Learning Distributed Representations for Statistical Language Modelling and Collaborative Filtering

by

Andriy Mnih

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

Abstract

Learning Distributed Representations for Statistical Language Modelling and Collaborative Filtering

Andriy Mnih

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2010

With the increasing availability of large datasets machine learning techniques are becoming an increasingly attractive alternative to expert-designed approaches to solving complex problems in domains where data is abundant. In this thesis we introduce several models for large sparse discrete datasets. Our approach, which is based on probabilistic models that use distributed representations to alleviate the effects of data sparsity, is applied to statistical language modelling and collaborative filtering.

We introduce three probabilistic language models that represent words using learned real-valued vectors. Two of the models are based on the Restricted Boltzmann Machine (RBM) architecture while the third one is a simple deterministic model. We show that the deterministic model outperforms the widely used n-gram models and learns sensible word representations.

To reduce the time complexity of training and making predictions with the deterministic model, we introduce a hierarchical version of the model, that can be exponentially faster. The speedup is achieved by structuring the vocabulary as a tree over words and taking advantage of this structure. We propose a simple feature-based algorithm for automatic construction of trees over words from data and show that the resulting models can outperform non-hierarchical neural models as well as the best n-gram models.

We then turn our attention to collaborative filtering and show how RBM models can be used to model the distribution of sparse high-dimensional user rating vectors efficiently, presenting inference and learning algorithms that scale linearly in the number of observed ratings. We also introduce the Probabilistic Matrix Factorization model which is based on the probabilistic formulation of the low-rank matrix approximation problem for partially observed matrices. The two models are then extended to allow conditioning on the identities of the rated items whether or not the actual rating values are known. Our results on the Netflix Prize dataset show that both RBM and PMF models outperform online SVD models.

Acknowledgements

Most of all, I would like to thank Geoffrey Hinton for being a great advisor. It was a pleasure to work with him. I also thank Radford Neal, Sam Roweis, and Rich Zemel, who were on my supervisory committee, as well as Yoshua Bengio, who served as the external examiner, for providing valuable and detailed feedback.

I am grateful to the members of the machine learning group, especially Ruslan Salakhutdinov, Ben Marlin, Roland Memisevic, Vlad Mnih, Iain Murray, Vinod Nair, Ilya Sutskever, Graham Taylor, and Tijmen Tieleman, for many interesting discussions.

Finally, I would like to thank my parents for their support and encouragement over the years.

This work was financially supported by NSERC, OGS, CIFAR, and CFI.

Contents

Chapter 1

Introduction

The increasing availability and variety of large datasets requires the development of sophisticated and efficient methods for processing them. Machine learning techniques are becoming an increasingly attractive alternative to expert-designed approaches to solving complex problems in domains with large amounts of data available. However, in order to be useful in practice, machine learning methods need to be both scalable and powerful enough to capture the rich structure of the data. In this thesis we introduce several models for large sparse discrete datasets. Our approach, which is based on probabilistic models that use distributed representations to alleviate the effects of data sparsity, is applied to statistical language modelling and collaborative filtering.

The discrete nature of the attributes, such as word or document IDs, makes modelling such datasets quite different from modelling datasets with real-valued attributes. Many tasks of interest for discrete datasets can be reduced to learning a mapping from vectors with discrete components to some (discrete or continuous) output space. If the number of vector components or the number of values each component can take is large, learning the mapping is difficult, because it amounts to estimating entries in a very large table, with very little or no data available for most entries. This is known as the data sparsity problem in statistical language modelling and collaborative filtering. In a continuous space, this kind of a problem can be easily addressed by smoothing the estimates by averaging over multiple datapoints near the point of interest. In a discrete space, however, it is often not clear how to implement this kind of smoothing because there is typically no natural distance metric and hence no concept of neighbourhood.

As an example, consider the task of finding documents similar to a given document. A popular approach to this task is based on representing each document using a vector of its word occurrence counts. The similarity between a pair of documents is then measured by applying a simple metric, such as TF-IDF, to the corresponding pair of word count vectors. While this approach can be quite effective, it does not handle synonyms properly, because it performs exact word matching. Thus documents that are similar in all respects other than word choice are unlikely to be judged similar by such an algorithm. This behaviour is a consequence of representing words using word indices, which corresponds to the assumption that each word is similar only to itself. A much more flexible word similarity scheme can be implemented by representing words using learned real-valued vectors, as we show in the statistical language modelling part of this thesis. Since the learned word representations capture word similarity information implicit in the training data, this approach alleviates the effects of data sparsity by ensuring automatic generalization across similar words.

Though generalization ability can be the primary criterion when selecting a model, the ability of the training algorithm to scale to large datasets is also important. In fact, a less powerful model trained on all of the available data can often achieve better predictive accuracy than a more powerful model trained on a subset of the data. To ensure scaling to large datasets, we train our models using either online or mini-batch-based gradient descent.

1.1 Thesis outline

In this thesis we apply probabilistic models based on distributed representations to two tasks that involve discrete data: statistical language modelling and collaborative filtering. The remaining two sections of this chapter provide an introduction to distributed representations and restricted Boltzmann machines (RBMs), which are used in both the language modelling and collaborative filtering parts of the thesis.

The next three chapters deal with statistical language modelling. Chapter 2 gives a brief overview of this area, introducing the standard approach to the problem based on modelling the distribution of the next word conditional on a fixed number of preceding words. The widely used n-gram language models as well as the recently developed neural language models are then outlined and the strengths and weaknesses of these two model types are discussed.

In Chapter 3 we introduce three novel non-hierarchical neural language models that represent words using learned real-valued feature vectors. Our first model, based on the RBM architecture, models the interaction between the preceding words and the next word using a vector of stochastic binary latent variables. We obtain the second model by augmenting the first model with temporal connections between the successive instantiations of the latent variables to increase the number of preceding words taken into account without significantly increasing the number of model parameters. We then introduce a simpler and faster deterministic log-bilinear (LBL) language model that performs linear prediction in the space of word feature vectors to define the distribution for the next word. We show that the temporal RBM model is competitive with backoff Kneser-Ney n-gram models, while the LBL model is superior to them and is capable of taking advantage of larger windows of preceding words.

In Chapter 4 we address the main shortcoming of the LBL language model, which is its long training and prediction times. We build on the approach of Morin and Bengio (2005), who proposed a hierarchical language model based on a binary tree over words. Their hierarchal model was two orders of magnitude faster than the non-hierarchical model it was based on, but performed considerably worse in spite of using a word tree created using expert knowledge. We introduce a fast hierarchical version of our LBL model along with a simple feature-based algorithm for automatic construction of word trees from the data. We then show that the resulting model can outperform non-hierarchical neural models as well as the best n-gram models.

Chapter 5 constitutes the collaborative filtering part of the thesis. We first introduce recommender systems and collaborative filtering (CF) and describe the two main types of CF algorithms. We take the model-based approach and introduce two probabilistic models for CF. The first model captures the joint distribution of user ratings using a vector of binary latent variables. We show how the RBM architecture can be extended to deal with sparse rating vectors so that the training and inference complexity is linear in the number of observed ratings. The RBM model is then extended so that it can take into account the identities of the items rated by a user even when the actual rating values are unknown, producing the conditional RBM model. We then introduce the Probabilistic Matrix Factorization (PMF) model which is based on a probabilistic formulation of the low-rank matrix approximation methods for partially-observed matrices. It can also be viewed as a probabilistic counterpart of the Maximum Margin Matrix Factorization model (Srebro and Jaakkola, 2003). We then extend PMF, in a manner similar to the conditional RBM model, to allow conditioning on the identities of the items rated by a user. The training time of both PMF and conditional PMF scales linearly in the number of observed ratings, which makes them capable of scaling to very large datasets. We evaluate our models on the Netflix Prize dataset consisting of about 100M movie ratings. Our results show that both RBM and PMF models perform quite well, outperforming online SVD models. Though PMF models perform better than RBM models, they are also considerably less compact, requiring many tens of millions of parameters to achieve optimal performance. Conditioning on the identities of the rated movies is shown to be a very effective technique for boosting performance of PMF models, with conditional PMF emerging as the best-performing model class in the comparison.

1.2 Distributed representations

Suppose we would like to represent objects or entities with binary vectors. Of the continuum of possible representations schemes, we will consider two extremes: the (fully) local representations and the (fully) distributed representations. In a local representation, each entity is represented by a different component of the vector, so there is a one-to-one mapping between entities and components. One-of-n (or one-hot) encoding, where entity i is represented by a binary vector with component i set to 1 and all other components set to 0, is a classic example of a local representation.

In a distributed representation, an entity is represented by the joint configuration of all the components of the vector (Hinton et al., 1986). Thus, each component takes part in representing every entity. Representing entity i with the base-2 representation of number i (padded with zeros to always have the same length) is an example of a distributed representation scheme. This scheme demonstrates the exponential efficiency of distributed representations, since it needs only n bits to represent 2^n entities. In contrast, local representations need 2^n bits to represent that many entities.

Since in a distributed representation an entity is represented by the joint configuration of all components, distributed representations can be used to encode similarity between entities by representing similar entities with vectors that differ in only a small fraction of their component values. Even though this approach is fairly crude since it discretizes the (dis)similarity between entities by encoding it using the Hamming distance between the corresponding representations, it is still considerably more flexible than any local-representation-based approach. Indeed, since in a local representation scheme all entities have equally (dis)similar representations, such a scheme can only encode that each entity is similar to itself and dissimilar to all other entities.

Real-valued distributed representations provide a more flexible way to encode entity similarity information than their discrete-valued counterparts, since the similarity scale is no longer integer-valued. More importantly, real-valued representations suitable for a particular task can be efficiently learned jointly with other model parameters from the training data for the task, using gradient descent (Hinton, 1986).

For tasks involving discrete sequences as inputs, using learned real-valued distributed representations for sequence elements can lead to automatic generalization for novel input sequences. Since elements that occur in similar sequences (i.e. sequences that correspond to identical outputs) will typically be assigned similar representations, replacing one of the sequence elements with an element with a similar representation will not change the model input significantly, and thus will have very little effect on the output. This effect is explained in the context of statistical language modelling in Sec. 2.2.

1.3 Restricted Boltzmann machines

A Boltzmann Machine (BM) is an undirected graphical model parameterized in terms of pairwise potentials (or interactions) between its variables (Hinton and Sejnowski, 1986). While a BM with only observed variables can learn to model only their first- and second-order statistics, introducing latent variables can allow the BM to capture higher-order statistics as well. The observed and the latent variables of a BM are usually referred to as the visible and the hidden units respectively. Though general BM models can be very powerful, exact inference in such models is intractable. By restricting the model topology, however, it is possible to define a subclass of BMs with efficient exact inference.

A Restricted Boltzmann Machine (RBM) is a BM with only visible unit-hidden unit interactions present. Thus if we represent units as nodes and interactions as edges of a graph, an RBM can be visualized as a bipartite graph with the visible and the hidden units forming two components with no within-component connections. This bipartite property makes the time complexity of RBM inference linear in the number of hidden units.

Though RBMs have been generalized to allow using arbitrary exponential family random variables (Welling et al., 2005), for simplicity we will present the classic case of an RBM with binary (or Bernoulli) visible and hidden units. Let v be a binary vector containing the configuration of the visible units. Let h be the corresponding vector for the hidden units.

The joint distribution for the visible and hidden units is defined by specifying the energy function for joint configurations of the visible and hidden units. This function typically has the form

$$E(v, h) = -v^T W h - a^T v - b^T h, \tag{1.1}$$

where W is the parameter matrix that specifies the interactions between the visible and hidden units. Vectors a and b contain the biases for the visible and hidden units respectively. The joint distribution for v and h is given by the Boltzmann distribution induced by the energy function:

$$P(v, h) = \frac{1}{Z} \exp(-E(v, h)), \tag{1.2}$$

where $Z = \sum_v \sum_h \exp(-E(v, h))$ is the normalizing constant, sometimes called the partition function. Note that computing Z is typically infeasible because it requires computing a sum of exponentially many terms.

The marginal distribution for the visible units is obtained by marginalizing out the hidden variables:

$$P(v) = \sum_h P(v, h). \tag{1.3}$$

1.3.1 Inference

The lack of hidden-to-hidden and visible-to-visible connections in an RBM results in convenient conditional independence properties which enable efficient inference and Gibbs sampling in the model. These properties are:

- hidden units are conditionally independent given the state of the visible units;
- visible units are conditionally independent given the state of the hidden units.

In other words, the conditional distributions $P(v|h)$ and $P(h|v)$ factorize:

$$P(v|h) = \prod_i P(v_i|h), \tag{1.4}$$

$$P(h|v) = \prod_j P(h_j|v). \tag{1.5}$$

The conditional distributions for the individual units are given by

$$P(v_i = 1|h) = \sigma\left(a_i + \sum_j W_{ij} h_j\right), \tag{1.6}$$

$$P(h_j = 1|v) = \sigma\left(b_j + \sum_i W_{ij} v_i\right), \tag{1.7}$$

where $\sigma(x)$ is the logistic function.

Because $P(h|v)$ factorizes, inference in an RBM takes time linear in the number of hidden units. To infer the posterior distribution over the hidden units for the given data vector v, we simply evaluate Eq. 1.7 for each of the hidden units.

Generating samples from an RBM is typically done using MCMC methods. The block Gibbs sampler that alternates between sampling from $P(h|v)$ and $P(v|h)$ is the MCMC method of choice for RBMs because their conditional independence properties make these sampling operations very efficient (and potentially parallelizable). Sampling from an RBM can be very time-consuming, however, because it can take the sampler a large number of steps[1] to converge to the model distribution.

[1]One step of the Gibbs sampler consists of sampling h from $P(h|v)$ and then sampling v from $P(v|h)$.

1.3.2 Learning

Though maximum likelihood learning is intractable in RBMs, we will describe it here because it serves as the basis for an efficient approximate learning algorithm. From Eqs. 1.2 and 1.3 it follows that the log-probability of a single datapoint v^o can be written as:

$$\log P(v^o) = \log\left(\sum_h \exp(-E(v^o,h))\right) - \log\left(\sum_v\sum_h \exp(-E(v,h))\right). \tag{1.8}$$

The gradient of the log-probability of v^o w.r.t. a parameter θ can be computed as follows:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log P(v^o) &= \frac{\partial}{\partial\theta}\log\left(\sum_h \exp(-E(v^o,h))\right) - \frac{\partial}{\partial\theta}\log\left(\sum_v\sum_h \exp(-E(v,h))\right)\\
&= \sum_h \frac{\exp(-E(v^o,h))\frac{\partial}{\partial\theta}(-E(v^o,h))}{\sum_h \exp(-E(v^o,h))} - \sum_v\sum_h \frac{\exp(-E(v,h))\frac{\partial}{\partial\theta}(-E(v,h))}{\sum_v\sum_h \exp(-E(v,h))}\\
&= \sum_h P(h|v^o)\frac{\partial}{\partial\theta}(-E(v^o,h)) - \sum_v\sum_h P(v,h))\frac{\partial}{\partial\theta}(-E(v,h))\\
&= E_D\left[\frac{\partial}{\partial\theta}(-E(v^o,h))\right] - E_M\left[\frac{\partial}{\partial\theta}(-E(v,h)),\right]
\end{aligned} \tag{1.9}$$

where $E_D[\cdot]$ and $E_M[\cdot]$ denote expectations w.r.t. the posterior distribution $P(h|v^o)$ conditional on the observed datapoint and the model distribution $P(v,h)$ respectively. The gradients of the negative energy function w.r.t. the model parameters are:

$$\begin{aligned}
\frac{\partial}{\partial W}(-E(v,h)) &= \frac{\partial}{\partial W}v^T Wh\\
&= vh^T, &(1.10)\\
\frac{\partial}{\partial a}(-E(v,h)) &= \frac{\partial}{\partial a}a^T v\\
&= v, &(1.11)\\
\frac{\partial}{\partial b}(-E(v,h)) &= \frac{\partial}{\partial b}b^T h\\
&= h. &(1.12)
\end{aligned}$$

Plugging in these energy gradients into Eq. 1.9 gives the gradients needed for learning:

$$\frac{\partial}{\partial W} \log P(v^o) = E_D \left[v^o h^T \right] - E_M \left[v h^T \right], \tag{1.13}$$

$$\frac{\partial}{\partial a} \log P(v^o) = v^o - E_M \left[v \right], \tag{1.14}$$

$$\frac{\partial}{\partial b} \log P(v^o) = E_D \left[h \right] - E_M \left[h \right]. \tag{1.15}$$

Unfortunately, computing these gradients exactly is intractable because evaluating expectations w.r.t. the model distribution requires summing over exponentially many terms. Though we can obtain unbiased estimates of $E_M[\cdot]$ by sampling from the model (e.g. using the Gibbs sampler) and averaging the quantity of interest over those samples, running the Markov chain to convergence before performing each parameter update is simply too time-consuming to be feasible. Contrastive divergence (CD) learning (Hinton, 2002) is an approximate learning procedure that does not require running the Markov chain to convergence. CD learning differs from the sampling-based implementation of maximum likelihood learning in two ways. First, instead of being initialized to a random configuration, the Markov chain is initialized to the current training case. Second, instead of running the sampler till the chain converges, it is run for only a small number of steps. In fact, even a single step of Gibbs sampling is often sufficient to produce a sensible learning signal. All of the RBM models in this thesis have been trained using this kind of one-step CD learning. The pseudo-samples produced by the brief sampling runs are then used instead of the true samples from the model to estimate the gradients using Eqs. 1.13 - 1.15. While the resulting learning algorithm does not optimize the log-likelihood, it does approximately optimize a different objective function and, more importantly, works well in practice (Carreira-Perpinan and Hinton, 2005).

Chapter 2

Introduction to language modelling

Statistical language modelling is concerned with building probabilistic models of word sequences. Such models can be used to discriminate probable sequences from improbable ones, a task important for performing speech recognition, information retrieval, and machine translation. The vast majority of statistical language models are based on the Markov assumption, which states that the distribution of a word depends only on some fixed number of words that immediately precede it. If we assume that only $n-1$ previous words have an effect on the distribution of the next word, the probability of a sequence of words $w_1, ..., w_N$ can be expressed as

$$P(w_{1:N}) = P(w_{1:n-1}) \prod_{i=n}^{N} P(w_i | w_{i-n+1:i-1}), \tag{2.1}$$

where $w_{1:i}$ denotes $w_1, ..., w_i$. As a result, the task of modelling the distribution of sequences of arbitrary length is reduced to modelling the distribution of the next word given $n-1$ preceding words, which are called the *context*. Though the Markov assumption is clearly false for natural languages, modelling $P(w_n | w_{1:n-1})$ can result in very effective statistical language models.

2.1 N-gram language models

N-gram models, which are the most popular statistical language models, are effectively (sparse) probability tables that store $P(w_n|w_{1:n-1})$. These models are estimated by counting n-tuples in the training data and normalizing the counts appropriately. Smoothing these naive estimates is essential for achieving good performance, since the number of model parameters is exponential in the context size. A number of effective smoothing methods have been developed specifically for n-gram models (Chen and Goodman, 1996). Such methods smooth the next word probability estimates for $n-1$ word contexts by interpolating them with the corresponding estimates for shorter contexts.

However, using larger context sizes (e.g. 5 and larger) does not usually lead to better models due to overfitting even when the best smoothing methods are used (Goodman, 2000). This problem is a consequence of the way n-gram models parameterize $P(w_n|w_{1:n-1})$: there is a separate free parameter for each context / next word pair. Since there are no a priori smoothness constraints on the parameters, there is no natural way to use information about similar contexts when estimating $P(w_n|w_{1:n-1})$ for a particular context. In other words, these models do not take advantage of the fact that similar words occur in similar contexts because they have no concept of similarity. Class-based n-grams (Brown et al., 1992) aim to address this issue by clustering words and/or contexts into classes based on their usage patterns and using this class information to improve generalization. While this approach can in some cases improve n-gram performance, it introduces a very rigid kind of similarity, since each word typically belongs to exactly one class.

2.2 Neural language models

An alternative and much more flexible approach to counteracting the data sparsity problem is to represent each word using a real-valued feature vector that captures its properties, so that words used in similar contexts will have similar feature vectors (Bengio et al., 2000). Then the conditional probability of the next word can be modelled as a smooth function of the feature vectors of the context words and the next word. This approach provides automatic smoothing, since for a given context similar words are now guaranteed to be assigned similar probabilities. Similarly, similar contexts are now likely to have similar representations resulting in similar predictions for the next word. Most models based on this approach use a feed-forward neural network that takes the feature vectors of the context words as input and produces the distribution for the next word as output (e.g. (Schwenk and Gauvain, 2002) and (Emami et al., 2003)). Perhaps the best known model of this type is the Neural Probabilistic Language Model (Bengio et al., 2003), which has been shown to outperform n-gram models on a dataset of about one million words.

Another attractive property of neural language models is that their memory requirements are linear in the vocabulary size and the context size. In contrast, the memory requirements of n-gram models are linear in the training set size because all non-zero counts for $w_{1:n}$ have to be stored explicitly, which can be prohibitive for large training sets.

2.2.1 The neural probabilistic language model

In this section we will describe the Neural Probabilistic Language Model of Bengio et al. (Bengio et al., 2000, 2003) in some detail because most models in the literature are based on it. The NPLM maps the given context to a distribution for the next word in two stages. First, it maps the given local representation of the context (i.e. word IDs) to a distributed representation by mapping the context word IDs $w_{1:n-1}$ to their distributed representations $C(w_{1:n-1}) = C(w_1), ..., C(w_{n-1})$. Then, it feeds the distributed representation of the context $C(w_{1:n-1})$ into a feed-forward neural network that outputs the predicted distribution for the next word. The neural network has one hidden layer of *tanh* units and a softmax output layer that ensures that the output of the network is always a normalized probability distribution. Note that the same mapping $C()$ is used for all context positions because the distributed representations of words are meant to capture the position-independent properties of words. In some versions of this model, there are also direct connections from the input layer to the output layer. Though adding such connections to a model reduces the number of epochs the network takes to converge, the resulting model generalizes slightly worse than the original model (Bengio et al., 2003).

The original version of the NPLM, also called the *direct* version, uses distributed representations for the context words but not for the predicted word. As a result, this architecture does not take advantage of word similarity among the predicted words. To address this drawback Bengio et al. (2003) proposed an alternative, energy-based *cycling* version of the NPLM that used the same mapping $C()$ from words to feature vectors for both the context words and the next word. This model uses a neural network to map the distributed representations of the context words and the candidate next word to an energy value, so that higher energies correspond to less likely context / next word combinations. After computing the energies for all the next word candidates, the energies are negated, exponentiated, and normalized to obtain a distribution for the next word. This model is slightly slower to train than the direct NPLM because each energy computation requires performing a forward pass in the neural network. The cycling NPLM also performs slightly worse than the direct NPLM according to Bengio et al. (2000). In this thesis, "NPLM" will refer to the direct NPLM.

2.2.2 Strengths and weaknesses of neural language models

While representing words using learned distributed representations makes neural language models more robust than n-gram models to data sparsity, there are drawbacks to this approach. By representing words using learned feature vectors, the ability to distinguish between words that are used in similar ways is greatly diminished. While this is also the reason for the superior generalization ability of neural language models, sometimes words that are very similar in usage are not interchangeable. For example, since word-for-word reproduction is the defining property of a quotation, replacing any word with its synonym makes the quotation much less probable. N-gram models excel in cases like this because they effectively memorize the n-tuples seen in the training data, after assigning each vocabulary word a unique ID.

This difference in behaviour highlights the fundamental difference in the nature of the two language model types. N-gram models are very local non-parametric models and thus are very expressive. In fact, in the limit of the infinite training set they will converge to the true distribution for the next word (given the previous $n-1$ words). Neural language models are less local and thus limited to representing a smaller class of distributions. They have a stronger inductive bias which allows them to generalize better given a small amount of training data but their bias also prevents them from capturing the true data distribution even in the limit of an infinite training set.

There are a number of choices to be made when using distributed representations in neural language models. For example, the context words and the predicted word can use different sets of word representations or they can share the same set. Also, assigning each word a single feature vector makes it difficult for the model to capture multiple distinct usage patterns (or senses) for a word. Assigning (some) words multiple feature vectors is a natural way to address this difficulty. Unfortunately, having multiple representations for context words seems to require computationally expensive inference for larger contexts. Handling multiple predictive representations per word, on the other hand is easy, since it effectively amounts to scaling the number of words in the (predictive) vocabulary by the number of senses per word.

The complementary strengths of the two language model types suggest that a combination of two models of different types might outperform the individual models. Traditionally, neural models have been combined with n-grams by training both models individually and then mixing the predictions from the two models using fixed mixing proportions. A more promising approach might be to either train the two models jointly (as components of the mixture), or to train one model on its own and then train the second model to optimize the predictive performance of the mixture.

2.3 Evaluating language models

In this thesis we evaluate language models based on their test set perplexity, which is the standard application-independent performance metric for probabilistic language models. Perplexity quantifies the (average) uncertainty the model has about the next word given its context. Perplexity is defined as

$$\mathcal{P} = \exp\left(-\frac{1}{N} \sum_{w_{1:n}} \log P(w_n | w_{1:n-1}) \right), \tag{2.2}$$

where the sum is over all subsequences of length n in the dataset, N is the number of such subsequences, and $P(w_n|w_{1:n-1})$ is the probability under the model of the n^{th} word in the subsequence given the previous $n-1$ words.

Chapter 3

Three non-hierarchical language models

The supremacy of n-gram models in statistical language modelling has recently been challenged by parametric models that use distributed representations to counteract the difficulties caused by data sparsity. Despite their promising results, however, such models have not received much attention and, as a result, only a small number of possible architectures have been explored. In this chapter, we will partially address this situation by introducing three novel probabilistic language models that use distributed word representations. We will start with a latent-variable language model that uses a large number of hidden binary variables to capture the desired conditional distribution. Then we will augment it with temporal connections between latent variables to increase the number of preceding words taken into account without significantly increasing the number of model parameters. Finally, we will introduce a simpler and faster deterministic language model that performs linear prediction in the space of distributed word representations.

3.1 The factored restricted Boltzmann machine language model

In this section we propose a probabilistic model for word sequences that uses distributed representations for words and captures the dependencies between words in a sequence using stochastic hidden variables. The main choice to be made here is between directed and undirected interactions between the hidden variables and the visible variables representing words. Blitzer et al. (2005a) proposed a model with directed interactions for this task. However, training their model required exact inference, which is exponential in the number of hidden variables. As a result, only a very small number of hidden variables can be used, which greatly limits the expressive power of the model.

In order to be able to handle a large number of hidden variables, we use a Restricted Boltzmann Machine (RBM) that has undirected interactions between multinomial visible units and binary hidden units. While maximum likelihood learning in RBMs is intractable, RBMs can be trained efficiently using contrastive divergence learning (Hinton, 2002) and the learning rule is unaffected when binary units are replaced by multinomial ones.

Assuming that the words we are dealing with come from a fixed vocabulary of size N_w, we model the observed words as multinomial random variables, each of which can take on one of N_w values. For convenience we will encode word identities using binary vectors of length N_w. Word w will be encoded by a vector v with 1 in the w^{th} position and zeros in all other positions.

