Efficient Training Strategies for Deep Neural Network Language Models

Holger Schwenk **Fethi Bougares** **Loïc Barrault**
LIUM, University of Le Mans
72085 Le Mans cedex, FRANCE
Holger.Schwenk@lium.univ-lemans.fr

Abstract

Many works have shown that neural network language models consistently achieve significant improvements in applications like speech recognition and statistical machine translation. However, little research is devoted to explore optimal training strategies. This paper presents an extensive study on the best-practice to train large neural network language models on a corpus of more than 5.5 billion words. We provide solutions to questions like: how to train a neural network on so many examples using the back-propagation algorithm and data selection ? Is careful initialization important ? How to speed-up training ? Can we benefit from deep architectures ? Our best neural network language model can be trained in less than 40 hours on a GPU card and achieves a 25% perplexity reduction. An important finding is that deep architectures systematically achieve better translation quality than shallow ones. We also investigate training of feed-forward architectures on context sizes of more than 30 words. By these means, we aim at achieving an *"unlimited history"* similar to recurrent architectures.

1 Introduction

Language models play a very important role in many natural language processing applications, in particular large vocabulary speech recognition (LVCSR) and statistical machine translation (SMT). For those applications, and many others, it was considered that back-off n-gram language models were the state-of-art when large amounts of training data are available. Neural network language models (NNLM), also called continuous space language models (CSLM), were introduced thirteen years ago [4]. During the last years, NNLMs became very popular and it was confirmed in many studies that they systematically outperform back-off n-gram models by a significant margin in SMT and LVCSR.

In the neural network community it is common to carefully optimize the meta-parameters of neural networks and it is well known that optimal training can have an important impact on the performance, in particular with deep architectures, e.g. [3, 12]. We are not aware of on extensive study on the optimal training of the neural networks used in an CSLM, often default parameters are used and they are trained on a subset of the potentially available data only. In this paper we will show that careful training of the neural network, and the use of deep architectures, can significantly improve the performance of an CSLM.

The goal of a statistical language modeling is to estimate prior probabilities of word strings $w = w_1, \ldots, w_L$:

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

It was considered unfeasible to estimate these probabilities for very long contexts since many events are never observed in the training data and the size of memory-based models increases very quickly. Therefore, we usually make the approximation to limit the context to the $n-1$ preceding words (see Eqn 1), the so-called n-gram back-off language models (LM). 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 comparison of those technique can be found in [6]

2 Continuous space language models

The main drawback of back-off n-gram LMs is the fact that the probabilities are estimated in a discrete space. This prevents any kind of interpolation in order to estimate LM probabilities of un-observed n-grams.

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 [4].

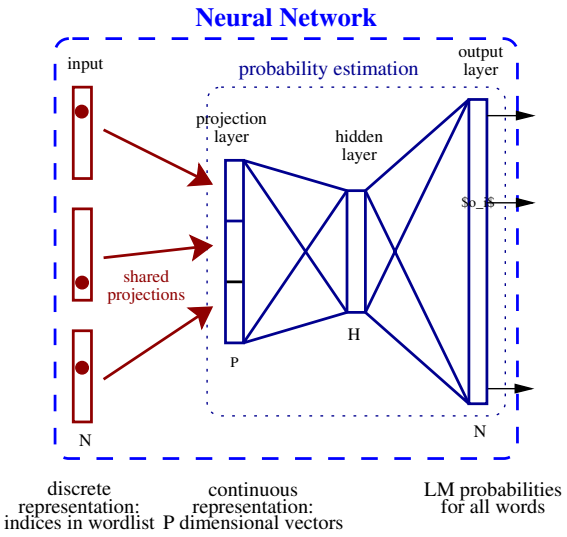

Figure 1: Architecture of the CSLM.

The basic architecture of this approach is shown in Figure 1. In the initial model a standard fully-connected multi-layer perceptron with two hidden layers 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: $P(w_j = i|h_j), \forall i \in [1, N]$, 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 (1st hidden layer). The remaining part of the network estimates the probabilities of all words in the vocabulary given that context. Usually one tanh hidden and a softmax output layer are used. In this paper, we will investigate much deeper architectures. Training is performed with standard back-prop minimizing a cross-entropy error and a weight decay regularizer.

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, called short-list, the other ones being predicted by a standard back-off LM [21]. All the words are still considered at the input layer. Other solutions are classes [15], an hierarchical decomposition [11] or noise contrastive estimation [16]. 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. A detailed description of the CSLM can be found in [21].

