NEURAL NETWORK BASED LANGUAGE MODELS FOR HIGHLY INFLECTIVE LANGUAGES

Tomáš Mikolov, Jiří Kopecký, Lukáš Burget, Ondřej Glembek and Jan "Honza" Černocký

Speech@FIT, Faculty of Information Technology, Brno University of Technology, Czech Republic
{imikolov|kopecky|burget|glembek|cernocky}@fit.vutbr.cz

ABSTRACT

Speech recognition of inflectional and morphologically rich languages like Czech is currently quite a challenging task, because simple n-gram techniques are unable to capture important regularities in the data. Several possible solutions were proposed, namely class based models, factored models, decision trees and neural networks. This paper describes improvements obtained in recognition of spoken Czech lectures using language models based on neural networks. Relative reductions in word error rate are more than 15% over baseline obtained with adapted 4-gram backoff language model using modified Kneser-Ney smoothing.

Index Terms— language modeling, neural networks, inflective languages.

1. INTRODUCTION

Statistical language models play important role in many applications, like machine translation, optical character recognition and speech recognition. The ultimate goal in language modeling is language understanding - however, current techniques are still quite far from it. The basic reason is that models like backoff n-grams can not describe some relationships in the data effectively. The other reason is that advanced techniques that are able to describe these relationships are computationally expensive and their training is still a big problem - even if the existence of solution is guaranteed, it may be practically impossible to find it using naive approaches. So in practice, basic trigrams are still commonly used for practical reasons.

On the other hand, n-gram backoff models do not work as well for inflectional languages as they work for English. It is easy to see that if we place several independent data into one symbol (like stem and ending to one word), n-grams will require much more parameters to estimate than is actually needed. This leads to techniques that try to decompose words into morphological parts. Still, this step may require development of a tagger and one may ask, whether it is actually needed.

The original idea of using neural networks that project words onto continuous space to reduce number of parameters that are needed to be estimated comes from Bengio [1]. Probably first results in speech recognition were produced by Schwenk [2]. In our approach, we use neural network to learn word structure in unsupervised manner, while learning bigram model. This structure is then used to learn n-gram neural network model. While computational requirements for neural network training are quite high, we train it on all data that are available and relevant for our task - recognition of spontaneous spoken lectures. Neural networks can be then applied to speech recognition in two ways: n-best list re-scoring (or lattice rescoring) and additional data generation. In this paper, we will discuss n-best list re-scoring, as it gives us the best results. Additional data generation by neural network, which can be seen as conversion of neural network model to standard backoff n-gram model, provides worse results, but is easier to use in practical recognizers - there is no need for re-scoring.

2. ARCHITECTURE

The first step in our architecture is training of bigram neural network. The task is relatively simple: given word w from vocabulary V, estimate probability distribution of the next word in text. We use neural network with input and output layer of the size $|V|$, where word w in the input layer uses "1 of n" coding (all input neurons are set to 0 except the one that corresponds to word w, which is set to 1). One hidden layer of variable size (usually 20-60 neurons) ensures that words that predict similar probability distribution in the output layer will share some of this distribution, because they will be automatically placed "near" in the highly dimensional space. Without hidden layer, no clustering would occur. In the hidden layer, sigmoid activation function is used. The output layer uses softmax that normalizes values of output neurons to sum up to 1.

2.1. Training algorithm

For training, standard back-propagation algorithm is used. Although [1] and [2] suggest that using weight decay is

This work was partly supported by Ministry of Trade and Commerce of Czech Republic under project FT-TA3/006 and by Ministry of Interior of Czech Republic under project VD20072010B16. The hardware used in this work was partially provided by CESNET under project No. 201/2006.

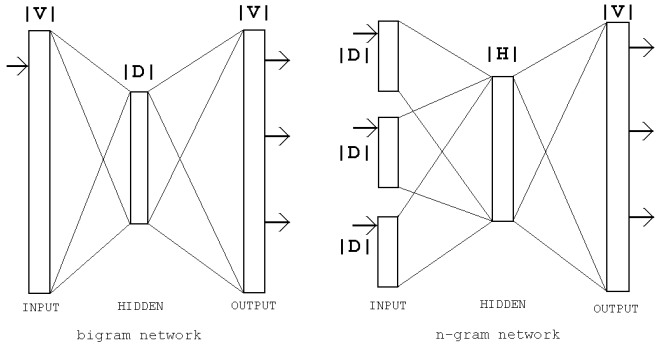

Fig. 1. Neural network architecture.

needed, in our early experiments weight decay improved results only slightly and we decided not to use it, because it requires additional parameter tuning.