Now we will define an RBM for modelling the distribution of the next word in a sentence given the word's context by specifying the energy function for joint configurations of the visible and hidden units. The first $(n-1)N_w$ visible units will represent the context words while the last N_w units will represent the next word. Perhaps the simplest energy function for this task is

$$E_0(w_n, h, w_{1:n-1}) = -\sum_{i=1}^{n} v_i^T G_i h, \tag{3.1}$$

where h is a vector containing the state of the N_h hidden variables. Here matrix G_i specifies the interaction between the multinomial visible unit v_i and the binary hidden units. For simplicity, we have ignored the bias terms for the visible and hidden units for now. Unfortunately, the parameterization in Eq. 3.1 requires nN_wN_h free parameters, which can be unacceptably high for vocabularies of even moderate size. Perhaps more importantly, in this parameterization of the model each word is associated with a different set of parameters for each of the n positions it can occupy, which means that the model does not have position-independent word representations.

Both of these drawbacks can be addressed by introducing distributed representations (i.e. feature vectors) for words. Generalization is made easier by sharing feature vectors across all sequence positions, and defining all of the interactions involving a word via its feature vector. This type of parameterization has been used in feed-forward neural networks for modelling symbolic relations (Hinton, 1986) and for statistical language modelling (Bengio et al., 2003).

We will represent each word using a real-valued feature vector of length N_f and make the energy depend on the word only through its feature vector. Let R be an $N_w \times N_f$ matrix with row i being the feature vector for the i^{th} word in the vocabulary. Then the feature vector for word w_i is given by $v_i^T R$. Using this notation, we define the joint energy of a sequence of words $w_1, ..., w_n$ along with the configuration of the hidden units h as

$$E(w_n, h, w_{1:n-1}) = -\sum_{i=1}^{n} v_i^T R W_i h - b_h^T h - b_v^T v_n. \tag{3.2}$$

Here matrix W_i specifies the interaction between the vector of hidden variables and the feature vector for the visible variable v_i. The vector b_h contains biases for the hidden units, while b_v is the vector of per-word biases. To simplify the notation we do not explicitly show the dependence of the energy functions and probability distributions on model parameters. In other words, we write $P(w_n|w_{1:n-1})$ instead of $P(w_n|w_{1:n-1}, W_i, R, ...)$.

Defining these interactions on the N_f-dimensional feature vectors instead of directly on the N_w-dimensional visible variables leads to a much more compact parameterization of the model, since typically N_f is much smaller than N_w. Using the same feature matrix R for all visible variables forces it to capture position-invariant information about words while further reducing the number of model parameters. With 18,000 words, 1000 hidden units and a context of size 2 ($n = 3$), the use of 100-dimensional feature vectors reduces the number of parameters by a factor of 25, from 54 million to a mere 2.1 million. As can be seen from Eqs. 3.1 and 3.2, the feature-based parameterization constrains each of the visible-hidden interaction matrices G_i to be a product of two low-rank matrices R and W_i, while the original parameterization does not constrain G_i in any way.

The joint conditional distribution of the next word and the hidden configuration h is defined in terms of the energy function in Eq. 3.2 as

$$P(w_n, h|w_{1:n-1}) = \frac{1}{Z_c} \exp(-E(w_n, h, w_{1:n-1})), \tag{3.3}$$

where $Z_c = \sum_{w_n} \sum_h \exp(-E(w_n, h, w_{1:n-1}))$ is a context-dependent normalization term. The conditional distribution of the next word given its context, which is the distribution we are ultimately interested in, can be obtained from the joint by marginalizing over the hidden variables:

$$P(w_n|w_{1:n-1}) = \frac{1}{Z_c} \sum_h \exp(-E(w_n, h, w_{1:n-1})). \tag{3.4}$$

Thus, we obtain a *conditional* model, which does not try to model the distribution of $w_{1:n-1}$ since we always condition on those variables.

3.1.1 Making predictions

One attractive property of RBMs is that the probability of a configuration of visible units can be computed up to a multiplicative constant in time linear in the number of

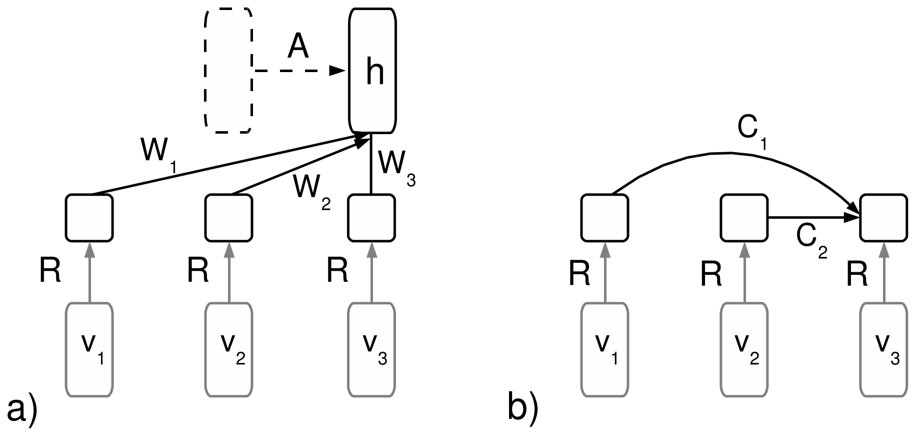

Figure 3.1: a) The diagram for the Factored RBM and the Temporal Factored RBM. The dashed part is included only for the TFRBM. b) The diagram for the log-bilinear model.

hidden units (Hinton, 2002). The normalizing constant, however, is usually infeasible to compute because it is a sum of an exponential number of terms (in the number of visible units). In the proposed language model, though, this computation is easy because normalization is performed only over w_n, resulting in a sum containing only N_w terms. Moreover, in some applications, such as speech recognition, we are interested in ratios of probabilities of words from a short list and as a result we do not have to compute the normalizing constant at all (Bengio et al., 2003).

The unnormalized probability of the next word can be efficiently computed using the formula

$$\begin{aligned}
P(w_n|w_{1:n-1}) &= \sum_h P(w_n, h|w_{1:n-1}) \\
&\propto \sum_h \exp(-E(w_n, h, w_{1:n-1})) \\
&= \sum_h \exp\left(\sum_{i=1}^{n} v_i^T R W_i h + b_h^T h + b_v^T v_n\right) \\
&= \exp\left(b_v^T v_n\right) \sum_h \exp\left(\sum_{i=1}^{n} v_i^T R W_i h + b_h^T h\right) \\
&= \exp\left(b_v^T v_n\right) \prod_{j=1}^{N_h} \sum_{h_j=0}^{1} \exp\left(\sum_{i=1}^{n} v_i^T R W_i^j h_j + b_{hj} h_j\right) \\
&= \exp\left(b_v^T v_n\right) \prod_{j=1}^{N_h} \left(1 + \exp\left(\sum_{i=1}^{n} v_i^T R W_i^j + b_{hj}\right)\right) \qquad (3.5)
\end{aligned}$$

where W_i^j is the j^{th} column of matrix W_i.

Since the normalizing constant for this distribution can be computed in time linear in the vocabulary size, exact inference in this model has time complexity proportional to the product of the number of hidden variables and the vocabulary size. This compares favourably with the exponential complexity of exact inference in the latent variable model proposed in (Blitzer et al., 2005a).

3.1.2 Learning

The model can be trained on a dataset D of word sequences using maximum likelihood learning. The log-likelihood (assuming IID sequences) simplifies to

$$L = \sum_{w_{1:n} \in D} \log P(w_n|w_{1:n-1}), \qquad (3.6)$$

where the sum is over all word subsequences $w_1, ..., w_n$ of length n in the dataset. L can be maximized w.r.t. model parameters using gradient ascent. The contribution made by a subsequence $w_1, ..., w_n$ from D to the gradient of L w.r.t. a parameter θ is given by

$$\frac{\partial}{\partial\theta}\log P(w_n|w_{1:n-1}) = E_D\left[\frac{\partial}{\partial\theta}(-E(w_n,h,w_{1:n-1}))\right] - E_M\left[\frac{\partial}{\partial\theta}(-E(w_n,h,w_{1:n-1}))\right], \tag{3.7}$$

where $E_D[\cdot]$ and $E_M[\cdot]$ denote expectations w.r.t. the posterior distribution $P(h|w_{1:n})$ conditional on the observed subsequence and the model distribution $P(w_n,h|w_{1:n-1}))$ respectively. The derivation of Eq. 3.7 is completely analogous to the derivation of Eq. 1.9.

The energy function gradients w.r.t. the model parameters are:

$$\begin{aligned}\frac{\partial}{\partial R}(-E(w_n,h,w_{1:n-1})) &= \frac{\partial}{\partial R}\sum_{i=1}^{n} v_i^T R W_i h \\ &= \sum_{i=1}^{n} v_i h^T W_i^T, \qquad (3.8)\\ \frac{\partial}{\partial W_i}(-E(w_n,h,w_{1:n-1})) &= \frac{\partial}{\partial W_i} v_i^T R W_i h \\ &= R^T v_i h^T. \qquad (3.9)\end{aligned}$$

Plugging in them into Eq. 3.7 gives

$$\frac{\partial}{\partial R}\log P(w_n|w_{1:n-1}) = E_D\left[\sum_{i=1}^{n} v_i h^T W_i^T\right] - E_M\left[\sum_{i=1}^{n} v_i h^T W_i^T\right], \tag{3.10}$$

$$\frac{\partial}{\partial W_i}\log P(w_n|w_{1:n-1}) = E_D\left[R^T v_i h^T\right] - E_M\left[R^T v_i h^T\right], \tag{3.11}$$

The gradient of the log-likelihood of the dataset w.r.t. each parameter is then simply the sum of the contributions by all n-word subsequences in the dataset.

Computing these gradients exactly can be computationally expensive because computing an expectation w.r.t. $P(v_n,h|w_{1:n-1})$ takes $\Theta(N_w N_h)$ time for each context. One alternative is to approximate the expectation using a Monte Carlo method by generating samples from $P(v_n,h|w_{1:n-1})$ and averaging over them. Unfortunately, generating an exact sample from $P(v_n,h|w_{1:n-1})$ is just as expensive as computing the original expectation. Instead of using exact sampling, we can generate samples from the distribution using a Markov chain Monte Carlo method such as Gibbs sampling which involves initializing v_n and h to some random configuration and alternating between sampling v_n and h from their respective conditional distributions given by

$$P(w_n|h, w_{1:n-1}) \propto \exp\left(v_n^T R W_n h + v_n^T b_v\right), \tag{3.12}$$

$$P(h|w_{1:n}) \propto \exp\left(\left(\sum_{i=1}^{n} v_i^T R W_i + b_h^T\right) h\right). \tag{3.13}$$

However, a large number of alternating updates might have to be performed to obtain a single sample from the joint distribution.

Instead of performing maximum likelihood learning, we use an approximate learning procedure called Contrastive Divergence (CD) learning which is much more efficient. It is obtained by making two changes to the MCMC-based ML learning method. First, instead of initializing v_n to a random configuration, we initialize it to the state corresponding to w_n. Second, instead of running the Markov chain to convergence, we perform three alternating updates (first h, then v_n, and then h again). While the resulting configuration (v_n, h) is not a sample from $P(v_n, h|w_{1:n-1})$, it has been shown empirically that learning still works well when such configurations (which we will call *CD samples*) are used instead of samples from $P(v_n, h|w_{1:n-1})$ in the learning rules given above (Hinton, 2002).

In this chapter, we train all our RBM models using CD learning. In some cases, we use a version of CD that, instead of sampling v_n from $P(w_n|h, w_{1:n-1})$ to obtain a binary vector with a single 1 in w_nth position, sets v_n to the vector of probabilities given by $P(w_n|h, w_{1:n-1})$. This can be viewed as using mean-field updates for the visible units and stochastic updates for the hidden units, which is common practice when training RBMs (Hinton, 2002). Using these mean-field updates instead of stochastic ones reduces the noise in the parameter derivatives allowing larger learning rates to be used.

3.2 The temporal factored RBM

The FRBM language model proposed above, like most statistical language models, is based on the assumption that given its context, the word is conditionally independent of all other preceding words. This assumption, which is clearly false, is made in order keep the number of model parameters relatively small. In n-gram models, for example, the number of parameters is exponential in context size, which makes such models unable to take advantage of large contexts. While the dependence of the number of model parameters on context size is usually linear for models that use distributed representations for words, the context size in such models is still fixed and has to be chosen in advance.

Ideally, a language model should be able to take advantage of indefinitely large contexts without needing a very large number of parameters. We propose a simple extension to the FRBM language model to achieve that goal, following Sutskever and Hinton (2007). Suppose we want to predict word w_{t+n} from $w_1, ..., w_{t+n-1}$ for some large t. We can apply a separate instance of our model (with the same parameters) to words $w_\tau, ..., w_{\tau+n-1}$ for each τ in $\{1, ..., t\}$, obtaining a distributed representation of the i^{th} n-tuple of words in the hidden state h^τ of the τ^{th} instance of the model.

In order to propagate context information forward through the sequence towards the word we want to predict, we introduce directed connections from h^τ to $h^{\tau+1}$ and compute the hidden state of model $\tau + 1$ using the inputs from the hidden state of model τ as well as its visible units. By introducing the dependencies between the hidden states of successive instances and specifying these dependencies using a shared parameter matrix A we make the distribution of w_{t+n} under the model depend on all previous words in the sequence.

3.2.1 Making predictions

Exact inference in the resulting model is intractable – it takes time exponential in the number of hidden variables in the model being instantiated. However, since predicting the next word given its (nearly-)infinite context is an online problem we take the filtering approach to the task, which requires storing only the last $n-1$ words in the sequence along with the hidden state distribution of a single model at any given time.

Unfortunately, exact filtering is intractable in this model. Though the distribution of the hidden state at time τ is factorial given the observed sequence and the hidden state at time $\tau-1$, marginalizing over a factorial distribution for $h^{\tau-1}$ results in a non-factorial and thus intractable distribution for h^{τ}.

To get around this problem, we treat the hidden state h^{τ} as fixed at p^{τ} when inferring the distribution $P(h^{\tau+1}|w_{1:\tau+n})$, where $p_j^{\tau} = P(h_j^{\tau} = 1|w_{1:\tau+n-1})$. Then, given a sequence of words $w_1, ..., w_{t+n-1}$ we infer the posterior over the hidden states of model instances using the following recursive procedure. The posterior for the first model instance is given by Eq. 3.13. Given the (factorial) posterior for model instance τ, the posterior for model instance $\tau+1$ is computed as

$$P(h^{\tau+1}|w_{1:\tau+n}) \propto \exp\left(\left(\sum_{i=1}^{n} v_i^T R W_i + (b_h + Ap^{\tau})^T\right) h^{\tau+1}\right). \qquad (3.14)$$

Thus, computing the posterior for model instance $\tau+1$ amounts to adding Ap^{τ} to that model's vector of hidden unit biases and performing inference in the resulting model using Eq. 3.13.

Finally, the predictive distribution over w_n is computed by applying the FRBM prediction procedure from Sec. 3.1.1 to model instance t after shifting its biases appropriately.

3.2.2 Learning

Maximum likelihood learning in the temporal FRBM model is intractable because it requires exact inference. Since we would like to be able to train models with large numbers of hidden variables we have to resort to an approximate algorithm.

Instead of performing exact inference we simply apply the filtering algorithm from the previous section to compute the approximate posterior for the model. For each model instance the algorithm produces a vector which, when added to that model's vector of hidden unit biases, makes the model posterior be the posterior produced by the filtering operation. Then we compute the parameter updates for each model instance separately using the usual CD learning rule and average them over all instances.

The temporal parameters are updated using the following rule applied to each training sequence separately:

$$\Delta A \propto \sum_{\tau=1}^{t-1} \left(p^{\tau+1} - \hat{p}^{\tau+1}\right)^T p^{\tau}. \tag{3.15}$$

Here $\hat{p}_i^{\tau+1}$ is the probability of the hidden unit i being on in the CD samples produced by model instance $\tau + 1$. See (Sutskever and Hinton, 2007) for a more detailed description of learning in temporal RBMs.

3.3 The log-bilinear language model

Unlike the RBM-based language models, the log-bilinear (LBL) language model is a simple deterministic model. Its main advantages are its simplicity and faster training.

Like virtually all neural language models, the LBL model represents words by real-valued feature vectors. We will denote the feature vector for word w by r_w and refer to the matrix containing all these N_f-dimensional feature vectors as R. To predict the next word w_n given the context words $w_{1:n-1}$, the model computes the predicted representation $\hat{r}$ for the next word by linearly combining the context word feature vectors:

$$\hat{r} = \sum_{i=1}^{n-1} C_i r_{w_i}, \tag{3.16}$$

where C_i is the weight matrix associated with the context position i. Then the similarity between the predicted representation and the feature vector for each word in the vocabulary is computed using the inner product. The similarities are then exponentiated and normalized to obtain the distribution over the next word:

$$P(w_n = w|w_{1:n-1}) = \frac{\exp(\hat{r}^T r_w + b_w)}{\sum_j \exp(\hat{r}^T r_j + b_j)}. \tag{3.17}$$

Here b_w is the bias for word w, which is used to capture the context-independent word frequency.

Computing the probability of the next word using Eqs. 3.17 and 3.16 takes $\Theta(N_f N_w + (n-1)N_f^2)$ time. Since typically the vocabulary size is quite large compared to the feature dimensionality, we have $(n-1)N_f \ll N_w$, which reduces the above complexity down to $\Theta(N_f N_w)$.

3.3.1 Relationship to other models

The LBL model can be interpreted as a special kind of a feed-forward neural network with one linear hidden layer and a soft-max output layer. The inputs to the network are the feature vectors for the context words, while the matrix of weights from the hidden layer to the output layer is simply the feature vector matrix R. The vector of activities of the hidden units corresponds to the predicted representation for the next word. Note that unlike the NPLM, the LBL model has no nonlinearities in the hidden layer.

Alternatively, the LBL model can be viewed in the energy-based framework as a kind of an “RBM without hidden units” or a fully-observed Markov Random Field. The energy function for the model is given by

$$E(w_n, w_{1:n-1}) = -\sum_{i=1}^{n-1} v_i^T R C_i R^T v_n - b_v^T v_n. \tag{3.18}$$

Here C_i specifies the interaction between the feature vector of w_i and the feature vector of w_n, while b_v is the vector of per-word biases as in Eq. 3.2. Just like the energy function for the factored RBM, this energy function defines a bilinear interaction. However, in the FRBM energy function the interaction is between the word feature vectors and the hidden variables, whereas in this model the interaction is between the feature vectors for the context words and the feature vector for the predicted word.

The LBL model can also be interpreted as a product of experts, where expert i predicts w_n based only on w_i:

$$P_i(w_n; w_i) \propto \exp\left(v_i^T R C_i R^T v_n + \frac{1}{n-1} b_v^T v_n\right). \tag{3.19}$$

Multiplying $P_1, ..., P_{n-1}$ and renormalizing yields the LBL model. Combining experts using the product of experts architecture allows the resulting model to produce predictions that are more confident than those of any individual expert in cases when several experts agree (Hinton, 2002). Note that the mixture of experts architecture does not have this property and predictions produced by a mixture will always be less confident than those of any constituent expert. Mixed-order Markov models (Saul and Pereira, 1997) can be seen as the mixture-of-experts counterparts to the LBL-model, that use skip-bigrams as their experts.

Collobert and Weston (2008) introduced a neural language model designed for learning word representations suitable for solving various natural language processing tasks. The architecture of the model is similar to that of the NPLM, except that it has several hidden layers (i.e. it is a deep model). The model is trained to discriminate fixed-length sequences taken from the training set from their corruptions obtained by replacing the middle word by a randomly-chosen word. This model is much faster to train than the traditional neural language models because it outputs a single probability instead of a distribution over the entire vocabulary.

3.3.2 Learning

Training the above model is considerably simpler and faster than training the FRBM models because no stochastic hidden variables are involved. To derive the gradients needed for learning we will use the energy-based formulation with the model with the energy function

$$E(w_n, w_{1:n-1}) = -\sum_{i=1}^{n-1} r_{w_i}^T C_i r_{w_n} - b_{w_n}. \tag{3.20}$$

The gradient of the log-probability of w_n w.r.t. a parameter θ is given by

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log P(w_n|w_{1:n-1}) &= \frac{\partial}{\partial\theta}\log\frac{\exp(-E(w_n, w_{1:n-1}))}{\sum_w \exp(-E(w, w_{1:n-1}))} \\
&= \frac{\partial}{\partial\theta}(-E(w_n, w_{1:n-1})) - \frac{\partial}{\partial\theta}\log\sum_w \exp(-E(w, w_{1:n-1})) \\
&= \frac{\partial}{\partial\theta}(-E(w_n, w_{1:n-1})) - \frac{\sum_w \exp(-E(w, w_{1:n-1}))\frac{\partial}{\partial\theta}(-E(w, w_{1:n-1}))}{\sum_w \exp(-E(w, w_{1:n-1}))} \\
&= \frac{\partial}{\partial\theta}(-E(w_n, w_{1:n-1})) - \sum_w P(w|w_{1:n-1})\frac{\partial}{\partial\theta}(-E(w, w_{1:n-1})) \\
&= \frac{\partial}{\partial\theta}(-E(w_n, w_{1:n-1})) - E_M\left[\frac{\partial}{\partial\theta}(-E(w, w_{1:n-1}))\right], \qquad (3.21)
\end{aligned}$$

where $E_M[\cdot]$ denotes the expectation w.r.t. the model distribution $P(w_n|w_{1:n-1})$.

The gradients of the negative energy function we need are

$$\begin{aligned}
\frac{\partial}{\partial C_i}(-E(w_n, w_{1:n-1})) &= \frac{\partial}{\partial C_i}\sum_{i=1}^{n-1} r_{w_i}^T C_i r_{w_n} \\
&= r_{w_i} r_{w_n}^T, \qquad (3.22) \\
\frac{\partial}{\partial r_w}(-E(w_n, w_{1:n-1})) &= \frac{\partial}{\partial r_w}\sum_{i=1}^{n-1} r_{w_i}^T C_i r_{w_n} \\
&= \sum_{i=1}^{n-1}\left(\delta_{w,w_n} C_i^T r_{w_i} + \delta_{w,w_i} C_i r_{w_n}\right), \qquad (3.23)
\end{aligned}$$

where $\delta_{w,u}$ is the Kronecker delta function.

Plugging in the above gradients into Eq. 3.21 gives following gradients needed for learning:

$$\frac{\partial}{\partial C_i}\log P(w_n|w_{1:n-1}) = r_{w_i} r_{w_n}^T - E_M\left[r_{w_i} r_{w_n}^T\right], \tag{3.24}$$

$$\begin{aligned}
\frac{\partial}{\partial r_w}\log P(w_n|w_{1:n-1}) = &\sum_{i=1}^{n-1}\left(\delta_{w,w_n} C_i^T r_{w_i} + \delta_{w,w_i} C_i r_{w_n}\right) - \\
&E_M\left[\sum_{i=1}^{n-1}\left(\delta_{w,w_n} C_i^T r_{w_i} + \delta_{w,w_i} C_i r_{w_n}\right)\right]. \qquad (3.25)
\end{aligned}$$

3.4 Experimental evaluation

We evaluated our models using the 16 million word APNews dataset consisting of Associated Press news stories from 1995 and 1996. The dataset has been preprocessed by replacing proper nouns and rare words with the special "proper noun" and "unknown word" symbols respectively, and treating punctuation marks as words, resulting in a vocabulary containing 17964 words. A more detailed description of preprocessing can be found in (Bengio et al., 2003).

We performed our experiments in two stages. First, we compared the performance of our models to that of n-gram models on a smaller subset of the dataset to determine which type of our models showed the most promise. Then we performed a more thorough comparison between the models of that type and the n-gram models on the full dataset.

3.4.1 Preliminary evaluation

In the first experiment, we used a 10 million word training set, a 0.5 million word validation set, and a 0.5 million word test set. We trained one non-temporal and one temporal FRBM, as well as two log-bilinear models. All of our models used 100-dimensional feature vectors and both FRBM models had 1000 hidden units.

The models were trained using mini-batches of 1000 examples each. For the non-temporal models with $n = 3$, each training case consisted of a two-word context and a vector of probabilities specifying the distribution of the next word for this context. These probabilities were precomputed on the training set and stored in a sparse array. When training the non-temporal FRBM, v_n was initialized with the precomputed probability vector instead of the usual binary vector with a single 1 indicating the single "correct" next word for this instance of the context. The use of probability vectors as inputs along with mean field updates for the visible unit, as described in Sec. 3.1.2, allowed us to use relatively high learning rates. Since precomputing and storing the probability vectors for all 5-word contexts turned out to be non-trivial, the model with a context size of 5 was trained directly on 6-word sequences.

Mini-batches for training the temporal FRBM were obtained by splitting the dataset into 1000 sequences of equal length and assigning the i^{th} context / next word pair from each sequence to the i^{th} mini-batch. During training, mini-batches were processed in the natural order to allow the model to propagate context information forward through the sequences. Model parameters were updated after each mini-batch.

All the parameters of the non-temporal models except for word biases were initialized to small random values. Per-word bias parameters b_v were initialized based on word frequencies in the training set.

The temporal FRBM model was initialized by copying the parameters from the trained FRBM and initializing the temporal weights (A) to zero. During training, stochastic updates were used for visible units and v_n was always initialized to a binary vector representing the actual next word in the sequences (as opposed to a distribution over words).

We regularized our models using weight decay, implemented by adding $-\lambda w$ term to the derivative of the log-likelihood w.r.t. each parameter w. We used weight decay of $\lambda = 10^{-4}$ for word representations and $\lambda = 10^{-5}$ for all other weights. No weight decay was applied to biases. Weight decay values as well as other learning parameters were chosen using the validation set. Model performance was not sensitive to the weight decay setting as long as it was not very high. Each model was trained until its performance on a subset of the validation set stopped improving significantly. We did not observe overfitting in any of our models, which suggests that using models with more parameters might lead to improved performance.

We compared our models to backoff n-gram models estimated using Good-Turing and modified Kneser-Ney discounting (Chen and Goodman, 1996). Training and testing of the n-gram models was performed using programs from the SRI Language Modelling toolkit

Table 3.1: Perplexity scores for the models trained on the 10M word training set. The mixture test score is the perplexity obtained by averaging the model's predictions with those of the modified Kneser-Ney backoff 6-gram model. The first four models use 100-dimensional feature vectors. The FRBM models have 1000 stochastic hidden units. GTn and KNn refer to backoff n-grams with Good-Turing and modified Kneser-Ney discounting respectively.

Model type	Context size	Model test score	Mixture test score
FRBM	2	169.4	110.6
Temporal FRBM	2	127.3	95.6
Log-bilinear	2	132.9	102.2
Log-bilinear	5	124.7	96.5
Back-off GT3	2	135.3	
Back-off KN3	2	124.3	
Back-off GT6	5	124.4	
Back-off KN6	5	116.2	

(Stolcke, 2002). To make the comparison fair, the n-gram models treated punctuation marks (including full stops) as if they were ordinary words, since that is how they were treated by the neural models.

Models were compared based on their perplexity (see Sec. 2.3) on the test set. To make the comparison between models of different context size fair, given a test sequence of length L, we ignored the first C words and tested the models at predicting words $C + 1, ..., L$ in the sequence, where $C = 9$ was the largest context size among the models compared.

For each network model we also computed the perplexity for a mixture of that model with the best n-gram model (modified Kneser-Ney backoff 6-gram). The predictive distribution for the mixture was obtained simply by averaging the predictive distributions produced by the network and the n-gram model (giving them equal weight of 0.5). The resulting model perplexities are given in Table 3.1.

The results show that three of the four network models we tested are competitive with n-gram models. Only the non-temporal FRBM is significantly outperformed by all n-gram models. However, adding temporal connections to the FRBM to obtain a temporal FRBM, improves the model dramatically as indicated by a 33% drop in perplexity, suggesting that the temporal connections do increase the effective context size of the model. The log-bilinear models perform quite well: their scores are on par with Good-Turing n-grams with the same context size.

