Recurrent Neural Network based Language Modeling in Meeting Recognition

Stefan Kombrink, Tomáš Mikolov, Martin Karafiát, Lukáš Burget

Speech@FIT, Brno University of Technology, Brno, Czech Republic

{kombrink,imikolov,karafiat,burget}@fit.vutbr.cz

Abstract

We use recurrent neural network (RNN) based language models to improve the BUT English meeting recognizer. On the baseline setup using the original language models we decrease word error rate (WER) more than 1% absolute by n-best list rescoring and language model adaptation. When n-gram language models are trained on the same moderately sized data set as the RNN models, improvements are higher yielding a system which performs comparable to the baseline. A noticeable improvement was observed with unsupervised adaptation of RNN models. Furthermore, we examine the influence of word history on WER and show how to speed-up rescoring by caching common prefix strings.

Index Terms: automatic speech recognition, language modeling, recurrent neural networks, rescoring, adaptation

1. Introduction

Neural network (NN) based language models as proposed in [12] have been continuously reported to perform well amongst other language modeling techniques. The best results on some smaller tasks were obtained by using recurrent NN-based language models [10], [11]. In RNNs, the feedback between hidden and input layer allows the hidden neurons to remember the history of previously processed words.

Neural networks in language modeling offer several advantages. In contrary to commonly used n-gram language models, smoothing is applied in an implicit way, and due to the projection of the entire vocabulary into a small hidden layer, semantically similar words get clustered. This explains, why n-gram counts of data sampled from the distribution defined by NN-based models could lead to better estimates for n-grams, which may have never been seen during training: Words get substituted by other words which the NN learned to be related. While no such relation could be learned by a standard n-gram model using the original sparse training data, we already showed in [1] how we can incorporate some of the improvements gained by RNN language models into systems using just standard n-gram language models: by generating a large amount of additional training data from the RNN distribution.

The purpose of this paper is to show, to what extent the current RNN language model is suitable for mass application in common LVCSR systems. We will show that the promising results of previously conducted experiments on smaller setups [10],[1] generalize to our state-of-the-art meeting recognizer and can be applied in fact in any other ASR system without too much effort. While RNN models effectively complement standard n-grams, they can be used also efficiently, even in systems where speed or memory consumption is an issue.

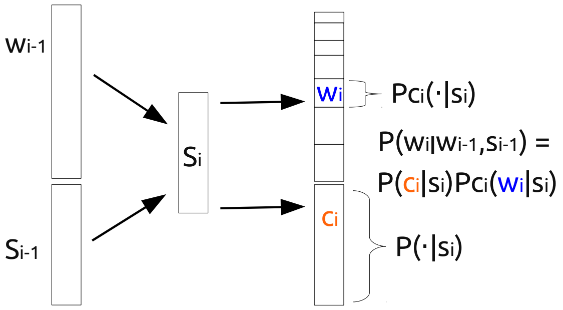

Figure 1: Architecture of the class-based recurrent NN.

In the following, we briefly introduce the utilized class-based RNN architecture for language modeling. A system description and details about used language models follows. Finally, we present our experiments in detail and conclude with a summary of our findings.

2. Class-based RNN language model

The RNN language model operates as a predictive model for the next word given the previous ones. As in n-gram models, the joint probability of a given sequence of words is factorized into the product of probability estimates of all words w_i conditioned on their history $h_i = w_1 w_2 ... w_{i-1}$:

$$P(w_1 w_2 ... w_n) = \prod_{i=1}^{n} P(w_i|h_i) \qquad (1)$$

The utilized RNN architecture is shown in figure 1. The previous word w_{i-1} is fed to the input of the net using 1-of-n encoding[1] together with the information encoded in the state vector s_{i-1} from processing the previous words. By propagating the input layer we obtain the updated state vector s_i so that we can write:

$$P(w_i|h_i) = P_{rnn}(w_i|w_{i-1}, s_{i-1}) = P_{rnn}(w_i|s_i) \qquad (2)$$

Usually, the posterior probability of the predicted word is estimated by using a softmax activation function on the output layer, which has the size of the vocabulary. The posterior probability for any given w_i can be read immediately from the corresponding output. Although often just the posterior probability of a particular w_i is required, the entire distribution has to

This work was partly supported by Technology Agency of the Czech Republic grant No. TA01011328, Czech Ministry of Education project No. MSM0021630528, Grant Agency of Czech Republic projects Nos. GP102/09/P635 and 102/08/0707, and by BUT FIT grant No. FIT-11-S-2.