After probability distribution is computed in the output layer (normalization is ensured by softmax), error vector is generated as $\mathbf{E} = \mathbf{D} - \mathbf{O}$, where $\mathbf{E}$ is error vector, $\mathbf{O}$ output vector and $\mathbf{D}$ desired vector. Vector $\mathbf{D}$ is encoded as "1 of n" and represents the next word. Error is then back-propagated through network and weights are modified according to standard back-propagation algorithm.

After each training epoch, neural network is used to compute probability of the validation data. If the probability increases, training continues. If the probability does not improve significantly (less than 1%), learning rate is halved and training continues. After learning rate is halved once, it is automatically halved in each following training epoch. When probability of training data does not rise again, training is stopped to prevent over-fitting. Usually it takes 10-20 epochs to train the neural network. Starting learning rate is 0.1.

N-gram network is trained in the same way, except that the input vector does not encode just previous word in history using 1 of V coding, but is formed by using $N-1$ projections from bigram network. To compute projection of word w onto continuous space of dimensionality D, half of the bigram network (first two layers) is used to compute values in hidden layer. Values from hidden layer of bigram network are used to form input layer of n-gram network. As D is around 20-60, it is possible to represent input for 4-gram network using less than 200 input neurons. In the following experiments, we will describe results obtained with n-gram neural networks and 4-gram backoff LMs.

Note that Bengio [1] claims that training word features together with neural network language model is better than independent training of features and n-gram NN LM. In our approach, we consider the first network as a feature extractor and do not train it together with the n-gram net. We suppose that this can prevent word features from learning cache-like relationships in the data, which tend to heavily optimize perplexity, but not recognition accuracy (Goodman [8]).

Data source	# words
Spontaneous speech	5 001 000
Technical lecture data	2 084 000
Lecture transcriptions	185 000

Table 1. Training data.

To improve speed of training, all words that occur in the training data less times than some threshold value are merged into one token in the same way as [1] does. This step effectively reduces size of the vocabulary, speeding up the training phase usually $3\times$ - $5\times$ (depends on threshold value) with only small impact on the final perplexity: words that occur very rarely in the training data can be hardly mapped onto the continuous space correctly without any additional information (for example POS tags). In the following experiments, words that occur less than 5 times are merged into one symbol, unless stated otherwise. This step reduces the vocabulary size from 204K words to 56K, resulting in $4\times$ speed-up. By merging all words that occur less than 10 times, vocabulary can be further reduced to 36K.

3. TRAINING DATA AND TEST DATA

Language modeling of spontaneous lecture speech in Czech is a challenging task, because commonly used corpora are based on newspaper articles / web data and are almost useless in our case. We needed to cover specific speaking style, non-literary words and technical terms. We were able to cover all these domains to some degree, as can be seen in Table 1 (overally we have 6 different corpora to cover these domains). However, the amount of data is quite small.

Note that by using additional text corpora based on web data and written out-of-domain text, less than 0.5% improvement in accuracy of the final recognizer can be obtained, while size of the language models rises dramatically. Our baseline adapted 4-gram language model is obtained by interpolating language models based on each corpora using interpolation weights chosen to minimize perplexity on validation data. We obtain perplexity 366 and OOV rate 1.9%. Although using modified Kneser-Ney smoothing reduces perplexity significantly against models that use Good-Turing smoothing, we observed no significant improvement in WER. Still, we use modified KN smoothing for baseline models as it is considered as the best smoothing method.

As test data we used half of a Systems and signals lecture (1.1 hour, 873 sentences total). The same setup was already used by Oparin [7] who worked on morphological random forests.

For decoding, we used acoustic models trained on Speecon and Temic databases (for more details, see [5]) and in-house STK decoder that is able to handle large vocabularies.

4. EXPERIMENTS AND RESULTS

In our experiments, we run decoder with bigram backoff LM and produce first-pass lattices. These lattices are then expanded by 4-gram LM with Kneser-Ney smoothing using SRI LM toolkit [6] to obtain baseline results. Neural networks are used to re-score N-best lists generated from first pass lattices (N=500 unless stated otherwise) and their score is interpolated with baseline LM score, with λ value of 0.6 for NN LM score (λ value was determined in previous work [3] and works well in most cases, as NN based LMs work better than backoff LMs). Neural network LMs are always trained on the same data as standard backoff LMs to provide fair comparison.