Averaging the predictions of any network model with the predictions of the best n-gram model produced better predictions than any single model, which suggests that the network and n-gram models make sufficiently different predictions in at least some cases. The best results were obtained by averaging with the temporal network model, resulting in 18% reduction in perplexity over the best n-gram model.

3.4.2 Final evaluation

Since the log-bilinear models performed well and were much faster to train than the FRBMs, we did not include FRBMs in our final evaluation. Similarly, we chose not to include the n-gram models with Good-Turing discounting as they performed significantly worse than the n-gram models with Kneser-Ney discounting. For this experiment we used the full APNews dataset which was split into a 14 million word training set, 1 million word validation set, and 1 million word test set. There was no overlap between the test set used in the preliminary evaluation and the test set used in the final evaluation. In fact, all data used in the preliminary evaluation was included in the final (14 million word) training set.

We trained two log-bilinear models: one with a context of size 5, the other with a

Table 3.2: Perplexity scores for the models trained on the 14M word training set. The log-bilinear models use 100-dimensional feature vectors. The mixture test score is the perplexity obtained by mixing the model's predictions with those of the Kneser-Ney 5-gram model using equal weights. The fitted mixture is obtained by mixing the two sets of predictions using the mixing proportion fitted on the validation set. The fitted mixing proportion column reports the mixing proportion of the neural model.

Model type	Context size	Model test score	Mixture test score	Fitted mixture test score	Fitted mixing proportion
Log-bilinear	5	117.0	97.2	96.9	0.57
Log-bilinear	10	107.8	92.2	91.4	0.62
Back-off KN3	2	129.8			
Back-off KN5	4	123.2			
Back-off KN6	5	123.5			
Back-off KN9	8	124.6			

context of size 10. The training parameters were the same as in the first experiment with the exception of the learning rate. After the validation perplexity of a model stopped improving when using the original learning rate of 10^{-2}, the learning rate was reduced to 10^{-3} and training was resumed until the validation perplexity stopped improving once again. This learning rate reduction was not performed in the preliminary evaluation, which might explain why the LBL models did not perform as well as the n-gram models in that experiment.

The results of the second experiment are summarized in Table 3.2. The perplexity scores clearly show that the log-bilinear models outperform the n-gram models. Even the log-bilinear model with the smaller context size (5) outperforms all the n-gram models. The results also show that increasing the context size of an n-gram model does not necessarily lead to better performance. In fact, the best n-gram performance on this dataset was achieved using a context of size 4. For the log-bilinear model, on the other hand, increasing the context size from 5 to 10 reduces perplexity by 8%.

In our second experiment, we used the same training, validation, and test sets as in (Bengio et al., 2003), which means that our results are directly comparable to theirs.[1] Bengio et al. (2003) trained a neural language model with a context size of 5 but did not report its individual test score. Instead they reported the score of 109 for the mixture of a Kneser-Ney 5-gram model and the neural language model. That score is significantly worse than the score of 97.2 obtained by the corresponding mixture in our experiments. Moreover, our best single model achieves the score of 107.8, outperforming their mixture.

So far we combined predictions of a neural model with those of an n-gram model by mixing them using the formula

$$P_{mix}(w_n|w_{1:n-1}) = \alpha P_{neural}(w_n|w_{1:n-1}) + (1-\alpha)P_{n-gram}(w_n|w_{1:n-1}) \tag{3.26}$$

with a fixed α of 0.5. Though using equal mixing proportions seems to work quite well, the mixture might perform better if we set the value of the mixing proportion α based on the models being combined. We use binary search to find the value of α that maximizes the validation log-likelihood for the mixture. The test set perplexity of the resulting mixtures as well as the values of α found are reported in Table 3.2. The results show that while fitting mixing proportions on the validation set does improve the test set perplexity, the improvement is quite small. In both fitted mixtures the LBL model contributes more than the n-gram model as indicated by the mixing proportion α being close to 0.6.

[1]Due to the limitations of the SRILM toolkit in dealing with very long strings, instead of treating the dataset as a single string, we broke up each of the test, training, and validations sets into 1000 strings of equal length. This might explain why the n-gram scores we obtained are slightly different from the scores reported in (Bengio et al., 2003).

Model performance analysis

It is well known that averaging predictions from a neural language model with those of an n-gram model usually results in better predictions than those of the individual models (Bengio et al., 2003; Schwenk, 2007). As the results in Tables 3.1 and 3.2 show, neural language models introduced in this thesis also benefit from such averaging. Such results suggest that neural models and n-gram models make different predictions that tend to complement each other.

In this section we investigate the ways in which the predictions from a neural language model differ from those of an n-gram model. We used two models from Table 3.2: the LBL model with a context of size 5 and the modified Kneser-Ney backoff 5-gram. Our analysis is based on the predictions made by these models on the 1M word test set.

First, we examined the distribution of the probability of the next word (i.e. $P(w_n|w_{1:n-1})$) on the test set under the two models. We partitioned the range of probability values into 8 bins, so that for $i = 1, ..., 7$, bin i contained the values between 10^{-i} and 10^{-i+1}, while bin 8 contained the values smaller than 10^{-7}. Thus, bin 1 contains the largest probability values, corresponding to the best predictions, while bin 8 contains the smallest probability values, corresponding to the least successful predictions. The top panel of Figure 3.2 shows the number of predictions contained in each bin for each model. The LBL model appears to produce more very successful (bin 1) as well as very unsuccessful predictions (bins 7 and 8) than the 5-gram model. The general trend is the same for both models though: more than half of the predictions fall into the first two bins while very few predictions fall into the last two bins. This suggests that both the LBL model and the 5-gram model assign a reasonably high probability to the next word in most cases. However, since perplexity is defined as $\mathcal{P} = \exp\left(-\frac{1}{N}\sum_{w_{1:n}} \log P(w_n|w_{1:n-1})\right)$, a less successful prediction contributes more to its value than the more successful one does. Thus it is instructive to look at the contribution to the negative log-probability of the test set (i.e. the sum of $-\log P(w_n|w_{1:n-1})$) by each of the bins. As can be seen from

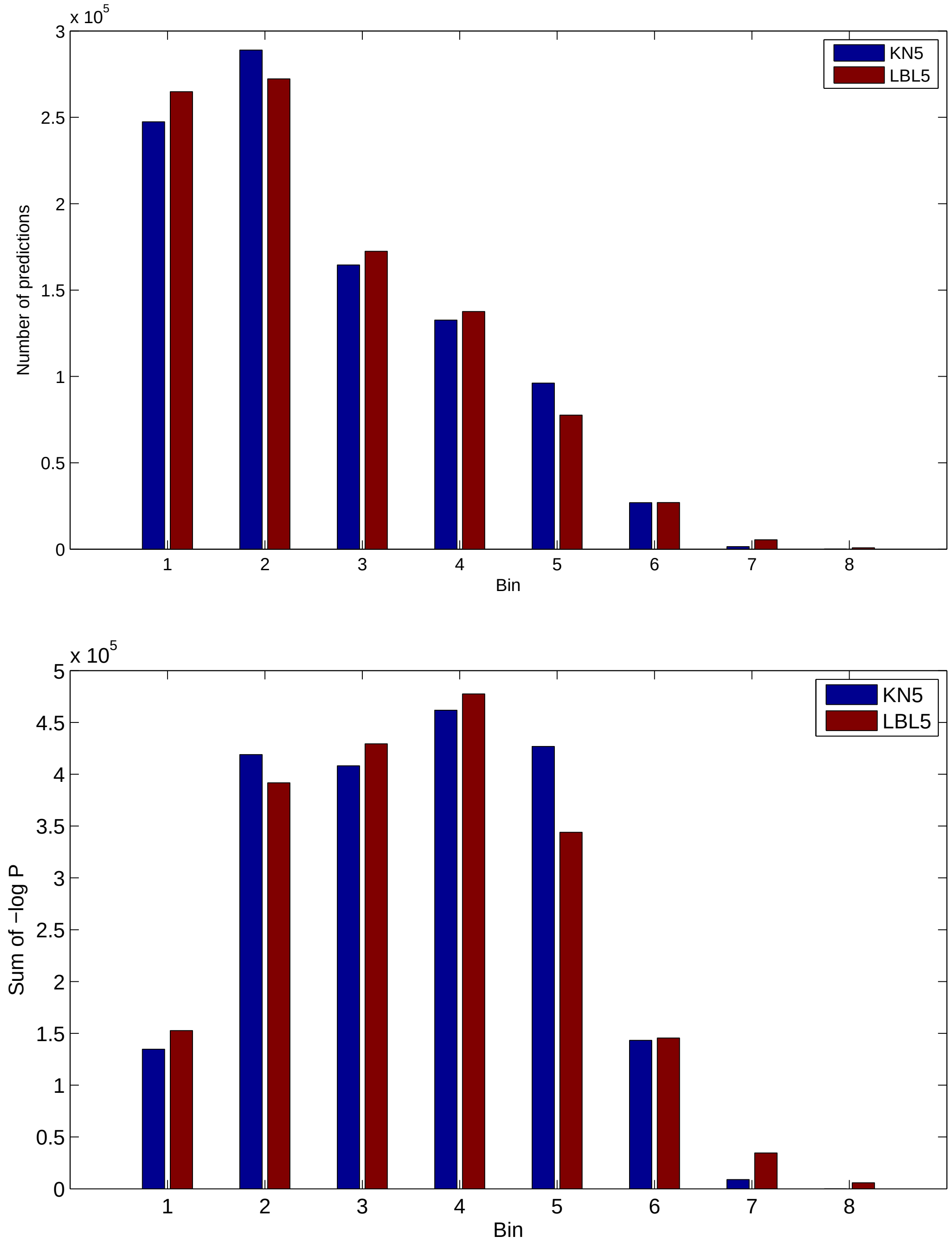

Figure 3.2: Top panel: Number of predictions $(P(w_n|w_{1:n-1}))$ on the test set as a function of their magnitude. Bottom panel: Contribution to the negative log-probability of the test set as a function of the prediction magnitude. Bin i (for $i = 1, ..., 7$) contains predictions between 10^{-i} and 10^{-i+1}. Bin 8 contains predictions smaller than 10^{-7}.

the bottom panel of Figure 3.2, bins 2-5 dominate the test set log-probability for both models. Thus the value of perplexity is mostly determined by mediocre predictions, with the most and the least successful predictions having little effect on it.

We then analyzed model performance as a function of the frequency of the word being predicted (i.e. the next word). We grouped the 17964 words in the vocabulary based on their training set frequency into 10 bins, so that each bin accounted for about 10% of the word occurrences in the training set. This vocabulary partitioning makes the per-bin results easier to interpret since the contribution of the words in each bin to the overall test set perplexity is roughly the same.[2] Note that the number of words in each bin is highly variable: the first bin contains 14199 (very rare) words, while the tenth bin contains only one word[3].

For each frequency bin, we computed its perplexity under both models by considering only those context / next word pairs in which the next word belonged to the frequency bin. To make evaluating the relative performance of the models easier, we also computed the ratio of the perplexity of the LBL model to that of the 5-gram model for each bin. The results of the evaluation are shown in Table 3.3.

As expected, our results show that both models are much better at predicting frequent words than rare words. However, as the per-bin perplexity ratio values indicate, the LBL model outperforms the 5-gram model on six of the ten bins. Interestingly, these bins are located at the extremes of the frequency range – they contain either the very rare (bin 1) or the very frequent words (bins 6-10). When predicting words of intermediate frequency the 5-gram model performs better than the LBL model.

We also analyzed model performance as a function of the frequency of the last word in the context, which is the word that immediately precedes the word we are trying to

[2]Note that the perplexity of the test set can be (approximately) computed by taking the *geometric* mean of the bin perplexities. This corresponds to taking the arithmetic mean of the per-bin log-probability values.

[3]This is the most frequent "word" in the dataset: ⟨proper_noun⟩.

Table 3.3: Model performance as a function of the frequency of the next word.

Bin number	Bin size (words)	Test set coverage (%)	LBL5 perplexity	KN5 perplexity	Perplexity ratio (PR)	$\log(PR)$
1	14199	10.1	11599.0	13041.0	0.89	-0.12
2	2428	10.0	2489.1	2336.3	1.07	0.06
3	870	10.1	937.2	819.7	1.14	0.13
4	340	9.8	394.7	349.8	1.13	0.12
5	92	9.7	125.2	120.6	1.04	0.04
6	22	10.3	30.2	32.0	0.94	-0.06
7	6	9.6	16.3	19.0	0.86	-0.15
8	4	12.4	10.6	12.8	0.83	-0.19
9	2	9.7	7.5	8.9	0.84	-0.18
10	1	8.2	8.8	10.5	0.84	-0.17

predict. We used the same frequency bin partitioning scheme as in the previous analysis. The results are shown in Table 3.4. For both models, conditioning on a context with a rare last word tends to result in a lower perplexity than conditioning on a context with a frequent last word, though the trend here is less pronounced than the one in the next word frequency vs. perplexity results. This result suggests that rare words are more informative than frequent words for predicting the word that follows them. The pattern of performance of the LBL model relative to the 5-gram model is very similar to the pattern from Table 3.3: the LBL model performs better when conditioning on very rare and very frequent words, while the 5-gram model performs better on words in the middle of the frequency range.

The superior performance of the LBL model when predicting or conditioning on rare words suggests that the model is able to learn usable 100-dimensional feature vectors even for words that occur only 30-332 times in the training set. More surprisingly,

Table 3.4: Model performance as a function of the frequency of the last context word.

Bin number	Bin size (words)	Test set coverage (%)	LBL5 perplexity	KN5 perplexity	Perplexity ratio (PR)	$\log(PR)$
1	14199	10.1	65.5	89.7	0.73	-0.32
2	2428	10.0	67.2	75.4	0.89	-0.12
3	870	10.1	62.9	63.5	0.99	-0.01
4	340	9.8	72.1	70.0	1.03	0.03
5	92	9.7	118.3	114.0	1.04	0.04
6	22	10.3	148.4	147.1	1.01	0.01
7	6	9.6	136.1	132.5	1.03	0.03
8	4	12.4	134.3	140.1	0.96	-0.04
9	2	9.7	315.7	342.3	0.92	-0.08
10	1	8.2	272.5	287.9	0.95	-0.05

the LBL model also outperforms the 5-gram model when dealing with the extremely frequent words (over 40,000 occurrences), which appears to indicate that the LBL model is sufficiently flexible to take advantage of large amounts of training data, in spite of being a parametric model.

To further investigate the strengths of both models we examined the context / next word pairs on which one of the models performed much better than the other. For every context / next word pair in the test set we computed the ratio of the probability of the next word under the LBL model to that of the 5-gram. Table 3.5 shows the cases with the largest value of the probability ratio, that is, the ones on which the LBL model performs disproportionately well, while Table 3.6 shows the cases that favour the 5-gram model. The results show that the 5-gram model excels at predicting the next word when it is a part of a frequently used word sequence (e.g. a compound proper noun) well represented in the training set. This is not at all surprising, since n-gram models effectively memorize the training set. In contrast, the LBL model does best relative to the 5-gram model when the context and the next word do not form a frequent pattern. In some of these cases, the context contains a part of a pattern that occurs frequently in the training set, but the next word does not fit that pattern, which causes the 5-gram to assign high probability to an incorrect answer. For example, the first example in Table 3.5, breaks a very frequent pattern because it does not have "a" between the monetary amount (#$n) and the next word ("troy_ounce"). Similarly, in the training set the context "unleaded , #n cents for" is always followed by "mid grade", which makes the 5-gram place a very low probability on "premium" being the next word. The LBL model, on the other hand, assigns a fairly high probability to "premium". In fact, given this context, the LBL model assigns fairly high probability (1×10^{-2} or greater) to words "mid", "premium", and "unleaded", which refer to gasoline types. In other cases, the LBL model outperforms the 5-gram model by generalizing across similar contexts and similar words. For example, though the phrase "houston symphony_orchestra" never occurs in the training set, the neural model is still able to assign "symphony_orchestra" a fairly high probability from the context "guest conductor with the houston" because "symphony_orchestra" is often preceded by a city name in the training set.

After seeing the LBL model perform poorly when predicting the next word in a frequently used expression, we inspected the distribution for the next word produced by the model after giving it a number of contexts with fairly unambiguous next words. Note that none of these contexts occur in the training set. We will illustrate the model behaviour using some of the contexts we looked at. The first context is "⟨proper_noun⟩ ⟨proper_noun⟩ , pontiac grand" was included in Table 3.6 as one of the contexts on which the 5-gram model does much better than the LBL model. The distribution for the next word produced by the LBL model shown in Table 3.7 includes only one plausible next word ("cherokee") in the top 20 words. The probability of the next word, which is "am" in this case, is very low (3×10^{-6}). It is not surprising that the LBL model fails in this

Table 3.5: The context / next word pairs from the test set most probable under the LBL model relative to the 5-gram model. The "Prob. ratio" column reports the ratio of the probability of the next word under the LBL model with a context of size 5 to that of a backoff 5-gram with modified Kneser-Ney smoothing. Contexts that start with "..." have been abbreviated to fit the table by replacing the first one or more words with "...".

Prob. ratio	LBL5 prob.	KN5 prob.	Context	Next word
3×10^6	8×10^{-3}	3×10^{-9}	silver traded_in london at #$n	troy_ounce
2×10^5	6×10^{-2}	3×10^{-7}	matched all five winning cards	drawn_in
8×10^4	1×10^{-1}	2×10^{-6}	: lotto #r . ⟨proper_noun⟩	supplementary
3×10^4	5×10^{-2}	2×10^{-6}	lotto #n #r . ⟨proper_noun⟩	supplementary
2×10^4	8×10^{-1}	3×10^{-5}	⟨unknown⟩ off the northwest african	coast
1×10^4	5×10^{-2}	4×10^{-6}	... committee chairman robert	livingston
1×10^4	6×10^{-2}	5×10^{-6}	unleaded , #n cents for	premium
1×10^4	7×10^{-1}	7×10^{-5}	... news_agency quoted fitzgerald	as_saying
1×10^4	3×10^{-2}	3×10^{-6}	guest conductor with the houston	symphony_orchestra
9×10^3	3×10^{-1}	3×10^{-5}	band of heavy showers with	thunderstorms
9×10^3	2×10^{-1}	2×10^{-5}	jurors deadlocked on two robbery	counts
8×10^3	5×10^{-1}	6×10^{-5}	- ⟨proper_noun⟩ ohio lottery will	pay_out
8×10^3	4×10^{-1}	5×10^{-5}	" the newspaper quoted webster	as_saying
7×10^3	9×10^{-3}	1×10^{-6}	the basque language acronym for	homeland
7×10^3	3×10^{-1}	4×10^{-5}	associate art professor at moscow	university
6×10^3	1×10^{-1}	2×10^{-5}	... american_stock_exchange 's	market_value
6×10^3	2×10^{-1}	3×10^{-5}	. ⟨proper_noun⟩ case is duncan	vs.
6×10^3	1×10^{-1}	2×10^{-5}	colleagues - moderates and liberals	alike
6×10^3	5×10^{-2}	9×10^{-6}	suddenly turned_on when the gas	went_off
6×10^3	3×10^{-2}	5×10^{-6}	most of ⟨proper_noun⟩ former french	colonies
5×10^3	9×10^{-2}	2×10^{-5}	, twelve , twenty-seven and	thirty-four

Table 3.6: The context / next word pairs from the test set most probable under the 5-gram model relative to the LBL-gram model. The "Prob. ratio" column reports the ratio of the probability of the next word under the LBL model with a context of size 5 to that of a backoff 5-gram with modified Kneser-Ney smoothing. Contexts that start with "..." have been abbreviated to fit the table by replacing the first one or more words with "...".

Prob. ratio	LBL5 prob.	KN5 prob.	Context	Next word
2×10^{-8}	6×10^{-14}	3×10^{-6}	latest weapon as peace talks	inch
2×10^{-7}	1×10^{-8}	6×10^{-2}	is #n . actor john	hurt
2×10^{-7}	5×10^{-8}	2×10^{-1}	⟨unknown⟩ , the chemical weapons	convention
4×10^{-7}	7×10^{-11}	2×10^{-4}	in #n , frederick r	law
5×10^{-7}	3×10^{-7}	7×10^{-1}	inappropriate for young children .	r
6×10^{-7}	2×10^{-7}	4×10^{-1}	assault charges , actor todd	bridges
6×10^{-7}	3×10^{-7}	5×10^{-1}	" ⟨proper_noun⟩ hunt for red	october
7×10^{-7}	4×10^{-7}	5×10^{-1}	such_as age , sex ,	region
2×10^{-6}	3×10^{-7}	2×10^{-1}	⟨proper_noun⟩ . #n . art	direction
2×10^{-6}	1×10^{-6}	7×10^{-1}	a #n) a #n	)
2×10^{-6}	5×10^{-9}	3×10^{-3}	's birthdays : actor victor	mature
2×10^{-6}	6×10^{-7}	3×10^{-1}	⟨proper_noun⟩ the benefit of the	doubt
2×10^{-6}	3×10^{-10}	1×10^{-4}	desert world , dependent_on an	off
2×10^{-6}	5×10^{-7}	2×10^{-1}	's jimmy page and robert	plant
3×10^{-6}	4×10^{-7}	1×10^{-1}	a basket of washington state	apples
3×10^{-6}	2×10^{-9}	5×10^{-4}	to return . ⟨proper_noun⟩ gen	guy
4×10^{-6}	3×10^{-9}	7×10^{-4}	research at wheat butcher &	singer
4×10^{-6}	3×10^{-6}	6×10^{-1}	... ⟨proper_noun⟩ , pontiac grand	am
4×10^{-6}	6×10^{-7}	1×10^{-1}	first come , first served	basis
5×10^{-6}	2×10^{-6}	4×10^{-1}	of #n . earth island	institute
5×10^{-6}	4×10^{-7}	8×10^{-2}	force , " " play	misty

case because the word “am” is used here as a part of a car name as opposed to its vastly more popular role as a form of the verb “to be”. Since neural models generally have only one feature vector per word, they can be expected to have difficulty dealing with words that have several very different usage patterns. However, if two quite different vectors are needed, their average can be much closer to both of them than to almost all other word vectors provided the vectors are high-dimensional.

However, the above example is rather atypical. The following two examples demonstrate a more typical situation. The next word distributions produced by the model when given contexts “a performance by the rolling” and “he has joined the rolling” are shown in Tables 3.8 and 3.9 respectively. In the first case, the correct next word (“stones”) is the fourth most probable prediction, while in the second case it is the most probable prediction. In both cases, the singular form of the word (“stone”) is also among the most probable words, which suggests that the model has learned similar representations for “stones” and “stone” from their similar usage patterns.

3.4.3 Visualization of word feature vectors

Our next step is to evaluate the distributed word representations learned by the LBL model. Since these feature vectors are high-dimensional (100D), it is infeasible to inspect them directly. Instead, we visualize them after reducing their dimensionality down to two dimensions using the recently developed t-SNE dimensionality reduction algorithm (van der Maaten and Hinton, 2008).[4] t-SNE is a nonlinear algorithm that reduces the dimensionality of the data while trying to preserve the local neighbourhood structure. Though we realize that some of the structure in the data will be lost or distorted by performing dimensionality reduction, we believe that analysis based on an imperfect condensation of the data is better than no analysis at all.

To evaluate the quality of the learned word feature vectors, we computed two embed-

[4]Perplexity, the only tunable parameter of the t-SNE algorithm, was set to 20 in our experiments.

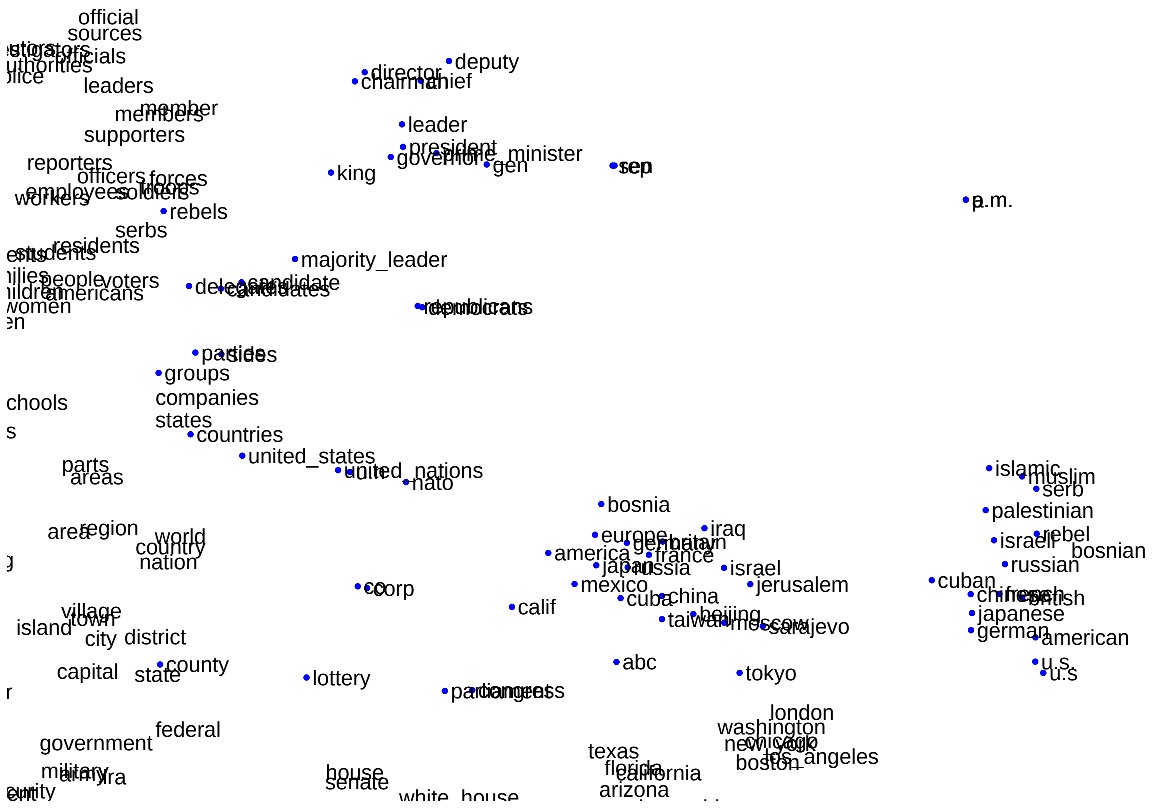

Figure 3.3: A fragment of a t-SNE embedding of the feature vectors (learned by an LBL model) of the most frequent 1000 words.

dings: one for the 1000 most frequent words and one for the 1000 rarest words. We used the word feature vectors from the LBL model with a context of size 5. Figures 3.3 and 3.4 show fragments of the two embeddings. The embedding of the top 1000 words exhibits strong local and mid-range structure. Days of the week, months, numbers, and various units of measurement form compact clusters, while geographic names and people's names form more diffuse clusters. Verbs and nouns tend to form large homogeneous regions. The quality of the embedding suggests that the feature vectors for the most frequent words capture word similarity very well.

