LSTM Neural Networks for Language Modeling

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney

Human Language Technology and Pattern Recognition, Computer Science Department, RWTH Aachen University, Aachen, Germany

{sundermeyer,schlueter,ney}@cs.rwth-aachen.de

Abstract

Neural networks have become increasingly popular for the task of language modeling. Whereas feed-forward networks only exploit a fixed context length to predict the next word of a sequence, conceptually, standard recurrent neural networks can take into account all of the predecessor words. On the other hand, it is well known that recurrent networks are difficult to train and therefore are unlikely to show the full potential of recurrent models.

These problems are addressed by a the Long Short-Term Memory neural network architecture. In this work, we analyze this type of network on an English and a large French language modeling task. Experiments show improvements of about 8 % relative in perplexity over standard recurrent neural network LMs. In addition, we gain considerable improvements in WER on top of a state-of-the-art speech recognition system.

Index Terms: language modeling, recurrent neural networks, LSTM neural networks

1. Introduction

In automatic speech recognition, the language model (LM) of a recognition system is the core component that incorporates syntactical and semantical constraints of a given natural language. While today mainly backing-off models ([1]) are used for the recognition pass, feed-forward neural network LMs, first introduced in [2], have become an important supplement to existing techniques in the rescoring stage ([3]).

Both approaches rely on the n-gram approximation where the probability $p(w_1^N)$ of a word sequence w_1^N is factorized as

$$p(w_1^N) = \prod_{m=1}^{N} p(w_m|h_m)$$

so that only the $n-1$ preceeding words $h_m := w_{m-n+1}^{m-1}$ are used to estimate the probability of the word at position m. However, neural network LMs overcome a major drawback of backing-off models ([5]): Whenever an n-gram (h, w) has not been observed in training, a backing-off model lacks an explicit estimate for the probability of this n-gram. Therefore it falls back on the estimate for the $(n-1)$-gram $(\bar{h}, w)$ where the left-most word of h has been removed to construct $\bar{h}$, and $\gamma(h)$ is a normalization constant:

$$p(w|h) = \gamma(h)p(w|\bar{h}).$$

In contrast to backing-off models, neural network LMs always estimate probabilities based on the full history, regardless of whether the n-gram was seen in training or not.

On the other hand, the n-gram assumption still leads to an inaccuracy in the modeling when feed-forward neural network LMs are used. According to the chain rule of probability theory, all predecessor words w_1^{m-1} need to be taken into account to predict the m-th word of a sentence:

$$p(w_1^N) = \prod_{m=1}^{N} p(w_m|w_1^{m-1}).$$

This can be remedied by replacing the feed-forward architecture by a recurrent neural network architecture which is appropriate for sequence modeling (see [6], [7]).

Unfortunately, recurrent neural networks are hard to train using backpropagation through time ([8]). The main difficulty lies in the well-known vanishing gradient problem ([9]) which means that the gradient that is propagated back through the network either decays or grows exponentially.

One approach to improved training of recurrent neural networks lies in better optimization algorithms that make use of higher-order information (see e. g. [10]). However, this usually comes at the price of significantly increased computational costs which makes these methods less attractive for language modeling where the amount of training data is extremely large.

An alternative solution called Long Short-Term Memory (LSTM) was proposed in [11]: The network architecture is modified such that the vanishing gradient problem is explicitly avoided, whereas the training algorithm is left unchanged.

In this work, we introduce LSTMs to the field of language modeling. We analyze its effectiveness on an English and a large French corpus in terms of perplexity and word error rate. Furthermore, we investigate techniques for decreased training times and compare different neural network LM architectures.

2. LSTM neural networks

In [11], the vanishing gradient problem was analysed in detail. Whenever the gradient of the error function of the neural network is propagated back through a unit of a neural network, it gets scaled by a certain factor. For nearly all practically relevant cases, this factor is either greater than one or smaller than one.

As a result, in a recurrent neural network, the gradient blows up or decays exponentially over time. (For the language modeling point of view, time steps correspond to word positions in a sentence.) Thus, the gradient either dominates the next weight adaptation step or effectively gets lost.

