rwthlm – The RWTH Aachen University Neural Network Language Modeling Toolkit

Martin Sundermeyer[1], *Ralf Schlüter*[1], *Hermann Ney*[1,2]

[1]Human Language Technology and Pattern Recognition,
Computer Science Department, RWTH Aachen University, Aachen, Germany
[2]Spoken Language Processing Group, LIMSI CNRS, Paris, France
{sundermeyer,schlueter,ney}@cs.rwth-aachen.de

Abstract

We present a novel toolkit that implements the long short-term memory (LSTM) neural network concept for language modeling. The main goal is to provide a software which is easy to use, and which allows fast training of standard recurrent and LSTM neural network language models.

The toolkit obtains state-of-the-art performance on the standard Treebank corpus. To reduce the training time, BLAS and related libraries are supported, and it is possible to evaluate multiple word sequences in parallel. In addition, arbitrary word classes can be used to speed up the computation in case of large vocabulary sizes.

Finally, the software allows easy integration with SRILM, and it supports direct decoding and rescoring of HTK lattices. The toolkit is available for download under an open source license.

Index Terms: speech recognition, language modeling, recurrent neural networks, long short-term memory

1. Introduction

Since their introduction in [1], neural networks have proven especially powerful for modeling the probability of a word sequence in natural language.

However, there are several aspects about neural network language models that make their application to the problem of language modeling difficult. Traditionally, a language model (LM) estimates the joint probability of a word sequence w_1^N

$$p(w_1^N) = \prod_{i=1}^{N} p(w_i|w_1^{i-1})$$

by factorizing it as the product of word posterior probabilities $p(w_i|w_1^{i-1})$. In modern speech recognition systems, the vocabulary of a recognizer usually contains hundreds of thousands of words. Then the corresponding neural network can easily consist of hundreds of millions or more parameters, which results in a huge computational complexity for training.

It is only by exploiting several speedup techniques and by an efficient implementation that such a neural network can be trained within reasonable time on large data sets. Usually, this means that it is prohibitive to use a standard neural network implementation for language modeling, and a specialized software is required.

There already exist several toolkits that can be used for neural network language modeling, namely `cslm` ([3]), `nplm` ([4]), and `rnnlm` ([5]). The first two of them implement a feedforward neural network, and concentrate on highly efficient training by using GPUs, or by giving up normalization, respectively. Currently, with `rnnlm` there is only one toolkit that supports a recurrent neural network (RNN) approach.

RNNs seem interesting from a research point of view, because it was observed that they improve over feedforward models ([6, 7, 8]). On the other hand, it was found that long-range dependences are difficult to learn with gradient-based training algorithms [9] in case of RNNs. To circumvent this problem, instead of refining the training algorithm, in [10] a revised RNN architecture was presented, which was subsequently improved in [11] and [12]. This architecture is known as long short-term memory (LSTM) neural network.

LSTMs have obtained state-of-the-art performance, especially in handwriting recognition ([13],[14],[15]), and acoustic modeling ([16]). It seems unclear to which extent long-range dependences play a role in language modeling. Nevertheless, LSTMs were found to perform significantly better than RNNs for such a task in [2] as well.

This paper presents `rwthlm`, a novel toolkit for language modeling with standard recurrent and LSTM networks. The software aims at being easy to use, and offers all features necessary to train a neural network LM and use it for rescoring.

The software relies on efficient BLAS libraries, and also allows parallelization in a way that multiple word sequences are processed at a time. It supports word classes to reduce the computational effort for training as well as rescoring. Word classes can be obtained by arbitrary techniques like word frequency clustering ([6]), the exchange algorithm for perplexity-based word classes ([17, 18]), or approaches relying on neural networks themselves ([19]).

With `rwthlm`, efficient Viterbi decoding of HTK lattices is possible, including pruning and look ahead techniques known from first pass speech decoding. More importantly, `rwthlm` can also output a rescored lattice incorporating the LSTM probabilities, so that later processing steps can make use of the neural network probabilities as well. With respect to lattice output, different algorithms are available, so that a user can trade lattice size for accuracy.

