Three New Graphical Models for Statistical Language Modelling

Andriy Mnih AMNIH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Canada

Abstract

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. We propose three new probabilistic language models that define the distribution of the next word in a sequence given several preceding words by using distributed representations of those words. We show how real-valued distributed representations for words can be learned at the same time as learning a large set of stochastic binary hidden features that are used to predict the distributed representation of the next word from previous distributed representations. Adding connections from the previous states of the binary hidden features improves performance as does adding direct connections between the real-valued distributed representations. One of our models significantly outperforms the very best n-gram models.

1. Introduction

One of the main tasks of statistical language modelling is learning probability distributions of word sequences. This problem is usually reduced to learning the conditional distribution of the next word given a fixed number of preceding words, a task at which n-gram models have been very successful (Chen & Goodman, 1996). Density estimation for discrete distributions is inherently difficult because there is no simple way to do smoothing based on input similarity. Since all discrete values are equally similar (or dissimilar) assigning similar probabilities to similar inputs, which is typically done for continuous inputs, does not work in the discrete case. Representing discrete structures such as words using continuous-valued distributed representations and then assigning probability to these structures based on their representations automatically introduces smoothing into the density estimation problem making the data sparsity problem less severe. Recently, significant progress in statistical language modelling has been made by using models that rely on such distributed representations. Feed-forward neural networks that operate on real-valued vector representations of words have been both the most popular and most successful models of this type (Bengio et al., 2003; Morin & Bengio, 2005; Emami et al., 2003). A number of techniques have been proposed to address the main drawback of these models – their long training times (Bengio & Senécal, 2003; Schwenk & Gauvain, 2005; Morin & Bengio, 2005). Hierarchical alternatives to feed-forward networks, which are faster to train and use, have been considered (Morin & Bengio, 2005; Blitzer et al., 2005b), but they do not perform quite as well. Recently, a stochastic model with hidden variables has been proposed for language modelling (Blitzer et al., 2005a). Unlike the previous models, it uses distributed representations that consist of stochastic binary variables as opposed to real numbers. Unfortunately, this model does not scale well to large vocabulary sizes due to the difficulty of inference in the model.

In this paper, we describe three new probabilistic language models that use distributed word representations to define the distribution of the next word in a sequence given several preceding words. We start with an undirected graphical model that uses a large number of hidden binary variables to capture the desired conditional distribution. Then we augment it with temporal connections between hidden units to increase the number of preceding words taken into account without significantly increasing the number of model parameters. Finally, we investigate a model that predicts the distributed representation for the next word using a linear function of the distributed representations of the preceding words, without using

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

any additional latent variables.

2. The Factored Restricted Boltzmann Machine Language Model

Our goal is to design 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) have proposed a model with directed interactions for this task. Unfortunately, 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 finite dictionary of size N_w, we model the observed words as multinomial random variables that take on N_w values.

To define an RBM model for the probability distribution of the next word in a sentence given the word's context $w_{1:n-1}$, which is the previous $n-1$ words $w_1, ..., w_{n-1}$, we must first specify the energy function for a joint configuration of the visible and hidden units. The simplest choice is probably

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

where v_i is a binary column vector of length N_w with 1 in the w_i^{th} position and zeros everywhere else, and h is a column vector of length N_h containing the configuration of the hidden variables. Here matrix G_i specifies the interaction between the multinomial[1] visible unit v_i and the binary hidden units. For simplicity, we have ignored the bias terms – the ones that depend only on h or only on v_n. Unfortunately, this parameterization requires nN_wN_h free parameters which can be unacceptably large for vocabularies of even moderate size. Another weakness of this model is that each word is associated with a different set of parameters for each of the n positions it can occupy.

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 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 dictionary. 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

$$E(w_n, h; w_{1:n-1}) = -(\sum_{i=1}^{n} v_i^T R W_i) h - b_h^T h - b_r^T R^T v_n - b_v^T v_n. \qquad (2)$$

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 the vectors b_v and b_r contain biases for words and word features respectively.[2] 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 as well as further reducing the number of model parameters. With 18,000

[1] Technically, we do not have to assume any distribution for $w_1, ..., w_{n-1}$ since we always condition on these random variables. As a result, the energy function can depend on them in an arbitrary, nonlinear, way.

[2] The inclusion of per-word biases contradicts our philosophy of using only the feature vectors to define the energy function. However, since the number of these bias parameters is small compared to the total number of parameters in the model, generalization is not negatively affected. In fact the inclusion of per-word biases does not seem to affect generalization but does speed up the early stages of learning.

