattle, chimpanzee, elephant, kangaroo, dinosaur
superclass 13	fox, porcupine, possum, raccoon, skunk
superclass 14	snail, worm, snake, caterpillar, ray
superclass 15	baby, boy, girl, man, woman
superclass 16	crocodile, lizard, turtle
superclass 17	hamster, mouse, rabbit, shrew, squirrel, beaver, otter
superclass 18	maple tree, oak tree, palm tree, pine tree, willow tree
superclass 19	streetcar, bus, motorcycle, road
superclass 20	lawn mower, pickup truck, tank, tractor
superclass 21	bicycle, motorcycle, clock
superclass 22	crab, spider
superclass 23	bridge, seal
superclass 24	rocket
superclass 25	lamp

Table B.2: Learned tree hierarchy for the CIFAR-100 dataset.

dropout, each unit was retained with probability $p = 0.8$ at each layer. We split the 25,000 labeled examples into 10,000 for training, 5,000 for validation and 10,000 for test.

B.2.2 Initial and Learned Trees

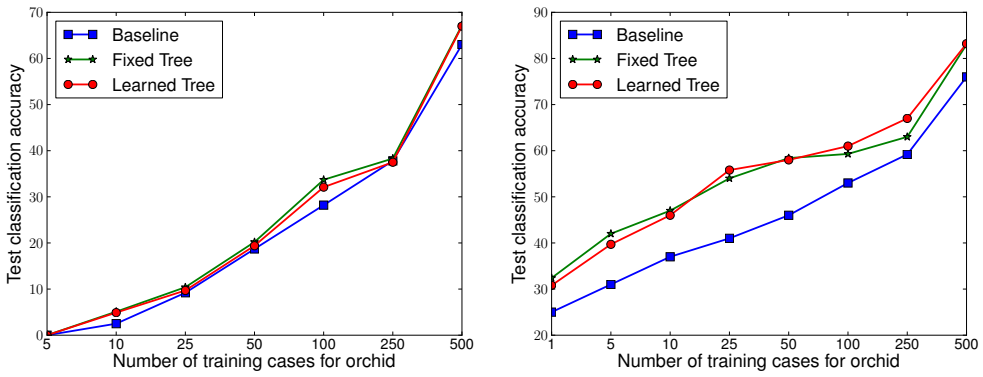
This dataset contains 38 classes, which can be categorized under a tree as shown in Table. B.3. The superclasses are based on the higher-level concepts as specified in Huiskes et al. (2010). The learned tree is shown in Table. B.4.

Superclass	Classes
animals	animals, bird, dog, bird*, dog*
people	baby*, people*, female*, male*, portrait*, baby, people, female, male, portrait
plants	flower, tree, plant life, flower*, tree*
sky	clouds, clouds*, sky
water	water, river, lake, sea, river*, sea*
time of day	night, night*, sunset
transport	transport, car, car*
structures	structures
food	food
indoor	indoor

Table B.3: Given tree hierarchy for the MIR-Flickr dataset.

Superclass	Classes
superclass 1	animals, dog, dog*, bird*
superclass 2	bird, sky, clouds, clouds*
superclass 3	baby*, baby, people*, female*, male*, portrait*, portrait
superclass 4	people, female, male
superclass 5	flower, flower*, tree*
superclass 6	tree, plant life
superclass 7	water, river, sea
superclass 8	lake, river*, sea*
superclass 9	night, night*
superclass 10	transport
superclass 11	car, car*
superclass 12	sunset
superclass 13	structures
superclass 14	food
superclass 15	indoor

Table B.4: Learned tree hierarchy for the MIR-Flickr dataset.