The embedding of the least frequent 1000 words is less structured. However, the

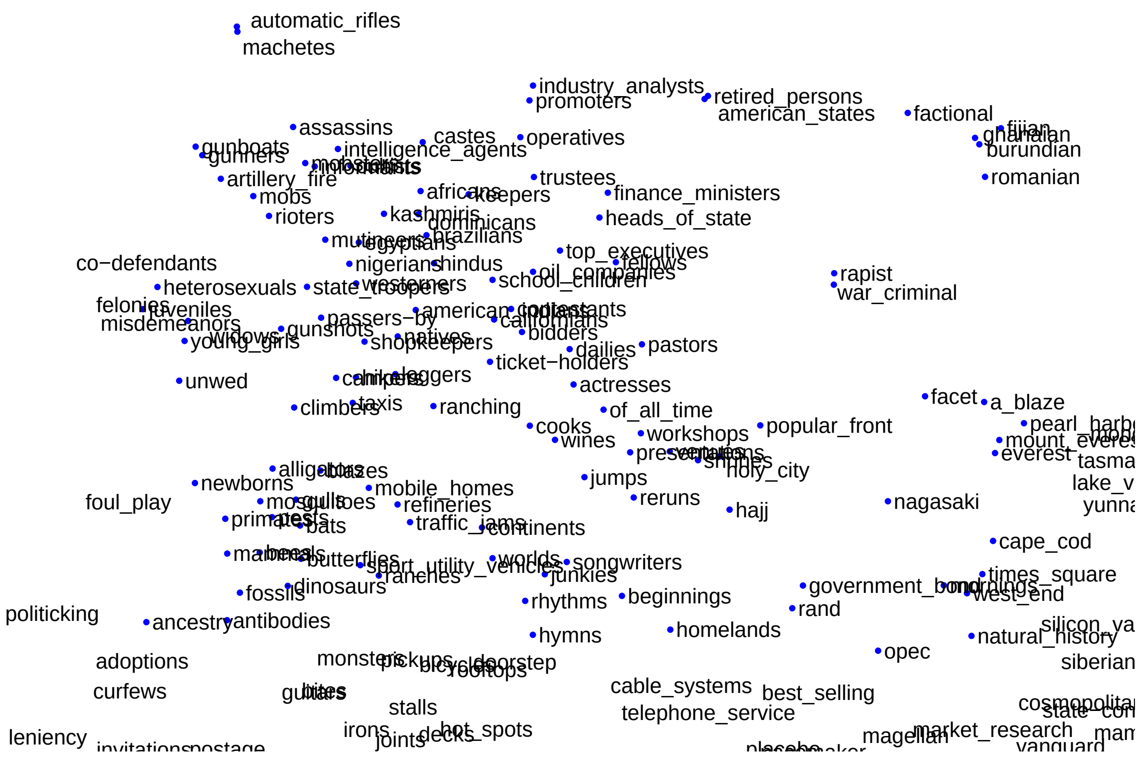

Figure 3.4: A fragment of a t-SNE embedding of the feature vectors (learned by an LBL model) of the least frequent 1000 words.

larger mostly homogeneous regions consisting predominantly of a single part of speech are still there, as are clusters of locations and names. The small clusters that were very prominent in the embedding of the frequent words are now less frequent and more diffuse. The loss of finer structure suggests that the feature vectors for rare words are not as well-estimated as those of their frequent counterparts. However, the amount of structure in the embedding suggests that even the feature vectors of the rarest words capture a fair amount of word similarity information.

Acknowledgements

We thank Yoshua Bengio for providing us with the preprocessed APNews dataset. An early version of this chapter has been published as (Mnih and Hinton, 2007).

Table 3.7: Most probable next words under the LBL model for the context "⟨proper_noun⟩ ⟨proper_noun⟩ , pontiac grand".

Rank	Probability	Next word
1	6.62×10^{-1}	⟨proper_noun⟩
2	4.23×10^{-2}	⟨unknown⟩
3	1.63×10^{-2}	forks
4	1.53×10^{-2}	,
5	7.97×10^{-3}	of
6	7.44×10^{-3}	and
7	7.01×10^{-3}	hall
8	5.60×10^{-3}	bell
9	5.17×10^{-3}	;
10	4.78×10^{-3}	cherokee
11	4.50×10^{-3}	grand
12	3.79×10^{-3}	kennedy
13	3.72×10^{-3}	for
14	3.40×10^{-3}	.
15	3.34×10^{-3}	city
16	3.15×10^{-3}	news
17	2.80×10^{-3}	silver
18	2.75×10^{-3}	bank
19	2.72×10^{-3}	campbell
20	2.66×10^{-3}	company

Table 3.8: Most probable next words under the LBL model for the context "a performance by the rolling".

Rank	Probability	Next word
1	7.85×10^{-2}	⟨unknown⟩
2	5.42×10^{-2}	⟨proper_noun⟩
3	2.75×10^{-2}	.
4	2.45×10^{-2}	stones
5	1.98×10^{-2}	stone
6	1.89×10^{-2}	,
7	1.72×10^{-2}	and
8	1.49×10^{-2}	on
9	1.48×10^{-2}	to
10	1.22×10^{-2}	west
11	1.04×10^{-2}	glass
12	9.32×10^{-3}	creek
13	9.29×10^{-3}	at
14	8.17×10^{-3}	house
15	8.01×10^{-3}	hills
16	7.87×10^{-3}	lake
17	7.81×10^{-3}	for
18	7.63×10^{-3}	valley
19	7.38×10^{-3}	river
20	7.27×10^{-3}	rock

Table 3.9: Most probable next words under the LBL model for the context "he has joined the rolling".

Rank	Probability	Next word
1	6.49×10^{-2}	stones
2	6.16×10^{-2}	on
3	4.76×10^{-2}	⟨unknown⟩
4	4.40×10^{-2}	⟨proper_noun⟩
5	3.85×10^{-2}	to
6	3.73×10^{-2}	stone
7	3.42×10^{-2}	at
8	2.56×10^{-2}	through
9	2.51×10^{-2}	in
10	2.32×10^{-2}	.
11	2.21×10^{-2}	for
12	2.19×10^{-2}	and
13	2.12×10^{-2}	across
14	1.97×10^{-2}	back
15	1.63×10^{-2}	hills
16	1.12×10^{-2}	by
17	1.11×10^{-2}	,
18	8.91×10^{-3}	around
19	8.09×10^{-3}	house
20	7.07×10^{-3}	west

Chapter 4

The hierarchical log-bilinear language model

In the previous chapter, we introduced the log-bilinear language model and showed that it can outperform n-gram models. Though its simplicity makes the LBL model faster to train and test than the more complex neural language models, training it can still take weeks on a single-core CPU.

To reduce the time complexity of training and making predictions with the NPLM[1], Morin and Bengio proposed a hierarchical language model built around a binary tree over words, which was two orders of magnitude faster than the non-hierarchical NPLM it was based on. However, it performed considerably worse than its non-hierarchical counterpart, in spite of using a word tree constructed using expert knowledge. In this chapter, we introduce a fast hierarchical language model, based on the log-bilinear model from the previous chapter, along with a simple feature-based algorithm for automatic construction of word trees from data. We then show that the resulting models can outperform non-hierarchical neural models as well as the best n-gram models.

[1]The Neural Probabilistic Language Model of Bengio et al. (2003) was described in Sec. 2.2.1.

4.1 The hierarchical neural probabilistic language model

The main drawback of neural language models, such as the NPLM and the LBL model, is that they are very slow to train and test (Morin and Bengio, 2005). Since computing the probability of the next word requires explicitly normalizing over all words in the vocabulary, the cost of computing the probability of a specific next word and the cost of computing the entire distribution for the next word are virtually the same – they take time linear in the vocabulary size. Since training such models requires repeatedly computing the probability of the next word given its context and updating the model parameters to increase that probability, training time is also linear in the vocabulary size. Typical natural language datasets have vocabularies containing tens of thousands of words, which means that training NPLM-like models the straightforward way is usually too computationally expensive in practice. One way to speed up the process is to use a specialized importance sampling procedure to approximate the gradients required for learning (Bengio and Senécal, 2003; Bengio and Sénécal, 2008). However, while this method can speed up training substantially, making predictions remains computationally expensive.

The hierarchical NPLM introduced by Morin and Bengio (2005) provides an exponential reduction in time complexity of learning and testing relative to the NPLM. It achieves this reduction by replacing the unstructured vocabulary of the NPLM by a binary tree that represents a hierarchical clustering of words in the vocabulary. The tree-based vocabulary decomposition is a generalization of the (word) class-based vocabulary decomposition proposed by Goodman (2001) as a way to speed up the training of maximum entropy language models. In a tree-based decomposition, each word corresponds to a leaf in the tree and can be uniquely specified by the path from the root to that leaf. If N is the number of words in the vocabulary and the tree is balanced, any word can be specified by a sequence of $\Theta(\log N)$ binary decisions indicating which of the two children of the current node is to be visited next. In this approach, one N-way decision is replaced by a sequence of $\Theta(\log N)$ binary decisions. In probabilistic terms, one N-way normalization is replaced by a sequence of $\Theta(\log N)$ local (binary) normalizations. As a result, a distribution over words in the vocabulary can be specified by providing the probability of visiting the left child at each of the nodes. In the hierarchical NPLM, these local probabilities are computed by applying a version of the NPLM to the feature vectors for the context words and the feature vector for the current node. The probability of the next word is then given by the probability of making the sequence of binary decisions that encodes to the path to that word.

When applied to a dataset of about one million words, this model outperformed class-based trigrams, but performed considerably worse than the NPLM (Morin and Bengio, 2005). The hierarchical NPLM however was more than two orders of magnitude faster than the NPLM. The main limitation of this work is the procedure used to construct the tree over words for the model. The tree was constructed by starting with the WordNet IS-A taxonomy and converting it into a binary tree through a combination of manual and data-driven processing. Our goal is to replace this procedure by an automated method for constructing trees from the training data without resorting to expert knowledge. We will also explore the benefits of using trees where each word can occur more than once.

4.2 The log-bilinear model

We will use the log-bilinear language model introduced in Sec. 3.3 as the foundation of our hierarchical model because of its simplicity and excellent performance. In this section, we will quickly review the LBL model. To compute the distribution for the word following the given context $w_{1:n-1}$, the model first computes the predicted feature vector $\hat{r}$ for the word by linearly combining the context word feature vectors using the context-position-dependent weight matrices $C_1, ..., C_{n-1}$:

$$\hat{r} = \sum_{i=1}^{n-1} C_i r_{w_i}. \tag{4.1}$$

Then the similarity between the predicted feature vector and the feature vector for each word in the vocabulary is computed using the inner product. The similarities are then exponentiated and normalized to obtain the distribution over the next word:

$$P(w_n = w|w_{1:n-1}) = \frac{\exp(\hat{r}^T r_w + b_w)}{\sum_j \exp(\hat{r}^T r_j + b_j)}. \tag{4.2}$$

Thus, the LBL model performs multinomial logistic regression to map the predicted word feature vector to a distribution over words in the vocabulary.

The probabilistic component of our hierarchical model will be used to model binary (left child / right child) decisions instead of the N-way decisions modeled by the LBL model. To achieve that we replace the softmax function used in Eq. 4.2 by the logistic function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{4.3}$$

4.3 The hierarchical log-bilinear model

Like the hierarchical NPLM, our hierarchical language model consists of two components: a binary tree over words and a probabilistic model of binary decisions for traversing the tree. The distinguishing features of our model are the use of the log-bilinear language model for computing the probabilities at each node and the ability to handle multiple occurrences of each word in the tree. The idea of using multiple word occurrences in a tree was proposed by Morin and Bengio (2005) but it was not implemented.

The first component of the hierarchical log-bilinear (HLBL) model is a binary tree with words at its leaves. For now, we will assume that each word in the vocabulary is at exactly one leaf. Then each word can be uniquely specified by a path from the root of the tree to the word's leaf node. The path itself can be encoded as a binary string d of decisions made at the nodes on the path, with $d_i = 1$ corresponding to the decision to visit the left child of the node being visited at time i. d_1 corresponds to the decision made at the root node. For example, the string "10" corresponds to a path that starts at the root, visits its left child, and then visits the right child of that child. This scheme allows each word to be represented by a binary string which we will call a *code*.

The second component of the HLBL model is the probabilistic model for making binary decisions at tree nodes, which in our case is a modified version of the LBL model. In the HLBL model, just like in its non-hierarchical counterpart, context words are represented using real-valued feature vectors. Each of the non-leaf nodes in the tree also has a feature vector associated with it for discriminating the words in the left subtree from the words in the right subtree of the node. Unlike the context words, the next word (i.e. the word to be predicted) is represented using its binary code that encodes its location in the word tree. Note that in spite of being discrete, this representation for the next word is quite flexible, since each binary digit in the code encodes a decision made at a node, which depends on that node's feature vector. The feature vector of a node specifies a hyperplane separating the feature vectors of words in its left subtree from those in its right subtree.[2]

Given a context, the probability of the next word being w is the probability of making the sequences of binary decisions specified by the word's code. Since the probability of making a decision at a node depends only on the predicted feature vector, determined by the context, and the feature vector for that node, we can express the probability of the next word as a product of probabilities of the binary decisions:

$$P(w_n = w | w_{1:n-1}) = \prod_i P(d_i | q_i, w_{1:n-1}), \tag{4.4}$$

where d_i is i^{th} digit in the code for word w, and q_i is the feature vector for the i^{th} node in the path corresponding to that code. The probability of each decision is given by

$$P(d_i = 1 | q_i, w_{1:n-1}) = \sigma(\hat{r}^T q_i + b_i), \tag{4.5}$$

where $\sigma(x)$ is the logistic function and $\hat{r}$ is the predicted feature vector computed using

[2]The representations referred to here are the *predicted* feature vectors produced by combining the context word feature vectors.

Eq. 4.1. b_i in the equation is the node's bias that captures the context-independent tendency to visit the left child when leaving this node.

The definition of $P(w_n = w|w_{1:n-1})$ can be extended to multiple codes per word by summing over all codes for w as follows:

$$P(w_n = w|w_{1:n-1}) = \sum_{d \in D(w)} \prod_i P(d_i|q_i, w_{1:n-1}), \tag{4.6}$$

where $D(w)$ is a set of codes corresponding to word w. Using multiple codes per word can allow better prediction of words that have multiple senses or multiple usage patterns. Allowing multiple codes per word also makes it easy to combine several trees over words into a single one to better capture the multifaceted nature of word similarity.

Using the LBL model instead of the NPLM for computing the local probabilities allows us to avoid computing the nonlinearities in the hidden layer which makes our hierarchical model faster at making predictions than the hierarchical NPLM. More importantly, the hierarchical NPLM needs to compute the hidden activities once for each of the $\Theta(\log N)$ decisions, while the HLBL model computes the predicted feature vector just once per prediction.

The quadratic time complexity ($\Theta(D^2)$, where D is the feature vector dimensionality) of computing the predicted feature vector in the HLBL model can make the use of high-dimensional feature vectors too computationally expensive. We make the time complexity linear in D by restricting the weight matrices C_i to be diagonal. Now the feature vector for the next word can be computed as

$$\hat{r} = \sum_{i=1}^{n-1} c_i \circ r_{w_i}, \tag{4.7}$$

where c_i is the weight *vector* for context position i and $\circ$ denotes the elementwise product of two vectors. Note that for a context of size 1, restricting the context weight matrices to be diagonal does not reduce the representational power of the model because the context weight matrix C_1 can be absorbed into the word feature vector matrix R. And while this restriction does make models with larger contexts slightly less powerful, we believe that this loss is more than compensated for by faster training times which allow using higher-dimensional feature vectors or more complex trees.

We train HLBL models by maximizing the penalized log-likelihood, with all model parameters regularized using an L_2 penalty. Since the probability of the next word depends only on the context weights, the feature vectors of the context words, and the feature vectors of the nodes on the paths from the root to the leaves containing the word in question, only a (logarithmically) small fraction of model parameters need to be updated for each training case.

4.4 Constructing trees over words

Before we can train a hierarchical language model, we need to construct a binary tree over words for it to use. This can be done by using data-driven methods, human expertise, or a combination of the two. For example, in (Morin and Bengio, 2005) the tree was constructed from the IS-A taxonomy DAG from WordNet (Fellbaum, 1998). After preprocessing the taxonomy by hand to ensure that each node had only one parent, data-driven hierarchical binary clustering was performed on the children of the nodes that had more than two children, resulting in a binary tree.

We are interested in using a pure learning approach applicable in situations where the expert knowledge is unavailable or inadequate. It should be noted that even when it is available, relying on human expertise exclusively does not necessarily lead to superior performance. Hierarchical binary clustering of words based on the their usage statistics is a natural choice for generating binary trees over words automatically. This task is similar to the task of clustering words into classes for training class-based n-gram models, for which a large number of algorithms have been proposed. We considered several of these algorithms before deciding to use our own algorithm which turned out to be surprisingly effective in spite of its simplicity. However, we will mention two existing algorithms that might be suitable for producing binary word hierarchies. Since we wanted an algorithm that scaled well to large vocabularies, we restricted our attention to the top-down hierarchical clustering algorithms, as they tend to have lower time complexity than their agglomerative counterparts (Goodman, 2000). The algorithm in (McMahon and Smith, 1996) produces exactly the kind of binary trees we need, except that its time complexity is cubic in the vocabulary size.[3] We also considered the distributional clustering algorithm (Pereira et al., 1993) but decided not to use it because of the difficulties involved in using contexts of more than one word for clustering. This problem is shared by most n-gram clustering algorithms, so we will describe it in some detail. Since we would like to cluster words for easy prediction of the next word based on its context, it is natural to describe each word in terms of the distribution of contexts that can precede it. The difficulty with this approach becomes apparent when we consider using larger contexts: the number of possible contexts is exponential in the context size. This is the very same data sparsity problem that affects the n-gram models, which is not surprising, since we are trying to describe words in terms of exponentially large (normalized) context count vectors. Thus, clustering words based on such large-context representations becomes non-trivial due to the computational cost involved as well as the statistical difficulties caused by the sparsity of the data.

We avoid these difficulties by operating on low-dimensional real-valued word representations in our tree-building procedure. Since we need to train a model to obtain word feature vectors, we perform the following bootstrapping procedure: we generate a *random* binary tree over words, train an HLBL model based on it, and then use the distributed representations it learns to compute the word representations used to construct the word tree. Though we could iterate the bootstrapping procedure and use the feature vectors learned by the model based on the constructed word tree to build another tree, we found

[3]More precisely, the time complexity of the algorithm is cubic in the number of the frequent words, but that is still too slow for our purposes.

that doing this resulted in only a slight improvement in model accuracy. As a result, in this thesis we perform the bootstrapping procedure only once.

Since each word is represented by a distribution over contexts it appears in, we need a way of compressing such a collection of contexts down to a low-dimensional vector. After training the HLBL model, we represent each context $w_{1:n-1}$ with the predicted feature vector produced from it using Eq. 4.1. Then, we map the distribution of contexts that precede a given word to a feature vector by computing the expectation of the predicted representation w.r.t. that distribution. Thus, for the purposes of clustering each word is represented by its average predicted feature vector.

Ideally, we would have preferred to cluster the next word feature vectors directly instead of clustering the average predicted feature vectors. However, since, unlike LBL, HLBL does not learn feature vectors for the next word, we have to derive a surrogate representation from the parameters the model does have, such as the context word feature vectors. Though it is possible to cluster the context word feature vectors learned by the model directly instead of using the average predicted feature vectors, we found that this approach does not work nearly as well. This is not very surprising because the context word representations capture the effect of the given context word on the next word (i.e. the word being predicted). It does not capture the information about the distribution of the contexts that can precede the given word, which is what we would like to cluster words based on. In other words, context word feature vectors are a poor replacement for next word feature vectors for building trees over words for HLBL.

After computing the low-dimensional real-valued feature vectors for words, we cluster them using a very simple recursive algorithm. At each step, we fit a mixture of two Gaussians to the feature vectors and then partition them into two subsets based on the responsibilities of the two mixture components for them. We then partition each of the subsets using the same procedure, and so on. The recursion stops when the current set contains a single word. We fit the mixtures by running the EM algorithm for 10 steps[4], updating both the means and the spherical covariances of the mixture components. Since the means of the components are initialized based on a random partitioning of the feature vectors, the algorithm is not deterministic and will produce slightly different clusterings on different runs. One appealing property of our hierarchical clustering algorithm is that each stage takes linear time in the vocabulary size, which is a consequence of representing words using feature vectors of fixed dimensionality. This property makes the algorithm suitable for handling very large vocabularies. In our experiments, the algorithm took only a few minutes to construct a tree for a vocabulary of nearly 18000 words based on 100-dimensional feature vectors.

In the implementation of the mixture fitting algorithm we used for all our experiments, mixture components do not share their covariance matrices. This can potentially lead to clusters in feature space that are optimally separated by quadratic decision boundaries that might be impossible to approximate accurately for the linear classifiers used at the tree nodes (see Eq. 4.5). Constraining the mixture components to use the same covariance matrix would eliminate this problem by ensuring that the optimal decision boundary between the classes is linear. However, enforcing this constraint explicitly did not seem to have any effect on the accuracy of the resulting models.

In order for hierarchical language models to generalize well and be fast to train and make predictions with, the trees over words used by the models should be well supported by the data and be reasonably balanced. To explore the trade-off between these two requirements, we explored several splitting rules in our tree-building algorithm. The rules are based on the observation that the responsibility of a mixture component for a datapoint can be used as a measure of confidence about the assignment of the datapoint to the component. Thus, when the responsibilities of both components for a datapoint

[4]Running EM for more than 10 steps did not result in a significant improvement in the quality of the resulting trees.

are close to 0.5, we cannot be sure that the datapoint should be in one component but not in the other.

Our simplest rule aims to produce a balanced tree at any cost. It sorts the responsibilities and splits the set of words into two disjoint subsets of equal size based on the sorted order. The second rule makes splits that are well-supported by the data even if that results in an unbalanced tree. It achieves that by assigning the word to the component with the higher responsibility for the word. The third and the most sophisticated rule is an extension of the second rule, modified to assign a point to both components whenever both responsibilities are within ϵ of 0.5, for some prespecified ϵ. This rule is designed to produce multiple codes for words that are difficult to cluster. We will refer to the algorithms that use these rules as BALANCED, ADAPTIVE, and ADAPTIVE(ϵ) respectively. Finally, as a baseline for comparison with the above algorithms, we will use an algorithm that generates random balanced trees. It starts with a random permutation of the words and recursively constructs the left subtree based on the first half of the words and the right subtree based on the second half of the words. We will refer to this algorithm as RANDOM.

4.5 Relationship to other models

As we have already seen in Sec. 4.3, the HLBL model is closely related to the hierarchical NPLM. However, there is another hierarchical neural language model that is related to both the hierarchical NPLM and the HLBL model. This model, introduced in (Blitzer et al., 2005b), is based on the hierarchical mixture of experts (HME) architecture (Jordan and Jacobs, 1994), and we will refer to it as the HMELM. As with most neural language models, context words in this model are represented using real-valued feature vectors. The distributed representation of the context is fed into a HME that is structured as a perfect binary tree. The leaf nodes of the tree store multinomial distributions over all words in the vocabulary, while the internal nodes are logistic regression models used to weight the contributions of the leaf experts. Since the distributions over words stored by the leaf nodes are context independent, both learning and making predictions in HMELM does not require the slow per-context normalization operation.

HMELM takes the idea of using multiple codes / leaves per word to the extreme. While HLBL models typically have a small number of codes per word, in HMELMs each word has as many codes as there are leaves in the HME tree. More importantly, since in HMELMs every word occurs at every leaf, all words share all these codes, so computing the probability of the given next word requires summing over all codes / leaves in the tree. This lack of structure in the next-word space makes it easy for HMELMs to overfit when large trees are used, a behaviour that was observed in (Blitzer et al., 2005b).

In terms of the number of codes per word, the hierarchical NPLM, with one code per word, and the HMELM, with many codes per word, define the two extremes of a spectrum, with the HLBL model located somewhere in between. The exact location of the HLBL model depends on the architecture of the tree over words that it is based on.

4.6 Experimental results

We evaluated our models on the 16 million word APNews dataset described in Sec. 3.4. The dataset consists of a 14 million word training set, a 1 million word validation set, and a 1 million word test set. The vocabulary size for this dataset is 17964.

Except where stated otherwise, the HLBL models used in the experiments had diagonal context weight matrices, 100-dimensional feature vectors and a context of size 5. All models were compared based on their perplexity score on the test set.

Table 4.1: Test set perplexity results for the HLBL models based on the trees generated using the RANDOM algorithm. LBL is the non-hierarchical log-bilinear model. The (H)LBL models have a context of size 5. KNn is an interpolated modified Kneser-Ney n-gram model. The perplexity value reported for each HLBL model is an average over four models based on different random trees.

Model type	Tree generating algorithm	Perplexity
HLBL	RANDOM	151.0
HLBL	RANDOM $\times 2$	140.3
HLBL	RANDOM $\times 4$	132.8
Int. KN2		174.2
Int. KN3		125.6
Int. KN6		119.2
LBL		117.0

4.6.1 Model training procedure

All of our neural models have been trained by maximizing the log-likelihood using stochastic gradient ascent. All model parameters other than the biases were initialized by sampling from a Gaussian of small variance. The biases for the tree nodes were initialized so that the distribution produced by the model with all the non-bias parameters set to zero matched the base rates of the words in the training set.

Models were trained using a learning rate of 10^{-3} until the perplexity on the validation set started to increase. Then the learning rate was reduced to 3×10^{-5} and training was resumed until the validation perplexity started increasing again. All model parameters were regularized using a small L_2 penalty.

4.6.2 Models based on random trees

In our first set of experiments, we evaluated the HLBL models that used random trees. In addition to random trees with one leaf per word, we also used 2× and 4× overcomplete trees that had two leaves per word and four leaves per word respectively. The 2× overcomplete trees were generated by running the RANDOM algorithm twice and combining the resulting trees by making them subtrees of a new root node. The 4× overcomplete trees were generated in an analogous manner. We generated four trees of each type (1×, 2×, and 4×) to find out whether different runs of the algorithm produce trees of significantly different quality. The test set perplexity results for these models are shown in Table 4.1. Since we trained four models per tree type, the perplexity values reported in the table are averages over the four models. There was little variability in the performance of models based on the same tree type, with the difference between the best and the worst model being under 1.7 perplexity points in all cases. Perplexity scores for interpolated modified Kneser-Ney n-grams are also reported in the table.

The results show that while HLBL models based on random trees perform reasonably well, they are not really competitive with Kneser-Ney n-gram models. Though the HLBL models use a context of size 5, they outperform only the Kneser-Ney bigram model. Using overcomplete trees clearly improves model performance, with the models based on 4× overcomplete tree being much closer to trigrams than bigrams in performance. However, none of the random-tree based models are competitive with the Kneser-Ney 6-gram or the LBL model. Thus, it is worthwhile to consider tree building algorithms more sophisticated than the RANDOM algorithm.

4.6.3 Models based on non-random trees

Next we explored the effect of the quality of the tree and the capacity of the binary decision model used on model performance. Since any distribution over leaf nodes in a

Table 4.2: The effect of the feature dimensionality and the word tree used on the test set perplexity of the model.

Feature dimensionality	Perplexity using a random tree	Perplexity using a non-random tree	Reduction in perplexity
25	191.6	162.4	29.2
50	166.4	141.7	24.7
75	156.4	134.8	21.6
100	151.2	131.3	19.9

tree can be obtained by setting the tree edge probabilities appropriately[5], for an infinitely powerful decision model trained on an infinitely large dataset the structure of the tree used does not matter. In practice, of course, both the model capacity and the training set size will be finite, so the quality of the tree is likely to be important.