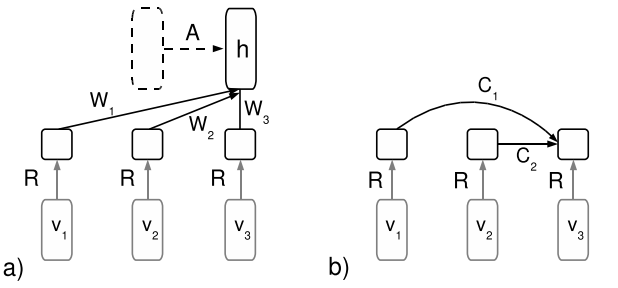

Figure 1. a) The diagram for the Factored RBM and the Temporal Factored RBM. The dashed part is included only for the TFRBM. b) The diagram for the log-bilinear model.

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. 1 and 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. 2 as

$$P(w_n, h|w_{1:n-1}) = \frac{1}{Z_c} \exp(-E(w_n, h; w_{1:n-1})), \quad (3)$$

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:

$$P(w_n|w_{1:n-1}) = \frac{1}{Z_c} \sum_h \exp(-E(w_n, h; w_{1:n-1})). \quad (4)$$

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.

2.1. 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 hidden units (Hinton, 2002). The normalizing constant, however, is usually infeasible to compute because it is a sum of the 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 v_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

$$P(w_n|w_{1:n-1}) \propto \exp(-b_r^T R^T v_n - b_v^T v_n) \prod_i (1 + \exp(T_i)), \quad (5)$$

where T_i is the total input to the hidden unit i when $w_1, ..., w_n$ is the input to the model.

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

2.2. Learning

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

$$L(D) = \sum \log P(w_n|w_{1:n-1}), \quad (6)$$

where P is defined by the model and the sum is over all word subsequences $w_1, ..., w_n$ of length n in the dataset. $L(D)$ can be maximized w.r.t. model parameters using gradient ascent. The contributions made by a subsequence $w_1, ..., w_n$ from D to the derivatives of $L(D)$ are given by

$$\frac{\partial}{\partial R} \log P(w_n|w_{1:n-1}) = \left\langle \sum_{i=1}^{n} v_i h^T W_i^T + v_n b_r^T \right\rangle_D - \left\langle \sum_{i=1}^{n} v_i h^T W_i^T + v_n b_r^T \right\rangle_M, \quad (7)$$

$$\frac{\partial}{\partial W_i} \log P(w_n|w_{1:n-1}) = \left\langle R^T v_i h^T \right\rangle_D - \left\langle R^T v_i h^T \right\rangle_M, \quad (8)$$

where $\langle\cdot\rangle_D$ and $\langle\cdot\rangle_M$ denote expectations w.r.t. distributions $P(h|w_{1:n})$ and $P(v_n, h|w_{1:n-1})$ respectively. The derivative 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 derivatives exactly can be computationally expensive because computing an expectation w.r.t. $P(v_n, h|w_{1:n-1})$ takes $O(N_w N_h)$ 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 the expression we are interested in 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 starting v_n and h in some random configuration and alternating between sampling v_n and h from their respective conditional distributions given by

$$P(w_n|h, w_{1:n-1}) \propto \exp((h^T W_n^T + b_r^T)R^T v_n + b_v^T v_n), \quad (9)$$

$$P(h|w_{1:n}) \propto \exp((\sum_{i=1}^{n} v_i^T R W_i + b_h^T)h). \quad (10)$$

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

Fortunately, there is 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 learning method. First, instead of starting v_n in a random configuration, we initialize v_n with 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) (called a "confabulation" or "reconstruction") is not a sample from $P(v_n, h|w_{1:n-1})$, it has been shown empirically that learning still works well when confabulations are used instead of samples from $P(v_n, h|w_{1:n-1})$ in the learning rules given above (Hinton, 2002).

In this paper, we train all our models that have hidden variables 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 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. The Temporal Factored RBM

The language model proposed above, like virtually all statistical language models, is based on the assumption that given a word's context, which is the $n-1$ words that immediately precede it, 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-grams, for example, the number of parameters is exponential in context size, which makes n-grams unsuitable for handling 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, for larger context sizes this number might still be very large.

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 factored RBM language model to achieve that goal (at least in theory) 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.

3.1. Making Predictions

Exact inference in the resulting model is intractable – it takes time exponential in the number of hidden variables (N_h) in the model being instantiated. However, since predicting the next word given its (near) infinite context is an online problem we take the filter-

ing approach to making this prediction, which requires storing only the last $n-1$ words in the sequence. In contrast, exact inference requires that the whole context be stored, which might be infeasible or undesirable.