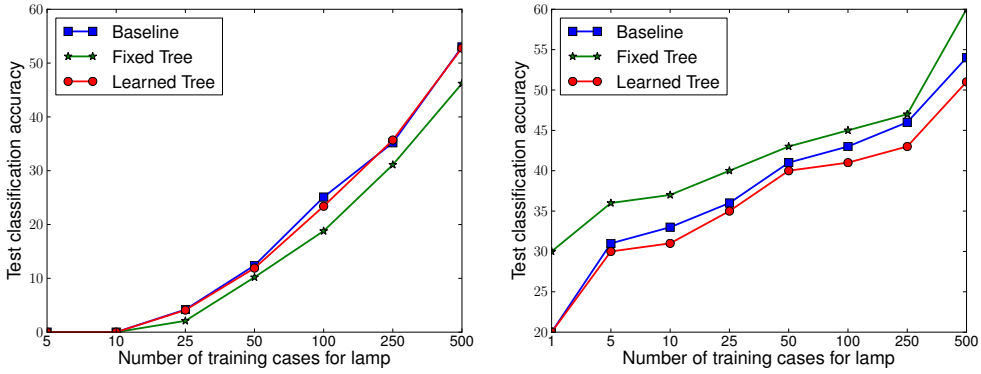
(a) (b)

Figure B.1: Results on CIFAR-100 with few examples for the *orchid* class. **Left**: Test set classification accuracy for different number of examples. **Right**: Accuracy when classifying a *orchid* as any other kind of flower is also considered correct.

B.3 Additional experiments on CIFAR-100 with few examples for one class

In this section we describe additional experiments in which we worked in a scenario where there are lots of examples for different classes, but only few examples of one particular class. In the paper, we presented details for the *dolphin* class. Here we include details of the same experiment with two other classes. The class presented in the paper is a typical case. Here, we pick two classes – one on which the model does quite well and the other on which it performs poorly. We take the *orchid* class which got a large positive transfer (+25%) and the *lamp* class which got a large negative one (-10%) when training with 10 examples per class for all classes, as described in the paper.

Fig. B.1 shows the results on the *orchid* class. Fig. B.1a shows the classification accuracy as the number of examples of *orchid* given to the algorithm increases from 5 to 500. We see that all the models do about the same, with the tree-based models doing slightly better. This is probably because



(a) (b)

Figure B.2: Results on CIFAR-100 with few examples for the *lamp* class. **Left**: Test set classification accuracy for different number of examples. **Right**: Accuracy when classifying a *lamp* as any other kind of household electrical device is also considered correct.

the given tree structure is quite accurate for the *flower* superclass which includes *orchid*. Learning the tree does not lead to much change. Fig. B.1b shows the classification accuracy when classifying an example as any other kind of flower is also considered correct. Here we see that the tree-based models perform significantly better, showing useful positive transfer. This is expected because the tree-based prior encourages the classification parameters for all kinds of flowers to be close together.

Fig. B.2 shows the results on the *lamp* class. Fig. B.2a shows the classification accuracy as the number of examples of *lamp* is increased. We see that the fixed tree model does worse than the other two. This is probably because *lamp* falls in the superclass *household electrical devices* which includes visually dissimilar objects such as keyboards and clocks. The learned tree model however does better. It learns to not group *lamp* along with these dissimilar classes and almost matches the baseline's performance. Fig. B.2b shows the accuracy when classifying *lamp* as any other household electrical device is also considered correct. In this case, the fixed tree model does much better. This is probably because the prior used by the fixed tree favours this loss function. This shows that the model is able to enforce relationships that are implied by the tree structure.

Bibliography

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, February 1998.

Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M. Kakade, and Matus Telgarsky. Tensor decompositions for learning latent variable models. *J. Mach. Learn. Res.*, 15(1):2773–2832, January 2014.

Pierre Baldi and Ronny Meir. Computing with arrays of coupled oscillators: An application to preattentive texture discrimination. *Neural Computation*, 2(4):458–471, 1990.

E. Bart, I. Porteous, P. Perona, and M. Welling. Unsupervised learning of visual taxonomies. In *CVPR*, pages 1–8, 2008.

Muhammet Bastan, Hayati Cam, Ugur Gudukbay, and Ozgur Ulusoy. Bilvideo-7: An mpeg-7- compatible video indexing and retrieval system. *IEEE Multimedia*, 17:62–73, 2010. ISSN 1070-986X.