We are not going to perform advanced study on perplexity, as we are more interested in recognition accuracy. As neural networks are trained on data that are not shuffled, they tend to overfit data that are at the end of training corpora (lecture transcriptions). Direct comparison between neural network (PPL 342) and standard unadapted 4-gram LM (PPL 528) shows big improvements in perplexity in favor of neural network. On the other hand, correctly weighted 4-gram LM (PPL 366) performs only slightly worse on perplexity than unadapted NN LM. Still, recognition accuracy is significantly higher with unadapted NN LM than with adapted 4-gram backoff LM. Simple experiments with NN LM adaptation by creating separate models for each corpus lead nowhere: while perplexity decreases as can be expected, recognition accuracy after interpolation with adapted LM decreases too. It seems that adapting just backoff LM is enough, as the final model is usually interpolation between NN LM and backoff LM. In previous work [3], we were able to obtain more than 20% improvement in PPL with NN LM against backoff LM when we trained models on homogeneous data.

4.1. Experiments with re-scoring

As our task involves relatively small amount of training data, it is possible to train neural network model directly on whole corpora. Still, our first experiments were performed on small subsets of the data, to better understand behavior of NN-based LMs. By adding more data, it is interesting to see that NN LMs provide more improvement (see Table 2).

In our experiments, we observed that the more training data we have, the bigger network must be used to obtain the best possible results. While for experiments with 0.1M and 0.2M words, 20 30 4 network configuration worked the best, for 1M words the best results were obtained with bigger networks.

Table 3 presents results obtained on full training data. As we expected, bigger networks work better. We performed experiments with order of n-gram NN language model - as can be seen, going from trigrams to 5-grams doesn't provide any improvement with 20 30 network configuration. While NN LMs were originally proposed to solve the data sparsity problem for long contexts [1], in our case, it is clear that improvement provided by NN LMs comes not from the long context, but from the short context information.

Model	Accuracy
0.1M words backoff LM	63.8%
20 30 4 neural network	64.8% (+1.0%)
0.2M words backoff LM	65.6%
20 30 4 neural network	67.2% (+1.6%)
30 50 4 neural network	67.3% (+1.7%)
1M words backoff LM	67.1%
10 15 4 neural network	68.8% (+1.7%)
20 30 4 neural network	69.5% (+2.4%)
30 50 4 neural network	69.9% (+2.8%)
50 80 4 neural network	69.9% (+2.8%)

Table 2. Models with various training data sizes and topologies: 20 30 4 neural network means that bigram NN has hidden layer with 20 neurons, n-gram NN 30 neurons and order of model is 4-gram. In these experiments, NNs are interpolated with 4-gram backoff baseline LM.

Model	Accuracy	Accuracy after interpolation
7.3M words LM (all data)	70.7%	70.7%
5 5 3 NN	69.1%	71.7% (+1.0%)
20 30 3 NN	71.4%	73.2% (+2.5%)
20 30 4 NN	71.2%	72.9% (+2.2%)
20 30 5 NN	71.6%	73.1% (+2.4%)
40 60 5 NN	71.9%	73.3% (+2.6%)
90 90 4 NN	72.6%	73.1% (+2.4%)
60 90 4 NN (mincount 10)	73.1%	73.9% (+3.2%)
60 150 4 NN (mincount 10)	72.9%	73.9% (+3.2%)

Table 3. Models trained on full data (except model 90 90 4), accuracies for standalone models and models interpolated with baseline 4-gram backoff LM.

While larger networks perform better, it takes much more time to train them - training 40 60 network is two times more time consuming than training 20 30 network. To compensate this, we tried to reduce vocabulary even more than we did in the previous experiments - instead of discarding words that occur less than 5 times, we discarded words that occur less than 10 times. This reduced vocabulary from 56K words to 36K. We expected that results should be a little worse, but this didn't happen: in fact, results of standalone models are a little better and after interpolation, we obtain the best results with these models. Possible explanation of this effect is that words that occur rarely require many parameters of the network to be estimated, and that cannot be done correctly with so little amount of training data. This assumption should be investigated in the future, as it may be possible to obtain even larger improvements than we present here. Other explanation might be that models with lesser vocabulary take big advatnage from using larger hidden layers.

