ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

Improving a statistical language model by modulating the effects of context words

Zhang Yuecheng, Andriy Mnih, Geoffrey Hinton

University of Toronto - Dept. of Computer Science
Toronto, Ontario, Canada

Abstract. We show how to improve a state-of-the-art neural network language model that converts the previous "context" words into feature vectors and combines these feature vectors to predict the feature vector of the next word. Significant improvements in predictive accuracy are achieved by using higher-level features to modulate the effects of the context words. This is more effective than using the higher-level features to directly predict the feature vector of the next word, but it is also possible to combine both methods.

1 Introduction

Given a sequence of n words, our goal is to design a statistical language model that uses distributed representations of words and exploits semantic and syntactic features of the first $n-1$ words to predict the n^{th} word. We will refer to the first $n-1$ words and the n^{th} word as the context words and the next word respectively. Our starting point is the log-bilinear model with a context size of 5 from [1] which we will call Model 0. Unlike other statistical language models based on distributed representations ([2], [3], [4], [5]), the log-bilinear model has been shown to outperform the best n-gram models on a large dataset [1] without resorting to model averaging.

This model learns to convert each word into a vector of real-valued features in such a way that the feature vectors of the $n-1$ context words are good at predicting the next word. The prediction is done by using $n-1$ different, learned weight matrices to linearly transform and combine the contextual feature vectors to produce a predicted feature vector, $\hat{r}$. The predicted probability of word i is defined to be proportional to $\exp(r_i^T \hat{r})$, where r_i is the distributed representation of word i.

It is convenient to define the conditional distribution over the next word w_n given the context words $w_1, ..., w_{n-1}$ by first specifying an energy function $E(w_n; w_{1:n-1})$ that assigns a score to each possible value of w_n. For notational convenience we stack the feature vectors for all the words in the dictionary to form a feature matrix R. We will represent the i^{th} word in the dictionary using a binary vector v with 1 in the i^{th} position and zeros in all other positions. Then the feature vector for word i is given by $R^T v$ and the energy function specifying

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

Model 0 can be written as[1]

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

where $\{C_i\}$ are the weight matrices used to combine the context feature vectors to obtain the predicted feature vector $\hat{r} = \sum_{i=1}^{n-1} C_i^T R^T v_i$. The distribution over w_n is then given by

$$P(w_n; w_{1:n-1}) = \frac{\exp(-E(w_n; w_{1:n-1})}{\sum_{w_n} \exp(-E(w_n; w_{1:n-1})}, \quad (2)$$

where the sum is over all the words in the dictionary. The matrix R containing word feature vectors and the weight matrices $\{C_i\}$ used to combine feature vectors are all learned by maximizing the log-likelihood using steepest descent.

As shown in [1], Model 0 is a very good language model, significantly beating the very best n-gram models [6] as well as the neural probabilistic language model of Bengio et al. [2], and our aim was to make it still better by removing some of its obvious limitations without adding too many free parameters.

2 Extending the log-bilinear model

A major limitation of Model 0 is that interactions between context words are not taken into account when computing the conditional probability of the next word. For example, in the 6-word[2] sequence, *"was Tuesday . We gathered together"*, due to the period, *"was"* and *"Tuesday"* are much less important than *"We gathered"* for predicting the next word. Hence, an effective model should reduce the effects of *"was"* and *"Tuesday"* and amplify the effects of *"We"* and *"gathered"* on the probability of the next word.

2.1 Model 1

To address this problem, we elaborate Model 0 by adding a gating network with one hidden layer that produces weighting coefficients, one per context word. Each coefficient represents the relative importance of the corresponding context word. The hidden layer in the gating network receives inputs from all the feature vectors of context words and the learned higher-order features are used to compute a weighting coefficient for each context word. The predictive effect of each context word is weighted by its coefficient before being incorporated into the predicted representation for the next word.

Let f be the vector obtained by concatenating the feature vectors for the context words in the order they appear in the context. The activity of the hidden unit h_j is a logistic function of its total input: $h_j = 1/(1+\exp(-\sum_k f_k A_{kj} - a_j))$,

[1] To simplify the notation we omitted the bias terms in the energy function. The complete energy function is given in [1].

[2] We treat punctuation marks as special words.