[1] The input vector has the same dimensionality as the vocabulary size. All inputs are set to zero except the one corresponding to the word which is set to one.

be computed because of the softmax. By assuming, that words can be mapped to classes surjectively, we can add a part for estimating the posterior probability of classes to the output layer, and hence estimate the probability for the predicted word as the product of two independent probability distributions - one over classes and the other one over words within a class:

$$P_{rnn}(w_i|s_i) = P_{rnn}(c_i|s_i)P_{rnn}^{c_i}(w_i|s_i) \tag{3}$$

This leads to speed-up both in training and testing because only the distribution over classes and then the distribution over words belonging to the class c_i of the predicted word have to be computed [11].

3. Setup

3.1. System description

Our state-of-the-art baseline speech recognition system uses acoustic and language models from the AMIDA Rich Transcription 2009 system [9]. Standard speaker adaptation techniques (VTLN and per-speaker CMLLR), fMPE MPE trained acoustic models and NN-bottleneck features [4] with CVN/CMN and HLDA are used. The output of two complementary branches (one based on PLP and the other based on posterior features) served for cross-adapting the system. In both branches, lattices are generated using a 2-gram language model and subsequently expanded up to 4-gram order. The estimated adaptation transformations are used in a lattice rescoring stage, whose lattices finally serve as input to RNN rescoring as performed later in the experiments.

Corpus	Words	**RT09**	**RT11**	**RNN**
Web data	931M	✓	–	–
Hub4	152M	✓	33M	–
Fisher 1/2	21M	✓	✓	✓
Swbd/CHE	3.4M	✓	✓	✓
Meetings	2.1M	✓	✓	✓
Total	1.1G	1.1G	60M	26.5M

Table 1: Language models utilized in the LVCSR system

3.2. Language Models

In Table 1 we show the corpora[2] used for training the baseline language models. RT09 and RT11 were 4-gram models using modified Kneser-Ney smoothing, and shared the same vocabulary of 50k words. RNN was a class-based recurrent network model trained online with 13 iterations of backpropagation through time (BPTT, [3]) and a learning rate of 0.1. It used 500 hidden neurons, 1000 classes and full vocabulary (without cut-offs, 65k words). Using only a moderately sized subset[3] of 26.5M words one iteration took approximately three days on a single CPU. The rt06seval data set (30k words) served as validation data in model training and combination. In our experiments we report speech recognition results in WER on the NIST rt05seval and rt07seval sets.

4. Experiments

In our first experiment we kept the existing LVCSR setup and just replaced the old n-gram models by models that used artificial RNN-sampled data in addition. Hence, no RNN language model is required in this system.

4.1. Adding RNN-generated data

Model	PPL	Data	#n-grams
RT11	82.5	see Table 1	14.4M
VA	81.7	300M words from RNN	35.5M
RT11+VA	76.6	interpolated RT11+VA	46.5M
RT09	72.2	see Table 1	51.2M
RT09+VA	69.2	interpolated RT09+VA	78.6M

Table 2: Interpolated language model perplexities (4-grams)

We sampled 300M additional words from the RNN language model and used this data to create improved n-gram language models. In Table 2 we show an overview of all n-gram model combinations in decreasing order of perplexity (PPL). It can be seen that the LM trained on the RNN data (VA) performs already comparable to the RT11 model. Both models still seem to be complementary: the RT11+VA model is an equally weighted mixture of the RT11 and VA model and shows decreased PPL. Its model size is almost comparable to the RT09 model which uses much more data. When the RNN data is used in combination with the RT09 model (RT09+VA) PPL decreases just slightly whereas the growth in number of n-grams (78.6M) turns out to be huge.