To avoid this scaling effect, the authors re-designed the unit of a neural network in such a way that its corresponding scaling factor is fixed to one. The new unit type that is obtained from this design goal is rather limited in its learning capabilities. Therefore, the unit was enriched by several so-called gat-

ing units. The final unit is depicted in Fig. 1, where we have included two modifications of the original LSTM unit proposed in [12] and [13].

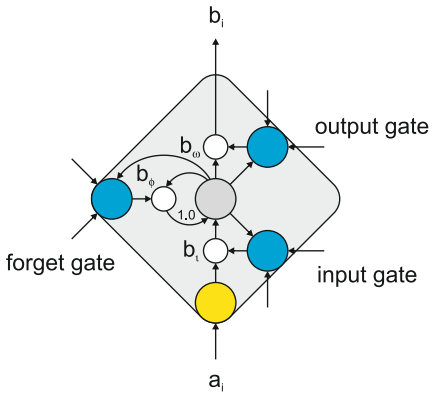

Figure 1: LSTM memory cell with gating units

A standard neural network unit i only consists of the input activation a_i and the output activation b_i which are related—when a $tanh$ activation function is used—by

$$b_i = \tanh(a_i).$$

The LSTM unit adds several intermediate steps: After applying the activation function to a_i, the result is multiplied by a factor b_ι. Then the inner activation value of the previous time step, multiplied by the quantity b_ϕ is added due to the recurrent self-connection. Finally, the result is scaled by b_ω and fed to another activation function, yielding b_i. The factors $b_\iota, b_\phi, b_\omega \in (0, 1)$, indicated by the small white circles, are controlled by additional units (depicted as blue circles) called input, output, and forget gate, respectively. The gating units sum the activations of the previous hidden layer and the activations of the current layer from the previous time step as well as the inner activation of the LSTM unit. The resulting value is squashed by a logistic sigmoid function which then is set to b_ι, b_ϕ, or b_ω, respectively.

For brevity, we omit the rather extensive equations describing the LSTM network. These can be found e. g. in [14][1].

The whole LSTM unit including the gating units may be interpreted as a differentiable version of computer memory ([14]). For this reason, LSTM units sometimes are also referred to as LSTM memory cells. Whether one adheres to the proposed interpretation of the gating units or not, the LSTM architecture solves the vanishing gradient problem at small computational extra-costs. In addition, it has the desirable property of including standard recurrent neural network units as a special case.

3. Neural network language models

Although there are several differences in the neural network language models that have been successfully applied so far, all of them share some basic principles:

- The input words are encoded by 1-of-K coding where K is the number of words in the vocabulary.
- At the output layer, a softmax activation function is used to produce correctly normalized probability values.
- As training criterion the cross entropy error is used which is equivalent to maximum likelihood.

We also follow this approach. It is generally advised to normalize the input data of a neural network ([15]) which means that a linear transformation is applied so that the data have zero mean and unit variance. When using 1-of-K coding, this is obviously not the case.

Giving up the sparseness of the input features (which is usually exploited to speed up matrix computations, cf. [16]), the data can easily be normalized because there exist closed-form solutions for the mean and variance of the 1-of-K encoded input features that depend only on the unigram counts of the words observed in the training data. On the contrary we observed that convergence was considerably slowed down by normalization. It seems that it suffices when the input data in each dimension lie in the same $[0, 1]$ range.

As the input features are highly correlated (e. g., we have $x_i = 1 - \sum_{i \neq j} x_i$) for the i-th dimension of an input variable x), applying a whitening transform to the features appears to be more promising. Because of the high dimensionality, this seems practically unfeasible.

Regarding the network topology, in [6] a single recurrent hidden layer was used, while in [3] an architecture with two hidden layers was applied, the first layer having the interpretation of projecting the input words to a continuous space. In a similar spirit, we stick to the topology shown in Fig. 2 where we plug in LSTM units into the second recurrent layer, combining it with different projection layers of standard neural network units.