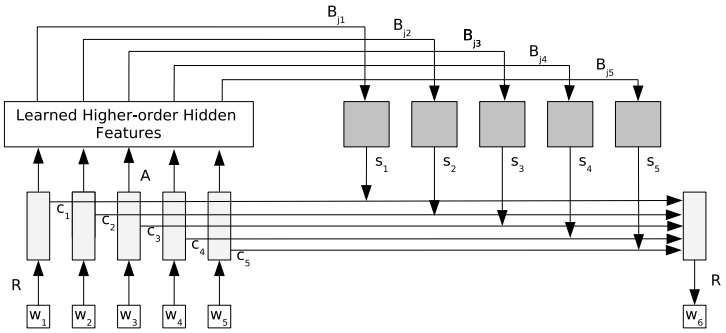

Fig. 1: The diagram for the model that uses a gating network to modulate the effects of context words (Model 1).

where A_{kj} is the weight on the connection from k to j, and a_j is the bias of unit j. The activity of a gating unit s_i is a logistic function of the total input it receives from the hidden layer, scaled by a factor of 2 to make the gating unit output 1 when its total input is 0: $s_i = 2/(1 + \exp(-\sum_j h_j B_{ji} - b_i))$, where B_{ji} is the weight on the connection from j to i, and b_i is the bias of the gating unit i. The gating unit activities are incorporated into the energy function as follows:

$$E(w_n; w_{1:n-1}) = -\left(\sum_{i=1}^{n-1} v_i^T R C_i s_i\right) R^T v_n, \tag{3}$$

where C_i represents the interaction between the features of the i^{th} context word and the features of the candidate word w_n. As before, the predictive distribution $P(w_n; w_{1:n-1})$ is obtained applying Eq. 1 to the energy function.

2.2 Model 2

Our second model extension also uses a hidden layer to capture the interactions between the context words when predicting the next word. However, instead of using the hidden layer as a part of a gating network, we use a linear combination of the hidden unit activities as an additional term in the predicted feature vector of the next word. Thus, this approach makes the predicted feature vector a fairly general non-linear function of the context feature vectors by making the energy function take to following form:

$$E(w_n; w_{1:n-1}) = -\left(\sum_{i=1}^{n-1} v_i^T R C_i + hS\right) R^T v_n. \tag{4}$$

Here h represents the vector of hidden unit activities and S is the matrix of weights between h and the predicted feature vector for the next word.

Table 1: Test set perplexity scores for the models trained on the APNews dataset. The reduction of perplexity for each model relative to the log-bilinear model is also shown. The mixture test perplexity was computed by averaging the predictions of the model with those of the Kneser-Ney 5-gram model. The score for NPLM, the language model of Bengio et al., is from [2]. All network models in the comparison have a context size of 5.

Model type	Number of hidden units	Test set perplexity	Perplexity reduction	Mixture test perplexity
KN 5-gram	–	123.2	−5.3%	123.2
NPLM Model	–	–	–	109
Model 0	–	117.0	–	97.3
Model 1	100	113.9	2.7%	96.0
Model 1	200	107.3	8.3%	93.3
Model 1	300	105.9	9.5%	92.7
Model 1	500	104.3	10.8%	91.9
Model 2	100	113.9	2.7%	96.0
Model 2	200	112.8	3.6%	95.7
Model 2	300	112.7	3.5%	95.9
Model 2	500	110.1	5.9%	96.9
Model 3	100	110.1	5.9%	94.5
Model 3	200	109.4	6.5%	94.3
Model 3	300	109.2	6.7%	93.6
Model 3	500	109.0	6.8%	93.6

2.3 Model 3

If each of the two extensions above improves model performance, it might be possible to achieve further performance gains by incorporating both extensions into a single model. Model 3 combines the gating network of Model 1 with the general non-linear prediction of the feature vector for the next word used by Model 2. The following energy function includes both extensions:

$$E(w_n; w_{1:n-1}) = -\left(\sum_{i=1}^{n-1} v_i^T R C_i s_i + hS\right) R^T v_n. \tag{5}$$

3 Experimental results

We trained our models on 6-word sequences from the Associated Press News (APNews) dataset used in [1]. The dataset has been preprocessed by replacing all proper nouns and rare words with special symbols, reducing the number of unique words to 17964. For a more detailed description of the preprocessing procedure see [2]. The dataset was split into a training set of 14 million words, a validation set of 1 million words, and a test set of 1 million words.

ESANN'2008 proceedings, European Symposium on
Artificial Neural Networks - Advances in Computational Intelligence and Learning
Bruges (Belgium), 23-25 April 2008, d-side publi., ISBN 2-930307-08-0.

