

RNNLM - Recurrent Neural Network Language Modeling Toolkit

Tomáš Mikolov ^{#1}, Stefan Kombrink ^{#2}, Anoop Deoras ^{*3}, Lukáš Burget ^{#4}, Jan “Honza” Černocký ^{#5}

[#] *Speech@FIT, Brno University of Technology, Brno, Czech Republic*

¹imikolov@fit.vutbr.cz, ²kombrink@fit.vutbr.cz, ⁴burget@fit.vutbr.cz, ⁵cernocky@fit.vutbr.cz

^{*} *Center for Language and Speech Processing, Johns Hopkins University, Baltimore, MD, USA*

³adeoras@jhu.edu

Abstract—We present a freely available open-source toolkit for training recurrent neural network based language models. It can be easily used to improve existing speech recognition and machine translation systems. Also, it can be used as a baseline for future research of advanced language modeling techniques. In the paper, we discuss optimal parameter selection and different modes of functionality. The toolkit, example scripts and basic setups are freely available at <http://rnnlm.sourceforge.net/>.

I. INTRODUCTION, MOTIVATION AND GOALS

Statistical language modeling attracts a lot of attention, as models of natural languages are an important part of many practical systems. Moreover, it can be estimated that with further research progress, language models will become closer to the human understanding of languages [1] [2], and completely new applications will become practically realizable. Immediately, any significant progress in language modeling can be utilized in the existing speech recognition and statistical machine translation systems.

However, the whole research field is struggling for decades to overcome very simple, but also effective models based on n-gram frequencies [3] [4]. Many techniques were developed to beat n-grams, but the improvements came at the cost of computational complexity. Moreover, the improvements were often reported for very basic systems, and after application to state-of-the-art setups with n-gram models trained on huge data sets, the improvements provided by many techniques vanished. This has led to skepticism among speech recognition researchers.

In our previous work, we have compared many well-known advanced language modeling techniques, and we found that neural network based language models (NNLM) perform the best on several standard setups [5]. Models of this type were introduced by Bengio [6] about ten years ago. Their main weaknesses were huge computational complexity, and non-trivial implementation. Successful training of neural network language models require a good choice of hyper-parameters, such as learning rate and size of a hidden layer.

To help to overcome these basic obstacles, we have decided to release our toolkit for training recurrent neural network based language models (RNNLM). We have shown that the recurrent architecture outperforms the feed-forward one on

several setups in [7]. The implementation is simple and easy to understand.

Most importantly, recurrent neural networks are very interesting from the research point of view, as they allow effective processing of sequences and patterns with arbitrary length - these models can learn to store past information in the hidden layer. Recurrent neural networks can have memory, and are thus an important step forward to overcome the most painful and often criticized drawback of n-gram models - statistical dependence on only a few previous words.

In this paper we present an open source and freely available toolkit for training statistical language models based on recurrent neural networks and hash-based maximum entropy models. The toolkit includes techniques for reducing computational complexity (classes in the output layer and direct connections between input and output layer). It has been designed to provide comparable results to the popular toolkit for training n-gram models, SRILM [8]. The main goals for the RNNLM toolkit are these:

- promotion of research of advanced language modeling techniques
- easy usage
- simple portable code without any dependencies on external libraries
- computational efficiency

In this paper we describe how to easily apply RNNLM to almost any speech recognition or machine translation system.

II. RECURRENT NEURAL NETWORK

The recurrent neural network architecture used in the toolkit is shown at Figure 1 (it is usually called Elman network, or simple RNN). The input layer uses the 1-of-N representation of the previous word $w(t)$ concatenated with the previous state of the hidden layer $s(t-1)$. The neurons in the hidden layer $s(t)$ use a sigmoid activation function. The output layer $y(t)$ has the same dimensionality as $w(t)$, and after the network is trained, it represents the probability distribution of the next word given the previous word and the state of the hidden layer in the previous time step [9]. The class layer $c(t)$ can be optionally used to reduce the computational complexity of the model, at a small cost of accuracy [7]. Training is performed

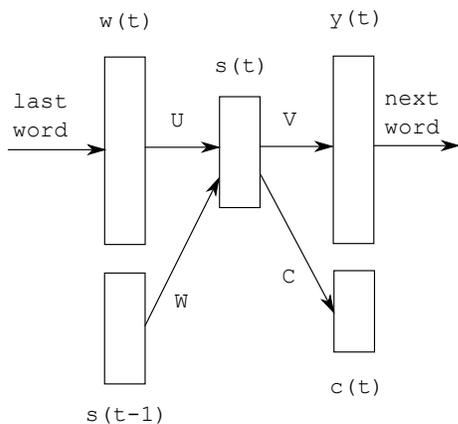


Fig. 1. Recurrent neural network based language model with classes.

by the standard stochastic gradient descent algorithm, and the matrix \mathbf{W} that represents recurrent weights is trained by the backpropagation through time algorithm (BPTT) [10].