Unfortunately, exact filtering is also intractable in this model. 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 $p^\tau_j = P(h^\tau_j = 1|w_{1:\tau+n-1})$ 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. 10. Given the (factorial) posterior for model instance τ, the posterior for model instance $\tau+1$ is computed as

$$P(h^{\tau+1}|w_{1:\tau+n}) \propto \exp((\sum_{i=1}^{n} v_i^T R W_i + (b_h + Ap^\tau)^T)h^{\tau+1}). \quad (11)$$

Thus, computing the posterior for model instance $\tau+1$ amounts to adding Ap^τ to that model's vector of biases for hidden units and performing inference in the resulting model using Eq. 10.

Finally, the predictive distribution over w_n is obtained by applying the procedure described in Section 2.1 to model instance t after shifting its biases appropriately.

3.2. Learning

Maximum likelihood learning in the temporal FRBM model is intractable because it requires performing 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 learned using the following rule applied to each training sequence separately:

$$\Delta A \propto \sum_{\tau=1}^{t-1} (p^{\tau+1} - \hat{p}^{\tau+1})^T p^\tau. \quad (12)$$

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

4. A Log-Bilinear Language Model

An alternative to using stochastic binary variables for modelling the conditional distribution of the next word given several previous words is to directly parameterize the distribution and thus avoid introducing stochastic hidden variables altogether. As before, we start by specifying the energy function:

$$E(w_n; w_{1:n-1}) = -\left(\sum_{i=1}^{n-1} v_i^T R C_i\right) R^T v_n - b_r^T R^T v_n - b_v^T v_n. \quad (13)$$

Here C_i specifies the interaction between the feature vector of w_i and the feature vector of w_n, while b_r and b_v specify the word biases as in Eq. 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. Intuitively, the model predicts a feature vector for the next word by computing a linear function of the context word feature vectors. Then it assigns probabilities to all words in the vocabulary based on the similarity of their feature vectors to the predicted feature vector as measured by the dot product. The resulting predictive distribution is given by $P(w_n|w_{1:n-1}) = \frac{1}{Z_c}\exp(-E(w_n; w_{1:n-1}))$, where $Z_c = \sum_{w_n}\exp(-E(w_n; w_{1:n-1}))$.

This model is similar to the energy-based model proposed in (Bengio et al., 2003). However, our model uses a bilinear energy function while their energy function is a one-hidden-layer neural network.

4.1. Learning

Training the above model is considerably simpler and faster than training the FRBM models because no stochastic hidden variables are involved. The gradients required for maximum likelihood learning are given by

$$\frac{\partial}{\partial C_i}\log P(w_n|w_{1:n-1}) = \left\langle R^T v_i v_n^T R\right\rangle_D - \left\langle R^T v_i v_n^T R\right\rangle_M, \quad (14)$$

Table 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 Kneser-Ney 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 back-off 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	

$$\frac{\partial}{\partial R}\log P(w_n|w_{1:n-1}) = \left\langle \sum_{i=1}^{n-1}(v_n v_i^T RC_i + v_i v_n^T RC_i^T) + v_n b_r^T \right\rangle_D - \left\langle \sum_{i=1}^{n-1}(v_n v_i^T RC_i + v_i v_n^T RC_i^T) + v_n b_r^T \right\rangle_M . \quad (15)$$

Note that averaging over the model distribution in this case is the same as averaging over the predictive distribution over v_n. As a result, these gradients are faster to compute than the gradients for a FRBM.

5. Experimental Results

We evaluated our models using the Associated Press News (APNews) dataset consisting of a text stream of about 16 million words. The dataset has been preprocessed by replacing proper nouns and rare words with the special "proper noun" and "unknown word" symbols respectively, while keeping all the punctuation marks, resulting in 17964 unique 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 to the n-gram models using the full dataset.

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 stochastic 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. During 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 Section 2.2, 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.

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

The temporal FRBM model was initialized by copying the parameters from the trained FRBM and initializing the temporal parameters (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).

For all models we used weight decay of 10^{-4} for word representations and 10^{-5} for all other weights. No weight decay was applied to network biases. Weight decay values as well as other learning parameters were chosen using the validation set. Each model was trained until its performance on a subset of the validation set stopped improving. 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 n-gram models estimated using Good-Turing and modified Kneser-Ney discounting. Training and testing of the n-gram models was performed using programs from the SRI Language Modelling toolkit (Stolcke, 2002). To make the comparison fair, the n-gram models treated punctuation marks (including full stops) is if they were ordinary

words, since that is how they were treated by the network models.

