CSLM - A modular Open-Source Continuous Space Language Modeling Toolkit

Holger Schwenk

LIUM, University of Le Mans, France

Holger.Schwenk@lium.univ-lemans.fr

Abstract

Language models play a very important role in many natural language processing applications, in particular large vocabulary speech recognition and statistical machine translation. For a long time, back-off n-gram language models were considered to be the state-of-art when large amounts of training data are available. Recently, so called continuous space methods or neural network language models have shown to systematically outperform these models and they are getting increasingly popular.

This article describes an open-source toolkit that implements these models in a very efficient way, including support for GPU cards. The modular architecture makes it very easy to work with different data formats and to support various alternative models. Using data selection, resampling techniques and a highly optimized code, training on more than five billions words takes less than 24 hours. The resulting models achieve reductions in the perplexity of almost 20%. This toolkit has been very successfully applied to various languages for large vocabulary speech recognition and statistical machine translation.

By making available this toolkit we hope that many more researchers will be able to work on this very promising technique, and by these means, quickly advance the field.

Index Terms: language modeling, continuous space methods, speech recognition, machine translation

1. Introduction

The goal of a statistical language modeling is to estimate prior probabilities of word strings. The dominant approach, since more than twenty years, are so-called n-gram back-off language models (LM). The main idea is to limit the context to a maximum of $n-1$ words:

$$P(w_1,\ldots,w_L) = \prod_{i=1}^{L} P(w_i|w_1,\ldots,w_{i-1}) \quad (1)$$

$$\approx \prod_{i=1}^{L} P(w_i|w_{i-n+1},\ldots,w_{i-1}) \quad (2)$$

To simplify notation, we have not considered border conditions (in practice, the first probability is an unigram, then a bigram and so on). The main challenge in n-gram language modeling is how to estimate the conditional probabilities for all possible word sequences, in particular those not observed in the training data. The standard techniques are backing-off to shorter contexts, interpolation, redistribution of probability mass and smoothing. A very comprehensive overview and comparison of the various methods can be found in [1]. In many applications, the best performing method is *modified Knesner-Ney smoothing* [2]. Many other language modeling techniques were proposed, but most of them failed to outperform back-off n-gram LMs, in particular when large amounts of training data are available. The only approaches we are aware of, are continuous space methods [3, 4] and the model M [5]. As far as we know, an open-source implementation of model M is not available. This paper describes a modular toolkit for continuous space language models (CSLM).

1.1. Continuous space language models

The main drawback of back-off n-gram language models is the fact that the probabilities are estimated in a discrete space. This prevents any kind of interpolation in order to estimate the LM probability of an n-gram which was not observed in the training data. In order to attack this problem, it was proposed to project the words into a continuous space and to perform the estimation task in this space. The projection as well as the estimation can be jointly performed by a multi-layer neural network [3, 4]. The basic architecture of this approach is shown in Figure 1.

A standard fully-connected multi-layer perceptron is used. The inputs to the neural network are the indices of the $n-1$ previous words in the vocabulary h_j=$w_{j-n+1},\ldots,w_{j-2},w_{j-1}$ and the outputs are the posterior probabilities of *all* words of the vocabulary:

$$P(w_j = i|h_j) \qquad \forall i \in [1,N] \quad (3)$$

where N is the size of the vocabulary. The input uses the so-called 1-of-n coding, i.e., the ith word of the vocabulary is coded by setting the ith element of the vector to 1 and all the other elements to 0. The ith line of the $N \times P$ dimensional projection matrix corresponds to the continuous representation of the ith word. Let us denote c_l these projections, d_j the hidden layer activities, o_i the outputs, p_i their softmax normalization, and m_{jl}, b_j, v_{ij} and k_i the hidden and output layer weights and the corresponding biases. Using these notations, the neural network performs the following operations:

$$d_j = \tanh\left(\sum_l m_{jl}\, c_l + b_j\right) \quad (4)$$

$$o_i = \sum_j v_{ij}\, d_j + k_i \quad (5)$$

$$p_i = e^{o_i} \,/\, \sum_{r=1}^{N} e^{o_r} \quad (6)$$

The value of the output neuron p_i is used as the probability $P(w_j = i|h_j)$. Training is performed with the standard back-