In the toolkit, we use truncated BPTT - the network is unfolded in time for a specified amount of time steps. For faster training, it is possible to unfold the recurrent part of the network after processing several time steps, which leads to significantly lower computational complexity during training.

Recurrent neural networks seem to be a very good choice for modeling sequential data. However, RNNs received much skepticism after it was shown that conventional training algorithms based on gradient descent suffer from vanishing and exploding gradients [11]. This has been the reason why RNNs have been sometimes considered to be difficult to train successfully just by gradient descent based methods.

In fact, the problematic part of algorithms such as backpropagation through time [10] can be the actual implementation, as it is easy to make a mistake and the algorithm is hard to debug. A good description of BPTT implementation can be found in [12]. Moreover, the training might diverge in some cases. The stability of the training can be improved by:

- using double instead of single precision of floating point numbers for weights
- limiting the maximum gradient to prevent explosion of gradients
- using regularization
- updating the recurrent weights in one big update [12]

Once the network is trained, the exact values of weights are no longer important - we have recently shown that the values of weights can be quantized to several bits without any significant loss of performance [13].

III. BASIC FUNCTIONALITY

The toolkit supports several functions, mostly for the basic language modeling operations: training RNN LM, training hash-based maximum entropy model (ME LM) and RNNME LM (jointly trained RNN and ME models [14]). For evaluation, either perplexity can be computed on some test data, or n-best lists can be rescored to evaluate impact of the models

on the word error rate or the BLEU score. Additionally, we support option to generate random sequences of words from the model, which can be useful for approximating the RNN models by n-gram models, at a cost of memory complexity [15].

A. Training phase

The input data are expected to be in a simple ASCII text format, with a space between words and end of line character at the end of each sentence. After specifying the training data set, a vocabulary is automatically constructed, and it is saved as part of the RNN model file. Note that if one wants to use limited vocabulary (for example for open-vocabulary experiments), the text data should be modified outside the toolkit, by first rewriting all words outside the vocabulary to `<unk>` or similar special token.

After the vocabulary is learned, the training phase starts (optionally, the progress can be shown if `-debug 2` option is used). Implicitly, it is expected that some validation data are provided using the option `-valid`, to control the number of the training epochs and the learning rate. However, it is also possible to train models without having any validation data; the option `-one-iter` can be used for that purpose. The model is saved after each completed epoch (or also after processing specified amount of words); the training process can be continued if interrupted.

B. Test phase

After the model is trained, it can be evaluated on some test data, and perplexity and \log_{10} probability is displayed as the result. The RNNLM toolkit was designed to provide results that can be compared to the results given by the popular SRILM toolkit. We also support an option to linearly interpolate the word probabilities given by various models. For both RNNLM and SRILM, the option `-debug 2` can be used to obtain verbose output during the test phase, and using the `-lm-prob` switch, the probabilities given by two models can be interpolated. We provide further details in the example scripts at the RNNLM webpage.

For n-best list rescoring, we are usually interested in the probabilities of whole sentences, that are used as a score during the re-ranking. The expected input for the RNNLM is a list of sentences to be scored, with a unique identifier as the first token in each hypothesis. The output is a list of scores for all sentences. This mode is specified by using the `-nbest` switch. Example of n-best list input file:

```
1 WE KNOW
1 WE DO KNOW
1 WE DONT KNOW
2 I AM
2 I SAY
```

IV. TYPICAL CHOICE OF HYPER-PARAMETERS

Due to huge computational complexity of neural network based language models, successful training of models in

a reasonable time can require some experience, as certain parameter combinations are too expensive to explore. There exist several possible scenarios, depending on if one wants to optimize the accuracy of the final model, the speed of the training, the speed of the rescoring or the size of the models. We will briefly mention some useful parameter configurations.