2. Recurrent Neural Network LMs

The focus of `rwthlm` lies on recurrent neural networks. As such, standard RNNs as well as the improved LSTM architecture are supported. In the simplest case, only a single RNN layer is used in addition to input and output layers. The equa-

tions defining such a network are then given by

$$
\begin{aligned}
y_{i-1} &= \sigma(A_1 x_{i-1} + R y_{i-2}) \\
p(c(w_i)|w_1^{i-1}) &= \varphi_{c(w_i)}(A_2 y_{i-1}) \\
p(w_i|c(w_i), w_1^{i-1}) &= \varphi_{w_i}(A_{c(w_i)} y_{i-1}) \\
p(w_i|w_1^{i-1}) &= p(c(w_i)|w_1^{i-1}) \cdot (w_i|c(w_i), w_1^{i-1})
\end{aligned}
$$

Here, by x_{i-1} we denote the one-hot encoded vector representation of the most-recent history word w_{i-1}, and y_{i-1} is the outgoing activation vector of the hidden layer. The matrices $A_1, A_2, A_{c(w_i)}$ contain the weights connecting the corresponding neural network layers, and R is the weight matrix for the recurrent connections. In the above formulas, we make use of word classes, where each word from the vocabulary is mapped to a unique class. By decomposing the word posterior probability into the product of the class posterior probability and the word posterior probability given its class, we can significantly speed up the computations ([20, 21]).

By σ and φ, we denote the logistic sigmoid and softmax function, respectively, applied element-wise to a vector

$$
\begin{aligned}
\sigma(x) &= \left(\left(1 + \exp(-x_1)\right)^{-1}, \ldots, \left(1 + \exp(-x_D)\right)^{-1} \right)^T \\
\varphi(x) &= \left(\frac{\exp(x_1)}{\sum_{j=1}^{D} \exp(x_j)}, \ldots, \frac{\exp(x_D)}{\sum_{j=1}^{D} \exp(x_j)} \right)^T
\end{aligned}
$$

where D is the dimension of x, and T indicates transposition.

More advanced architectures including LSTM layers can be created easily. In our experiments, we obtained good results with the architecture depicted in Fig. 1. We diverge from the simple RNN by replacing the recurrent layer with LSTMs. Furthermore, we add a so-called projection layer between the input and hidden layer, with an identity activation function.

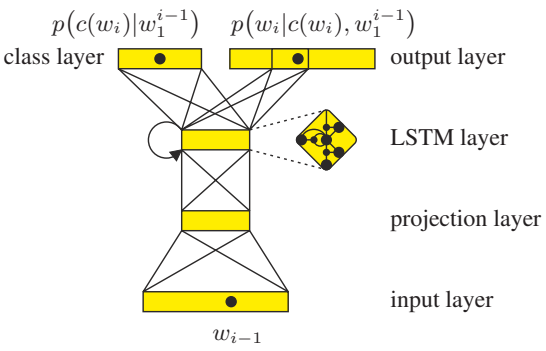

Figure 1: Architecture of a recurrent LSTM neural network language model that can be trained with `rwthlm`.

We did not find such a linear projection layer to have an impact on performance for LSTMs. However, it greatly reduces the number of parameters of the resulting neural network: An LSTM cell has four inputs and a single output. Thus, it should be avoided to directly connect the input layer to the LSTM layer. Due to space restrictions, we omit the details of LSTM networks, and refer to [2, 16] for more details.

3. Training of Neural Network LMs

3.1. Supported Neural Network Architectures

In `rwthlm`, there are no restrictions on building up neural networks of complex architectures: Neural network layers can be combined in arbitrary ways (except that it is not supported to connect an LSTM layer to the input layer directly, for the reasons discussed in the previous section). In particular, it is possible to build deep recurrent LSTM neural network architectures, as proposed in [16] for acoustic modeling. The layers themselves can be of type feedforward, standard recurrent, or LSTM, and different activation functions are available. Also, it can be chosen whether a bias for the layers is used or not.

3.2. Training Algorithm Details

Recurrent neural networks are most commonly trained with stochastic gradient descent (SGD), where the gradient is computed with the backpropagation through time (BPTT) algorithm ([22, 23, 24]). This method is implemented in `rwthlm` as well.

Our training receipe is closely related to the situation that is met in rescoring: First, we reset the activations which serve as the internal memory of the neural network. Then, we compute the gradient on a sequence of words, and we update a neural network weight α_{ij} according to the formula

$$
\alpha_{ij} := \alpha_{ij} - \lambda \cdot \frac{\partial F}{\partial \alpha_{ij}},
$$

where F denotes the objective function, and λ is the learning rate. In `rwthlm`, F is fixed to maximum likelihood, sometimes also denoted as cross entropy.