Ultimately, we have taken four largest networks (see Table

N	RT	Accuracy
1	-	70.2%
10	0.07	73.3%
100	0.6	73.8%
500	2.0	73.9%
3000	5.0	74.0%

Table 4. Real time factor for re-scoring N-best lists with 60 150 4 network interpolated with 4-gram backoff LM; 1-best corresponds to best path in lattices decoded with bigram backoff model.

LM	Accuracy	LM size
backoff 4-gram (gzipped)	70.7%	37M
20 30 3 network	71.4%	17M
60 150 4 network (mincount 10)	72.9%	39M

Table 5. LM sizes.

3) and combined their score using loglinear interpolation. We assigned the same weight to each network (0.25 in this case). We re-scored 3000 best lists to obtain the best possible results - 74.6% accuracy. After adding information from backoff 4-gram baseline LM, accuracy didn't go much higher - only to 74.7% with the best interpolation weight of 0.05. This means that in our case adapted backoff LM is so much worse than mix of NN LMs that it is almost useless, even if it contains adaptation information that is unknown to NN LMs.

4.2. Real-time factor

Many promising language modeling techniques are not widely used because of their high computational requirements. Our implementation of training phase is not optimized in any way, except vocabulary size reduction by merging rare words. Training times are thus quite high: still, we are able to train full NN LM model in less than 2 weeks, the biggest 60 150 4 network was trained in approximately 4 weeks. More important factor for practical use is time needed to re-score N-best lists - see Table 4.

To obtain reasonable re-scoring times, two level cache was implemented. Since N-best variants of utterances share large amount of n-grams that need to be estimated, it is useful to remember already computed n-gram probabilities. The second level of cache is storing computed probability distributions - this cache is more memory hungry and so the number of stored entries is much smaller. With our cache model, we obtain cache hit 68% when rescoring 10-best lists and 92% when re-scoring 3000 best list.

Another interesting fact is that neural networks may take less disk space than conventional backoff LM (Table 5).

5. CONCLUSION

In our experiments, we compared adapted 4-gram backoff language model with modified Kneser-Ney smoothing that is currently considered as standard baseline language model against mix of neural networks. Absolute improvement 4.0% in accuracy (more than 15% reduction of WER) is quite high, in comparison to usual improvements obtained by using advanced language modeling techniques. We performed manual comparison of recognition results to see where neural networks help this much. It was no surprise that some improvement comes from fixing endings of words. Recognized text after neural network re-scoring is much more smooth and understandable than baseline results.

Neural networks seem to be very useful for tasks where we struggle with inflectional languages and limited data amounts. While performance of our implementation is not high, [2] presents various optimization techniques that lead to significant improvements in required computational time for training and lattice re-scoring.

Future work should be focused on speeding up training phase, which is currently our worst problem. To improve results further, it may be also meaningful to use morphological information, which can allow neural network to "understand" even rare words.

6. REFERENCES

[1] Yoshua Bengio, Réjean Ducharme and Pascal Vincent. 2003. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137-1155

[2] Holger Schwenk and Jean-Luc Gauvain. Training Neural Network Language Models On Very Large Corpora. in Proc. Joint Conference HLT/EMNLP, October 2005.

[3] Tomáš Mikolov: Language Modeling for Speech Recognition in Czech, Master's thesis, Brno, FIT BUT, 2007

[4] Tomáš Mikolov: Language Models for Automatic Speech Recognition of Czech Lectures, in Proc. EEICT 2008, Brno, `https://wis.fit.vutbr.cz/FIT/db/vav/view_pub.php?id=8749`

[5] Jiří Kopecký, Ondřej Glembek and Martin Karafiát: Advances in Acoustic Modeling for the Recognition of Czech, in Proc. Text, Speech and Dialogue (TSD) 2008, Brno, Czech Republic, September 2008, pp. 357 – 363.

[6] Andreas Stolcke: SRILM - an extensible language modeling toolkit. in Proc. ICASLP, Denver, Colorado, USA, 2002.

[7] Oparin Ilya, Glembek Ondřej, Burget Lukáš, Černocký Jan: Morphological random forests for language modeling of inflectional languages, in Proc. 2008 IEEE Workshop on Spoken Language Technology, Goa, IN, IEEESP, 2008

[8] Goodman Joshua T. (2001). A bit of progress in language modeling, extended version. Technical report MSR-TR-2001-72.