A. Options for the best accuracy

To achieve the best possible accuracy, it is recommended to turn off the classes by `-class 1`, and to perform training for as long as any improvement on the validation data is observed, using the switch `-min-improvement 1`. Next, the BPTT algorithm should run for at least 6 steps (`-bptt 6`). The size of the hidden layer should be as large as possible. It is useful to train several models with different random initialization of the weights (by using the `-rand-seed` switch) and interpolate the resulting probabilities given by all models together [5].

B. Parameters for average-sized tasks

The above parameter choice would be very time consuming even for small data sets. With 20-50 million of training words, it is better to sacrifice a bit of accuracy for lower computational complexity. The most useful option is to use the classes (`-class`), with about $\sqrt{|V|}$ classes, where $|V|$ is the size of the untruncated vocabulary (typically, the amount of classes should be around 300-500). It should be noted that the user of the toolkit is required to specify just the amount of the classes, and these are found automatically based on unigram frequencies of words. The BPTT algorithm should run in a block mode, for example by using `-bptt-block 10`.

The size of the hidden layer should be set to around 300-1000 units, using the `-hidden` switch. With more data, larger hidden layers are needed. Also, the smaller the vocabulary is, the larger the hidden layer should be to ensure that the model has sufficient capacity. The size of the hidden layer affects the performance severely; it can be useful to train several models in parallel, with different sizes of the hidden layers, so that it can be estimated how much performance can be gained by using larger hidden layer.

C. Parameters for very large data sets

For data sets with 100-1000 million of words, it is still possible to train RNN models with a small hidden layer in a reasonable time. However, this choice severely degrades the final performance, as networks trained on large amounts of data with small hidden layers have insufficient capacity to store information. In our previous work, it proved to be very beneficial to train RNN model jointly with a maximum entropy model (which can be seen as a weight matrix between the input and the output layers in the original RNN model). We denote this architecture as RNNME [14] and it should be noted that it performs very differently than just interpolation of RNN and ME models - the main difference is that both models are trained jointly, so that the RNN model can focus on discovering complementary information to the ME model.

A hash-based implementation of ME can be enabled by specifying the amount of parameters that will be reserved

for the hash by using the `-direct` switch (this option just increases the memory complexity, not the computational complexity) and the order of n-gram features for the ME model is specified by `-direct-order`. The computational complexity increases linearly with the order of the ME model, and for model with order N it is about the same as for RNN model with N hidden neurons. Typically, using ME with up to 4-gram features is sufficient. Due to the hash-based nature of the implementation, higher orders might actually degrade the performance if the size of the hash is insufficient. The disadvantage of the RNNME architecture is in its high memory complexity.

V. APPLICATION TO ASR/MT SYSTEMS

The toolkit can be easily used for rescoring n-best lists from any system that can produce lattices. The n-best lists can be extracted from the lattices for example by using the `lattice-tool` from SRILM. A typical usage of RNNLM in an ASR system consists of these steps:

- train RNN language model(s)
- decode utterances, produce lattices
- extract n-best lists from lattices
- compute sentence-level scores given by the baseline n-gram model and RNN model(s)
- perform weighted linear interpolation of log-scores given by various LMs (the weights should be tuned on the development data)
- re-rank the n-best lists using the new LM scores

One should ensure that the input lattices are wide enough to obtain any improvements - this can be verified by measuring the oracle word error rate. Usually, even 20-best list rescoring can provide majority of the achievable improvement, at negligible computational complexity. On the other hand, full lattice rescoring can be performed by constructing full n-best lists, as each lattice contains a finite amount of unique paths. However, such approach is computationally complex, and a more effective approach for lattice rescoring with RNNLM is presented in [16], together with a freely available tool¹.

A self-contained example demonstrating RNN rescoring on an average-sized Wall Street Journal ASR task using a Kaldi speech recognition toolkit is provided in the download section under <http://rnnlm.sourceforge.net>.

Alternatively, one can approximate the RNN language model by an n-gram model. This can be accomplished by following these steps:

- train RNN language model
- generate large amount of random sentences from the RNN model
- build n-gram model based on the random sentences
- interpolate the approximated n-gram model with the baseline n-gram model
- decode utterances with the new n-gram model

This approach has the advantage that we do not need any RNNLM rescoring code in the system. This comes at a cost of

¹http://www.cisp.jhu.edu/~adeoras/HomePage/Code_Release.html

additional memory complexity (it is needed to generate large amount of random sentences) and by using the approximation, in the usual cases it is possible to achieve only about 20%-40% of the improvement that can be achieved by the full RNNLM rescoring. We describe this technique more closely in [15] [17].