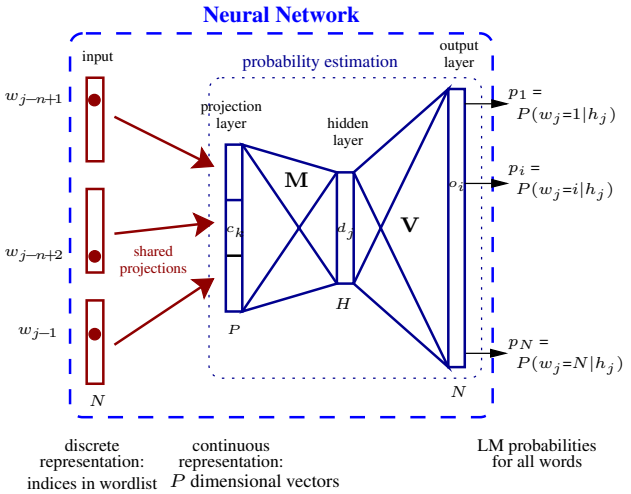

Figure 1: Architecture of the continuous space LM. h_j denotes the context $w_{j-n+1}, \ldots, w_{j-1}$. P is the size of one projection and H,N is the size of the hidden and output layer respectively. When short-lists are used the size of the output layer is much smaller then the size of the vocabulary.

propagation algorithm minimizing the following error function:

$$E = \sum_{i=1}^{N} t_i \log p_i + \beta \left(\sum_{jl} m_{jl}^2 + \sum_{ij} v_{ij}^2 \right) \qquad (7)$$

where t_i denotes the desired output, i.e., the probability should be 1.0 for the next word in the training sentence and 0.0 for all the other ones. The first part of this equation is the cross-entropy between the output and the target probability distributions, and the second part is a regularization term that aims to prevent the neural network from overfitting the training data (weight decay). The parameter β has to be determined experimentally. The CSLM has a much higher complexity than a back-off LM, mainly because of the high dimension of the output layer. One solution is to limit the size of the output layer to the most frequent words, the other ones being predicted by a standard back-off LM [6]. All the words are still considered at the input layer.

It is important to note that the CSLM is still an n-gram approach, but the notion of backing-off to shorter contexts does not exist any more. The model can provide probability estimates for any possible n-gram. It also has the advantage that the complexity only slightly increases for longer context windows, while it is generally considered to be unfeasible to train back-off LMs on billions of words for orders larger than five. A detailed description of the CSLM van be found in [7].

During the last years, several extensions of the basic architecture were proposed in the literature, for instance:

- Mikolov and his colleagues are working on the use of recurrent neural networks instead of multi-layer feed-forward architecture [8, 9].
- A simplified calculation of the short-list probability mass and the addition of an adaptation layer [10, 11]
- the so-called SOUL architecture which allows to cover all the words at the output layer instead of using a short-list [12, 13], based on work by [14, 15].
- alternative sampling in large corpora [16]

Despite significant and consistent gains in large vocabulary continuous speech recognition (LVCSR) and statistical machine translation (SMT), CSLMs are not yet in widespread use. Possible reasons for this could be the large computational complexity which requires flexible and carefully tuned software so that the models can be build and used in an efficient manner. Therefore we provide the freely available CSLM toolkit, hoping to foster research on this approach. The CSLM toolkit is distributed under the LGPL license, version 3.

2. Design and implementation

The CSLM toolkit was designed to be at the same time modular and efficient. For this we haven chosen C++ as the programming language and we rely on highly optimized math libraries for the computational intensive parts.