During the last years, several extensions of the basic architecture were proposed in the literature. Given the fact that we want to estimate the probability of the next word given all the preceding ones, see Eqn 1, recurrent architectures seem to be particularly appropriate. This was investigated in the literature. T. Mikolov introduced recurrent neural networks, [14] and follow-up work. More recently, LSTM were also used for language modeling, e.g. [26]. From our point of view, it is still

an open question whether these recurrent architectures outperform feed-forward networks. In some work, better performance was observed, e.g. [13, 25], while in other studies no significant difference was observed, e.g. [10]. For many languages, huge amounts of training data are available, usually billions of words. After our experience, taking advantage of all data, using appropriate training techniques, has an important impact on the performance of the neural network LM. In general, it is more challenging to train recurrent architectures on large amounts of data. For these reasons, we investigate feed-forward architectures only in this paper. However, we will show in section 5 that standard feed-forward architectures can very efficiently learn long distance dependencies.

3 Statistical machine translation

The standard metric to evaluate the quality of an LM is to report perplexity on some test set. Perplexity measures the ability of the model to predict the next word in the word sequence. The smaller the value, the less uncertainty the model has, the better it is. We also integrate the CSLM into a full SMT system. Suppose we want to translate a sentence in the source language $\mathbf{s} = s_1 \dots s_i$ to a sentence in the target language $\mathbf{t} = t_1 \dots t_j$. Then, the fundamental equation of SMT is:

$$t^* = \arg\max_{\mathbf{t}} P(\mathbf{t}|\mathbf{s}) = \arg\max_{\mathbf{t}} \frac{P(\mathbf{s}|\mathbf{t})P(\mathbf{t})}{P(\mathbf{s})} = \arg\max_{\mathbf{t}} P(\mathbf{s}|\mathbf{t})P(\mathbf{t}) \tag{2}$$

The translation model $P(\mathbf{s}|\mathbf{t})$ is estimated from bilingual sentence aligned data and the language model $P(\mathbf{t})$ from monolingual data in the target language. This equation can be generalized in order to introduce additional models h_i, in a log-linear framework:

$$\mathbf{t}^* = \arg\max_{\mathbf{t}} P(\mathbf{t}|\mathbf{s}) = \arg\max_{t} \{exp(\sum_i \lambda_i h_i(\mathbf{t}, \mathbf{s}))\} \tag{3}$$

The coefficients λ_i are numerically optimized to maximize a scoring function on a development set [18]. In this work, we add an additional model, the CSLM. In practice, we first create with the baseline system a list of the 1000 most likely translations of each source sentence. For each of these hypothesis, we calculate a score with the CSLM, a new global score is calculated according to equation 3 and the best scoring hypothesis is extracted. We use the freely available Moses SMT toolkit [9].

Many automatic quality measures were proposed in the SMT community, the most popular being BLEU [19] and TER [23]. Both compare the hypothesis to a reference translation. BLEU is an n-gram precision metric, i.e. higher values are better, while TER is an error rate, i.e. lower values are better. TER is an adaptation of the well known word error rate (which counts substitutions, insertion and deletions of words) to machine translation. One of major challenges in the evaluation of machine translation is the fact that word order is usually not preserved. Therefore, TER also consider shifts of word sequences. Our main metric is the combination of both, (TER-BLEU)/2, since we aim at simultaneously maximizing BLEU and minimizing TER.

4 Efficient training of deep architectures

In this first part, we report results on a large SMT task, translating from Arabic into English. This research is performed in the framework of the DARPA BOLT project.[1]. This projects aims in obtaining major breakthroughs in the automatic translation of dialectal Arabic and informal Chinese into English. Therefore the LM must be able to deal with phenomena of this genre: repetitions, errors in grammar and word choice, unusual expressions, etc. In the next two sections, we first describe the available resources for this tasks and the procedure to create the reference back-off LM.

4.1 Task and resources

Table 1 summarizes the resources available to train the LM. In total, we have more than 5.6 billion words out of which only a very small fraction can be considered as in-domain, i.e. of informal genre. The translation model of an SMT system is trained on parallel data, i.e. text in the source

[1] **B**road **O**perational **L**anguage **T**ranslation

Training data	genre	words	subset
Parallel	informal	7.1M	7.1M
(target side)	mixed	178M	19.7M
Monolingual	news	4 800M	173M
	informal	666M	33.3M
Total		**5 651M**	**233M**

Metric	Back-off 4-gram px=76.3	7-gram CSLM px=61.6
BLEU	26.64	27.22
TER	58.38	57.94
TB2	15.87	**15.36**

Table 1: Available corpora for language modeling (left) and performance of the baseline systems on the test set (right). TB2 stands for (TER-BLEU)/2.

language and their translations. The target side of these corpora can be used for language modeling (parallel in Table 1). This is complemented by huge collections of news texts and informal forum discussions. All English data is lower-cased since the capitalization is done by a post-processing module (not discussed here).