Å. Björck and C. Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM J. Numer. Anal.*, 8:358–364, 1971.

David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, 2012.

David M. Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

David M. Blei, Thomas L. Griffiths, and Michael I. Jordan. The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *J. ACM*, 57(2), 2010.

A Bosch, Andrew Zisserman, and X Munoz. Image classification using random forests and ferns. *IEEE 11th International Conference on Computer Vision*, 23:1–8, 2007.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML*, 2012.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. *CoRR*, abs/1605.09673, 2016.

Charles F. Cadieu and Bruno A. Olshausen. Learning intermediate-level representations of form and motion from natural movies. *Neural Computation*, 24(4):827–866, 2012.

K. Canini, L. Shi, and T. Griffiths. Online inference of topics with latent Dirichlet allocation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, 2009.

S. Chiappa, S. Racaniere, D. Wierstra, and S. Mohamed. Recurrent environment simulators. In *ICLR*, 2017.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, pages 1724–1734, 2014.

Stephen Cox, Richard Harvey, Yuxuan Lan, Jacob Newman, and Barry Theobald. The challenge of multispeaker lip-reading. In *International Conference on Auditory-Visual Speech Processing*, pages 179–184, September 2008.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

Hal Daumé, III. Bayesian multitask learning with latent hierarchies. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 135–142. AUAI Press, 2009.

Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems 28*, pages 1486–1494, 2015.

Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk, and H. J. Reitboeck. Coherent oscillations: a mechanism of feature linking in the visual cortex? Multiple electrode and correlation analyses in the cat. *Biological cybernetics*, 60(2):121–130, 1988.

Alan Edelman, Tomás A. Arias, and Steven T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM J. Matrix Anal. Appl.*, 20(2):303–353, April 1999.

SM Ali Eslami, Nicolas Heess, Christopher KI Williams, and John Winn. The shape boltzmann machine: a strong model of object shape. *International Journal of Computer Vision*, 107(2):155–176, 2014.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *ACM SIGKDD*, 2004.

Li Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(4):594–611, April 2006.

Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *CoRR*, abs/1605.07157, 2016.

Simone Fiori. A theory for learning by weight flow on stiefel-grassman manifold. *Neural Computation*, 13(7):1625–1647, July 2001.

William M. Fisher, George R. Doddington, and Kathleen M. Goudie-Marshall. The DARPA Speech Recognition Research Database: Specifications and Status. In *Proceedings of DARPA Workshop on Speech Recognition*, pages 93–99, 1986.

Yoav Freund and David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1994.

Lars Gislén, Carsten Peterson, and Bo Söderberg. Rotor neurons: Basic formalism and dynamics. *Neural Computation*, 4(5):737–745, 1992.

Ian Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Multi-prediction deep boltzmann machines. In *Advances in Neural Information Processing Systems 26*, pages 548–556, 2013a.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1319–1327, 2013b.

Hermann Grassmann. *Die Ausdehnungslehre*. verlag von Th. Chr. Fr. Enslin, 1862.

Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. DRAW: A recurrent neural network for image generation. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1462–1471, 2015.

T. Griffiths and M. Steyvers. Finding scientific topics. In *Proceedings of the National Academy of Sciences*, volume 101, pages 5228–5235, 2004.

M. Guillaumin, J. Verbeek, and C. Schmid. Multimodal semi-supervised learning for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 902–909, 2010.

Mihai Gurban and Jean-Philippe Thiran. Information theoretic feature extraction for audio-visual speech recognition. *IEEE Transactions on Signal Processing*, 57(12):4765–4776, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40(1–3):185 – 234, 1989.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1711–1800, 2002.

Geoffrey E. Hinton, Brian Sallans, and Zoubin Ghahramani. A hierarchical community of experts. In *Learning in Graphical Models*, pages 479–494. MIT Press, Cambridge, MA, USA, 1999.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012a.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Daniel Hsu and Sham M. Kakade. Learning mixtures of spherical gaussians: Moment methods and spectral decompositions. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 11–20, New York, NY, USA, 2013. ACM.