A neural network can have many different architectures and it is important for a toolkit to be modular so that feature ideas can be implemented without making major changes in the code. In our framework, a neural work is composed of several machines which can be combined in many ways. Each machine is implemented as a C++ class using inheritance when applicable. The basic building blocks are the well known linear, sigmoidal, tanh and softmax machines. Other types can be easily added. Each machine provides a function to propagate the input to the output and to back-propagate the error in the opposite direction. Multiple machines can be combined in several ways:

sequential machine: the input of the second machine is the output of the first machine and so on. This allows to build the standard feed-forward neural networks.

parallel machine: all the machines work in parallel. The dimension of the input and output layers is the sum of the dimensions of the layers of the individual machines. This is used in the CSLM to create the projection layer which combines all the context words.

splitting machine: all the machines share the same input, but have individual outputs which are concatenated.

join machine: all the machines have individual input layers, but they share the same output layer.

Machines can be arbitrarily combined, e.g. a sequential machine can contain a parallel machine or a parallel machine can contain several sequential machines of different lengths. For instance, it was straight forward to use the same toolkit to implement a continuous space translation model for a phrase-based SMT system [17]. In that architecture, we predict several target words given several source words at the input context.

A well known trick is to process several examples at once during back-propagation training (mini-batch). By these means, one can use matrix-matrix operations instead of matrix-vector ones. Matrix-matrix operations can be very aggressively optimized on multi-threaded CPUs and GPU cards, resulting in speed ups of almost one order of magnitude. By changing a single flag in the makefile, all neural network architectures can be either run on a standard CPU with multi-threading support or on Nvidia GPU cards.

2.1. Data handling

For many languages, large amounts of training data are available which can easily sum up to several billions of words. Often data comes from many different sources (e.g. in-domain versus out-of domain) and it is well known that those sources should

be accordingly weighted. The standard procedure for back-off LMs is to build separate models and to merge them into one using interpolation coefficients estimated with an EM procedure.

Neural network training involves cycling many times through the training data. Therefore, we use a binary representation of the texts in order to speed up the processing. Before training, each word is replaced with its index in the wordlist. Multiple heterogeneous sources are supported with a generic resampling procedure [18]. Below, we give an example of a *data description file*. It enumerates all the used binary data files, together with a resampling coefficient. Resampling is implemented by sequentially advancing through the data. An example is skipped if a random number is bigger than the resampling coefficient.

```
DataDescr 1
Preload
ResamplMode 0
ResamplSeed 12345678
ShuffleMode 10
#  <file> <resampl_coeff> <order> <mode>
DataNgramBin indomain.btxt  1.00 4 3
DataNgramBin generic1.btxt  0.30 4 3
DataNgramBin generic2.btxt  0.10 4 3
```

The toolkit supports different data formats: binarized text files, but also standard ASCII data to specify the input and target values of the neural network. Other can be easily added. The above described resampling algorithm will automatically work for any datafile format. By these means, the CSLM toolkit can be also used as a highly efficient library to train neural networks for other applications, e.g. to calculate neural acoustic features or to use deep neural networks for acoustic modeling.

2.2. Training algorithms

Currently, we have only implemented back-propagation training with weight decay regularization (see Eqn 7). Several error function are provided, like MSE, MCE or cross-entropy. Other extensions are under work, in particular deep learning algorithms. In the CSLM toolkit, neural network training is implemented using a `Trainer` class. It builds on classes to access the next training data, to perform a forward pass through the network, to calculate the error and gradients at the output layer, and to perform a backward pass and weight update. The training algorithms are completely independent of the network architecture, as long as the input and output layer dimension matches the on of the training data.