As shown in table 3, by just replacing the original n-gram model by an improved n-gram model using sampled RNN data we can keep the original LVCSR setup and yet achieve some improvement. RNN data sampling decreases WER in case of the smaller RT11 model, but does not work for the RT09 model, which already uses plenty of training data. The RT09+VA model showed no improvement over the RT09 model which is why we did not use it in the following experiments at all.

Test set	RT11	RT11+VA	RT09	RT09+VA
rt07seval	22.2	21.5	20.3	20.4
rt05seval	19.0	18.5	17.7	17.7

Table 3: WER reduction due to the use of RNN sampled data

4.2. RNN rescoring

Further improvements are obtained by running a RNN rescoring stage. In n-best list rescoring, the RNN model re-estimated a log-likelihood score for each n-best hypothesis s:

$$\log L(s) = n \cdot wp + \sum_{i=1}^{n} asc_i + lms \sum_{i=1}^{n} \log P_x(w_i|h_i) \tag{4}$$

where n is the number of words, wp is the word insertion penalty, asc_i is the acoustic score for word w_i, h_i the history $w_1...w_{i-1}$ and lms the language model scale applied in the generation of the input lattices. P_x is the combined probability estimate of standard 4-gram and RNN models, which was obtained by linear interpolation:

$$P_x(w_i|h) = \lambda P_{rnn}(w_i|h) + (1-\lambda)P_{ng}(w_i|h) \tag{5}$$

[2] The web data actually consists of four separate data sets described more thoroughly in [8].

[3] AMI meetings + Fisher1/2 + CallHome English + Switchboard

rt07seval - 2.25 hours - 4527 utterances

n-gram model	baseline	RNN	Adapt
RT09	20.3	19.6	19.4
RT11+VA	21.5	20.5	20.2
RT11	22.2	20.7	20.4

rt05seval - 2.00 hours - 3130 utterances

n-gram model	baseline	RNN	Adapt
RT09	17.7	16.9	16.6
RT11+VA	18.5	17.4	17.1
RT11	19.0	17.4	17.2

Table 4: Word error rates (WERs) on the rt05seval and rt07seval test sets using RNN rescoring and adaptation

Table 4 shows the n-gram models used in the system and their performance gained by RNN rescoring. The 4-gram lattices (4-gram) constituted the baseline which was used to extract n-best lists. The improvement gained in our best system (RT09) is 0.7-0.8% absolute, in the system enhanced by RNN data sampling (RT11+VA) 1.0-1.1% and in the light-weight RT11 system up to 1.6%.

4.3. LM Adaptation

rt07seval - 8 meetings - 19 speakers

WER	Adaptation
19.6	RNN rescoring, no adaptation
19.7	on entire 1-best using one model
19.4	on 1-best per meeting using 8 models

rt05seval - 10 meetings - 50 speakers

WER	Adaptation
16.9	RNN rescoring, no adaptation
16.8	on entire 1-best using one model
16.6	on 1-best per meeting using 10 models

Table 5: Influence of data ordering in adaptation (RT09 system)

Earlier experiments have already shown potential improvements in case the language models gets adapted. The adaptation process for the RNN model is a one-iteration retraining on the 1-best output from the rescored n-best lists. In the following, rescoring is performed a second time using the adapted RNN model, and an improved recognition output is obtained. Two criteria were tried to determine the learning rate for RNN adaptation: best PPL on 1-best and best PPL on the validation data. The optimal learning rate[4] in terms of WER was often found in between the estimates using both criteria.

In Table 5 we compare two ways of adaptation using our best system. Adapting only one RNN model on the entire recognition output did not work reliably. But when we adapted one model per meeting and applied it in a second RNN rescoring to the respective n-best lists only, we obtained considerable improvements. As shown in Table 4 we decreased WER further between 0.2-0.3% on all system variants. In case the RT11 and RT11+VA models are used, the system obtains finally performs comparable to the original baseline (RT09) without any RNN post-processing being used. These models require approximately 18 times less training data than the large RT09 model.