VI. CONCLUSION AND FUTURE WORK

The presented toolkit for training RNN language models can be used to improve existing systems for speech recognition and machine translation. We have designed the toolkit to be simple to use and to install - it is written in simple C/C++ code and does not depend on any external libraries (such as BLAS). The main motivation for releasing the toolkit is to promote research of advanced language modeling techniques - despite significant research effort during the last three decades, the n-grams are still considered to be the state of the art technique, and we hope to change this in the future.

We have previously shown that the RNN models are significantly better than n-grams for speech recognition, and that the improvements are increasing with more training data. Thus from the practical point of view, the main problem is to perform fast training of these models on very large corpora. Despite its simple design, the RNNLM toolkit can be used to train very good RNN language models in a few days on corpora with hundreds of million of words.

Future work might focus on incremental improvements, ie. parallelization of the training algorithm [18], training of RNN on a GPU [19], optimized rescoring [16], decreasing memory complexity of the RNNME architecture [20], compression of RNNLMs [13]. However, we also hope that the toolkit will boost research of language models, and will bring into attention some very interesting research problems and questions - whether the language can be learned unsupervisedly from raw textual data, the need for memory in models that process sequential data, questionable usefulness of linguistic knowledge in statistical language modeling, training of advanced RNN architectures that can discover long-range regularities etc. The strategy 'more data is better' has been dominant in the statistical language modeling (and in the automatic speech recognition and machine translation in general) for quite some time; however, by following it, we do not seem to get any closer to human-level performance.

ACKNOWLEDGMENT

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 project No. 102/08/0707, and by Czech Ministry of Trade and Commerce project No. FR-TI1/034. Anoop Deoras was partly funded by HLT-COE Johns Hopkins University.

REFERENCES

[1] J. T. Goodman, "A Bit of Progress in Language Modeling Extended Version," Microsoft Research, Tech. Rep. MSR-TR-2001-72, 2001.
 [2] M. Hutter, "The Human knowledge compression prize," 2006.

[3] F. Jelinek, "Up From Trigrams! The struggle for improved language models," in *Proceedings of Eurospeech*, 1991.
 [4] R. Rosenfeld, "Two decades of statistical language modeling: where do we go from here?" *Proceedings of the IEEE*, vol. 88, pp. 1270-1278, 2000.
 [5] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Černocký, "Empirical evaluation and combination of advanced language modeling techniques," in *Proceedings of Interspeech*, 2011.
 [6] Y. Bengio, R. Ducharme, P. Vincent *et al.*, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137-1155, 2003.
 [7] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proceedings of ICASSP*, 2011.
 [8] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proceedings of ICSLP*, 2002.
 [9] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proceedings of Interspeech*, 2010.
 [10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Mit Press Computational Models Of Cognition And Perception Series*, pp. 318-362, 1986.
 [11] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult." *IEEE transactions on neural networks*, vol. 5, pp. 157-166, 1994.
 [12] M. Bodén, "A guide to recurrent neural networks and backpropagation," in *In the Dallas project, SICS Technical Report T2002:03, SICS*, 2002.
 [13] T. Mikolov, I. Sutskever, A. Deoras, H. S. Le, S. Kombrink, and J. Černocký, "Compression of Language Models Using Subword Neural Networks," in *Submitted to ICASSP*, 2012.
 [14] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, "Strategies for Training Large Scale Neural Network Language Models," in *Accepted to ASRU*, 2011.
 [15] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát, and S. Khudanpur, "Variational Approximation of Long-Span Language Models for LVCSR," in *Proceedings of ICASSP*, 2011.
 [16] A. Deoras, T. Mikolov, and K. Church, "Fast Rescoring Strategy to Capture Long Distance Dependencies," in *Proceedings of EMNLP*, 2011.
 [17] S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent Neural Network based Language Modeling in Meeting Recognition," in *Proceedings of Interspeech*, 2011.
 [18] H. Schwenk, "Continuous space language models," *Comput. Speech Lang.*, vol. 21, pp. 492-518, July 2007.
 [19] I. Sutskever, J. Martens, and G. Hinton, "Generating Text with Recurrent Neural Networks," in *Proceedings of ICML*, 2011.
 [20] P. Xu, S. Khudanpur, and A. Gunawardana, "Randomized Maximum Entropy Language Models," in *Accepted to ASRU*, 2011.