2.3. Inference algorithms

Once the neural network is trained, it can be used in several ways. It is possible to calculate the perplexity on some test data so that it can be compared to back-off LMs, but it is more important to judge its performance in real applications. The toolkit implements very efficient algorithms to rescore n-best lists or lattices. The former ones are usually used in SMT (with the Moses decoder), while it is very common to rely on a lattice representation in LVCSR (using the HTK format). Both representations of the search space are supported by the CSLM toolkit. Rescoring n-best lists and lattices for large tasks usually requires the evaluation of millions of n-grams, however many of them appear multiple times, in particular in n-best lists. Therefore, we have implemented a very efficient algorithm:

- go through the lattice or n-best list and collect all n-gram LM probability requests. The corresponding LM probabilities are not immediately calculated, but we just memorize the address where to store the value.
- all the n-gram probability requests are sorted with respect to the context (n-1)-gram. N-grams with the same contexts, but different following words are grouped together (since all the results are simultaneously available at the output layer of the neural network)
- These n-gram are propagated through the neural network in mini-batches (the usual size is 128 examples) and the probabilities are stored in the memorized addresses.

This procedure allows to substantially improve the speed of lattice and n-best rescoring, in comparison to sequential processing of the n-gram LM probability requests. By these means, it is even possible to use the CSLM toolkit in a real-time LVCSR system. A typical 1000-best list of an SMT system for a test set of 1400 sentences requires 18M 7-gram LM probability requests. This resulted in less than twelve thousand forward passes through the neural network and in average 1500 probabilities were collected at the output layer for each n-gram.

2.4. Executables tools

All the classes to support neural networks, training algorithm, data handling, etc are compiled into one library `libcslm.a`. Based on this library, the following tools are provided:

`cslm_train` this is the main tool to train an CSLM. The main command line arguments are the network to be trained, the training and development data as well various learning parameters (learning rate, number of iterations, etc).

`cslm_eval` evaluation of an existing CSLM on some development data.

`nbest` rescoring of n-best lists with a back-off or CSLM.

`cslm_tool` lattice rescoring with a back-off or CSLM.

In addition, the following tools are provided:

`text2bin` convert text files to the internal binary representation of the CSLM toolkit.

`nn_info` display information on a neural network.

2.5. Revision history

Free software to train and use CSLM was first proposed in [19]. The first version of this toolkit provided no support for short lists or other means to train CSLMs with large output vocabularies. Therefore, it was not possible to use it for LVCSR and large SMT tasks. We extended our tool with full support for short lists during training and inference [20]. This version was available in June 2012. Short lists are implemented as proposed in [10], i.e. we add one extra output neuron for all words that are not in the short list. This probability mass is learned by the neural network from the training data. However, we do not use this probability mass to renormalize the output distribution, we simply assume that it is sufficiently close to the probability mass reserved by the back-off LM for words that are not in the short list. In summary, during inference words in the short-list are predicted by the neural network and all the other ones by a standard back-off LM. No renormalization is performed. We have performed some comparative experiments with renormalization during inference and we could not observe significant differences.

This paper describes a new version V3 available in August 2013. This version adds support to rescore lattices in HTK format so that it can be easily applied to LVCSR tasks. The toolkit supports back-off LMs in the standard ARPA format and the binary formats of the SRILM [21] and KENLM toolkit [22]. We have also substantially improved the training speed on Nvidia GPU cards, in particular the new Tesla Kepler K20 generation.

In our experiments, we achieved best performance on multi-threaded Intel CPUs using Intel's MKL math library, but other open-source math libraries are also supported, in particular Atlas.

3. Experimental results

The CSLM toolkit was used for a large variety of tasks in LVCSR and SMT, achieving each time substantial improvements in perplexity with respect to a carefully tuned state-of-the-art huge back-off LM. These perplexity gain always seem to carry over to significant improvements when the CSLM is integrated into an application. In the following, we give an example of an LVCSR and an SMT task.