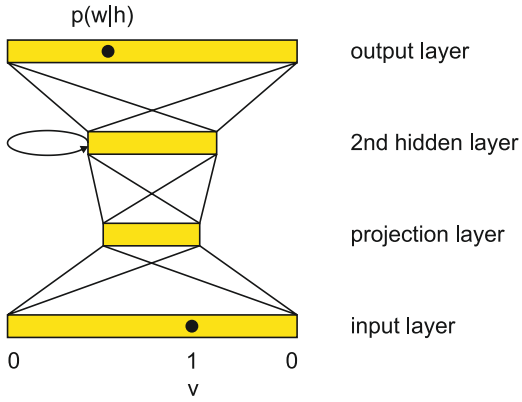

Figure 2: Neural network LM architecture

For large-vocabulary language modeling, training is strongly dominated by the computation of the input activations a_i of the softmax output layer which in contrast to the input layer is not sparse:

$$a_i = \sum_{j=1}^{J} \omega_{ij} b_j.$$

Here, J denotes the number of nodes in the last hidden layer, ω_{ij} are the weights between the last hidden layer and the output layer, and $i = 1, \ldots, V$, where V is the vocabulary size.

To reduce the computational effort, in [17] (following an idea from [18]), it was proposed to split the words into a set of disjoint word classes. Then the probability $p(w_m|w_1^{m-1})$ can be factorized as follows:

$$p(w_m|w_1^{m-1}) = p\big(w_m|c(w_m), w_1^{m-1}\big)p\big(c(w_m)|w_1^{m-1}\big)$$

[1] As opposed to our LSTM version, in [14] the gating units do not receive the activations of the previous hidden layer

where $w_m \in c(w_m)$, and $c(w_m)$ is the class of word w_m. How to define a reasonable set of classes is described in [19]. Using this identity, the computational complexity can be significantly reduced.

4. Experimental results

For the experimental results, we concentrated on two corpora: The English Treebank-3 Corpus and the French corpora from the Quaero[2] project. Details can be found in Table 1.

LM	train	dev1	dev2	test
Treebank	930 K	74 K	–	82 K
Quaero French	27 M	46 K	36 K	35 K

Table 1: Corpus sizes in number of running words; the vocabulary size of the Treebank corpus is 10 K, for Quaero French it is 170 K; dev1 was used as validation data for neural network training, dev2 for optimizing the LM scale

The results for the Treebank-3 corpus are summarized in Fig. 3. First, we trained a recurrent neural network LM with the architecture shown in Fig. 2 except that we omitted the projection layer. For the recurrent hidden layer, once we chose standard units with a sigmoid activation function and once LSTM units, see Fig. 3 (a). We found that the perplexity of the models was constantly lower by about 8 % compared to the standard recurrent neural network. The perplexities we obtained with the sigmoidal recurrent network closely match those obtained with the rnnlm toolkit ([20]).

The training times for these two models are of similar order. However, the corresponding model sizes actually are quite different for a given number of hidden layer nodes: While the LSTM version with 150 hidden nodes corresponds to 7.6 M parameters, the sigmoidal network has only 3.0 M weight parameters. On the other hand, when increasing the model size of the sigmoidal network until a comparable number of parameters is reached, no significant improvement can be obtained (350 hidden nodes correspond to 7.1 M parameters). In addition, when a projection layer is used and the vocabulary size is huge, the overhead in model size of the LSTM variant is negligible.

In a second set of experiments, we tried to find out whether an additional projection layer gives further improvements, cf. Fig. 3 (b). Unfortunately, compared to the raw LSTM version, neither a linear layer (where the activation function is the identity) nor a sigmoidal layer led to lower perplexities. Our interpretation of the results is that such a projection layer creates smeared input features that complicate the learning task for the LSTM units.

For the results we have shown so far, a single input sentence was presented to the network during training and testing. This means that the maximum context length is limited to about 21 words which is the average sentence length in the Treebank corpus. However, unlike for standard recurrent neural networks, LSTM nets might be able to exploit even longer context sizes. Therefore we increased the size of the input sequences by concatenating a fixed number of consecutive sentences. The effect on the performance can be seen in Fig. 3 (c).

We observe that in case of a single hidden layer with LSTM cells, a small improvement is possible when switching from one to two concatenated sentences. The same holds true for an LSTM with a sigmoidal projection layer. Interestingly, in contrast to single sentences, a linear projection layer helps for longer input sequences.