Mark J. Huiskes and Michael S. Lew. The MIR Flickr retrieval evaluation. In *ACM International Conference on Multimedia Information Retrieval*, 2008.

Mark J. Huiskes, Bart Thomee, and Michael S. Lew. New trends and ideas in visual concept detection: the MIR flickr retrieval evaluation initiative. In *11th ACM International Conference on Multimedia Information Retrieval*, pages 527–536, 2010.

Jarmo Hurri and Aapo Hyvärinen. Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691, 2003.

A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(1):221–231, Jan 2013.

Ian T. Jolliffe. Rotation of principal components: choice of normalization constraints. *Journal of Applied Statistics*, 22(1):29–35, 1995.

I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *CoRR*, abs/1610.00527, 2016.

Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 521–528, June 2011.

Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

Seyoung Kim and Eric P. Xing. Tree-guided group lasso for multi-task regression with structured sparsity. In *ICML*, pages 543–550, 2010.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems 28*, pages 3294–3302, 2015.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Toronto, Ontario, Canada, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, 2012.

H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. HMDB: a large video database for human motion recognition. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2011.

Zhen-Zhong Lan, Ming Lin, Xuanchong Li, Alexander G. Hauptmann, and Bhiksha Raj. Beyond gaussian pyramid: Multi-skip feature stacking for action recognition. *CoRR*, abs/1411.6660, 2014.

Hugo Larochelle and Stanislas Lauly. A neural autoregressive topic model. In *Advances in Neural Information Processing Systems 25*, pages 2717–2725, 2012.

Q. V. Le, W. Zou, S. Y. Yeung, and A. Y. Ng. Learning hierarchical spatio-temporal features for action recognition with independent subspace analysis. In *CVPR*, 2011.

H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, 2009a.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, 2009b.

M Lengyel and Peter Dayan. Uncertainty, phase and oscillatory hippocampal recall. In *Advances in Neural Information Processing Systems 19*, pages 833–840. MIT Press, 2007.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *CoRR*, abs/1605.08104, 2016.

Patrick Lucey and Sridha Sridharan. Patch-based representation of visual speech. In *Proceedings of the HCSNet workshop on Use of vision in human-computer interaction - Volume 56*, VisHCI '06, pages 79–85. Australian Computer Society, Inc., 2006.

B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, and A. Yamada. Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):703 –715, 2001.

Michaël Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *CoRR*, abs/1511.05440, 2015.

Iain Matthews, Timothy F. Cootes, J. Andrew Bangham, Stephen Cox, and Richard Harvey. Extraction of visual features for lipreading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):198–213, February 2002.

Roland Memisevic. Learning to relate images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1829–1846, 2013.

Roland Memisevic and Geoffrey E. Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, June 2010.

Vincent Michalski, Roland Memisevic, and Kishore Konda. Modeling deep temporal dependencies with recurrent grammar cells. In *Advances in Neural Information Processing Systems 27*, pages 1925–1933, 2014.

George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

D. Mimno and A. McCallum. Topic models conditioned on arbitrary features with dirichlet-multinomial regression. In *UAI*, pages 411–418, 2008.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 737–744, New York, NY, USA, 2009. ACM.

A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 2011.

Radford M. Neal. Connectionist learning of belief networks. *Artif. Intell.*, 56(1):71–113, July 1992.

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

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

Y. Nishimori. Learning algorithm for independent component analysis by geodesic flows on orthogonal group. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 2, pages 933–938 vol.2, Jul 1999.

A. J. Noest. Associative memory in sparse phasor neural networks. *EPL (Europhysics Letters)*, 6(5): 469, 1988.

André J. Noest. Phasor neural networks. In *Proceedings of the 1987 International Conference on Neural Information Processing Systems*, pages 584–591, 1987.