It is often advised in neural network literature to shuffle the training samples before each training epoch. This is not possible in case of recurrent neural networks, as the words of a sequence have to be processed in order. In `rwthlm`, the sequences are shuffled instead.

The toolkit offers some flexibility regarding how to define a sequence that is used for training. Three variants are distinguished:

1. A sequence is defined to be a sentence from the training data. As a result, sequences can be quite different in length, especially in case of conversational speech transcriptions.
2. A sequence represents the concatenation of multiple sentences up to a given maximum length. In this way, the neural network can potentially learn across-sentence dependences, and each sequence starts with a *sentence-begin* token.
3. A sequence consists of a fixed number of consecutive words. This means that the text is split into sequences at arbitrary positions. The network can learn across-sentence dependences, and the sequence may start with any word.

The second and third definitons of a sequence rely on a maximum sequence length that must be specified in advance. When computing perplexities, the sequences are prepared in the exact same way as for training.

3.3. Parallelization

For language modeling, most of the time large amounts of data are available, and the performance of a neural network LM usually improves when training with more data. As the computational costs for training are high, parallelization is important to speed up the training process. For this purpose, `rwthlm` allows the parallel evaluation of multiple sequences. For a simple implementation, we stick to a parallelization scheme that is

mainly based on matrix-matrix operations, similar to the techniques presented in [25]. To this end, the RNN and LSTM equations from [16] have to be transformed from a matrix-vector into a matrix-matrix formulation.

The splitting of the training data into sequences interacts with the efficiency of the parallelization. In case where all sequences are split such that they have the same length, parallelization obviously works best. However, we found that the difference in runtime between sequence splitting strategies is rather small and did not exceed 20 % in the cases we considered.

3.4. Learning Rate Tuning

One of the crucial parameters of SGD training is the learning rate λ. Conceptually, a high value for λ is preferable, because it will lead to faster convergence of the tranining, but if the value is chosen too large, perplexity will start to fluctuate.

This behaviour of the SGD learning algorithm is usually handled by starting from a high λ value, and decreasing it as soon as a full epoch of training leads to a degradation in perplexity on a held-out data set.

This strategy was not effective in our case. The main observation is that the larger the gradient, the smaller the learning rate must be to avoid fluctuations. However, the size of the gradient can vary greatly depending on the length of the sequence as well as the number of sequences that are evaluated in parallel. For this reason, each time a good initialization for the learning rate has to be found, which can be tedious. Following [26], we implemented a simple algorithm that guesses an initial learning rate: Starting from a fixed λ value, we run over a small portion of the training data, computing an on-the-fly average of the training error. If the error decreases continuously, we keep the current λ as a candidate, and iterate the same process with an increased learning rate.

The procedure of increasing the learning rate terminates as soon as the training error starts fluctuating. If we found a candidate λ until then, we stop and use this value for full training. Otherwise, we continue by decreasing λ until a suitable value is found. While there is no guarantee that this strategy works in all cases, most of the time it helps finding a good guess, and the algorithm for finding an initial learning rate is also parameterizable such that it can be adjusted to improve its reliability, at the cost of a longer initialization phase.

4. Lattice Decoding and Rescoring

To improve the performance of a speech recognition system, `rwthlm` supports decoding of n-best lists as well as word lattices in HTK format. In case of lattices, the unlimited context size of the neural network poses problems for decoding: Unless the lattice has a prefix-tree like structure, where the sequence of words leading to a certain lattice node is unique, multiple neural network LM probabilities correspond to a single word arc in the lattice. For this reason, an exact decoding pass cannot be performed in such a way that the probabilities on all the lattice arcs are replaced by the neural network LM estimate and the best path is obtained afterwards.

As an alternative, an approximative decoding ([27, 28]) can be performed which closely resembles the methods used for first pass speech decoding. This approach is also implemented in `rwthlm`. There is a need for efficient pruning techniques during rescoring, where the following are supported by `rwthlm`:

- Cardinality pruning: At each lattice node, only the best k hypotheses are retained.
- Beam pruning: At each time step, only those hypotheses are kept whose probability is not smaller than the current best one, multiplied by a certain factor.
- Recombination pruning: Even though there is no fixed context size of an RNN, we can still enforce recombination, keeping only the best hypothesis for a given LM context of a certain order.
- Acoustic and LM look ahead: In a lattice, both the acoustic and (count) LM probabilities of the full word sequences are available. Thus, the probabilities of future words can be incorporated into the pruning decision at the current time step. It can be distinguished whether the sum over all future paths or only the single best path is considered for look ahead.