Probably the number of different words the neural network has to distinguish at the output layer is too large for learning complex long-range dependencies. Therefore we observe the general tendency that the perplexity significantly increases when the context length exceeds a certain threshold, regardless of any preprocessing.

However it seems that the smearing of the input features introduced by the linear layer is beneficial for long input sequences, and we obtained best perplexities when LSTM cells were combined with this type of projection layer.

Finally, we investigated the interaction between LSTM networks and a clustered output layer. For this experiment, we used an LSTM network with 200 hidden nodes and no projection layer.

As shown in Fig. 3 (d), the impact of clustering on perplexities is only moderate, while large speed ups are possible for training (as well as for testing). In theory, the speed up should be largest for $C = \sqrt{V}$ where C denotes the number of classes, and V is the vocabulary size. It turns out that this behaviour is not exactly matched in practice because classes have different sizes.

Apart from the results for the comparatively small English corpus, we also applied the LSTM networks to a large vocabulary French speech recognition task. Within the Quaero research project, yearly evaluations are held where speech recognition systems are evaluated on broadcast conversational podcast data.

We took our best French recognition system which showed to be competitive in the 2011 evaluation. The system included state-of-the-art acoustic models including cross-adaptation, MLP-features, and discriminate training. The backing-off LM was trained on more than 4 B words.

From the lattices created by the speech recognizer, we extracted n-best lists of size $n = 1000$. We trained an LSTM LM using 300 hidden nodes and 27 M running words of in-domain training data. Although the Kneser-Ney (KN) backing-off model was trained on more than a hundred times more data, by interpolation, we obtained improvements in word error rate of 0.5 % on the development and 0.3 % on the test data of the 2011 evaluation.

LM	dev2	test
KN 4-gram	19.7 %	17.6 %
KN 4-gram + LSTM	19.2 %	17.3 %

Table 2: Word error rate results for Quaero French.

5. Conclusions

In this paper, we applied the LSTM neural network architecture to two language modeling tasks. This network type is especially well-suited to language modeling as in theory it allows the exact modeling of the probability of a word sequence. As opposed to previous approaches, it does not suffer from conceptual problems of standard recurrent neural network training.

We explored several different neural network topologies and analyzed the importance of the wide-spread use of an additional hidden projection layer. We showed that the LSTM network can be combined with existing clustering techniques to gain large speed ups in training and testing times at a small loss in performance.