To investigate the effect of tree quality on model performance, we compared the performance of models based on a random tree to that of models based on a more carefully constructed tree. The non-random tree was generated by running the BALANCED algorithm on the feature vectors obtained by training a random-tree based model with 100-dimensional feature vectors. For each of these trees, we then trained models of various feature vector dimensionality to see whether a more expressive probabilistic model can compensate for using a tree of lower quality. The test scores for the resulting models are given in Table 4.2. As can be seen from the scores, using a tree constructed based on learned feature vectors results in much better model performance than using a random tree. Though the performance gap can be reduced by increasing the dimensionality of feature vectors, using a feature-based tree drastically improves performance even for

[5]Supposed we are given a distribution over leaves in a tree. Let $S(n)$ be the sum of the probabilities of the leaves that are descendants of node n. Then for every node n, set the probability of choosing the left child to $\frac{S(left\ child\ of\ n)}{S(n)}$.

the model with 100-dimensional feature vectors. In fact, when 100-dimensional feature vectors are used, the model that uses a feature-based tree performs as well as a model based on a 4× overcomplete random tree. These results clearly show that the quality of the tree matters even when fairly powerful binary decision models are used. Since increasing the feature dimensionality beyond 100 did not result in a substantial reduction in perplexity, we used models with 100-dimensional feature vectors in all of the remaining experiments.

Next we compared the predictive performance of models that used trees generated by our more sophisticated tree-building algorithms, ADAPTIVE and ADAPTIVE(ϵ), to that of models based on trees generated by the simpler algorithms, RANDOM and BALANCED. To do that, we used the RANDOM, BALANCED, and ADAPTIVE algorithms to generate one tree each. The ADAPTIVE(ϵ) algorithm was used to generate two trees: one with ϵ set to 0.25 and the other with ϵ set to 0.4. Note that trees generated using ADAPTIVE(ϵ) using $\epsilon > 0$ result in models with more parameters due to the greater number of non-leaf nodes (and thus node feature vectors), as compared to trees generated using methods producing one leaf / code per word. We then generated a 2× overcomplete tree by running the ADAPTIVE(0.4) algorithm twice and creating a tree with a root node that had the two generated trees as its subtrees. Finally, we generated a 4× overcomplete tree using the same approach. Models based on overcomplete trees are more powerful than their non-overcomplete counterparts because they have more codes per word (and more parameters) to work with. However, it not clear that this extra capacity will necessarily translate to better test set performance.

Table 4.3 lists the generated trees along with some of their characteristics. Table 4.4 shows the test set scores for the resulting models along with the scores for the LBL model from Sec. 3.4.2 and the interpolated modified Kneser-Ney n-gram models. The results show that the performance of the HLBL models based on non-random trees is comparable to that of the Kneser-Ney trigram model. As expected, constructing word

Table 4.3: Trees over words used in the experiments. The mean code length is the sum of the lengths of the codes associated with a word, averaged over the word distribution in the training set. The run-time complexity of the hierarchical model is linear in the mean code length of the tree used. The mean number of codes per word refers to the number of codes per word averaged over the training data distribution. Since each non-leaf node in a tree has its own feature vector, the number of free parameters associated with the tree is linear in this quantity.

Tree label	Generating algorithm	Mean code length	Mean number of codes per word	Number of non-leaf nodes
T1	RANDOM	14.2	1.0	17963
T2	BALANCED	14.3	1.0	17963
T3	ADAPTIVE	16.1	1.0	17963
T4	ADAPTIVE(0.25)	24.2	1.3	22995
T5	ADAPTIVE(0.4)	29.0	1.7	30296
T6	ADAPTIVE(0.4) $\times$ 2	69.1	3.4	61014
T7	ADAPTIVE(0.4) $\times$ 4	143.2	6.8	121980

trees adaptively improves model performance. The general trend that emerges is that models based on larger trees tend to perform better. For example, a model based on a single tree produced using the ADAPTIVE(0.4) algorithm, performs better than the Kneser-Ney bigram model but not as well as the trigram model or the non-hierarchical LBL model. However, using a 2$\times$ overcomplete tree generated using the same algorithm results in a model that outperforms both the n-gram models and the LBL model, and using a 4$\times$ overcomplete tree leads to a further reduction in perplexity.

For each of the neural models in Table 4.4 we also report the test set perplexity for two mixtures of that model with the Kneser-Ney 6-gram. In the first mixture, the neural

Table 4.4: Test set perplexity results for the hierarchical LBL models. All (H)LBL models used 100-dimensional feature vectors and a context of size 5. LBL is the non-hierarchical log-bilinear model. KNn is an interpolated modified Kneser-Ney n-gram model. Mixture perplexity is the perplexity obtained by mixing the model's predictions with those of the Kneser-Ney 6-gram model using equal mixing proportions. The fitted mixture is obtained by mixing the two sets of predictions using the mixing proportion fitted on the validation set. The fitted mixing proportion column reports the mixing proportion for the neural model.

Model type	Tree used	Tree generating algorithm	Perplex.	Mixture perplex.	Fitted mixture perplex.	Fitted mixing proportion
HLBL	T1	RANDOM	151.2	107.2	106.0	0.34
HLBL	T2	BALANCED	131.3	99.9	99.7	0.43
HLBL	T3	ADAPTIVE	127.0	98.3	98.2	0.45
HLBL	T4	ADAPTIVE(0.25)	124.4	97.5	97.4	0.47
HLBL	T5	ADAPTIVE(0.4)	123.3	97.2	97.1	0.47
HLBL	T6	ADAPTIVE(0.4) $\times$ 2	115.7	95.3	95.3	0.52
HLBL	T7	ADAPTIVE(0.4) $\times$ 4	112.1	94.4	94.3	0.54
LBL			117.0	94.0	94.0	0.50
Int. KN2			174.2			
Int. KN3			125.6			
Int. KN6			119.2			

model and the 6-gram model are combined using equal mixing proportions (0.5). In the second mixture, the mixing proportion is chosen to optimize the validation set perplexity of the mixture as described in Sec. 3.4.2. In all cases, a mixture of a neural model and the 6-gram considerably outperforms the models being mixed. This suggests that the predictions of neural models differ significantly from the predictions of the 6-gram model. Though the HLBL model based on the ADAPTIVE(0.4) $\times$ 4 tree is the best-performing individual model in the table, its mixture with the 6-gram model is slightly outperformed by the mixture of the non-hierarchical LBL model with the 6-gram. The results also indicate that optimizing the mixing proportion on the validation set does not significantly improve the test test perplexity of the mixture. The fact that almost all of the optimized values of the mixing proportions are close to our default value of 0.5, suggests that the perplexity of the mixture is not very sensitive to the exact setting of the mixing proportion in that region. The general trend is that when the mixing proportions for a mixture of two models are fitted, the model with the lower perplexity gets the larger mixing proportion.

The time-per-epoch statistics reported for the neural models in Table 4.5 shows the great speed advantage of the HLBL models over the LBL model. Indeed, the slowest of our HLBL models is over 200 times faster than the LBL model.

4.6.4 The effect of diversity in overcomplete trees

As we have just seen, using overcomplete trees results in better model performance. In our previous experiments, we generated the overcomplete trees by combining several independently-generated non-overcomplete trees. Since all of our tree-building algorithms are stochastic to some degree, this approach constructed overcomplete trees from non-identical subtrees. To determine whether the use of non-identical subtrees is important, we evaluated five models based on $4\times$ overcomplete trees, with only one of the trees constructed from non-identical subtrees. That is, we generated four different subtrees S1, S2, S3, and S4 using the same algorithm and constructed five different $4\times$ overcomplete trees by combining these subtrees in different ways. The first four trees were obtained by combining four replicas of the same subtree Si. The fifth tree was obtained by combining all

Table 4.5: Training time per epoch for the hierarchical LBL models. All of the models used 100-dimensional feature vectors and a context of size 5. LBL is the non-hierarchical log-bilinear model.

Model type	Tree used	Tree generating algorithm	Minutes per epoch
HLBL	T1	RANDOM	4
HLBL	T2	BALANCED	4
HLBL	T3	ADAPTIVE	4
HLBL	T4	ADAPTIVE(0.25)	6
HLBL	T5	ADAPTIVE(0.4)	7
HLBL	T6	ADAPTIVE(0.4) $\times$ 2	16
HLBL	T7	ADAPTIVE(0.4) $\times$ 4	32
LBL			6420

four subtrees. We performed this procedure using the ADAPTIVE(0.4), RANDOM, and NOISY-ADAPTIVE(0.4) tree-building algorithms. NOISY-ADAPTIVE(ϵ) is a modification of the ADAPTIVE(ϵ) algorithm designed to produce more variable trees. When the responsibility p of a component for a word is more than ϵ away from 0.5, NOISY-ADAPTIVE(ϵ) assigns the word to that component with probability p and to the other component with probability $1-p$. The test set scores of the resulting models are shown in Table 4.6. For all three tree-building algorithms, the five models perform very similarly, which implies that the boost in model performance we have seen when using overcomplete trees is not due to the diversity of the subtrees being combined into a single tree. Instead, it appears to be a consequence of using a larger tree with each word occurring in multiple leaves and the significant increase in the number of tree node feature vectors that accompanies it.

Table 4.6: The effect of diversity in overcomplete trees. Four non-overcomplete subtrees (S1-S4) were generated using the same algorithm. Then five models were trained using five different 4$\times$ overcomplete trees. The first four trees were obtained by combining four replicas of the same subtree Si. The fifth tree was obtained by combining the four subtrees S1-S4. The procedure was performed using the ADAPTIVE(0.4), RANDOM, and NOISY-ADAPTIVE(0.4) tree-building algorithms.

Tree composition	ADAPTIVE(0.4) perplexity	RANDOM perplexity	NOISY-ADAPTIVE(0.4) perplexity
S1 $\times$4	112.6	134.0	111.8
S2 $\times$4	112.2	136.0	112.7
S3 $\times$4	112.3	134.9	113.1
S4 $\times$4	112.7	134.6	112.1
S1+S2+S3+S4	112.1	133.9	111.3

4.6.5 The effect of the context size

One important advantage of neural language models over n-gram models is their ability to use large contexts. As we have shown in Sec. 3.4.2, increasing the context size of an LBL language model from 5 to 10 significantly reduces the model's perplexity on the APNews dataset. To determine whether HLBL models behave in a similar manner, we trained six HLBL models based on the ADAPTIVE(0.4) $\times$ 4 overcomplete tree using context sizes from 1 to 20. The perplexity scores for those models along with the scores for interpolated modified Kneser-Ney n-gram models are shown in Table 4.7.

The scores indicate that the perplexity of HLBL models improves monotonically in context size (at least until the context becomes over twenty words long). In contrast, the best n-gram model perplexity is achieved by the 5-gram model and increasing the context size beyond that value increases perplexity slightly.

Table 4.7: Test set perplexity as a function of the context size. The HLBL models used a 4 $\times$ overcomplete tree generated using the ADAPTIVE(0.4) algorithm. KNn is an interpolated modified Kneser-Ney n-gram model.

Model type	Context size	Perplexity	Mixture perplex.	Fitted mixture perplex.
HLBL	1	170.5	115.9	112.8
HLBL	2	128.7	103.6	103.4
HLBL	3	119.4	98.7	98.7
HLBL	5	112.1	94.4	94.3
HLBL	10	104.3	89.6	89.2
HLBL	20	99.1	86.3	85.6
Int. KN2	1	174.2		
Int. KN3	2	125.6		
Int. KN6	5	119.2		
Int. KN8	7	120.2		

An additional benefit of neural language models such as HLBL is that using larger contexts does not significantly increase the time or space complexity of using such models, because computing the predicted vector from the context is only a small fraction of the computation. Thus, the advantages of using larger contexts in such models essentially come for free.

4.6.6 Bias-variance analysis

To study the effect of the choice of the training set on the resulting model, we performed a bias-variance analysis of the HLBL model based on the ADAPTIVE(0.4) $\times$ 4 tree and the interpolated Kneser-Ney 6-gram model.

In order to estimate bias and variance of a model, we need to compute expectations w.r.t. the distribution of training sets. Since we do not know this distribution, we approximate it by a collection of datasets derived from the original training set. We generated eight new training sets by subdividing the original training set into eight parts and leaving out one part at a time. On each of the new training sets, we then trained one model of each type. Bias and variance values were then estimated based on the predictions made by the resulting models on the test set. We used the same test set for this analysis as for all other experiments in this chapter.

Unfortunately, there is no standard bias-variance decomposition for the multinomial log-likelihood objective function. Here we will derive a simple decomposition which we believe is sufficient for our purposes. Let $P_i = P_i(w_n|w_{1:n-1})$ be the probability of the next word under the model trained on the i^{th} training set. Let N be the number of training sets used. Then the expected negative log-probability of the next word, averaged over the distribution of training sets, is

$$\begin{aligned}
E[-\log P_i] &= -\frac{1}{N}\sum_{i=1}^{N}\log P_i \\
&= \left(-\log\frac{1}{N}\sum_{i=1}^{N}P_i + \log\frac{1}{N}\sum_{i=1}^{N}P_i\right) - \frac{1}{N}\sum_{i=1}^{N}\log P_i \\
&= -\log\frac{1}{N}\sum_{i=1}^{N}P_i + \left(\log\frac{1}{N}\sum_{i=1}^{N}P_i - \log\prod_{i=1}^{N}P_i^{\frac{1}{N}}\right) \\
&= -\log\frac{1}{N}\sum_{i=1}^{N}P_i + \log\frac{\frac{1}{N}\sum_{i=1}^{N}P_i}{\prod_{i=1}^{N}P_i^{\frac{1}{N}}}. \qquad (4.8)
\end{aligned}$$

Since the arithmetic mean is always greater than or equal to the geometric mean, the second term in Eq. 4.8 in always non-negative. That term is also equal to zero if and only if all P_i's are equal. Based on these properties, we will associate this term with variance. The term $-\log\frac{1}{N}\sum_{i=1}^{N}P_i$ will be associated with bias.

Table 4.8: Bias and variance estimates for an HLBL model based on the ADAPTIVE(0.4)×4 tree and the interpolated Kneser-Ney 6-gram model.

Model	Bias	Variance
HLBL	4.629	0.111
Int. KN6	4.771	0.042

Based on this decomposition, we compute the estimates of bias and variance as follows:

$$Bias = -\frac{1}{K}\sum_{w_{1:n}} \log \frac{1}{N}\sum_{i=1}^{N} P_i(w_n|w_{1:n-1}), \tag{4.9}$$

$$Variance = \frac{1}{K}\sum_{w_{1:n}} \log \frac{\frac{1}{N}\sum_{i=1}^{N} P_i(w_n|w_{1:n-1})}{\prod_{i=1}^{N} P_i(w_n|w_{1:n-1})^{\frac{1}{N}}}, \tag{4.10}$$

where summation over $w_{1:n}$ denotes summing over all n-tuples in the test set and K is the number of such n-tuples. Note that under this definition, bias is equal to the logarithm of perplexity for the model that mixes the N predictors using equal mixing proportions.

The obtained bias and variance estimates shown in Table 4.8, indicate that the HLBL model has slightly lower bias and much higher variance than the Kneser-Ney 6-gram. This is somewhat surprising since the 6-gram model has many more parameters than the HLBL model, which might suggest that it has lower bias and higher variance. It appears that Kneser-Ney smoothing is very effective at regularizing the 6-gram model, greatly reducing its variance at the cost of increasing its bias.

Table 4.9 shows the perplexity scores for the eight models of each type trained on the subsets of the training set. The perplexity scores obtained by averaging the predictions from the eight models is also reported. Averaging predictions from models trained on the different subsets of the training set is much more effective at reducing perplexity for the HLBL model than for the 6-gram model. This provides further evidence that the HLBL model has higher variance than the 6-gram model.

Table 4.9: Test set perplexity for an HLBL model based on the ADAPTIVE(0.4)$\times$4 tree and the interpolated Kneser-Ney 6-gram model trained on subsets of the training set. The "combined" perplexity score was obtained using the average prediction for the eight models.

Training subset	HLBL test perplexity	6-gram test perplexity
1	115.1	124.7
2	114.6	123.5
3	114.1	122.8
4	113.9	122.7
5	114.0	122.8
6	114.1	123.2
7	114.2	122.5
8	114.7	122.3
Combined	102.4	118.0

4.6.7 Word-frequency based performance analysis

In this section we compare the predictive performance of an HLBL model to that of an LBL model, as a function of the frequency of the context words and the next word. Our goal is to see whether we give anything up (as far as predictive accuracy is concerned) when we replace the flat vocabulary of LBL models with a tree over words.

First, we compare the HLBL model based on the ADAPTIVE(0.4) $\times$ 4 tree, which is the best-performing model from Table 4.4, to the LBL model listed in the same table. Both of these models have contexts of size 5 and 100-dimensional feature vectors.

As in Sec. 3.4.2, we partition the context / next word pairs into ten bins based on the frequency of the next word so that each bin accounts for roughly 10% of the pairs in the

Table 4.10: Test set perplexity as a function of the frequency of the next word. The HLBL model is the ADAPTIVE(0.4) $\times$ 4 model from Table 4.4. The LBL model is the model with the context of size 5 from Table 3.2.

Bin number	Bin size (words)	Test set coverage (%)	HLBL perplexity	LBL perplexity	Perplexity ratio (PR)	$\log(PR)$
1	14199	10.1	14353.4	11599.0	1.24	0.21
2	2428	10.0	2612.4	2489.1	1.05	0.05
3	870	10.1	917.8	937.2	0.98	-0.02
4	340	9.8	371.8	394.7	0.94	-0.06
5	92	9.7	115.3	125.2	0.92	-0.08
6	22	10.3	27.0	30.2	0.89	-0.11
7	6	9.6	14.9	16.3	0.92	-0.08
8	4	12.4	9.8	10.6	0.92	-0.08
9	2	9.7	6.7	7.5	0.89	-0.11
10	1	8.2	7.7	8.8	0.88	-0.13

test set. We then compute the perplexity for the pairs in each bin under both models. We also compute the ratio of the perplexity under the HLBL model to the perplexity under the LBL model as well as the (natural) logarithm of that ratio (which we will call $\log(PR)$). The results shown in Table 4.10 indicate that the HLBL model outperforms the LBL model on eight of the ten bins – the ones that contain the more frequent words. It appears that the hierarchical model is at its weakest when predicting the very rare words.

To gain more insight into this phenomenon, we compared the above HLBL model (HLBL1) to the RANDOM $\times$ 4 HLBL model (HLBL2) from Table 4.1. Table 4.11 shows the result of this comparison. Though HLBL1 achieves lower perplexity than HLBL2 for each frequency bin, the gap in performance (as indicated by the perplexity ratio),

Table 4.11: Test set perplexity as a function of the frequency of the next word. HLBL1 is the ADAPTIVE(0.4) $\times$ 4 HLBL model from Table 4.4. HLBL2 is the RANDOM $\times$ 4 HLBL model from Table 4.1.

Bin number	Bin size (words)	Test set coverage (%)	HLBL1 perplexity	HLBL2 perplexity	Perplexity ratio (PR)	$\log(PR)$
1	14199	10.1	14353.4	23327.1	0.62	-0.49
2	2428	10.0	2612.4	3671.1	0.71	-0.34
3	870	10.1	917.8	1199.0	0.77	-0.27
4	340	9.8	371.8	450.3	0.83	-0.19
5	92	9.7	115.3	132.1	0.87	-0.14
6	22	10.3	27.0	29.3	0.92	-0.08
7	6	9.6	14.9	16.0	0.93	-0.07
8	4	12.4	9.8	10.3	0.95	-0.06
9	2	9.7	6.7	7.0	0.96	-0.04
10	1	8.2	7.7	7.8	0.99	-0.01

largest when predicting the least frequent words, becomes smaller as the frequency of the word being predicted increases. This trend suggests that the structure of the tree is much more important for predicting rare words. This is not surprising, since for frequent words there is much more training data available to learn the feature vectors of the nodes on the paths to those words. In contrast, there is very little training data available for each rare word, which makes sharing of statistical strength between neighbouring words in a tree much more important. And, of course, such sharing can only be beneficial if the neighbouring words occur in similar contexts or, in other words, the tree is well constructed. This suggests that in order for HLBL models to compete with LBL models when predicting rare words, better algorithms for tree construction need to be found.

We also analyzed model performance as a function of the frequency of the last context

Table 4.12: Test set perplexity as a function of the frequency of the last context word. The HLBL model is the ADAPTIVE(0.4) $\times$ 4 model from Table 4.4. The LBL model is the model with the context of size 5 from Table 3.2.

Bin number	Bin size (words)	Test set coverage (%)	HLBL perplexity	LBL perplexity	Perplexity ratio (PR)	$\log(PR)$
1	14199	10.1	68.4	65.5	1.04	0.04
2	2428	10.0	66.9	67.2	1.00	-0.00
3	870	10.1	60.5	62.9	0.96	-0.04
4	340	9.8	68.4	72.1	0.95	-0.05
5	92	9.7	112.3	118.3	0.95	-0.05
6	22	10.3	139.6	148.4	0.94	-0.06
7	6	9.6	127.4	136.1	0.94	-0.07
8	4	12.4	123.5	134.3	0.92	-0.08
9	2	9.7	299.9	315.7	0.95	-0.05
10	1	8.2	261.4	272.5	0.96	-0.04

word. As the perplexity scores in Table 4.12 show, the HLBL model achieves lower perplexity than the LBL model on all but one of the frequency bins. The superior performance of the LBL model on the bin containing the least frequent words might be a consequence of the use of the same set of feature vectors for both the context words and the next word. This parameter sharing scheme allows the feature vectors for rare words to be estimated more accurately. The HLBL model, on the other hand, does not use parameter sharing of this type because it uses real-valued feature vectors to represent the context words but not the next word. Thus context word feature vectors might be estimated less accurately in the HLBL model.

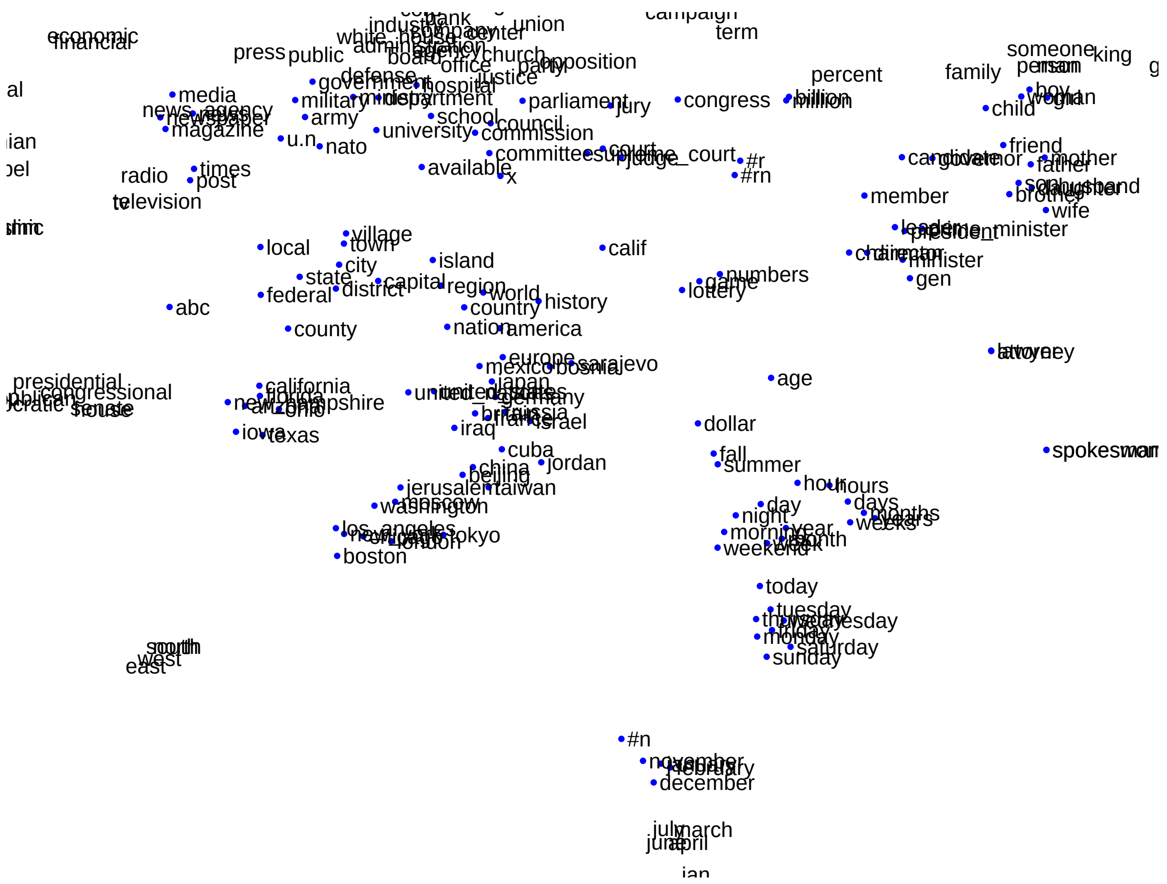

Figure 4.1: A fragment of a t-SNE embedding of the feature vectors (learned by an HLBL model) of the most frequent 1000 words.

4.6.8 Visualization of word feature vectors

Unlike LBL models which use the same set of feature vectors to represent the context words and the next word, only the context words are represented with feature vectors by HLBL models. The next word is represented using binary codes (that encode word location in the tree) along with the collection of feature vectors for the tree nodes. Since this representation of the next word does not naturally lend itself to embedding in a Euclidean space, we are going to visualize only the feature vectors for the context words.

To evaluate the quality of the learned word feature vectors, we computed two 2D

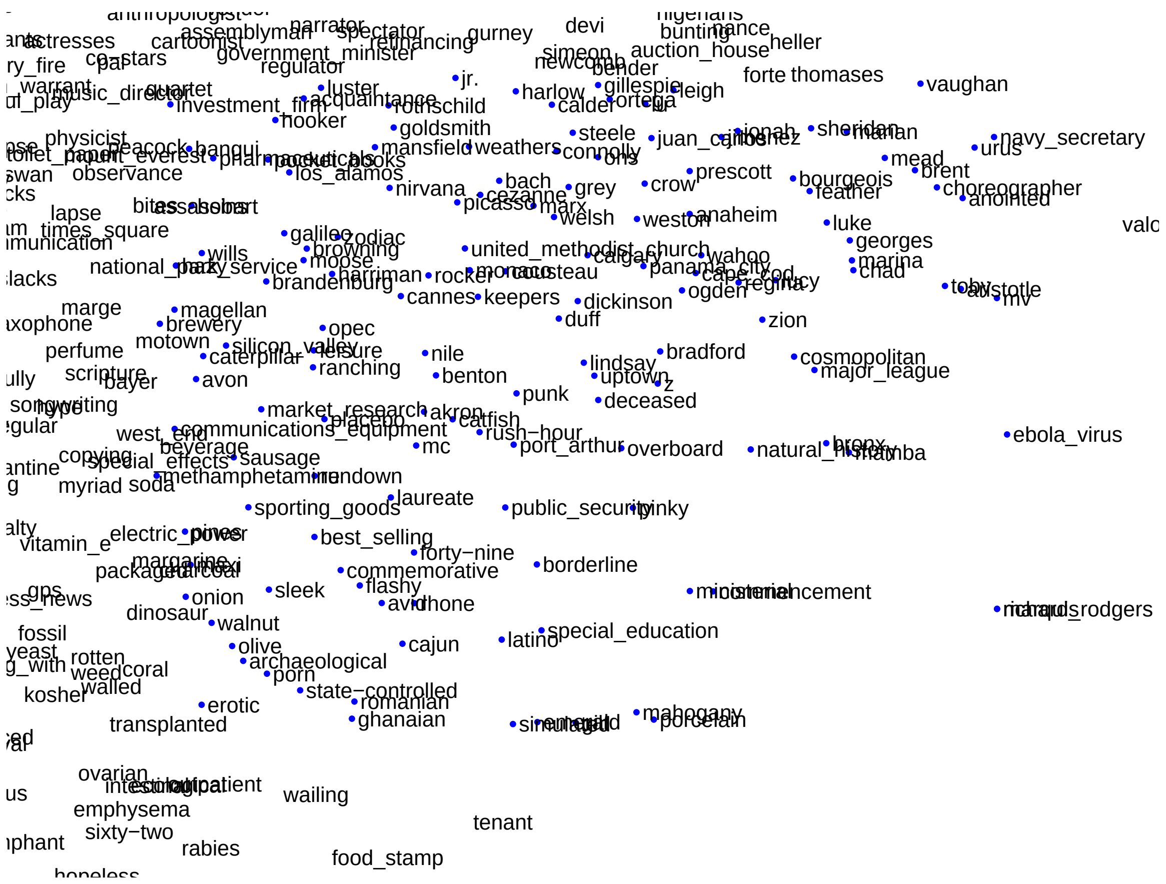

Figure 4.2: A fragment of a t-SNE embedding of the feature vectors (learned by an HLBL model) of the least frequent 1000 words.

embeddings: one of the 1000 most frequent words and one of the 1000 least frequent words. The feature vectors used were learned by the ADAPTIVE(0.4) × 4 HLBL model from Table 4.4. As can be seen from Figure 4.1, the embedding of the most frequent words is highly structured on both short-range and mid-range scales. It does not differ noticeably in quality from the corresponding embedding based on the feature vectors from an LBL model (Figure 3.3). However, the embedding of the least frequent 1000 words, shown in part on Figure 4.2, exhibits very little coherent structure, though sensible clusters of three or four words can still be found. This embedding is noticeably inferior to the embedding of the rare word feature vectors from the LBL model (Figure 3.4).