The best path obtained from direct decoding can be stored in standard NIST format.

A unique feature of `rwthlm` is that it is also possible to obtain a lattice that incorporates the neural network LM probabilities. Two possible options exist: As an approximation, the original lattice structure can be kept. Interestingly, this only leads to a slight degradation in the Viterbi decoding result of the rescored lattice in comparison to direct decoding. By using confusion network decoding on the rescored lattice, in all of our experiments we obtained at least the same performance as in case of direct decoding, or even improved.

The second option allows to write back a lattice that may be larger than the original one. It is derived from the paths considered during direct decoding. This lattice is guaranteed to have the same Viterbi word error rate as would be obtained by direct decoding. Using confusion network decoding on this lattice gives additional improvements on top, which lie in the same range as observed on lattices created by first pass speech decoders. The overhead in lattice size can be tuned directly by adjusting the pruning parameters. More details can be found in [28].

5. Implementational Aspects

The software is implemented in C++ 11. It relies on efficient mathematical libraries (Intel MKL and AMD ACML libraries are both supported, GPU support may be added in the future). In addition, it uses some functions that are part of the boost library. Both float and double precision are available, but for our experiments, we only used double precision. However, single precision may be interesting as it speeds up the computations by a factor of 1.6. The software is released under the *RWTH ASR License* which allows free usage including redistribution and modification for non-commercial use. It can be downloaded from `http://www-i6.informatik.rwth-aachen.de/web/Software/rwthlm.php`.

For the implementation of neural networks and especially LSTM networks, it is helpful to verify the results that are computed by the software. As noted in [29], this can be done by comparing the gradient, as obtained by BPTT, with the symmetric finite difference, where we have

$$\frac{\partial F}{\partial \alpha_{ij}} = \frac{F(\alpha_{ij} + \epsilon) - F(\alpha_{ij} - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

for a constant ϵ. The tests may also be helpful for developing new extensions of `rwthlm` that affect the underlying mathematical model.

6. Experimental Results

We conducted experiments on two corpora, namely the standard Treebank corpus and a French corpus, where we also obtained speech recognition results. Table 1 lists the corresponding training data.

Corpus		Running Words	Vocabulary
Treebank	Train	890 K	10 K
	Dev	70 K	
	Test	79 K	
Quaero French	Train	100 M	188 K
	Dev	35 K	
	Test	41 K	

Table 1: Training data used for the experiments.

For the Treebank corpus, we only investigate the performance of the neural networks themselves, without additional techniques such as direct connections ([31]) or contextual features ([32]), which could be added to both a standard RNN as well as an LSTM. Table 2 depicts the perplexity results of `rnnlm` and `rwthlm`. For `rwthlm` results, we always optimize over the different sequence types on the development data. The LSTM was trained with a projection layer and an LSTM layer of size 200 each.

As `rnnlm` does not offer support for a projection layer, we also trained a neural network LM with `rwthlm` including a projection layer and a standard recurrent layer of the same dimensions. In this case, we obtained a perplexity of 122.4 which is similar to the `rnnlm` result.

We also investigated a larger French corpus. Results can be found in Table 3. The Kneser-Ney 4-gram (KN4) model was trained on 16 times more data than the LSTM. As the vocabulary size (200 K) of the KN4 model is larger than the number of distinct words in the LSTM training data, we normalized the LSTM perplexities to a vocabulary size of 200 K as proposed in [8].

Corpus	Mod. KN5	`rnnlm`	`rwthlm`	
			RNN	LSTM
Dev	146.8	–	129.4	113.9
Test	140.7	124.7	122.4	108.0

Table 2: Perplexity results on the Treebank corpus for a modified Kneser-Ney-smoothed 5-gram, `rnnlm` and `rwthlm`. The result for `rnnlm` is taken from [30]. The `rwthlm` neural networks include a projection layer.

The hidden layer size was set to 300 for both the projection and the hidden LSTM layer, which is rather small in view of the amount of training data. For the output layer, 1000 word classes were trained based on a perplexity criterion with the exchange algorithm [17].