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

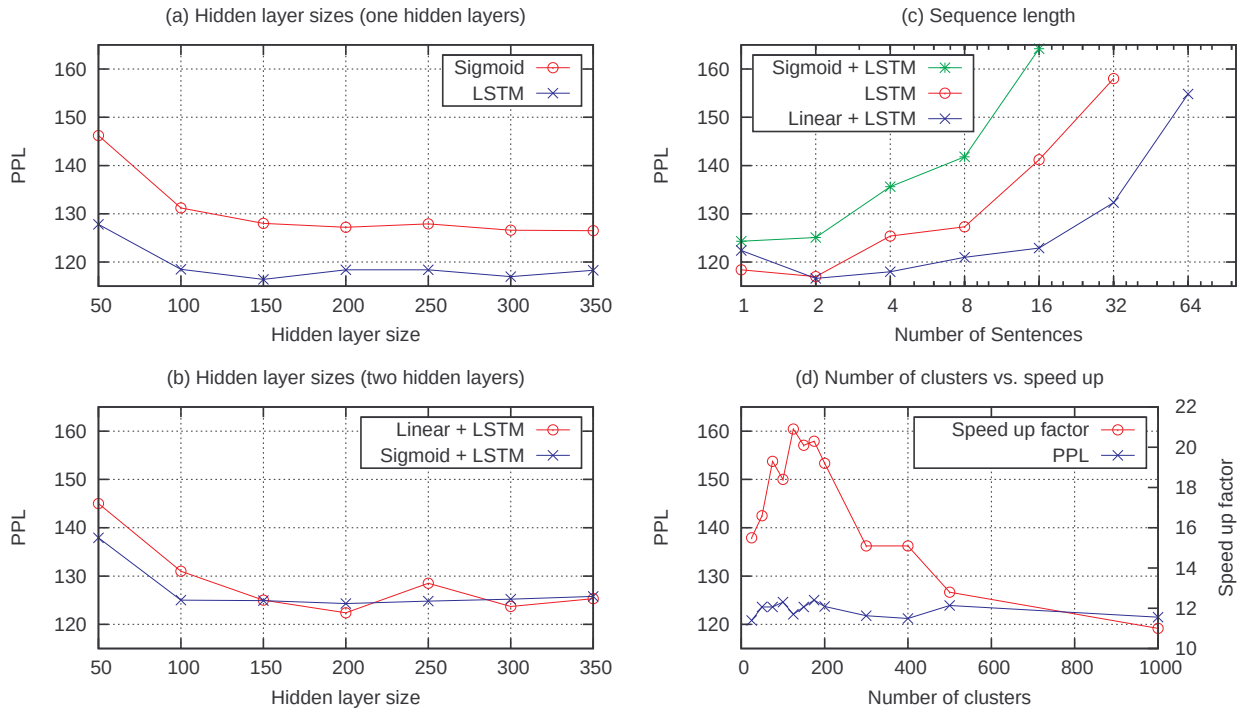

Figure 3: Experimental results on the Treebank corpus; for (c) and (d), 200 nodes were used for the hidden layers.

Experiments suggest that the performance of standard recurrent neural network architectures can be improved by about 8 % relative in terms of perplexity. Finally, comparatively large improvements were obtained when interpolating an LSTM LM with a huge Kneser-Ney smoothed backing-off model on top of a state-of-the-art French recognition system.

For future work, it seems interesting to analyze the differences between standard and LSTM networks and the impact on the recognition quality of a speech recognizer.

6. Acknowledgment

This work was partly realized as part of the Quaero programme, funded by OSEO, French State agency for innovation.

7. References

[1] Kneser, R., and Ney, H., "Improved Backing-Off For M-Gram Language Modeling", Proc. of ICASSP 1995, pp. 181–184

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

[3] Schwenk, H., "Continuous space language models", Computer Speech and Language 21 (2007), pp. 492–518

[5] Oparin, I., Sundermeyer, M., Ney, H., Gauvain, J.-L., "Performance Analysis of Neural Networks in Combination with n-Gram Language Models", Proc. of ICASSP 2012, accepted for publication

[6] Mikolov, T., Karafiát, M., Burget, L., Černoczký, J. H., and Khudanpur, S., "Recurrent neural network based language model" Proc. of Interspeech 2010, pp. 1045–1048

[7] Elman, J., "Finding Structure in Time", Cognitive Science 14 (1990), pp. 179–211

[8] Rumelhart, D. E., Hinton, G. E., Williams, R. J., "Learning representations by back-propagating errors", Nature 323 (1986), pp. 533–536

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

[10] Martens, J., Sutskever, I., "Learning Recurrent Neural Networks with Hessian-Free Optimization", Proc. of the 28th Int. Conf. on Machine Learning 2011

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

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

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

[14] Graves, A., Schmidhuber, J., "Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures", Neural Networks, Vol. 18, Issue 5–6, 2005, pp. 602–610

[15] Bishop, C., "Neural Networks for Pattern Recognition", Clarendon Press, Oxford, 1995

[16] Le, H. S., Allauzen, A., Wisniewski, G., Yvon, F., "Training continuous space language models: some practical issues", Proc. of the 2010 Conf. on Emp. Methods in NLP, pp. 778–788

[17] Morin, F., Bengio, Y., "Hierarchical Probabilistic Neural Network Language Model", Proc. of the 10th Int. Workshop on Artificial Intelligence and Statistics

[18] Goodman, J., "Classes for fast maximum entropy training", Proc. of the ICASSP, 2001

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

[20] Mikolov, T., Kombrink, S., Deoras, A., Burget, L., Černocký, J., "RNNLM – Recurrent Neural Network Language Modeling Toolkit", Proc. of the 2011 ASRU Workshop, pp. 196–201