This provides evidence for our theory from the previous section that the feature vectors for the rare context words are estimated less accurately in HLBL models than in their non-hierarchical counterparts.

4.7 Discussion

Beygelzimer et al. (2009) have recently proposed an algorithm for estimating the probability of a label in multiclass classification in time logarithmic in the number of classes. Their approach can be seen as an application-agnostic online version of our approach to constructing trees. Their algorithm does not have a separate tree building phase: when a new label value is observed during training, it is inserted into the binary tree over labels in a way that tries to keep the tree balanced and make the classification problems for the binary classifiers at the internal tree nodes as easy as possible. It would be interesting to see how HLBL models trained using this algorithm would perform.

Another way of using a hierarchy over classes has been proposed by Shahbaba and Neal (2007). Here the hierarchy is used to introduce correlations between the weight vectors for different classes in a multinomial regression model. The authors show that their approach can produce more accurate models than the tree-based factorization approach (i.e. the one used by HLBL). We think that incorporating this kind of hierarchical parameterization into HLBL models is worth investigating, since it might improve their predictive performance through better regularization.

Acknowledgments

An early version of this chapter has been published as (Mnih and Hinton, 2009). We thank Martin Szummer for his comments on an earlier version of this chapter.

Chapter 5

Restricted Boltzmann machines and probabilistic matrix factorization for collaborative filtering

5.1 Introduction to collaborative filtering

5.1.1 Recommender systems: the two approaches

The number of products such as books and DVDs available to consumers online, which is already much larger than the number of such products available in brick-and-mortar stores, keeps increasing every month. This growth makes choosing products worth one's attention increasingly difficult. Recommender systems attempt to make the product selection process easier by filtering the list of products based on customer preferences.

There are two main approaches to recommender systems. The first approach, called content-based filtering, recommends items based on their content or description (Adomavicius and Tuzhilin, 2005). Thus items are represented by vectors of their intrinsic attributes, which for a book could be its title, author, publisher, genre, and so on. The drawback of this approach is that recommendation accuracy is heavily dependent on the comprehensiveness of item descriptions. In contrast, collaborative filtering (Marlin, 2004a) does not use any information about items other than the rating information. This means that items are treated as abstract entities and are represented using arbitrary item IDs. As a result, collaborative filtering systems cannot recommend items that have not been rated. The content-based systems do not suffer from this drawback, called the cold-start problem, because they take advantage of the intrinsic item attributes. On the other hand, content-based systems treat items with similar attributes as similar, while collaborative filtering systems can learn to view any pair of items as similar (or dissimilar) as necessary given a sufficient amount of training data.

In practice, because the strengths of the collaborative and context-based systems are complementary, very few systems are purely collaborative or purely content-based. In this thesis we will concentrate on (almost pure) collaborative filtering algorithms. This choice was motivated by the recent release of a very large dataset of movie ratings, associated with the Netflix Prize competition (Bennett and Lanning, 2007), which is about ten times larger than the previous largest publicly available dataset. The size of this dataset makes it feasible to consider more sophisticated collaborative filtering algorithms than was possible before, while at the same time making algorithm scalability an important design requirement.

5.1.2 Collaborative filtering

While the ultimate goal of recommender systems is recommending items to users, most collaborative filtering systems do not approach this task directly. Typically such systems implement the rating-based approach, where they first learn to predict ratings users assign to items and then make recommendations based on the predicted ratings (Adomavicius and Tuzhilin, 2005). The advantage of this approach is its simplicity, since it reduces the complex problem of recommending (or ranking) items to regression, which is a simpler and better-understood task.

Collaborative filtering algorithms are usually classified into two broad classes: memory-based algorithms and model-based algorithms (Breese et al., 1998). In machine learning terms, these two classes roughly correspond to non-probabilistic non-parametric algorithms and (semi-)parametric algorithms.

Memory-based algorithms

Memory-based algorithms predict the rating given to an item by a user by aggregating the ratings given by the same user to other items (or the ratings given to the item by other users). Such algorithms use a prespecified similarity metric to find some fixed number of items rated by the user that are most similar to the item of interest. The most popular similarity metrics for rating vectors are the Pearson correlation coefficient and the cosine-based similarity metric (Adomavicius and Tuzhilin, 2005). The drawback of memory-based algorithms is that they are almost entirely based on heuristics, which means that they require extensive hand-tuning on new datasets. They also have higher memory requirements than model-based algorithms because they need to have access to the entire dataset in order to make predictions.

Model-based algorithms

Model-based algorithms predict ratings using a parametric or semi-parametric model trained on the rating data. A large number of model types have been used in such algorithms. Mixture models (Breese et al., 1998), belief networks (Breese et al., 1998), the aspect model (Hofmann, 2001), the user rating profile model (Marlin, 2004b), and the multiple multiplicative factor model (Marlin and Zemel, 2004) are some of the probabilistic models employed. Low-rank matrix factorization (Srebro and Jaakkola, 2003; Srebro et al., 2004; Rennie and Srebro, 2005) is the most popular non-probabilistic model-based approach.

Collaborative filtering algorithms proposed in this thesis will follow the probabilistic model-based approach because it is based on a principled and flexible framework widely used in machine learning and applied statistics. In the following sections we will refer to items as movies because we will be presenting our models in the context of the Netflix Prize dataset.

5.2 Restricted Boltzmann machines

A set of ratings assigned to movies by users can be arranged into a matrix with rows corresponding to users and columns to movies. Most entries in this matrix will be missing (or unobserved) because typically users rate only a small fraction of the available movies. Collaborative filtering can be viewed as a matrix completion task, where the goal is to fill in the missing entries of the rating matrix based on the observed entries. In this thesis we will assume that ratings can take on integral values between 1 and K, with 1 being the lowest rating and K the highest.

One popular model-based approach to collaborative filtering is to define a latent variable model of users, where the variables encode user preferences (Canny, 2002; Marlin, 2004b; Marlin and Zemel, 2004). The appeal of this approach is that it provides fully generative models of (fully observed) user rating vectors. Since users are represented by vectors of latent variables, user profiles can be recomputed to take into account newly-observed ratings without having to retrain the entire model. Similarly, such models can handle new users without retraining by inferring the latent representation for them from the new ratings. Unfortunately, exact inference in almost all such models is intractable, so approximate inference methods, such as variational inference (Neal and Hinton, 1998; Jordan et al., 1999), have to be used. As a result, even when approximate methods are used, inference in such models is an iterative and fairly expensive process, which is not ideal for large-scale real-world applications. Moreover, in cases when variational inference is used there is the concern that its approximate nature might compromise the predictive performance of the model.

In contrast, exact inference in Restricted Boltzmann Machines (Sec. 1.3) is very efficient due to the combination of the model's bipartite structure and the use of undirected interactions. In fact, inferring the latent representation for an observed datavector in an RBM amounts to performing a multiplication of the input vector by a matrix followed by evaluating the logistic function for every component of the resulting vector. We propose a model for collaborative filtering based on the exponential-family Restricted Boltzmann Machine architecture (Welling et al., 2005) and demonstrate its scalability by applying it to the Netflix Prize dataset consisting of 100M movie ratings.

5.2.1 Model formulation

Ideally, ratings that take on ordered discrete values should be modelled using ordinal random variables. However, since to the best of our knowledge, no exponential family is ideal for modelling this type of random variables, we chose the simplest option which was to model ratings with multinomial random variables. This is a popular choice in the collaborative filtering literature used in mixture models (Breese et al., 1998), and various directed graphical models (e.g. Marlin, 2004b). The drawback of this approach is that while it captures the fact that rating values are discrete, it fails to capture the fact that they are ordered. However, we do not believe that this is a serious flaw, because given enough training data the model will be able to learn the correct ordering of rating values.

For notational convenience, we will represent ratings with binary vectors using one-hot encoding. Let e_i denote a vector of length K with component i set to 1 and all other components set to 0. Then rating i given to movie m by user v will be represented by $r_{vm} = e_i$. If movie m has not been rated by user v, r_{vm} is a vector of zeros. We will denote the matrix obtained by stacking these vectors by r_v.

Now we will define an RBM model with multinomial visible units and binary hidden units for modelling the joint distribution of ratings given to movies by a user. We start by specifying the joint energy function for the visible vector of observed ratings and the hidden variable vector:

$$E_v(r_v, h_v; \mathcal{R}_v) = -\sum_{m \in \mathcal{R}_v} r_{vm}^T (W_m h_v + b_m) - b_h^T h_v. \tag{5.1}$$

Here $\mathcal{R}_v$ is the set of movies rated by user v, W_m is the matrix of interaction weights between the rating for movie m and the hidden variables. b_h and b_m are the vectors of biases for the hidden units and for the rating values for movie m respectively. The joint distribution for r_v and h_v is then given by the Boltzmann distribution induced by the energy function:

$$P_v(r_v, h_v | \mathcal{R}_v) = \frac{\exp(-E_v(r_v, h_v; \mathcal{R}_v))}{\sum_{r'_v} \sum_{h_v} \exp(-E_v(r'_v, h_v; \mathcal{R}_v))}, \tag{5.2}$$

where the summation w.r.t. r'_v in the denominator is performed only over ratings for movies rated by the user.

The marginal distribution of user rating vectors is then obtained by marginalizing out the hidden variables:

$$P_v(r_v | \mathcal{R}_v) = \sum_{h_v} P_v(r_v, h_v | \mathcal{R}_v). \tag{5.3}$$

To simplify our notation, we will not explicitly show conditioning on $\mathcal{R}_v$ from now on, though it will always be implied.

Note that the energy function for a user rating vector does not depend on the entries in r_v that correspond to the movies not rated by user v. Similarly, normalization in Eq. 5.2 is performed only over the ratings for the rated movies. This means that for the given user, we model the distribution of ratings only for movies that have been rated by the user. Thus we learn one RBM per user under the constraint that the interaction weights $\{W_m\}$ and unit biases $(\{b_m\}, b_h)$ are shared among the RBMs. This parameter sharing is essential for avoiding overfitting, since we have only one datapoint per RBM. Now that we have explained the "family of RBMs" view of the model, from now on we will talk about it as if it were a single RBM.

The primary advantage of modelling ratings only for movies rated by the user is efficiency. Inference for a single user takes $\Theta(N_r N_h)$ time, where N_r is the number of movies rated by the user and N_h is the number of hidden units in the RBM. Modelling the distribution of ratings for all movies for every user makes the time complexity of inference linear in the number of movies in the dataset, which is typically orders of magnitude larger than the number of movies rated by an average user. Thus, modelling the distribution of the observed ratings only is essential for making the model scale to larger datasets.

Learning

Since, as explained in Sec. 1.3, computing the log-likelihood gradient for RBMs is intractable, we train our models using contrastive divergence learning. The CD update rule for the weights is given by

$$\Delta W_m = \epsilon \left(E_D \left[r_{vm} h_v^T \right] - E_T \left[r_{vm} h_v^T \right] \right), \tag{5.4}$$

where ϵ is the learning rate, and $E_D[\cdot]$ and $E_T[\cdot]$ denote expectations with respect to the posterior distribution $P_v(h|r_v)$ conditional on the observed ratings for the user and the distribution of T-step reconstructions respectively. The corresponding rules for updating the movie and hidden unit biases are similar:

$$\Delta b_m = \epsilon \left(E_D \left[r_{vm} \right] - E_T \left[r_{vm} \right] \right), \tag{5.5}$$

$$\Delta b_h = \epsilon \left(E_D \left[h_v \right] - E_T \left[h_v \right] \right). \tag{5.6}$$

Our models were trained by using the above updates in the online learning setting. That is, we iterated through the users in the training set, updating model parameters based on the ratings of the current user.

The expectations with respect to the distribution of T-step reconstructions were computed by running the Gibbs sampler for T steps, starting at the data, and then averaging the quantity of interest over the resulting reconstructions. The conditional distributions used by the Gibbs sampler are given by

$$P_v(r_{vm} = e_i|h) = \frac{\exp(e_i^T(b_m + W_m h_v))}{\sum_j \exp(e_j^T(b_m + W_m h_v))}, \tag{5.7}$$

$$P_v(h_i = 1|r_v) = \sigma\left(b_{hi} + \sum_{m \in \mathcal{R}_v} r_{vm}^T W_{mi}\right), \tag{5.8}$$

where $\sigma(x)$ is the logistic function and W_{mi} is the i^{th} column of matrix W_m.

Since the distribution over the hidden unit states in an RBM is factorial, inference in an RBM amounts to computing $P_v(h_i = 1|r_v)$ for each hidden unit using Eq. 5.8.

Making predictions

The downside of modelling ratings for rated movies only is that, strictly speaking, we cannot predict user ratings for movies that were not rated. We considered two approaches to solving this problem. One approach would be to treat all movies we would like to predict ratings for as rated with a missing rating.[1] Then during learning we would treat these ratings as proper latent variables and marginalize them out. Since exact marginalization takes time exponential in the number of missing observations, we would sample the values of the missing ratings to approximate the expectation of interest. The drawback of this approach is that we need to know the movies of interest for each user before training the model. Moreover, if the number of such movies is large, training time might increase significantly.

We take an alternative approach, where we train the RBMs as if we were only interested in modelling the observed ratings. Then to predict ratings for movies the given user RBM was not trained on, we expand the RBM to include the visible units corresponding

[1]This is a sensible approach for the Netflix dataset because we know that the ratings for movies we would like to predict come from roughly the same distribution as the movies we train on. However, this approach is unlikely to work well when the two distributions are different (see (Marlin and Zemel, 2007)).

to the movies of interest.[2]

Before we derive the procedure for predicting ratings, we need to extend our notation. Let $r_{v\backslash m}$ be the rating matrix r_v with all the entries in the row corresponding to r_{vm} set to zero (as if movie m were not rated by v). Then we can compute the distribution of rating r_{vm} as follows:

$$\begin{aligned} P_v(r_{vm} = e_i | r_{v\backslash m}) &\propto \sum_h P_v(r_{vm} = e_i, r_{v\backslash m}, h) = \sum_h \exp(-E_v(r_{vm} = e_i, r_{v\backslash m}, h)) \\ &\propto \exp(e_i^T b_m) \prod_{j=1}^{N_h} \sum_{h_j=0}^{1} \exp\left(\sum_{l \in \mathcal{R}_{v\backslash m}} r_{vl}^T W_l h_{vj} + e_i^T W_{mj} h_{vj} + b_{hj} h_{vj} \right) \\ &= \exp(b_{mi}) \prod_{j=1}^{N_h} \left(1 + \exp\left(\sum_{l \in \mathcal{R}_{v\backslash m}} r_{vl}^T W_l + e_i^T W_{mj} + b_{hj} \right) \right) \end{aligned} \tag{5.9}$$

Normalizing the above expression over the K possible rating values gives the distribution for the rating of interest. However, computing the distribution over n ratings $r_{vm_1}, ..., r_{vm_n}$ exactly would requires computing $P_v(r_{vm} | r_{v\backslash m_1,...,m_n})$ for K^n possible rating combinations, which is infeasible for all but the smallest values of n. If we assume that the ratings to be predicted are conditionally independent given the observed ratings, we can compute the distribution for each rating separately using Eq. 5.9. Though the time complexity of predicting n ratings is now linear in n, we still need to compute the quantity in Eq. 5.9 K times for each prediction.

We propose a faster way of making predictions based on the mean field update equations. This approximate method is based on the assumption that the hidden unit probabilities are not significantly affected by the value of a single rating. We expect this method to perform well as long as the number of observed ratings is not very small. For any number of ratings to predict, we first compute the probability of being on for each

[2]The (shared) weights to those units have been learned by the RBMs that had those movies in their training data.

hidden unit:

$$p_i = \sigma\left(b_{hi} + \sum_{m \in \mathcal{R}_v} r_{vm}^T W_{mi}\right). \tag{5.10}$$

Then for each rating to be predicted, we compute its distribution based on the hidden units probabilities:

$$P_v(r_{vm} = e_i|p) = \frac{\exp(e_i^T(b_m + W_m p))}{\sum_j \exp(e_j^T(b_m + W_m p))}, \tag{5.11}$$

where p is the vector of probabilities computed using Eq. 5.10. Since p has to be computed only once per user, this approach leads to considerable computational savings.

In this thesis we will evaluate our collaborative filtering models based on the root mean squared error (RMSE) of their predictions. We chose to use RMSE because it is the standard evaluation metric on the Netflix Prize dataset. Since the mean of a random variable is the optimal estimator for it under the RMSE metric, once we have computed the distribution for the rating we want to predict, we take the mean of the distribution as our prediction.

5.2.2 Conditional RBM

Sometimes we know that certain movies have been rated by a user even though we do not know the actual rating values. Alternatively, we might know all movies rented by a user who entered ratings only for some of the movies. This type of information should be useful for inferring user preferences because knowing what movies someone has rented (or rated) can tell us quite a bit about their taste in movies. For example, someone who rented the third movie in a trilogy most likely did not hate the first two movies. We can take this kind of implicit preference information into account by conditioning on it and thus modelling the distribution of user ratings given the identities of movies rated by the user.

The particular parameterization for the multinomial visible units used by the RBM allows it to detect whether the rating for a particular movie is among the observed ratings. This can be seen by comparing the parameterization of $P_v(r_{vm}|h)$ and $P_v(h|r_v)$ given by Eqs. 5.7 and 5.8 respectively. Note that adding the same value w to the K weights from a hidden unit to the K possible rating values for movie m does not affect $P_v(r_{vm} = e_i|h)$ because both the numerator and the denominator are multiplied by the same value. In other words, the conditional distribution for the visible units given the hidden units is overparameterized. However, as can be seen from Eq. 5.8, adding w to those weights does have an effect on the hidden unit involved, as its total input increases by w as long as the rating r_v is observed. Decreasing the bias for the hidden unit by w in addition to shifting the weights produces an RBM that can detect whether the rating for movie m is present in its input. Thus, in theory, an RBM might be able to learn to take advantage of the rating presence / absence information encoded in the user rating vectors in the training set. In practice, however, this ability is likely to be compromised by regularization applied to the weights, which would encourage the weights to be small in magnitude. Moreover, even if the regularization effect were to be disregarded, the RBM would not be able to take advantage of the rated information[3] for movies with unknown ratings.

We propose conditioning on the identities of movies rated by the user by making the hidden unit biases of the user RBM a linear function of the rated indicator vector. Let f_v be a binary vector with entries encoding the identities of the movies rated by user v.[4] The vector of hidden biases for the user RBM is then given by

$$b'_v = b_h + U f_v, \tag{5.12}$$

where U is the matrix of conditioning weights shared among all the user RBMs. The resulting conditional RBM models the joint distribution of r_v and h conditional on f_v: $P_v(r_v, h|f_v)$.

[3]In this thesis *rated information* refers to the identities of the movies rated by a user. This information does not include the actual rating values.

[4]That is the i^{th} entry of f_v is set to 1 if and only if the user rated movie i.

The conditional distributions needed for Gibbs sampling in this model are:

$$P_v(r_{vm} = e_i | h, f_v) = \frac{\exp(e_i^T(b_m + W_m h_v))}{\sum_j \exp(e_j^T(b_m + W_m h_v))}, \tag{5.13}$$

$$P_v(h_i = 1 | r_v, f_v) = \sigma\left(b_{hi} + U_i f_v + r_{vm}^T W_{mi}\right), \tag{5.14}$$

where U_i is the i^{th} row of the conditioning matrix U.

We learn U using a rule similar to the one for learning W:

$$\Delta U = \epsilon\left(E_D\left[h_v f_v^T\right] - E_T\left[h_v f_v^T\right]\right). \tag{5.15}$$

The rules for learning other parameters of the model are the same as for the original RBM model.

Though we used a linear function to map f_v to b'_v, the mapping from the rated indicators to the biases of the hidden units does not have to be linear. For example, a feed-forward neural network can be used to map f_v to b'_v. The parameters of the network can then be learned by backpropagating the gradient estimator given by $E_D[h_v] - E_T[h_v]$.

5.3 Probabilistic matrix factorization

Latent variable user models, already mentioned in Sec. 5.2, are based on the assumption that rating preferences of a user are determined by a small number of unobserved factors. Such models (e.g. Hofmann, 1999; Marlin, 2004b; Marlin and Zemel, 2004) can be viewed as graphical models with directed connections from marginally independent user-specific latent variables to the variables representing the observed ratings. Since exact inference in models of this type is typically intractable (Welling et al., 2005), approximate inference methods have to be resorted to. In Sec. 5.2, we proposed a model with efficient exact inference, based on the Restricted Boltzmann Machine architecture, as an alternative to directed latent variable models. The key to making exact inference tractable was replacing the directed connections between the latent variables and the observed variables by undirected connections. The drawback of this approach is that it makes the latent variables marginally dependent and the process of generating data from the model less intuitive.

Matrix factorization methods form the other major class of model-based collaborative filtering methods. These methods approximate the rating matrix with a product of two low-rank matrices: a user factor matrix U and an item factor matrix V. As a result, predicting a rating in the rating matrix amounts to computing the inner product between the feature vectors for the user and the item, which is a very efficient operation. In contrast to the traditional latent variable models that use discrete-valued factors, the low-rank matrices found by matrix factorization methods are real-valued. Moreover, such methods treat factors as parameters to be learned as opposed to latent variables to be inferred, thus avoiding all inference-related difficulties.

In the (hypothetical) case of a fully-observed rating matrix, the best approximation of the specified rank in the least squares sense can be found using Singular Value Decomposition (SVD). However, as most real-world datasets are sparse, the sum of squares distance is computed only for the observed entries of the rating matrix. This seemingly minor modification results in a difficult non-convex optimization problem which cannot be solved using standard SVD implementations, as was shown by Srebro and Jaakkola (2003). Instead of constraining the rank of the approximation matrix, or equivalently the number of factors, Srebro et al. (2004) proposed a method called Maximum Margin Matrix Factorization that is based on penalizing the norms of the factor matrices U and V. Training this model, however, requires solving a sparse semi-definite program (SDP), making this approach infeasible for all but the smallest datasets. While the faster gradient-based version of the algorithm scales much better than the SDP-based version (Rennie and Srebro, 2005), it still cannot handle a dataset of tens of millions of ratings.[5]

In this section, we will present the Probabilistic Matrix Factorization (PMF) model,

[5]For example, the fast MMMF training algorithm took about 15 hours on a dataset of about 2 million ratings on a (single-core) 3 Ghz Pentium 4 (Rennie and Srebro, 2005).

which is a straightforward probabilistic formalization of the low-rank matrix approximation methods for partially-observed matrices. As such, it can be viewed both as a latent variable user model and as a matrix factorization model. Even though more sophisticated algorithms for training PMF models exist, we will restrict our attention to the gradient-based online training method, since model scalability is our primary concern here. The training time for this algorithm is linear in the number of observed ratings, which makes the proposed method easily scalable to datasets of hundreds of millions of observed ratings.

5.3.1 Model formulation

We will now formulate PMF as a directed latent variable model. In PMF users and movies are represented by D-dimensional real-valued feature vectors. We will refer to the feature vectors for the i^{th} user and j^{th} movie as U_i and V_j respectively. The feature vectors are assumed to have Gaussian prior distributions:

$$P(U_i|\sigma_U^2) = \mathcal{N}(U_i|0, \sigma_U^2 I), \tag{5.16}$$

$$P(V_j|\sigma_V^2) = \mathcal{N}(V_j|0, \sigma_V^2 I). \tag{5.17}$$

To reflect the fact that the mean rating can be drastically different for different users and different movies, we incorporate per-user and per-movie biases into the model:

$$P(b_i|\sigma_b^2) = \mathcal{N}(b_i|0, \sigma_b^2), \tag{5.18}$$

$$P(c_j|\sigma_c^2) = \mathcal{N}(c_j|0, \sigma_c^2). \tag{5.19}$$

The rating given by user i to movie j is assumed to be Gaussian-distributed, with the distribution mean given by the inner product between the corresponding feature vectors added to the corresponding biases:

$$P(R_{ij}|U_i, V_j, b_i, c_j, \sigma^2) = \mathcal{N}(R_{ij}|U_i^T V_j + b_i + c_j, \sigma^2). \tag{5.20}$$

The entries of R are assumed to be independent given the model parameters, which results in the following conditional distribution for the rating matrix:

$$P(R|U,V,b,c,\sigma^2) = \prod_i \prod_j \mathcal{N}(R_{ij}|U_i^T V_j + b_i + c_j, \sigma^2). \tag{5.21}$$

For simplicity, all hyperparameters in the model are assumed to be fixed.

Even though all conditional distributions used to define PMF are Gaussian, exact inference in the model is intractable. This is the case because $U_i^T V_j$ is not Gaussian-distributed, since a product of two Gaussian random variables is not a Gaussian. Since exact inference is intractable, we need to choose an approximate inference method. In the interests of scalability, we avoided the full Bayesian approach that involves computing the posterior over model parameters using either Markov chain Monte Carlo (Neal, 1993) or variational methods (Neal and Hinton, 1998; Jordan et al., 1999). Instead, we find the MAP point estimate of the parameters using online gradient descent. The time complexity of computing the gradient of log-likelihood w.r.t. to the model parameters is linear in the number of observed ratings because the terms corresponding to unobserved ratings drop out. Thus one pass through the dataset takes $\Theta(DN)$ time where D is the feature vector dimensionality and N is the number of ratings in the training set.

5.3.2 Conditional PMF

Conditional PMF extends PMF to take into account the identities of the movies rated by a user, by conditioning on those identities. This extension is based on the assumption that users who have rated similar sets of movies are likely to have similar preferences. The resulting model is the PMF counterpart of the conditional RBM model from Sec. 5.2.2.

We condition on ratings by making the prior mean of the user feature vector a linear function of the rated indicator vector as follows:

$$\mu_{U_i} = \frac{\sum_j I_{ij} W_j}{\sum_j I_{ij}}, \tag{5.22}$$

where W_j is the vector of conditioning weights for movie j and I_{ij} is an indicator function that is equal to 1 if and only if user i rated movie j. The prior distribution for U_i is now

$$P(U_i|\sigma_U^2) = \mathcal{N}(U_i|\mu_{U_i}, \sigma_U^2 I). \tag{5.23}$$

The conditioning weight vectors are given zero-mean spherical Gaussian priors:

$$p(W_i|\sigma_W^2) = \mathcal{N}(W_i|0, \sigma_W^2 I). \tag{5.24}$$

This kind of conditioning on rated information allows the model to use the information that a user rated some movie even if the rating value is not known. Moreover, the model does not have to be retrained to use the identities of the newly-rated movie for making predictions. Intuitively, the i^{th} column of the W matrix captures the effect of a user having rated a particular movie has on the prior mean of the user's feature vector. As a result, users that have seen the same (or similar) movies will have similar prior distributions for their feature vectors.