By rescoring 100-best lists with `rwthlm`, we were able to improve the word error rate of a French speech recognition system from 15.9 % to 14.8 %. This system obtained the best result in the final evaluation of the Quaero project[1]. More details can be found in [28]. By rescoring lattices, we arrived at a final word error rate of 14.2 % on the test data. For all word error rates, confusion network decoding was used.

[1] http://www.quaero.org

Type	WER [%]		Perplexity	
	Dev	Test	Dev	Test
KN4	14.1	15.9	102.9	122.0
100-best	13.4	14.8	79.9 (98.6)	94.4 (114.9)
1000-best	13.1	14.6		
Lattice	12.6	14.2		

Table 3: Perplexity and word error rate (WER) results for French. Numbers in parentheses indicate LSTM perplexities without interpolation, normalized to a 200 K vocabulary.

We also analyzed the training time of `rwthlm`. We switched to a subset of the French training data, comprising 27 M running words, and used 2000 Brown classes. Including parallelization over four sequences, a single epoch took 226 minutes on two 6-core Intel Westmere CPUs.

It is rather difficult to consistently compare the computational effort for training a neural network LM with `rwthlm` with that of other toolkits. In case where we train an `rnnlm` with the same amount of classes and the same dimension of the hidden layer, this would take 895 minutes on the same machine, even when switching to a block mode of 10 and backpropagating for one time step only. By contrast, in `rwthlm`, the backpropagation is carried out over the full sequence and the model is significantly more complex, so the performance difference is surely due to the fact that `rwthlm` takes advantage of optimized math libraries and parallelization. For this experiment, the physical memory consumption of `rwthlm` was 1.2 GB vs. 2.4 GB for `rnnlm`.

7. Conclusion

In this work, we presented `rwthlm`, a novel toolkit for training recurrent and LSTM neural network LMs.

The software allows training of arbitrary types of recurrent neural network LMs and in particular LSTM models, that were found to perform significantly better than simple RNNs on a standard language modeling task. The toolkit supports fast BLAS libraries and also implements a parallelization scheme where multiple sequences are forwarded through the network.

Furthermore, the processing of lattices in HTK format is supported, where the best Viterbi path can be obtained directly, or a rescored lattice containing the neural network LM probabilities can be created instead. The toolkit is available for download under an open source license.

With these features, `rwthlm` can help to facilitate research in the area of neural network language modeling, building new models and techniques on top of a solid baseline.

8. Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreements nos. 287658 and 287755. H. Ney was partially supported by a senior chair award from DIGITEO, a French research cluster in Ile-de-France. Experiments were performed with computing resources granted by JARA-HPC from RWTH Aachen University under project 'jara0085'.

9. References

[1] Bengio, Y., and Ducharme, R., "A neural probabilistic language model", Proc. of Advances in Neural Information Processing Systems (2001), vol. 13., pp. 932–938

[2] Sundermeyer, M., Schlüter, R., and Ney, H., "LSTM Neural Networks for Language Modeling", Proc. of Interspeech 2012

[3] Schwenk, H., "CSLM - A modular Open-Source Continuous Space Language Modeling Toolkit", Proc. of Interspeech 2013, pp. 1198–1202

[4] "Decoding with large-scale neural language models improves translation", Vaswani, A., Zhao, Y., Fossum, V., and Chiang, D., Proc. of EMNLP 2013, pp. 1387–1392

[5] Mikolov, T., Kombrink, S., Deoras, A., Burget, L., and Černocký, J., "RNNLM - Recurrent Neural Network Language Modeling Toolkit", ASRU 2011 Demo Session

[6] Mikolov, T., Kombrink, S., Burget, L., Černocký, J., and Khudanpur, S., "Extensions of Recurrent Neural Network Language Model", Proc. of ICASSP 2011, pp. 5528–5531

[7] Arısoy, E., Sainath, T. N., Kingsbury, B., and Ramabhadran, B., "Deep Neural Network Language Models", Proc. of NAACL-HLT 2012 Workshop, pp. 20–28

[8] Sundermeyer, M., Oparin, I., Gauvain, J.-L., Freiberg, B., Schlüter, R., and Ney, H., "Comparison of Feedforward and Recurrent Neural Network Language Models", Proc. of ICASSP 2013, pp. 8430–8434