We used the log-bilinear model (Model 0) with 100-dimensional word feature vectors and a context size of 5 from [1] both as a baseline for comparison as well as the starting point from which we trained our extended models. That is, in all of our experiments we initialized R and $\{C_i\}$ by copying the corresponding values from Model 0. The remaining parameters of Model 1 were initialized as follows: the weights A on the connections between the context feature vectors and the hidden units were set to small random values, while all other parameters not present in Model 0 (i.e. B, a, b) were initialized to zero. This initialization procedure ensured that initially Model 1 weighted all context words equally by setting $s_i = 1$ and hence produced exactly the same predictions as Model 0. Steepest descent was then used to learn all model parameters except for the word features (R) which were kept fixed to reduce the time required to reach convergence. Weight decay of 10^{-5} was applied to all parameters, including biases. We used the following learning rates: 10^{-3} for C, 10^{-1} for A, and 10^{-5} for B. Momentum of 0.5 was used for updating context weights, while for all other parameters the value of momentum depended on the number of hidden units: 0.9 was used for 100 hidden units, 0.8 for 200, 0.8 for 300, and 0.5 for 500.

We considered two approaches to training Model 2. The first approach involved copying the weights A between the context feature vectors and the hidden units from the trained Model 1 and learning only the weights S from the hidden units to the predicted feature vector. The second approach involved learning A, S, and C after initializing A to small random values. In both cases S and all the biases were initialized to zero. We compared the speed with which the two approaches reduced the perplexity of a validation set using a model with 100 hidden units. A learning rate of 10^{-5}, momentum of 0.5, and weight decay of 10^{-4} were used for S in both experiments. In the second method, the same learning rate, momentum, and weight decay as in the experiments for Model 1 were used for A and C. Our experiments showed that the first approach reduced model perplexity twice as quickly as the second one, which lead us to use the first approach when training various instances of Model 2 for the main comparison.

In the experiments with Model 3, we used the same initialization and learning parameters for C, A, and B as for Model 1 and the same initialization and learning parameters for S as for Model 2.

In all of our experiments parameters were updated after processing each mini-batch of 1000 training cases. Each model was trained until its performance on the validation set stopped improving.

Table 1 shows the test set perplexities for the trained models. We report two perplexity scores as the baselines for comparison. First is the score for the back-off 5-gram with modified Kneser-Ney discounting fitted using the SRILM toolkit [7], which is the best-performing standard n-gram model [6] on this dataset. The second baseline score is for the log-bilinear model from [1] (Model 0) which achieves state-of-the-art performance, outperforming the n-gram model by over 5%. The results show that both extensions to the log-bilinear model result in a reduction in perplexity. Interestingly, when the number of hidden units is at least 200, adding the gating network to Model 0 leads to greater reductions

in perplexity than introducing a seemingly more general second extension. The best results are achieved adding a gating network with 500 hidden units to Model 0, which reduces its perplexity by almost 11%. The perplexity of the resulting model is over 15% lower than the perplexity of the 5-gram. Using both extensions in the same model does not lead to significant reductions in perplexity. In fact, in most cases a model with both extensions does not perform as well as a model that only has a gating network (with the same number of hidden units). For all model types increasing the number of hidden units lead to better predictive performance. We also included the score for the neural probabilistic language model (NPLM) of Bengio at el. from [2]. Since that paper includes the score for the mixture of the model and an 5-gram on the APNews dataset but not the score for the NPLM on its own, we computed the corresponding mixture scores for our models as well. The scores show that when mixed with a 5-gram our models outperform the corresponding NPLM mixture, sometimes by as much as 15%. More impressively, Model 1 with 200 or more hidden units, even when not mixed with a 5-gram, outperforms the NPLM mixture.

4 Discussion

Our results suggest that using a gating network to modulate the effects of the context words is a very effective way to improve the performance of a neural network language model. Incorporating a gating network into a log-bilinear model resulted in a language model that had 15% lower test set perplexity than the best n-gram model. Surprisingly, using a more general model to capture the interaction between the context words when predicting the next word appears to be counterproductive as it does not usually lead to an improvement over a model that modulates the context word effects.

References

[1] Andriy Mnih and Geoffrey E. Hinton. Three new graphical models for statistical language modelling. In *ICML*, pages 641–648, 2007.

[2] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[3] John Blitzer, Kilian Weinberger, Lawrence Saul, and Fernando Pereira. Hierarchical distributed representations for statistical language modeling. In *Advances in Neural Information Processing Systems 18*, Cambridge, MA, 2005. MIT Press.

[4] John Blitzer, Amir Globerson, and Fernando Pereira. Distributed latent variable models of lexical co-occurrences. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, January 2005.

[5] Holger Schwenk and Jean-Luc Gauvain. Training neural network language models on very large corpora. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 201–208, 2005.

[6] Stanley F. Chen and Joshua Goodman. 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, San Francisco, 1996.

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