Learning

We avoid recomputing the conditioning contribution to the user feature vector for each user/movie/rating triple by training on user rating vectors instead of triples. Given a vector of ratings for a particular user, we compute the conditioning term using Eq. 5.22 and, keeping it fixed, accumulate the parameter gradients for all the ratings in the vector before updating the model parameters.

5.4 Experimental results

5.4.1 The Netflix Prize dataset

We evaluated our models on the Netflix Prize dataset, which is the largest publicly available collaborative filtering dataset. The dataset contains over 100M ratings given by 480,189 users to 17,770 movies and is split into a 99M rating training set and a 1.4M validation set. Additionally, 2.8M user / movie pairs are provided as the test set. The ratings corresponding to these pairs are not publicly available,[6] though Netflix provides an online service that computes the test score given a set of predictions.[7] The ratings, which are integers from 1 to 5, have been provided by Netflix customers between October 1998 and December 2005. Each user / movie pair is accompanied by the date on which the rating was entered. The users included in the dataset have been chosen uniformly at random from the Netflix customer base. While no information outside their ratings has been provided for the users, the dataset includes the titles and release years for the movies.

This dataset in interesting for several reasons. First, it is large and very (98.8%) sparse. It includes users with over 10,000 ratings as well as users with fewer than 5 ratings. More importantly, the provided validation and test sets do not exclude the very rare users, which makes the results obtained on this dataset much more representative of the real-world situation. The validation and the test set were created by taking nine most recent ratings entered by each user and assigning each one of them with probability $\frac{1}{3}$ to the validation set and with probability $\frac{2}{3}$ to the test set.[8] As a result of this selection scheme, ratings from users with few ratings are overrepresented in the validation and test sets, relative to the training set. This makes predictive accuracy on users with few ratings very important for achieving high overall predictive accuracy.

[6]These rating are not available as of time of this writing. They will be released once the competition is over.

[7]The test score reported is actually computed on a subset of the test set. Netflix did not disclose which test pairs are included in the subset to prevent model selection based on the test score.

[8]If a user had fewer than 18 ratings in total, only a half of those ratings were split between the validation set and the test set.

5.4.2 Details of training

RBM and CRBM models were trained on vectors of user ratings, with model parameters updated after each user vector (i.e. online). A learning rate of 10^{-3} for weights and 10^{-4} for biases was used. The L_2 regularization parameter was set to 10^{-2} for weights and to 10^{-3} for biases.

PMF and online SVD models were trained on user/movie/rating triples, with parameters updated after each triple. The learning rates for feature vectors and biases were 10^{-2} and 10^{-3} respectively. The L_2 regularization parameter was set to 10^{-2} for feature vectors and to 10^{-1} for biases.

Like RBMs, conditional PMF models were trained on vectors of ratings. Each user vector was broken up into blocks of 100 ratings each and model parameters were updated after each block. The learning rate and the regularization parameter for the conditioning weights were 5×10^{-3} and 3×10^{-3} respectively.

Early stopping on the validation set was used for all models. We found that using early stopping in conjunction with L_2 weight regularization produced better models (as measured by the validation set RMSE) than either regularization method on its own. Our training algorithm performed 250 passes through the training set, keeping track of the parameter values that produced the best validation score, which was computed after each pass. Early stopping was implemented by returning the model with the best validation score. While this is not the most efficient way to implement early stopping, it is simpler and more robust to the random fluctuations in the validation score than implementations that make the stopping decision based on the last few score values. The L_2 regularization parameters were chosen based on the validation set score they produced in conjunction with the early stopping procedure described above.

5.4.3 Details of evaluation

In our evaluation we compared models based on their RMSE on the validation set. We chose to use the validation set score instead of the test set score as a basis for comparison because some of our performance analysis methods require computing RMSE on subsets of the evaluation set, which cannot be done for the test set since the answers for it have not been released by Netflix at the time of this writing.[9] We did not split the validation set into a new validation set and a new test set because we wanted to make sure that our reported scores were directly comparable to those in the literature, where the score on the standard validation set is frequently reported either in addition to or instead of the test score. Finally, while the reported validation scores are likely to be slightly optimistic compared to the corresponding test scores, this is not a serious problem since we are primarily interested in determining how our models perform relative to each other.

5.4.4 Experiments with RBMs

To determine how the predictive performance of RBMs depends on the number of hidden units used we tried four different numbers of hidden units: 25, 50, 100, and 200. For each of these numbers, we trained one plain RBM and two conditional RBMs that differed in the set of movies conditioned on: the first CRBM (CRBM1) conditioned only on the movies rated by the user in the training set, while the second CRBM (CRBM22) conditioned on the movies from the validation set and the test set in addition to those from the training set.[10] The validation set RMSE scores for the resulting models are given in Table 5.1. As can be seen from the results, increasing the number of hidden units results in better performance for both the plain RBM and CRBM2, with lowest RMSE scores achieved by the models with 200 hidden units. For CRBM1, RMSE decreases as

[9]The web service provided by Netflix of course does not allow choosing the subset of the test set to compute the RMSE on.

[10]Though it is possible to condition on one set of rated indicators when training a CRBM and on another when testing it, in these experiments the same set was used during training and testing.

Table 5.1: Validation set RMSE for RBM models with various numbers of hidden units. The second column shows the scores for the basic RBM model. The conditional RBM in the third column conditions on the rated information in the training set. The conditional RBM in the fourth column conditions on the rated information in the validation and test sets in addition to that in the training set.

Number of hidden units	RBM	CRBM1 cond. on train	CRBM2 cond. on train + valid + test
25	0.9277	0.9290	0.9276
50	0.9180	0.9191	0.9179
100	0.9127	0.9146	0.9121
200	0.9125	0.9151	0.9116

the number of hidden units goes from 25 to 50 to 100 and then slightly increases when the number of units becomes 200. For all three RBM types, however, the difference in performance between the model with 100 hidden units and the model with 200 hidden units is very small. In all cases, models with 100 and 200 hidden units outperform models with 25 hidden units (and, to a lesser extent, models with 50 hidden units) by a large margin.

For all numbers of hidden units in the table, the plain RBM outperforms CRBM1, while CRBM2 outperforms the RBM. We believe that CRBM1 does not perform as well as the plain RBM because of overfitting. After all, when conditioning on the rated information from the training set alone, a CRBM is not any more powerful than the plain RBM it is based on, in spite of having more parameters, as explained in Sec. 5.2.2. However, conditioning on the rated information not already present in the training set does make the CRBM more powerful than the underlying RBM as demonstrated by CRBM2 slightly outperforming the plain RBM for all numbers of hidden units. The

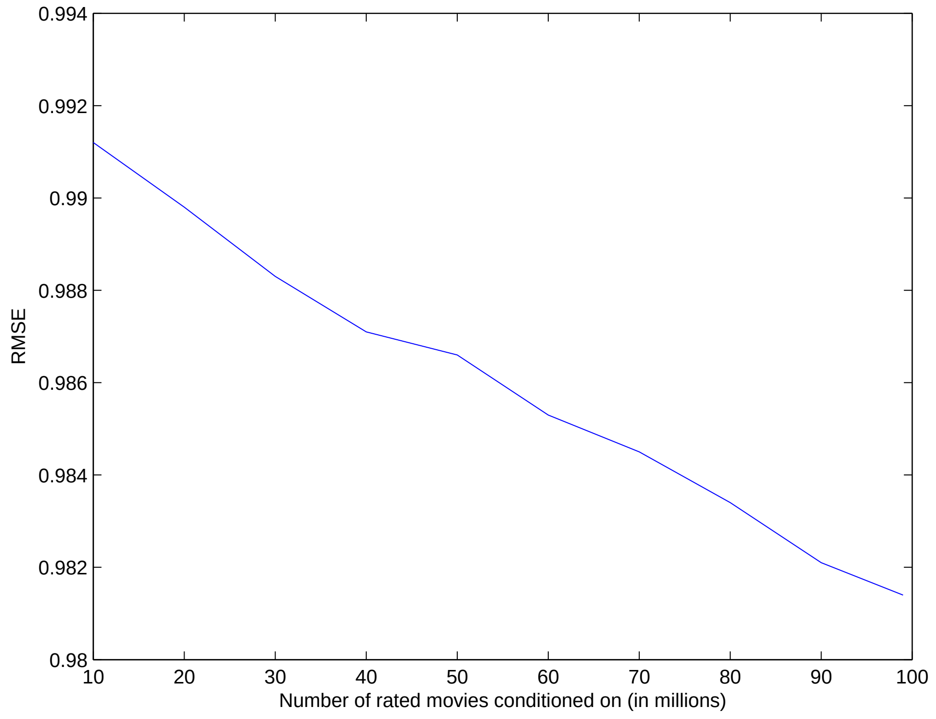

Figure 5.1: Validation set RMSE of a conditional RBM with 50 hidden units as a function of the size of the rated set conditioned on. The model was trained on a 10M rating training set. The x-axis gives the size of the rated set conditioned on during training and testing by the RBM.

small gap in performance between CRBM2 and the plain RBM is explained by the fact that only a small fraction of the rated information CRBM2 conditions on does not come from the training set, since the training set contains 99M rated entries, while the validation set and the test set contain only 4.2M entries in total.

The results reported in Table 5.1 demonstrate that conditioning on more rated information than is present in the training set alone, allows CRBMs to outperform plain RBMs. To test our theory that the improvement in RMSE is so small because only a small fraction of the rated information came from outside the training set, we performed

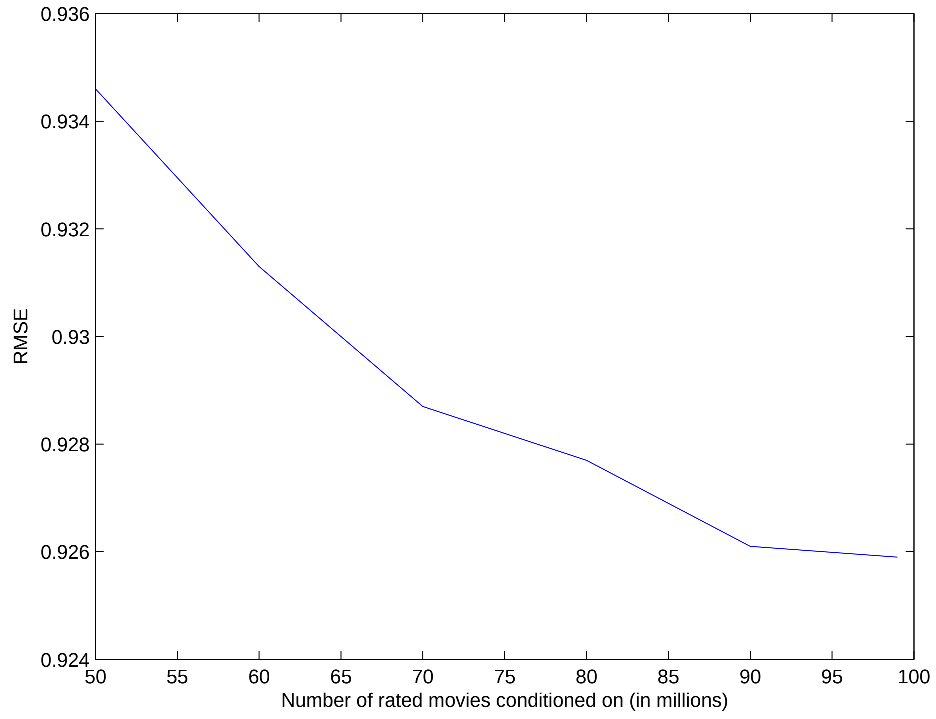

Figure 5.2: Validation set RMSE of a conditional RBM with 50 hidden units as a function of the size of the rated dataset conditioned on. The model was trained on a 50M rating training set. The x-axis gives the size of the rated set conditioned on during training and testing by the RBM.

two additional experiments. Ideally, we would have preferred to train our models on the entire Netflix training set while conditioning on a large collection of additional rated information. Since no such collection was available, we randomly split the Netflix training set into two parts: a training set containing 10M user/movie/rating triples and a conditioning set containing 89M of user/movie tuples. We then trained a sequence of ten conditional RBMs with 50 hidden units on the training set, conditioning on progressively larger subsets of the rated information. The first CRBM conditioned only on the rated information from the training set, while the tenth CRBM conditioned on all 99M rated entries. As can be seen from Figure 5.1, the RMSE score is effectively linear in the number of the rated entries conditioned on, going from 0.9912 for the first model down to 0.9814 for the last one. For comparison, the plain RBM trained on the 10M training set achieved a score of 0.9914. These numbers confirm that rated information can be used to improve RBM performance substantially, as long as there is a sufficient amount of such information available. On the other hand, using more rating data is likely to be a more effective way of boosting model performance. For example, doubling the training set size to 20M ratings reduces the RMSE for the plain RBM down to 0.9646.

We performed a similar experiment using a larger training set of 50M ratings. The remaining 49M ratings were discarded leaving only the rated information that was partitioned into 5 sets of about 10M entries each. We trained a sequence of six CRBMs with the first one using only the rated information from the training set and the last one using all available rated information. As in the previous experiment, the models had 50 hidden units. The baseline plain RBM had a score of 0.9339, while the first and the last CRBMs in the sequence achieved 0.9346 and 0.9259 respectively. These results (shown in Figure 5.2) indicate that even when we have a fairly large training set using a set of rated information of a comparable size can still lead to a substantial reduction in RMSE.

5.4.5 Experiments with PMF

We investigated the effect of feature vector dimensionality on the performance of PMF and conditional PMF models by training models with 10-, 25-, 50-, 100-, 200-, and 500-dimensional feature vectors. As baselines for comparison, we used unregularized PMF models with and without biases. Models of this type can be viewed as an online version of SVD that operates on partially observed matrices. These baselines have been chosen to emphasize the importance of regularization and biases in PMF. We also included three weaker baselines: a user-bias model, a movie-bias model, and a model with both user and

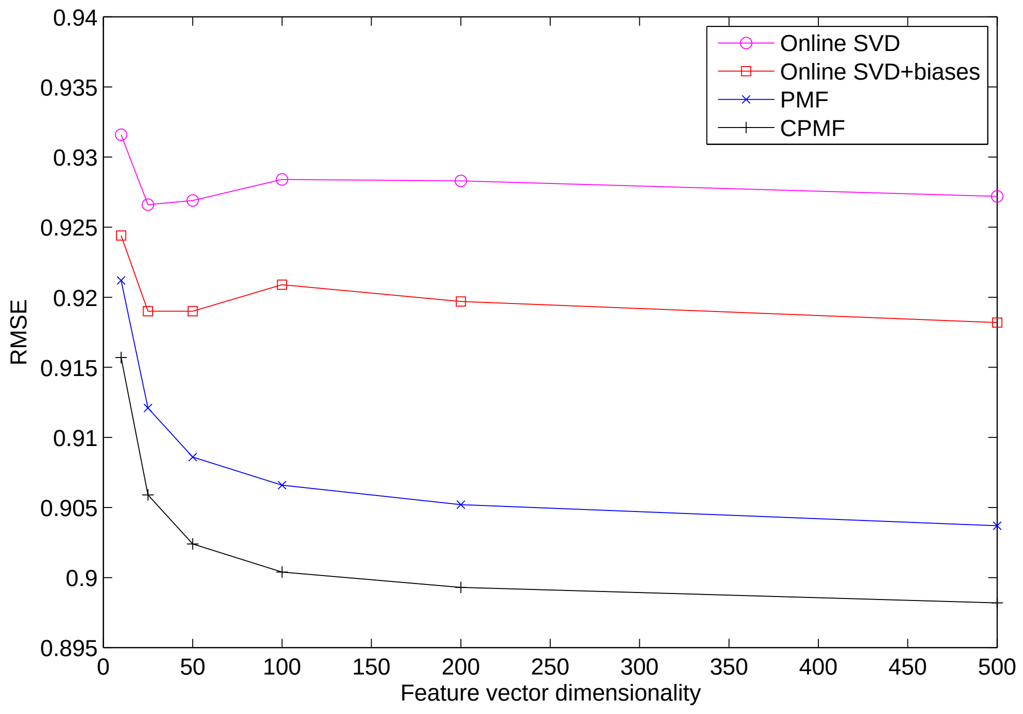

Figure 5.3: Validation set RMSE for various PMF and PMF-like models.

movie biases.[11] All models were trained of the entire Netflix training set. The conditional PMF models conditioned on all available rated information (training, validation, and test) during training and testing. Early stopping on the validation set was used. The RMSE scores for the feature-vector-based models are plotted on Figure 5.3. Those scores along with the scores for the bias-only models, which did not fit on the figure, are also shown in Table 5.2.

The worst performing models in the comparison are the bias-only models, which is not surprising considering how few parameters they have. The model with both user and movie biases performs much better than the two models that have biases of a single type. In terms of RMSE, it is about half-way between the other two bias-only models and the

[11]These bias-only models can be viewed as PMF models without feature vectors.

Table 5.2: Validation set RMSE for various PMF and PMF-like models.

	Feature vector dimensionality					
Model type	10	25	50	100	200	500
Online SVD	0.9316	0.9266	0.9269	0.9284	0.9283	0.9272
Online SVD w. biases	0.9244	0.9190	0.9190	0.9209	0.9197	0.9182
PMF	0.9212	0.9121	0.9086	0.9066	0.9052	0.9037
Conditional PMF	0.9157	0.9059	0.9024	0.9004	0.8993	0.8982
User-bias	1.0634					
Movie-bias	1.0528					
User and movie bias	0.9820					

PMF models. The online SVD models without biases outperform the bias-only models by a large margin but perform considerably worse than the corresponding SVD models with biases. The gap in performance due to biases grows from 0.0072 for models with 10D feature vectors to 0.0090 for models with 500D feature vectors. More impressively, the worst-performing SVD model with biases outperforms the best-performing SVD model without biases. This clearly demonstrates that movie and user biases have a strong positive effect on predictive performance.

For each feature vector dimensionality we used, the online SVD model with biases was outperformed by the PMF model, which, in turn, was outperformed by the conditional PMF model. The gap in RMSE between the online SVD model with biases and the PMF model grew from 0.0032 for 10D feature vectors to 0.0145 for 200D and 500D feature vectors. These numbers demonstrate the importance of regularization for matrix factorization models, since the online SVD model with biases is simply a PMF model with regularization turned off. As expected, regularization becomes more important as the number of free parameters in the model increases. The gap in RMSE between the PMF model and the conditional PMF model, on the other hand, changes only slightly as the dimensionality of the feature vectors grows, staying in the range 0.0055-0.0062. Thus the benefit from conditioning on the rated information does not seem to depend on the feature vector dimensionality of the model.

Figure 5.3 shows that predictive performance of PMF and conditional PMF models improves monotonically as feature vector dimensionality increases. However, this is not the case for the online SVD models (with and without biases), for which model performance degrades significantly when going from 50D feature vectors to 100D feature vectors before starting to improve again. This behaviour is a consequence of the lack of a regularization term in the objective function for the online SVD models. Since early stopping is the only regularization method used by these models, the differences in learning dynamics resulting from different feature vector dimensionalities can have a dramatic effect on the parameter values found by the training algorithm. In contrast, the regularization effect produced by the feature vector priors (Eqs. 5.16 and 5.17) of PMF models makes PMF training less sensitive to the peculiarities of learning dynamics.

The effect of conditioning on rated information

To determine the effect of conditioning on rated information in PMF models on prediction accuracy for users of different rating frequency, we grouped users into ten bins and computed the model RMSE for each bin. Users were assigned to bins based on the number of ratings they had in the training set, so that each bin contained the same number of users. This is a reasonable partitioning scheme because users of all frequencies have roughly the same number of ratings in the validation set. The plots of the per-bin RMSE scores for PMF and conditional PMF models with 10D and 500D feature vectors are shown on Figure 5.4. From the figure it is clear that both models perform much better on users with more ratings. For users with few ratings (the first two bins), the conditional PMF models outperform their non-conditional counterparts by over 0.01, but this gap decreases as the number of training set ratings per user increases. For the models

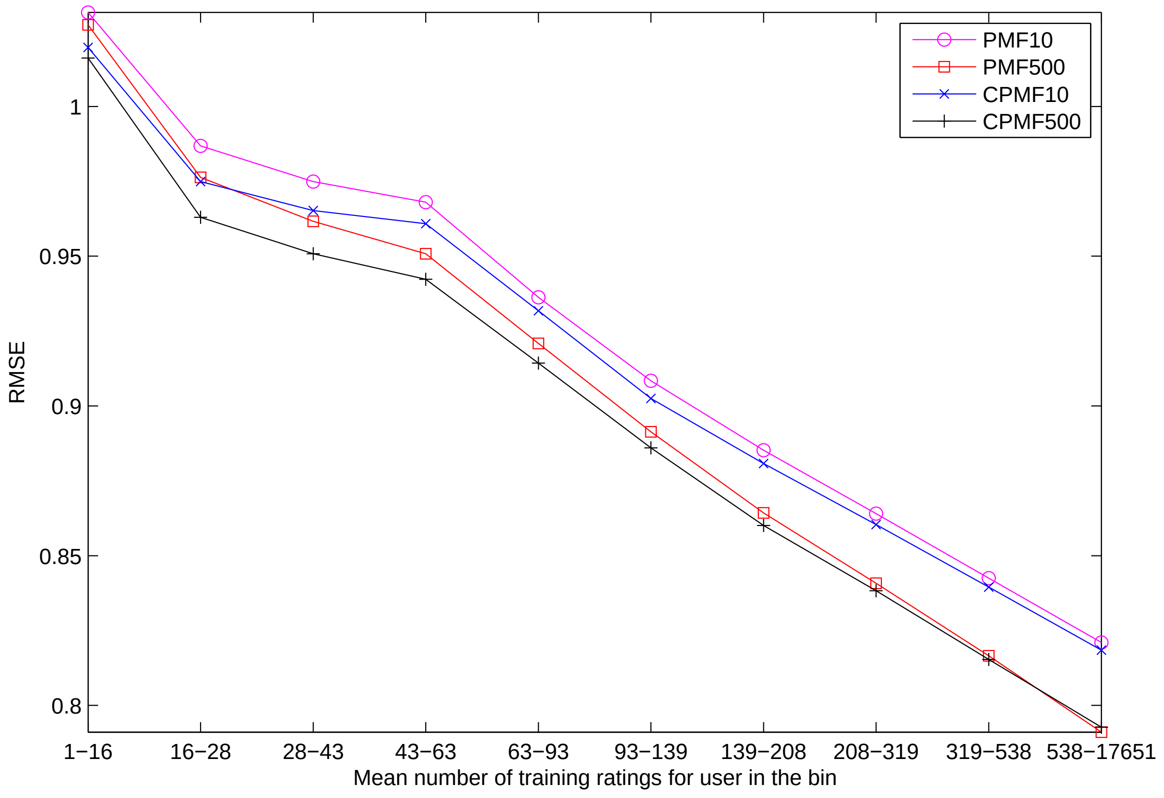

Figure 5.4: Validation set RMSE of PMF and CPMF models as a function of the number of ratings for the user. PMFn is a PMF model with n-dimensional feature vectors. CPMFn is a conditional PMF model with n-dimensional feature vectors.

with 10D feature vectors, when predicting ratings for users with the most ratings (i.e. in the last bin) the gap is only 0.026. For the models with 500D feature vectors, the PMF model actually outperforms the conditional PMF model by 0.0017 on the last bin. However, on users with the fewest ratings contained in the first bin, the conditional PMF model with 10D feature vectors outperforms the non-conditional PMF model with 500D feature vectors by 0.0075. These results show that conditioning on rated information tends to improve PMF performance for almost all users, though users with few (e.g.

1-100) ratings benefit the most.

One of the advantages of conditional PMF is that it can make non-trivial predictions for users for which we know the identities of the rated items but not the actual rating values. To determine how much conditioning on the identities of the rated items helps, we measured the accuracy of the model when predicting the test ratings for users with no ratings in the training set, while conditioning on the identities of the rated movies. We started by randomly dividing users into five groups of equal size. For each group, we generated a modified training set by leaving out the ratings for the users in the group from the original training set. The corresponding validation sets were produced in an analogous manner. The test set for each group was obtained by leaving out the ratings for users from other groups from the original validation set. On each modified training set, we trained a 100D conditional PMF model and measured its RMSE on the corresponding test set. Predictions on the test set were made while conditioning on the identities of the movies rated by the user in the original training set. As a baseline for comparison, we used a movie-bias model trained on the same training sets.

Figure 5.5 shows the mean RMSE for each model type as a function of the number of movies known to be rated by the user. For each frequency bin, the RMSE value was computed based on the total SSE value for the five models. The conditional PMF model clearly outperforms the movie-bias model for all user frequency bins. The overall RMSE scores are 1.0134 for CPMF and 1.0529 for the movie-bias model. The gap in RMSE between the two models is the smallest (0.0121), though still quite substantial, for users with the fewest movies rated (the first bin). The gap between the two models is much larger (0.0332-0.0505) for users in other frequency bins. It achieves its largest value of 0.0505 for users in the third bin (28-43 movies rated), before beginning to decrease down to 0.0332 for the last bin (538-17651 movies rated). Thus when making predictions for users without known rating values, users with an intermediate number of movies rated (28-63) benefit the most from conditioning on the movie identities.

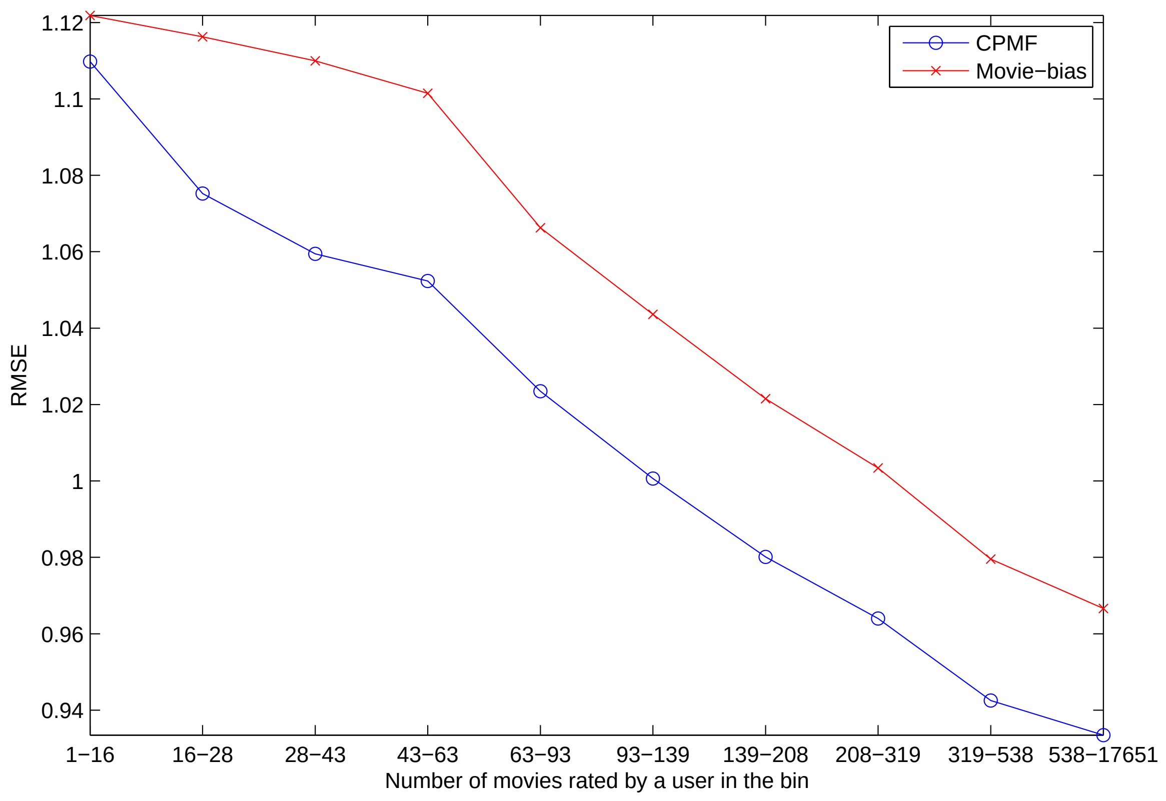

Figure 5.5: Validation set RMSE for users without known rating values as a function of the number of movies known to be rated by the user. CPMF is a conditional PMF model with 100D feature vectors. Movie-bias is a PMF-like model without feature vectors and user biases. CPMF conditions on the identities of the movies rated by the user, while the movie-bias model does not.