Models were compared based on their perplexity on the test set. Perplexity is a standard performance measure for probabilistic language models, which is computed as

$$\mathcal{P} = \exp(-\frac{1}{N}\sum_{w_{1:n}} \log P(w_n|w_{1:n-1})), \qquad (16)$$

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 nth word in the subsequence given the previous $n-1$ words. 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 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 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). The resulting model perplexities are given in Table 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. Adding temporal connections to it, however, 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 complement each other in at least some cases. The best results were obtained by averaging with the temporal network model, resulting in 21% reduction in perplexity over the best n-gram model.

Since the log-bilinear models performed well and, compared to the FRBMs, were fast to train, we used only log-bilinear models in the second experiment. Similarly, we chose to use n-grams with Kneser-Ney discounting as they significantly outperformed the n-grams with Good-Turing 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.

Table 2. Perplexity scores for the models trained on the 14M word training set. The mixture test score is the perplexity obtained by averaging the model's predictions with those of the Kneser-Ney 5-gram model. The log-bilinear models use 100-dimensional feature vectors.

Model type	Context size	Model test score	Mixture test score
Log-bilinear	5	117.0	97.3
Log-bilinear	10	107.8	92.1
Back-off KN3	2	129.8	
Back-off KN5	4	123.2	
Back-off KN6	5	123.5	
Back-off KN9	8	124.6	

We trained two log-bilinear models: one with a context of size 5, the other with a context of size 10. The training parameters were the same as in the first experiment with the exception of the learning rate, which was annealed to a lower value than before.[3] The results of the second experiments are summarized in Table 2. The perplexity scores clearly show that the log-bilinear models outperform the n-gram models. Even the smaller log-bilinear model, with a context of size 5, outperforms all the n-gram models. The table also shows using n-grams with a larger context size does not necessarily lead to better results. In fact, the best performance on this dataset was achieved for the context size of 4. Log-bilinear models, on the other hand, do benefit from larger context sizes: increasing the context size from 5 to 10 decreases the model 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[4]. In (Bengio et al., 2003), a neural language model with a context size of 5 is trained but its individual score on the test set is not reported. Instead the score of 109 is reported for the mixture of a Kneser-Ney 5-gram model and the neural language

[3]This difference is one of the reasons for the log-bilinear model with a context of size 5 performing considerably better in the second experiment than in the first one. The increased training set size and the different test set are unlikely be the only reasons because the n-gram models actually performed slightly better in the first experiment.

[4]Due to 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. That score is significantly worse than the score of 97.3 obtained by the corresponding mixture in our experiments. Moreover, our best single model has a score of 107.8 which means it performs at least as well as their mixture.

6. Discussion and Future Work

We have proposed three new probabilistic models for language one of which achieves state-of-the-art performance on the APNews dataset. In two of our models the interaction between the feature vectors for the context words and the feature vector for the next word is controlled by binary hidden variables. In the third model the interaction is modelled directly using a bilinear interaction between the feature vectors. This third model appears to be preferable to the other two models because it is considerably faster to train and make predictions with. We plan to combine these two interaction types in a single model, which, due to its greater representational power, should perform better than any of our current models.

Unlike other language models, with the exception of the energy based model in (Bengio et al., 2003), our models use distributed representations not only for context words, but for the word being predicted as well. This means that our models should be able to generalize well even for datasets with very large vocabularies. We intend to test this hypothesis by comparing our models to n-gram models on the APNews dataset without removing rare words first.

In this paper we have used only a single layer of hidden units, but now that we have shown how to use factored matrices to take care of the very large number of free parameters, it would be easy to make use of the greedy, multilayer learning algorithm for RBMs that was described in (Hinton et al., 2006).

Acknowledgements

We thank Yoshua Bengio for providing us with the APNews dataset. This research was funded by grants from NSERC, CIAR and CFI.

References

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

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

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

Blitzer, J., Weinberger, K., Saul, L., & Pereira, F. (2005b). Hierarchical distributed representations for statistical language modeling. *Advances in Neural Information Processing Systems 18*. Cambridge, MA: MIT Press.

Chen, S. F., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics* (pp. 310–318). San Francisco: Morgan Kaufmann Publishers.

Emami, A., Xu, P., & Jelinek, F. (2003). Using a connectionist model in a syntactical based language model. *Proceedings of ICASSP* (pp. 372–375).

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

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

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

Morin, F., & Bengio, Y. (2005). Hierarchical probabilistic neural network language model. *AISTATS'05* (pp. 246–252).

Schwenk, H., & Gauvain, J.-L. (2005). Training neural network language models on very large corpora. *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing* (pp. 201–208). Vancouver, Canada.

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

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