Aude Oliva and Antonio Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42:145–175, 2001.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016a.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016b.

Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1717–1724, 2014.

G. Papandreou, A. Katsamanis, Vassilis Pitsikalis, and P. Maragos. Multimodal fusion and learning with uncertain features applied to audiovisual speech recognition. In *IEEE 9th Workshop on Multimedia Signal Processing*, pages 264–267, 2007.

G. Papandreou, A. Katsamanis, Vassilis Pitsikalis, and P. Maragos. Adaptive multimodal fusion by uncertainty compensation with application to audiovisual speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(3):423–435, 2009.

Eric K. Patterson, Sabri Gurbuz, Zekeriya Tufekci, and John N. Gowdy. Cuave: A new audio-visual database for multimodal human-computer interface research. In *International Conference on Audio Speech and Signal Processing*, pages 2017–2020, 2002.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Marc'Aurelio Ranzato, Arthur Szlam, Joan Bruna, Michaël Mathieu, Ronan Collobert, and Sumit Chopra. Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, abs/1412.6604, 2014.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder network. In *Advances in Neural Information Processing Systems 28*, 2015.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

David P. Reichert and Thomas Serre. Neuronal synchrony in complex-valued deep networks. *CoRR*, abs/1312.6115, 2013.

Danilo J. Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1278–1286, 2014.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Stat.*, 22:400–407, 1951.

R. Salakhutdinov, J. Tenenbaum, and A. Torralba. Learning to learn with compound hierarchical-deep models. In *NIPS*. MIT Press, 2011a.

R. Salakhutdinov, A. Torralba, and J. Tenenbaum. Learning to share visual appearance for multiclass object detection. In *CVPR*, 2011b.

R. R. Salakhutdinov and G. E. Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 12, 2009a.

Ruslan Salakhutdinov and Geoff Hinton. A better way to pretrain deep boltzmann machines. In *Advances in Neural Information Processing Systems 25*, pages 2456–2464, 2012.

Ruslan Salakhutdinov and Geoffrey Hinton. Replicated softmax: an undirected topic model. In *Advances in Neural Information Processing Systems 22*, pages 1607–1614, 2009b.

Babak Shahbaba and Radford M. Neal. Improving classification when a class hierarchy is available using a hierarchy-based prior. *Bayesian Analysis*, 2(1):221–238, 2007.

K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, 2014a.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014b.

P. Smolensky. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 194–281. MIT Press, Cambridge, MA, USA, 1986.

k. Soomro, A. Roshan Zamir, and M. Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. In *CRCV-TR-12-01*, 2012.

Suvrit Sra. *Matrix Nearness Problems in Data Mining*. PhD thesis, University of Texas at Austin, 8 2007.

Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, Toronto, Ontario, Canada, 2013.

Nitish Srivastava and Ruslan Salakhutdinov. Discriminative transfer learning with tree-based priors. In *Advances in Neural Information Processing Systems 26*, pages 2094–2102, 2013.

Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep Boltzmann machines. *Journal of Machine Learning Research*, 15:2949–2980, 2014.

Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep Boltzmann machines. In *Advances in Neural Information Processing Systems 25*, pages 2222–2230, 2012.

Nitish Srivastava, Ruslan Salakhutdinov, and Geoffrey Hinton. Modeling documents with deep Boltzmann machines. In *UAI*. AUAI Press, 2013.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using LSTMs. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 843–852, 2015.

Eduard Stiefel. Richtungsfelder und fernparallelismus in n-dimensionalen mannigfaltigkeiten. *Commentarii Mathematici Helvetici*, 8(1):305–353, 1935.

J. Susskind, R. Memisevic, G. Hinton, and M. Pollefeys. Modeling the joint density of two images under a variety of transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Ilya Sutskever, Geoffrey E. Hinton, and Graham W. Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems 21*, pages 1601–1608, 2009.