5.4.6 Analysis of RBM and PMF models

In this section we will compare the predictive performance of RBM models to that of PMF models. Table 5.3 summarizes the results from the two previous sections. The table does not include conditional RBMs because they performed only marginally better than the plain RBM models. As can be seem from the table, PMF and conditional PMF models with feature vector dimensionality 25 and higher outperform all RBM models. The best PMF model achieves an RMSE score that is 0.0088 lower than the best RBM score. The gap between the scores of the best conditional PMF model and the best RBM model is even larger at 0.0143. However, the best-performing (C)PMF models have many more parameters than any of the RBM models. Performance of RBM and plain PMF models with similar numbers of parameter are quite similar. Conditional PMF models still outperform the corresponding RBM models even if we control for the number of parameters. Thus, whether or not model memory usage is a concern, conditional PMF appears to be the best-performing model class.

Though, as we have seen, PMF models tend to outperform RBM models in terms of the overall RMSE score, it is not clear that PMF models are better across the entire spectrum of users. To investigate this question, we grouped users into bins based on the number of ratings in the training set and computed the per-bin RMSE. We use the same binning setup as in Sec. 5.4.5. We compared the best RBM (200 hidden units) to the 50D PMF model as well as to the 500D PMF model, which was the best-performing PMF model. The 50D PMF model was chosen because it had a comparable number of parameters to that of the RBM (about 42% more parameters).

The plots of the per-bin RMSE results are shown on Figures 5.6 and 5.7. The large gap between the score of the PMF model and the RBM model on the last three bins on both plots indicates the PMF models excel at predicting ratings for users with many (e.g. over 150) ratings. On bins 2-4, the RBM performs better than the 50D PMF model and about as well at the 500D PMF model. On the first bin however, the PMF models outperform the RBM once again. The superior performance of the PMF models on users with many ratings can be explained by the fairly large number of per-user parameters these models have. Having many user-dependent parameters makes such models very flexible but results in more training data being required for good performance. The use of per-user biases in PMF models explains why the RBM, which does not have any

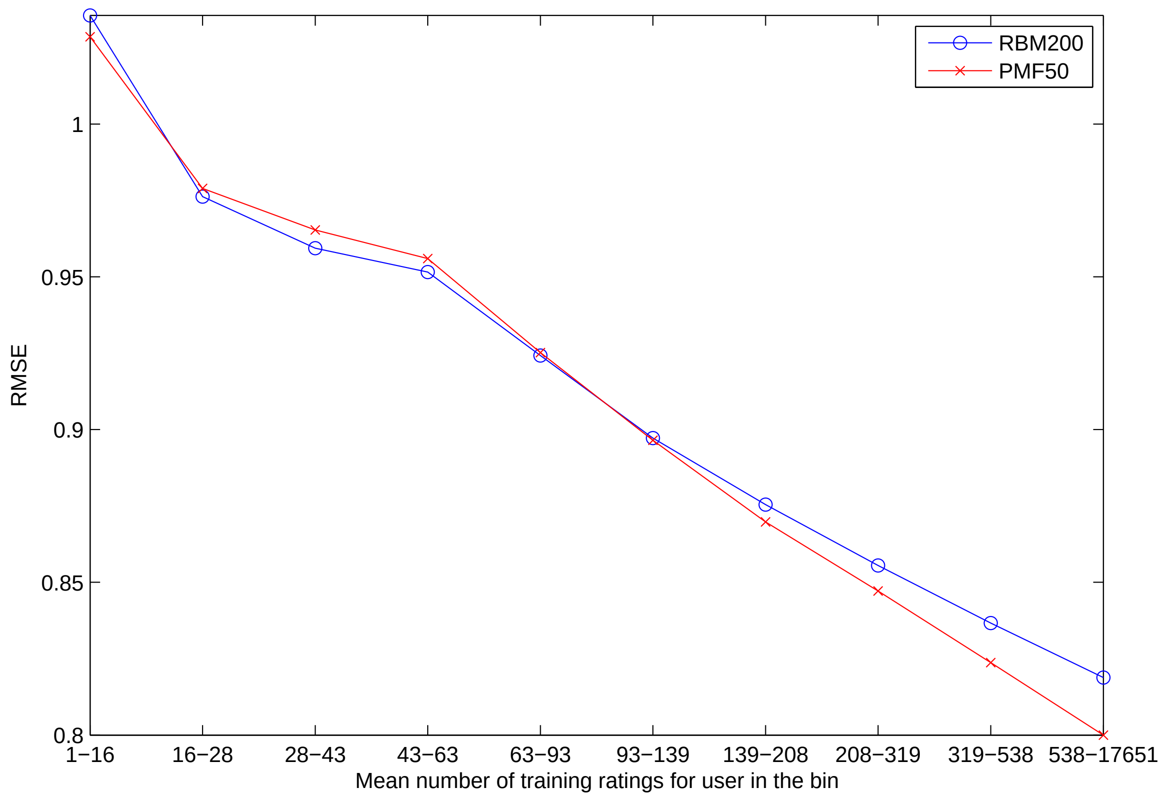

Figure 5.6: Validation set RMSE of RBM and PMF models as a function of the number of ratings for the user. RBM200 is an RBM model with 200 hidden units. PMF50 is a PMF model with 50-dimensional feature vectors.

user-dependent parameters, is outperformed by all PMF models on the bin containing users with the fewest ratings. Figure 5.8 showing the per-bin results for a 10D PMF model provides more evidence for this theory, since the RBM model outperforms this PMF model on all bins but the first one.

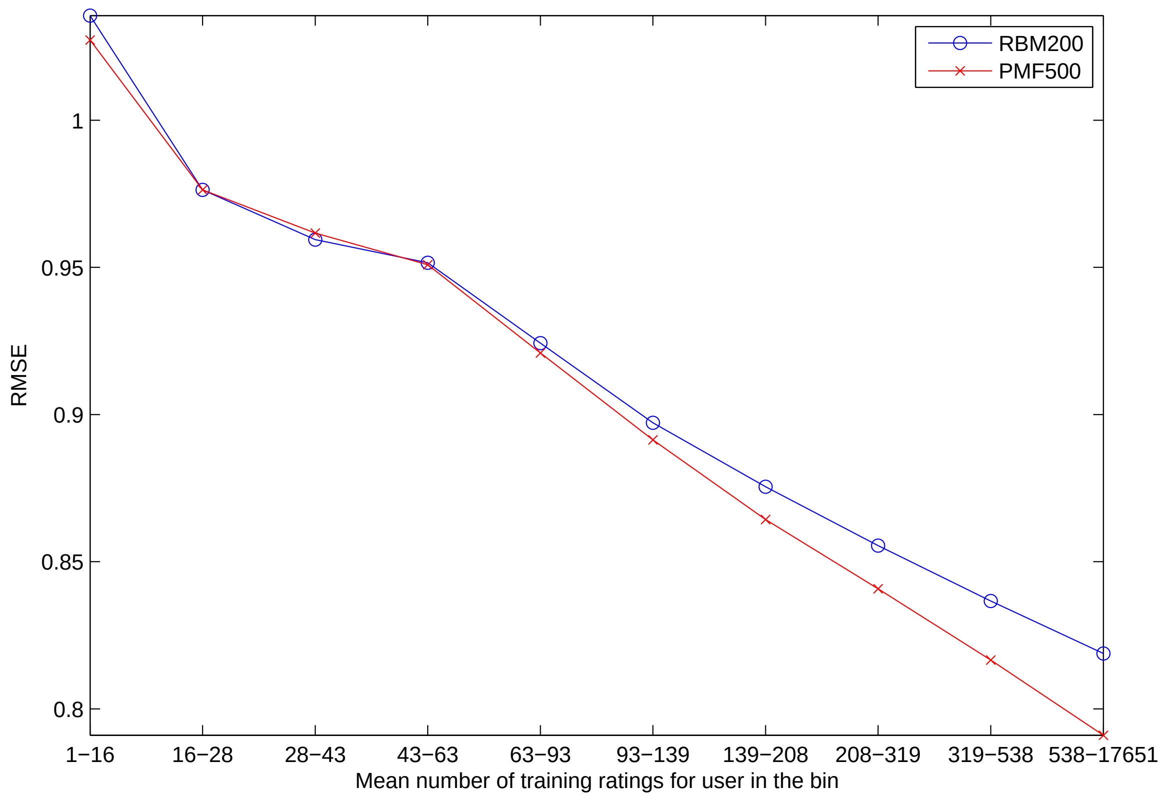

Figure 5.7: Validation set RMSE of RBM and PMF models as a function of the number of ratings for the user. RBM200 is an RBM model with 200 hidden units. PMF500 is a PMF model with 500-dimensional feature vectors.

5.5 Discussion

The release of the Netflix Prize dataset has resulted in an increased interest in large-scale collaborative filtering in the machine learning community. Matrix factorization (MF) models have received by far the most attention due to their simplicity and remarkably good performance on the dataset.

The common feature of the best-performing MF models is conditioning on the identities of the rated movies, first introduced for RBM models in (Salakhutdinov et al., 2007).

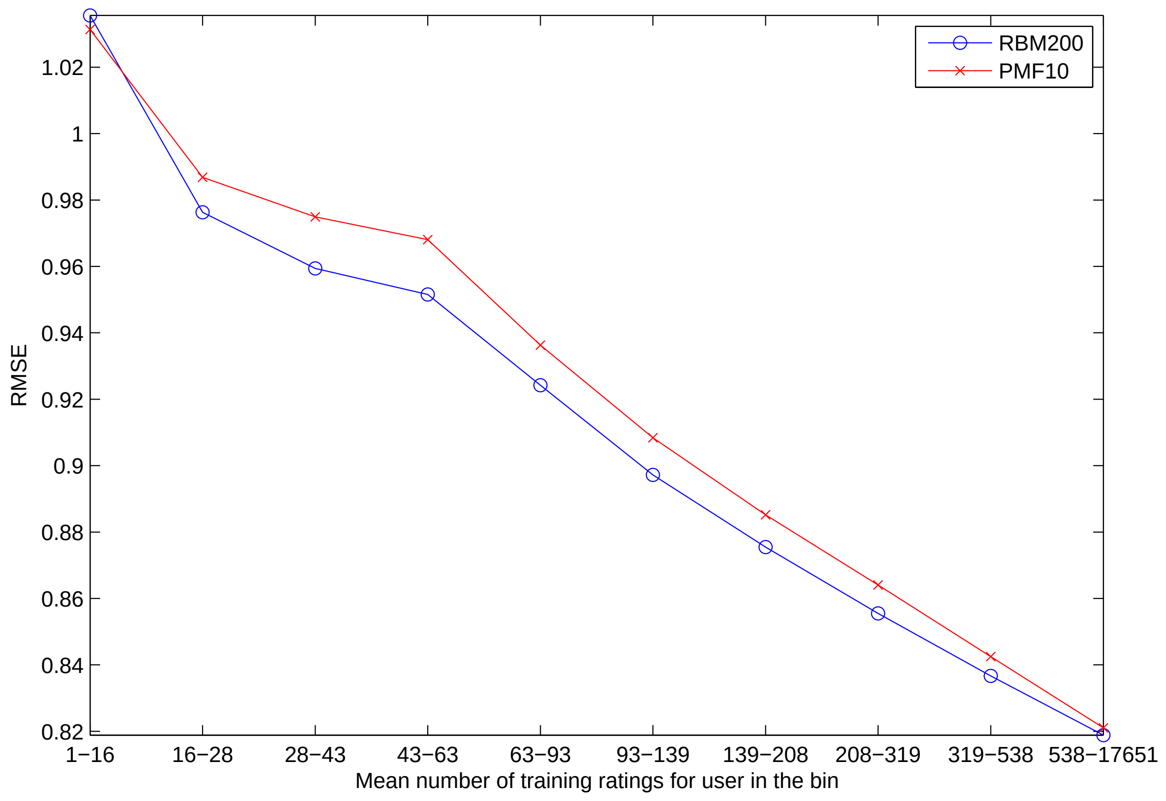

Figure 5.8: Validation set RMSE of RBM and PMF models as a function of the number of ratings for the user. RBM200 is an RBM model with 200 hidden units. PMF10 is a PMF model with 10-dimensional feature vectors.

This technique was used by Paterek (2007) to eliminate user feature vectors in an MF model by replacing them with a linear function of the rated indicator vector. The resulting model performed considerably worse than the unconstrained model because it had several times fewer parameters. The conditional PMF model introduced in (Salakhutdinov and Mnih, 2008b) was the first MF model that had both the conditional component and user feature vectors, resulting in performance superior to that of the non-conditional MF models. Similar models have since been proposed in (Koren, 2008; Takács et al., 2008), where they were augmented with non-parametric components designed capture the local structure in the data.

A number of Bayesian MF approaches to collaborative filtering have also been explored. Both variational Bayes (Lim and Teh, 2007) and MCMC-based training (Salakhutdinov and Mnih, 2008a; Zhu et al., 2008) have been shown to produce more accurate models than MAP-based training.

Boltzmann machines as models for collaborative filtering have received relatively little attention. Truyen et al. (2009) introduced "ordinal Boltzmann machines" designed for modelling ordinal rating values. The basic idea is to represent ratings using vectors of indicator variables of the form $[rating < i]$ and $[rating > i]$ to ensure that rating values that are close numerically will have similar representations. The authors also introduced a product-of-RBMs model that models the joint distribution of the rows and columns of the rating matrix.

Acknowledgments

We thank Vinod Nair, Ilya Sutskever, and Tijmen Tieleman for many helpful discussions. We thank Netflix for making the Netflix Prize dataset available.

Early versions of parts of this chapter have been published as (Salakhutdinov, Mnih, and Hinton, 2007) and (Salakhutdinov and Mnih, 2008b).

Table 5.3: Validation set RMSE for RBM and (C)PMF models. Representation dimensionality refers to the number of hidden units in RBM models and the dimensionality of the feature vectors in (C)PMF models. Number of parameters is the number of model parameters in millions.

Model type	Representation dimensionality	Number of parameters ($\times 10^6$)	RMSE
RBM	25	2.3	0.9277
RBM	50	4.5	0.9180
RBM	100	9.0	0.9127
RBM	200	17.9	0.9125
PMF	10	5.5	0.9212
PMF	25	13.0	0.9121
PMF	50	25.4	0.9086
PMF	100	50.3	0.9066
PMF	200	100.1	0.9052
PMF	500	249.5	0.9037
CPMF	10	5.6	0.9157
CPMF	25	13.4	0.9059
CPMF	50	26.3	0.9024
CPMF	100	52.1	0.9004
CPMF	200	103.6	0.8993
CPMF	500	258.4	0.8982

Chapter 6

Conclusions

In this thesis we proposed and evaluated a number of probabilistic models for discrete data. Our approach was based on the use of learned distributed representations for discrete inputs as a way of dealing with sparse high-dimensional data. We concentrated on two application areas: statistical language modelling and collaborative filtering.

6.1 Language modelling

The first part of the thesis was concerned with statistical language modelling. In Chapter 3 we introduced three non-hierarchical language models. Our first model, based on the RBM architecture, used a vector of binary latent variables to capture the interaction between the feature vectors for the context words and the feature vector for the next word. Though this model did not perform well, we showed that augmenting it with temporal connections between different instantiations of the latent variable vector made it competitive with n-gram models. The log-bilinear model, which was simpler and faster than the RBM-based models, was shown to be superior to the n-gram models on the 14M word APNews dataset.

Though the LBL model achieves high predictive accuracy, the time complexity of training and making predictions with the model is linear in the number of words in the vocabulary, which makes it too slow for most applications. We addressed this drawback in Chapter 4, where, building on work of Morin and Bengio (2005), we introduced the hierarchical version of the LBL model which is exponentially faster than its non-hierarchical counterpart. The speedup is achieved by structuring the vocabulary as a binary tree over words, making the time complexity of computing the probability of the next word logarithmic in the vocabulary size.

We have demonstrated that HLBL models can outperform the best n-gram models and perform as well as LBL models on the APNews dataset. We found that the key to making a hierarchical model perform well is using a carefully constructed hierarchy over words. We have presented a simple and fast feature-based algorithm for automatic construction of such hierarchies. Creating hierarchies in which every word occurred more than once was essential to getting the models to perform well.

An inspection of trees generated by our adaptive algorithm showed that the words with the largest numbers of replicas in a tree did not typically have multiple distinct senses. Instead, the algorithm appeared to replicate the words that occurred relatively infrequently in the data and were therefore difficult to cluster. The failure to use multiple codes for words with several very different senses is probably a consequence of summarizing the distribution over contexts with a single mean feature vector when clustering words. The "sense multimodality" of context distributions would be better captured by using a small set of feature vectors found by clustering the contexts.

Our results show that the hierarchical clustering algorithm we use to build trees over words works quite well in practice. However, the objective function that this algorithm optimizes is unrelated to the log-likelihood on the training data, which is the objective function we optimize when we train the model. Developing a tree-building algorithm that optimizes the same objective function as the training algorithm is likely to produce word hierarchies better suited for use by HLBL models, which in turn is likely to produce better performing models.

HLBL is based on the idea of linear prediction in the space of word feature vectors. While this approach works well with high-dimensional feature vectors, it is possible that comparable or superior performance can be achieved by performing non-linear prediction in a lower-dimensional feature space. Using a deep neural network (Hinton et al., 2006; Bengio and Lecun, 2007) for making probabilistic decisions on a tree is another promising direction, since it was shown that deep networks are more effective than shallow networks at mapping text documents to compact codes (Ranzato and Szummer, 2008).

Though in this thesis we emphasized the advantages of using distributed representation for words, there are drawbacks to this approach. By representing words using learned feature vectors, the ability to distinguish between words that are used in similar ways is greatly diminished. While this is also the reason for the superior generalization ability of neural language models, sometimes words that are very similar in usage are not interchangeable. For example, since word-for-word reproduction is the defining property of a quotation, replacing any word with its synonym makes the quotation much less probable. N-gram models excel in cases like this because they effectively memorize the n-tuples seen in the training data. The complementary strengths of the two language model types suggest that a combination of models of different types might outperform the individual models. Traditionally, neural models have been combined with n-grams by training the models separately and then mixing their predictions using fixed mixing proportions. A more promising approach might be to either train the models jointly (as components of the mixture), or to train the neural model on its own and then train the n-gram model to optimize the predictive performance of the mixture.

6.2 Collaborative filtering

In the second part of the thesis we concentrated on collaborative filtering. We showed how RBM models can be used to model the distribution of sparse high-dimensional user rating vectors efficiently, presenting inference and learning algorithms that scale linearly in the number of observed ratings. We also introduced the PMF model which is based on the probabilistic formulation of the low-rank matrix approximation problem for partially observed matrices. Conditional versions of both model types were also introduced to allow conditioning on the identities of the movies rated whether or not the actual rating values are known. We evaluated our collaborative filtering models on the Netflix Prize dataset and showed that both PMF and RBM models outperform online SVD models. Of all models we evaluated, conditional PMF models performed best in our comparison, achieving excellent performance even when using relatively low-dimensional feature vectors.

Though RBMs typically use exponential family random variables, the multinomial distribution is not the only reasonable choice for modelling ratings, since Binomial, Gaussian, and Beta distributions can also be used. Moreover, extending RBMs that have visible units of one of these types to condition on rated information is likely to be much more beneficial then for RBMs with multinomial units, since it will actually make the models more expressive.

There is considerably more flexibility with the choice of the rating noise distribution in PMF than in RBMs. For example, PMF can model ordinal ratings by using the ordered multinomial logit parameterization at the output (Gelman and Hill, 2007, Chap. 6). Similarly, PMF can be extended to model multinomial observations with many possible values efficiently by organizing the variable values into a hierarchy as was done with words in the HLBL model. This extension would be useful for modelling the distribution of movies rated by a user. Unfortunately, this tree-based parameterization of a multinomial distribution cannot be used efficiently in an RBM due to the restrictions on the form of its energy function (Hinton et al., 2006; Bengio, 2009).

Since PMF models performed best in our experiments when they had hundreds of millions of parameters, developing better ways of regularizing such models is important.

Making the prior mean of a user vector a linear function of the rated indicator variables as was done in conditional PMF is one such regularization technique. Another possibility is to define flexible hierarchical priors for user and movie vectors.

Like most model-based collaborative filtering methods, PMF assumes that each user has a single set of preferences. While in most cases this is a reasonable assumption, accounts for many users in the Netflix Prize dataset are shared by multiple people with distinct movie tastes. Because of its probabilistic nature, PMF can be easily extended to associate each user (account) with multiple feature vectors. The model would generate the rating given by a user to a movie by picking one of the user feature vectors and then generating the rating the way original PMF does. As a result, the rating for any given user/movie pair would have a mixture of Gaussians distribution.

Bibliography

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127.

Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. In *NIPS*, pages 932–938.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Bengio, Y. and Lecun, Y. (2007). Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., Decoste, D., and Weston, J., editors, *Large-Scale Kernel Machines*. MIT Press.

Bengio, Y. and Senécal, J.-S. (2003). Quick training of probabilistic neural nets by importance sampling. In *AISTATS'03*.

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

Bennett, J. and Lanning, S. (2007). The Netflix prize. In *Proceedings of KDD Cup and Workshop.*

Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G. B., and Strehl, A. L. (2009). Conditional probability tree estimation analysis and algorithms. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence.*

Blitzer, J., Globerson, A., and Pereira, F. (2005a). Distributed latent variable models of lexical co-occurrences. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics.*

Blitzer, J., Weinberger, K., Saul, L., and Pereira, F. (2005b). Hierarchical distributed representations for statistical language modeling. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 18.*

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 43–52.

Brown, P., Mercer, R., Della Pietra, V., and Lai, J. (1992). Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

Canny, J. F. (2002). Collaborative filtering with privacy via factor analysis. In *SIGIR*, pages 238–245.

Carreira-Perpinan, M. and Hinton, G. (2005). On contrastive divergence learning. In *10th Int. Workshop on Artificial Intelligence and Statistics (AISTATS'2005).*

Chen, S. F. and Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine learning*.

Emami, A., Xu, P., and Jelinek, F. (2003). Using a connectionist model in a syntactical based language model. In *Proceedings of ICASSP*, volume 1, pages 372–375.

Fellbaum, C. (1998). *WordNet: An Electronical Lexical Database*. The MIT Press, Cambridge, MA.

Gelman, A. and Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press New York.

Goodman, J. (2000). A bit of progress in language modeling. Technical report, Microsoft Research.

Goodman, J. (2001). Classes for fast maximum entropy training. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1.

Hinton, G. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12.

Hinton, G., McClelland, J., and Rumelhart, D. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 77–109. MIT Press, Cambridge.

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

Hinton, G. E., Osindero, S., and Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge.

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the 15th Conference on Uncertainty in AI*, pages 289–296.

Hofmann, T. (2001). Learning what people (don't) want. In *Proceedings of the 12th European Conference on Machine Learning*, pages 214–225.

Jordan, M. and Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2):181–214.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434.

Lim, Y. and Teh, Y. (2007). Variational Bayesian approach to movie rating prediction. In *Proceedings of KDD Cup and Workshop*.

Marlin, B. (2004a). Collaborative filtering: A machine learning perspective. Master's thesis, University of Toronto.

Marlin, B. (2004b). Modeling user rating profiles for collaborative filtering. In *Advances in Neural Information Processing Systems*, volume 16, pages 627–634.

Marlin, B. and Zemel, R. S. (2004). The multiple multiplicative factor model for collaborative filtering. In *Proceedings of the Twenty-first International Conference on Machine Learning*.

Marlin, B. M. and Zemel, R. S. (2007). Collaborative filtering and the missing at random assumption. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence.*

McMahon, J. G. and Smith, F. J. (1996). Improving statistical language model performance with automatically generated word hierarchies. *Computational Linguistics*, 22(2):217–247.

Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. *Proceedings of the 24th international conference on Machine learning*, pages 641–648.

Mnih, A. and Hinton, G. (2009). A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems*, volume 21.

Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In Cowell, R. G. and Ghahramani, Z., editors, *AISTATS'05*, pages 246–252.

Neal, R. M. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.

Neal, R. M. and Hinton, G. E. (1998). A view of the EM algorithm that justifies incremental, sparse and other variants. In Jordan, M. I., editor, *Learning in Graphical Models*, pages 355–368.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop.*

Pereira, F., Tishby, N., and Lee, L. (1993). Distributional clustering of English words. *Proceedings of the 31st conference on Association for Computational Linguistics*, pages 183–190.

Ranzato, M. and Szummer, M. (2008). Semi-supervised learning of compact document representations with deep networks. In *ICML*, pages 792–799.

Rennie, J. D. M. and Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 713–719.

Salakhutdinov, R. and Mnih, A. (2008a). Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the International Conference on Machine Learning*, volume 25.

Salakhutdinov, R. and Mnih, A. (2008b). Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the Twenty-fourth International Conference on Machine Learning*.

Saul, L. and Pereira, F. (1997). Aggregate and mixed-order Markov models for statistical language processing. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 81–89.

Schwenk, H. (2007). Continuous space language models. *Computer Speech & Language*, 21(3):492–518.

Schwenk, H. and Gauvain, J. (2002). Connectionist language modeling for large vocabulary continuous speech recognition. *Acoustics, Speech, and Signal Processing, 2002. Proceedings.(ICASSP'02). IEEE International Conference on*, 1.

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

Srebro, N. and Jaakkola, T. (2003). Weighted low-rank approximations. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 720–727.

Srebro, N., Rennie, J. D. M., and Jaakkola, T. (2004). Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems*.

Stolcke, A. (2002). SRILM – an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, volume 2, pages 901–904.

Sutskever, I. and Hinton, G. E. (2007). Learning multilevel distributed representations for high-dimensional sequences. In *AISTATS'07*.

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2008). Matrix factorization and neighbor based algorithms for the netflix prize problem. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 267–274. ACM New York, NY, USA.

Truyen, T. T., Phung, D. Q., and Venkatesh, S. (2009). Ordinal Boltzmann machines for collaborative filtering. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*.

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

Welling, M., Rosen-Zvi, M., and Hinton, G. (2005). Exponential family harmoniums with an application to information retrieval. In *NIPS 17*, pages 1481–1488.

Zhu, S., Yu, K., and Gong, Y. (2008). Stochastic relational models for large-scale dyadic data using MCMC. In *Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems*, pages 1993–2000.