3.1. Large vocabulary continuous speech recognition

The CSLM was used in a LVCSR system developed for the French Repere evaluation.[1] The described system was ranked first in this evaluation. The system is based on the Sphinx decoder and many extension from LIUM. About 500 hours were available to train the acoustic models and 1 billion words for the LM. The CSLM was trained on all the data and then used to rescore lattices produced by the baseline system. The results are summarized in Table 1. The CSLM achieved a significant reduction of the word error rate of 1.1% absolute. A detailed description of this system can be found in [23].

Order:	Back-off 4-gram	CSLM 5-gram
WER Test:	18.06%	16.90%

Table 1: WER scores for a back-off LM and the CSLM (French LVCSR system for the 2013 Repere evaluation).

3.2. Statistical machine translation

The CSLM was used in an Arabic/English statistical machine translation system developed for the NIST 2012 OpenMT evaluation. This system is based on the Moses SMT toolkit [24] The system was ranked among the best ones.[2] For this task, about 5 billion words of English texts are available, mainly coming from LDC's Gigaword corpus. We used the data selection method of Moore and Lewis [25] to extract the most relevant data for the task (about 550M words). An interpolated 4-gram back-off LM was trained with the SRILM toolkit [21] using modified Kneser-Ney smoothing. Neither training the back-off LM on all the data nor using longer contexts did improve its perplexity.

We build one CSLM resampling simultaneously in all the corpora (about 15M words at each epoch) . The short list was set to 16k – this covers about 92% of the n-gram requests. Since it is very easy to use large context windows with an CSLM, we trained up to 7-grams. The networks were trained for 20 epochs. This can be done in about 20 hours on a standard PC with an Nvidia GTX580 GPU card. Table 2 summarizes the results with a 4-gram back-off LM and CSLM of order 4 to 7. The CSLM is used to rescore 1000-best lists. The best CSLM achieves a perplexity reduction of 20% relative and an improvement of 1.5 BLEU on the test set. This is a very significant gain with respect to a carefully tuned back-off LM. BLEU is a commonly used measure the judge the performance of a machine translation system [26]. This is a precision metric and higher values are better. More details on these experiments can be found in [20].

Order:	Back-off 4-gram	CSLM 4-gram	CSLM 5-gram	CSLM 6-gram	CSLM 7-gram
Px Dev:	71.1	63.9	59.5	57.6	56.9
BLEU Dev:	58.66	59.76	60.11	60.29	60.26
BLEU Test:	50.75	51.91	51.85	52.23	52.28

Table 2: Perplexity and BLEU scores for a back-off LM and the CSLM (NIST Open MT'12 Ar/En SMT system).

These two examples show that the CSLM can achieve significant improvements with respect to carefully back-off LMs trained on huge amounts of data.

4. Conclusion

We described an open-source toolkit to train continuous space language models and tools to rescore n-best lists and lattices. Training and rescoring is very efficient using either high performance math libraries to support modern multi-threaded CPUs or Nvidia GPU cards. By the means, we are able to train CSLMs on billions of words in about one day. Those CSLMs significantly outperformed huge, carefully tuned, back-off LMs. As an example, we report an 1.1% WER reduction for a French LVCSR system and an 1.5 improvement in the BLEU score for an Arabic/English SMT system. Both systems have achieved excellent rankings in international evaluations.

The toolkit is written in C++ using a modular architecture and freely available under under the LGPL license, version 3, at `http://www-lium.univ-lemans.fr/~cslm`.

We hope that this toolkit will foster research on continuous space language modeling, and that it will give many research sites the opportunity to apply it to large tasks within state-of-the-art LVCSR and SMT systems and to perform research in that area. In particular, we hope that the toolkit will be the framework for many contributions. We plan ourselves several extensions, namely class-based or hierarchical output layer structures, and deep learning algorithms.

5. Acknowledgments

Lattice rescoring for LVCSR systems was provided by Paul Delglise and Yannick Estve from LIUM. The development of the CSLM toolkit has been partially funded by the French Government under the project COSMAT (ANR-09-CORD-004), the European Commission under the projects FP7 EuromatrixPlus and MateCat, and DARPA under the BOLT project.