Ilya Sutskever, Oriol Vinyals, and Quoc V. V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

Graham W. Taylor, Rob Fergus, Yann LeCun, and Christoph Bregler. Convolutional learning of spatio-temporal features. In *Eurpean Conference on Computer Vision*, 2010.

Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

Y. W. Teh, K. Kurihara, and M. Welling. Collapsed variational inference for HDP. In *Advances in Neural Information Processing Systems 21*, 2008.

T. Tieleman. Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1064–1071, 2008.

T. Tieleman and G.E. Hinton. Using Fast Weights to Improve Persistent Contrastive Divergence. In *Proceedings of the 26th International Conference on Machine learning*, pages 1033–1040, 2009.

Tijmen Tieleman. *Optimizing neural networks that generate images*. PhD thesis, Department of Computer Science, University of Toronto, 2014.

Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3D: generic features for video analysis. *CoRR*, abs/1412.0767, 2014.

P.J. Uhlhaas, G. Pipa, B. Lima, L. Melloni, S. Neuenschwander, D. Nikolic, and W. Singer. Neural synchrony in cortical networks: history, concept and current status. *Frontiers in Integrative Neuroscience*, 17(3), 2009.

Gary Ulrich. Computer generation of distributions on the m-sphere. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 33(2):pp. 158–163, 1984.

Harri Valpola. From neural PCA to deep unsupervised learning. *CoRR*, abs/1411.7783, 2014.

J. H. van Hateren and D. L. Ruderman. Independent component analysis of natural image sequences yields spatio-temporal filters similar to simple cells in primary visual cortex. *Proceedings. Biological sciences / The Royal Society*, 265(1412):2315–2320, 1998.

A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

Jakob Verbeek, Matthieu Guillaumin, Thomas Mensink, and Cordelia Schmid. Image annotation with TagProp on the MIRFLICKR set. In *11th ACM International Conference on Multimedia Information Retrieval*, pages 537–546, 2010.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *CoRR*, abs/1411.4555, 2014.

Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *CoRR*, abs/1606.07873, 2016.

Chong Wang and David M. Blei. Variational inference for the nested chinese restaurant process. In *Advances in Neural Information Processing Systems 22*, pages 1990–1998, 2009.

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. *CoRR*, abs/1505.00687, 2015.

M. Welling, M. Rosen-Zvi, and G. E. Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems 18*, pages 1481–1488, 2005.

Laurenz Wiskott and Terrence J. Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

Andrew T.A Wood. Simulation of the von mises fisher distribution. *Communications in Statistics - Simulation and Computation*, 23(1):157–164, 1994.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Eric P. Xing, Rong Yan, and Alexander G. Hauptmann. Mining associated text and images with dual-wing harmoniums. In *UAI*, pages 633–641, 2005.

Tianfan Xue, Jiajun Wu, Katherine L. Bouman, and William T. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. *CoRR*, abs/1607.02586, 2016.

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with dirichlet process priors. *J. Mach. Learn. Res.*, 8:35–63, May 2007.

L. Younes. Parameter inference for imperfectly observed Gibbsian fields. *Probability Theory Rel. Fields*, 82:625–645, 1989.

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

A. L. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems*, 2004.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.

Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013.

Richard Zemel, Christopher K. I. Williams, and Michael C. Mozer. Lending direction to neural networks. *Neural Networks*, 8:503–512, 1995a.

Richard S. Zemel, Christopher K. I. Williams, and Michael Mozer. Directional-unit boltzmann machines. In *Advances in Neural Information Processing Systems 5*, pages 172–179, 1992.

Richard S. Zemel, Christopher K. I. Williams, and Michael C. Mozer. Lending direction to neural networks. *Neural Networks*, 8(4):503–512, June 1995b.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016.

Guoying Zhao, M. Barnard, and M. Pietikainen. Lipreading with local spatiotemporal descriptors. *IEEE Transactions on Multimedia*, 11(7):1254–1265, 2009.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15:2006, 2004.

Alon Zweig and Daphna Weinshall. Hierarchical regularization cascade for joint learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 37–45, May 2013.