4.4. Influence of History

In previous experiments in [10] cache models still complemented the RNN model. This suggested, that the simple RNN architecture we are currently using still does not allow to learn very long contexts. Hence, we tested the influence of the history length used in RNN rescoring. In general, the state vector is conditioning the posterior probability estimate of the predicted word on the preceding words. But the history can effectively be "forgotten" by initializing this vector to some random default value. Table 6 shows three different ways of using history in RNN rescoring and their influence on WER:

Full history is used if the entire data set is processed sequentially where the state vector potentially can represent the entire history. The state vector is just initialized once in the beginning. For every utterance, all hypotheses are processed by initializing the RNN with the state from the winning hypothesis of the utterance processed previously. The drawback is that the data set cannot be processed easily in parallel. *Binned history* is used if the entire data set is split into equally sized bins which are processed independently of each other. The state vector for each bin gets initialized at the beginning of the processing. In our experiments we ran RNN rescoring in parallel used bins containing as few as 10-20 utterances without noticeable degradation. *Hypothesis history* can be seen as binned history with bins containing only one utterance. Although the considered history comes already close to what is used in (high order) n-gram models, it increased WER only by 0.1%. We conclude, that the probability estimate of words is almost independent of words in previous sentences.

Test-set	Full	Binned	Hypothesis
rt07seval	19.6	19.6	19.7
rt05seval	16.9	16.9	17.0

Table 6: Influence of history in WER (RT09 system)

4.5. Speeding up rescoring

A well-known technique to speed up NN/RNN training and evaluation is the use of shortlists as done in [5]. Whereas shortlist are known to degrade results, we use the factorization into classes [11], which leads to faster processing and results without large degradation. Another proposed speed-up is the block operation: several words get propagated through the net in a single matrix×matrix operation, which can be performed faster than a sequence of matrix×vector operations. While that method can be applied even to recurrent neural networks, the speedup is smaller when class factorization is used, because the location of the second softmax output layer is different for every word. Hence, its softmax computation has do be done exclusively.

Locally caching hypotheses of single utterances

n-best size	10	100	1000
speed-up	2.1	2.6	3.3
avg cache size	55	354	2575
max cache size	438	3336	31392

Table 7: Trade-off between speed-up and cache size when state vectors of common prefix strings on the rt07seval set are cached

Nevertheless, we can still take advantage of the fact that n-best lists contain many hypotheses sharing common prefix

[4] Since this value is close to the inital learning rate used for RNN training we suggest to use it as a guideline.

strings. We can precompute the set of prefix strings that occur at least twice and cache their corresponding state vectors on-the-fly. By doing so, we obtain a speed-up by factor 2-3.

The cache accumulates state vectors and the posterior probabilities of the sequence of words that has been cached. The size of the cache should be kept down, because a medium-sized RNN model (1250 hidden neurons) requires already 5 KB per cached state. Therefore, it is appropriate to apply the caching locally i.e. just within hypotheses of each single utterance. That way, the precomputation of the prefix strings just needs to process the hypotheses of a single utterance which allows to run online decoding and online rescoring. Cache size will be always limited by the number of hypotheses and their length. The last history state vector can be passed easily across utterances to optimally preserve word history and obtain full accuracy in rescoring.

Another speed-up of factor 2-3 was observed in rescoring when floats were used instead of doubles. In that case, the RNN model also consumed just half the memory. By now, 1000-best rescoring can be done in 0.5×RT even for larger models (e.g. hidden layer size of 500, factored by 1000 classes) on a 3 GHz single core using 512 MB memory without loss in accuracy and without the mentioned prefix caching.

Consequently, RNN rescoring of 10- or 100-best could be used even in light-weight ASR setups, which due to memory and CPU limitations usually work based on n-grams. Our software for training RNN models and rescoring n-best lists can be downloaded from *http://www.fit.vutbr.cz/~imikolov/rnnlm*. An example package to repeat parts of the reported experiment is also available under the given link.

5. Conclusions

We recommend the use of RNN language models as easy mean to improve an existing LVCSR system, either by improving n-gram models using data sampled from an RNN or by performing the proposed rescoring and adaptation postprocessing steps.