[9] Bengio, Y., Simard, P., and Frasconi, P., "Learning long-term dependencies with gradient descent is difficult" IEEE Transactions on Neural Networks 5 (1994), pp. 157–166

[10] Hochreiter, S., and Schmidhuber, J., "Long Short-Term Memory", Neural Computation 9 (8), 1997, pp. 1735–1780

[11] Gers, F. A., "Learning to Forget: Continual Prediction with LSTM", Proc. of the 9th Int. Conf. on Artificial Neural Networks, 1999, pp. 850–855

[12] Gers, F. A., Schraudolph, N. N., and Schmidhuber, J., "Learning Precise Timing with LSTM Recurrent Networks", Journal of Machine Learning Research 3, 2002, pp. 115–143

[13] Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., and Schmidhuber, J., "A novel connectionist system for unconstrained handwriting recognition", IEEE Trans. on PAMI 2009, Vol. 31, No. 5, pp. 855–868

[14] Grosicki, E., and El Abed, H., "ICDAR 2009 handwriting recognition competition", Proc. of ICDAR 2009, pp. 1398–1402

[15] Kozielski, M., Doetsch, P., and Ney, H. "Improvements in RWTH's system for off-line handwriting recognition", Proc. of ICDAR 2013, pp. 935–939

[16] Graves, A., Mohamed, G., Hinton, G., "Speech Recognition with Deep Recurrent Neural Networks", Proc. of ICASSP 2013, pp. 6645–6649

[17] Kneser, R., and Ney, H., "Forming Word Classes by Statistical Clustering for Statistical Language Modelling", Proc. of QUALICO 1991, pp. 221–226

[18] Brown, P. F., deSouza, P. V., Mercer, R. L., Della Pietra V. J., and Lai, J. C., "Class-Based n-gram Models of Natural Language", Computational Linguistics 18 (4), pp. 467–479

[19] Mikolov, T., Sutskever, I., Chen, K., Corrado, C., and Dean, J., "Distributed Representations of Words and Phrases and their Compositionality", Proc. of NIPS 2013

[20] Goodman, J., "Classes for fast maximum entropy training", Proc. of the ICASSP 2001, pp. 561–564

[21] Morin, F., and Bengio, Y., "Hierarchical Probabilistic Neural Network Language Model", Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics 2005, pp. 246-252

[22] Rumelhart, D. E., Hinton, G. E., Williams, R. J., "Learning Internal Representations by Error Propagation", in: McClelland, J. L., Rumelhart, D. E., PDP Research Group, The, "Parallel Distributed Processing", The MIT Press, 1986, pp. 318–362.

[23] Werbos, Paul J., "Backpropagation Through Time: What It Does and How to Do It", Proceedings of the IEEE 1990, Vol. 78, No. 10, pp.1550–1560

[24] Williams, R. J., Zipser, D., "Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity", in: Chauvain, Y., Rumelhart, D. E., "Backpropagation: Theory, Architectures, and Applications", Psychology Press, 1995, pp. 433–486

[25] Schwenk, H., "Continuous Space Language Models", Computer Speech and Language 21 (2007), pp. 492–518

[26] Hinton, G., "Neural Networks for Machine Learning", Online Course at coursera.org, Lecture 6, 2012

[27] Auli, M., Galley, M., Quirk, C., and Zweig, G., "Joint Language and Translation Modeling with Recurrent Neural Networks", Proc. of EMNLP 2013, pp. 1044–1054

[28] Sundermeyer, M., Tüske, Z., Schlüter, R., and Ney, H., "Lattice Decoding and Rescoring with Long-Span Neural Network Language Models", Interspeech 2014, accepted for publication

[29] Graves, A., "Supervised Sequence Labelling with Recurrent Neural Networks" Springer 2012, Chapter 3

[30] Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Cěrnocký, J. H., "Empirical Evaluation and Combination of Advanced Language Modeling Techniques", Proc. of Interspeech 2011, pp. 605–608

[31] Mikolov, T., Deoras, A., Povey, D., Burget, L., Cernocky, J., "Strategies for Training Large Scale Neural Network Language Models", Proc. of ASRU 2011, pp. 196–201

[32] Mikolov, T., and Zweig, G., "Context dependent recurrent neural network language model", Proc. of SLT 2012, pp. 234–239