[1] `http://www.defi-repere.fr`

[2] All the official results are available at `http://www.nist.gov/itl/iad/mig/openmt12results.cfm`

6. References

[1] S. F. Chen and J. T. Goodman, "An empirical study of smoothing techniques for language modeling," *Computer Speech & Language*, vol. 13, no. 4, pp. 359–394, 1999.

[2] J. T. Goodman, "A bit of progress in language modeling," *Computer Speech & Language*, vol. 15, pp. 403–434, 2001.

[3] Y. Bengio and R. Ducharme, "A neural probabilistic language model," in *NIPS*, vol. 13, 2001, pp. 932–938.

[4] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *JMLR*, vol. 3, no. 2, pp. 1137–1155, 2003.

[5] S. F. Chen, "Shrinking exponential language models," in *NAACL*, 2009.

[6] H. Schwenk, "Efficient training of large neural networks for language modeling," in *IJCNN*, 2004, pp. 3059–3062.

[7] ——, "Continuous space language models," *Computer Speech and Language*, vol. 21, pp. 492–518, 2007.

[8] T. Mikolov, M. Karafit, L. Burget, J. ernock, and S. Khudanpur, "Recurrent neural network based language model," in *Interspeech*, 2010, pp. 1045–1048.

[9] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *ICASSP*, 2011, pp. 5528–5531.

[10] J. Park, X. Liu, M. J. F. Gales, and P. C. Woodland, "Improved neural network based language modelling and adaptation," in *Interspeech*, 2010, pp. 1041–1044.

[11] X. Liu, M. J. F. Gales, and P. C. Woodland, "Improving LVCSR system combination using neural network language model cross adaptation," in *Interspeech*, 2011, pp. 2857–2860.

[12] H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language model," in *ICASSP*, 2011, pp. 5524–5527.

[13] H.-S. Le, I. Oparin, A. Messaoudi, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Large vocabulary SOUL neural network language models," in *Interspeech*, 2011.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

[15] A. Mnih and G. Hinton, "A scalable hierarchical distributed language model," in *NIPS*, 2008.

[16] P. Xu, A. Gunawardana, and S. Khudanpur, "Efficient subsampling for training complex language models," in *EMNLP*, 2011, pp. 1128–1136.

[17] H. Schwenk, "Continuous space translation models for phrase-based statistical machine translation," in *Coling*, 2012, pp. 1071–1080.

[18] H. Schwenk and J.-L. Gauvain, "Training neural network language models on very large corpora," in *EMNLP*, 2005, pp. 201–208.

[19] H. Schwenk, "Continuous space language models for statistical machine translation," *The Prague Bulletin of Mathematical Linguistics*, no. 93, pp. 137–146, 2010.

[20] H. Schwenk, A. Rousseau, and M. Attik, "Large, pruned or continuous space language models on a GPU for statistical machine translation," in *NAACL-HLT workshop on the Future of Language Modeling for HLT*, 2012, pp. 11–19. [Online]. Available: http://www.aclweb.org/anthology/W12-2702

[21] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *ICSLP*, 2002, pp. II: 901–904.

[22] K. Heafield, "KenLM: Faster and smaller language model queries," in *Sixth Workshop on SMT*, 2011.

[23] F. Bougares, P. Delglise, Y. Estve, and M. Rouvier, "LIUM ASR system for etape french evaluation campaign: experiments on system combination using open-source recognizers," in *International Conference on Text, Speech and Dialogue*, 2013.

[24] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst, "Moses: Open source toolkit for statistical machine translation," in *ACL, demonstration session*, 2007.

[25] R. C. Moore and W. Lewis, "Intelligent selection of language model training data," in *ACL*, 2010, pp. 220–224.

[26] K. Papineni, S. Roukos, T. Ward, and W. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *ACL*, 2002, pp. 311–318.