Previous experiments in [10] and [1] already showed the advantage of RNN language models on simple ASR systems using limited training data. While RNN training times are still a bottleneck, we showed, that improvements can be obtained even in a state-of-the-art ASR system using n-gram language models trained on much more data than the RNN model. If the system uses about 18 times less than the original language modeling data, it still reaches a performance similar to the baseline.

Thus, RNN models are interesting also in cases of low-resource ASR. RNN data sampling is an easy way to increase the amount of training data than retrieving domain-relevant web data, in case of low-resource languages it may be the only way. The existing system can be improved without the need to change the structure at all, just n-gram LMs needs to be replaced. Improvements may however vanish if RNN rescoring is applied.

It was already pointed out in [2] that continuous space models should adapt better on little data than n-gram models. Unsupervised adaptation of the RNN model on a meeting level provided noticeable improvements in addition to RNN rescoring. We think there might be still potential e.g. by using dynamic adaptation or the adaptation of just a subset of all RNN weights.

Still, the current RNN architecture can hardly exploit context longer than a sentence. Further improvement could possibly be obtained by using long short term memory (LSTM) RNNs [7] or temporal kernel [6].

Its fast rescoring process make class-based RNNs interesting for light-weight and real-time ASR systems. We proposed the caching of common prefix strings as an easy way to get a speed-up of factor 2-3. Rescoring could be still made even faster by combining block operation and prefix caching or parallelization. Prefix caching without block operation seems suitable for very light-weight (e.g. 10-best) online systems.

6. References

[1] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát and S. Khudanpur. Variational Approximation of Long-Span Language Models in LVCSR. In *IEEE Intl. Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, CZ, May 2011.

[2] M. Afify, O. Siohan, and R. Sarikaya. Gaussian mixture language models for speech recognition. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–29. IEEE, 2007.

[3] M. Bodén. A guide to recurrent neural networks and backpropagation. In *THE DALLAS PROJECT, SICS TECHNICAL REPORT T2002:03, SICS*, 2002.

[4] F. Grezl, M. Karafiát and L. Burget. Investigation into bottle-neck features for meeting speech recognition. In *Proc. Interspeech 2009*, number 9, pages 2947–2950, Brighton, GB, 2009. International Speech Communication Association.

[5] H. Schwenk and J.L. Gauvain. Building continuous space language models for transcribing european languages. pages 737–740, Lisbon, Portugal, 2005.

[6] M.C. Mozer. *A focused backpropagation algorithm for temporal pattern recognition*, pages 137–169. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[7] S. Hochreiter and J. Schmidhuber. LSTM Can Solve Hard Long Time Lag Problems. In *Advances in Neural Information Processing Systems 9*, pages 473–479. MIT Press, 1997.

[8] T. Hain, L. Burget, J. Dines, G. Garau, M. Karafiát, D.v. Leeuwen, M. Lincoln and V. Wan. The 2007 AMI(DA) system for meeting transcription. In *Proc. Rich Transcription 2007 Spring Meeting Recognition Evaluation Workshop*, Baltimore, Maryland USA, May 2007.

[9] T. Hain, L. Burget, J. Dines, N.P. Garner, A.H. El, M. Huijbregts, M. Karafiát, M. Lincoln and V. Wan. The AMIDA 2009 Meeting Transcription System. In *Proc. of INTERSPEECH 2010*, volume 2010, pages 358–361, Makuhari, Chiba, JP, 2010. International Speech Communication Association.

[10] T. Mikolov, M. Karafiát, L. Burget, J. Černocký and S. Khudanpur. Recurrent neural network based language model. In *Proc. of INTERSPEECH 2010*, number 9, pages 1045–1048, Makuhari, Chiba, JP, 2010. International Speech Communication Association.

[11] T. Mikolov, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur. Extensions of Recurrent Neural Network Language Models. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, CZ, May 2011.

[12] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, T. Hofmann, T. Poggio and J. Shawe-taylor. A neural probabilistic language model. In *Journal of Machine Learning Research*, volume 3, pages 1137–1155, 2003.