**rwthlm – The RWTH Aachen University**

**Neural Network Language Modeling Toolkit**

**Martin Sundermeyer**, **Ralf Schlüter**, **Hermann Ney**

**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
\[
y_{i-1} = \sigma(A_1 x_{i-1} + R y_{i-2})
\]
\[
p(c(w_i)|w_{i-1}) = \varphi_{c(w_i)}(A_2 y_{i-1})
\]
\[
p(w_i|x_i, w_{i-1}) = \varphi_{w_i}(A_{c(w_i)} y_{i-1})
\]
\[
p(w_i|w_{i-1}) = p(c(w_i)|w_{i-1}), (w_i|c(w_i), w_{i-1})
\]

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 \(\sigma\) and \(\varphi\), we denote the logistic sigmoid and softmax function, respectively, applied element-wise to a vector
\[
\sigma(x) = \left(1 + \exp(-x_1), \ldots, 1 + \exp(-x_D)\right)^\top
\]
\[
\varphi(x) = \left(\frac{\exp(x_1)}{\sum_{j=1}^D \exp(x_j)}, \ldots, \frac{\exp(x_D)}{\sum_{j=1}^D \exp(x_j)}\right)^\top
\]
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.

![Architecture of a recurrent LSTM neural network language model that can be trained with rwthlm.](image)

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 receive 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 \(\alpha_{ij}\) according to the formula
\[
\alpha_{ij} := \alpha_{ij} - \lambda \cdot \frac{\partial F}{\partial \alpha_{ij}}
\]
where \(F\) denotes the objective function, and \(\lambda\) 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 sentence 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 dependencies, 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 dependencies, and the sequence may start with any word.

The second and third definitions 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 $\lambda$. Conceptually, a high value for $\lambda$ is preferable, because it will lead to faster convergence of the training, 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$ until then, we stop and use this value for full training. Otherwise, we continue by decreasing $\lambda$ 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, rwtlhm 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 rwtlhm. There is a need for efficient pruning techniques during rescoring, where the following are supported by rwtlhm:

- 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 rwtlhm 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/rwtlhm.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 $\epsilon$. The tests may also be helpful for developing new extensions of rwtlhm 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.

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Running Words</th>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treebank</td>
<td>Train 890 K</td>
<td>10 K</td>
</tr>
<tr>
<td></td>
<td>Dev 70 K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 79 K</td>
<td></td>
</tr>
<tr>
<td>Quaero French</td>
<td>Train 100 M</td>
<td>188 K</td>
</tr>
<tr>
<td></td>
<td>Dev 35 K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test 41 K</td>
<td></td>
</tr>
</tbody>
</table>

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 \texttt{rnnlm} and \texttt{rwthlm}. For \texttt{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 \texttt{rnnlm} does not offer support for a projection layer, we also trained a neural network LM with \texttt{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 \texttt{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].

<table>
<thead>
<tr>
<th>Type</th>
<th>WER [%]</th>
<th>Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN4</td>
<td>Dev 14.1</td>
<td>102.9</td>
</tr>
<tr>
<td></td>
<td>Test 15.9</td>
<td>122.0</td>
</tr>
<tr>
<td>100-best</td>
<td>Dev 13.4</td>
<td>79.9</td>
</tr>
<tr>
<td></td>
<td>Test 14.8</td>
<td>94.4</td>
</tr>
<tr>
<td>1000-best</td>
<td>Dev 13.1</td>
<td>(98.6)</td>
</tr>
<tr>
<td></td>
<td>Test 14.6</td>
<td>(114.9)</td>
</tr>
</tbody>
</table>

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.

word error rate of 14.2 % on the test data. For all word error rates, confusion network decoding was used.

We also analyzed the training time of \texttt{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 \texttt{rwthlm} with that of other toolkits. In case where we train an \texttt{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 back-propagating for one time step only. By contrast, in \texttt{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 \texttt{rwthlm} takes advantage of optimized math libraries and parallelization. For this experiment, the physical memory consumption of \texttt{rwthlm} was 1.2 GB vs. 2.4 GB for \texttt{rnnlm}.

7. Conclusion

In this work, we presented \texttt{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, \texttt{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’.

1http://www.quaero.org
9. References


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