Data selection

It is well known that it is sub-optimal to directly build the LM on the concatenation of the individual corpora – instead the relevance of each source should be taken into consideration. For this reason, we build a back-off LM on each corpus and then interpolate and merge them into one final LM. The interpolation coefficients are estimated using an EM procedure to minimize the perplexity on the development data.[2] We use modified Kneser-Ney smoothing which is reported to consistently obtain the best performance. The SRILM toolkit was used for these tasks [24]. Building LMs on all the available data (5.6G words) is a computational challenge and leads to huge models (in the order of 40GB). Recent work has shown that it is actually better to select a subset of the most relevant data. We applied the method presented in [17]. The main idea is to focus on sentences which are close to the domain of the task, using a criterion based on perplexity difference. In our case, about 230M words were selected out of a pool of 5.6G. An 4-gram back-off trained on that subset achieves a perplexity of 76.3. The back-off LM trained on all the 5.6G words achieves the same perplexity. We also build a 5-gram back-off LM using the same procedure. It's perplexity is 75.7 – only a minor improvement.

4.2 Optimizing the meta-parameters of the CSLM

All our CSLMs are trained on exactly the same subset of 233M words than the reference back-off LM. This is substantially larger than in many other application of NNLMs. The networks are trained for 20 epochs consisting of about 30M randomly sampled examples at each epoch.

Network initialization

Proper initialization of the parameters is known to be quite important, in particular with deep architectures. Indeed, we were not able to properly train networks with many hidden layers using a simple random initialization in the range $[-0.1, 0.1]$, independently of the size of the layers. In our case, scaling the random values for each layer in function of its input and output dimensions [7] has shown to be very useful to train deep architectures. The main idea is to scale the random values separately for each layer in function of its input and output dimensions (Equation 4).

$$range = \pm\sqrt{\frac{6}{\text{input\_dim} + \text{output\_dim}}} \tag{4}$$

Batch mode and learning rate

We first tempted to speed-up training by processing several examples at once, also known as mini batch. Values between 128 and 2048 were analyzed. Larger values result in faster processing on GPUs (twice as fast for 2048 w/r to 128), but the networks need more epochs to converge. The

[2] this is performed by the script `compute-best-mix` in the SRILM toolkit.

best compromise was a batch size of 256. With this setting, the overall training takes about 40h on an Nvidia Tesla GPU K20x GPU. The initial learning rate was set to 0.06 which was scaled by the square root of the batch size. The learning rate decreases with the epochs.

Dealing with short n-grams

The CSLM can be easily trained on contexts longer than the usual 4-grams since its complexity only slowly increases. For instance, in [22] improvements are reported when increasing the context from 3 to 6 words. However, the input of a standard CSLM is of constant dimension, i.e. it is only trained on 7-grams and can't directly estimate the probabilities of shorter n-grams. This variable length problem is of course nicely addressed by recurrent neural network LMs [14], but it can be also handled with fixed-size feed-forward multi-layer networks. For this, we introduced a special token, `NULL_WORD`, to indicate shorter n-grams. These are used during training and inference. In our case, about 24% of the n-grams during training, and 21% during n-best list rescoring have a shorter context. Handling these ones by the CSLM lead to a reduction of the perplexity by almost two points.

Network architecture

One of the most important meta-parameter of neural networks is probably its architecture: should we use one or more hidden layers ? How many neurons per layer ? During the last years, there is a large body of research showing that deep architectures, i.e. with many hidden layers, demonstrate improved performance compared to shallow architectures for many tasks, e.g. [2, 8].

We are only aware of one research studying deep architectures for NNLMS. Deep neural networks were shown to outperform shallow architectures for a LVSCR task [1]. However, only a very small corpus was used to train the neural networks (about 23.5M words). In the following, we provide an extensive analysis of different neural network architectures, ranging from one to four hidden layers. For each one we give the perplexity on the development data, the number of parameters,[3] the time to perform one epoch through the training data, and the BLEU and TER score on our test set (22k words). We consider the number of hidden layers used for the **probability estimation task**, not the number of total layers, i.e. the projection layer itself is not included. The network depicted in Figure 1 is therefore considered to have one hidden layer. In preliminary experiments, we analyzed different sizes of the short list: 8k, 16k and 32k. A value of 8k has an negative impact on the performance, but we observed only minor differences between 16k and 32k. Therefore, all following experiments were done with a short list of 16k.

From Table 2, it can be clearly seen that networks with only one hidden layer perform badly, even when the dimension of this hidden layer is substantially increased. The large network with one 2048-dimensional hidden layer performs worse than a network with two hidden layers and half of the parameters. The perplexities are almost 2.5 points lower when two hidden layers are used. However, increasing the size of the layers has no notable impact – the neural network is not able to efficiently use this additional capacity: the architecture `3072-1536` has twice as many parameters than the network `1024-1024`, but achieves identical performance with respect to perplexity and translation quality. We did not observe a notable change in the perplexity when three hidden layers are used for the probability estimation task. However, those networks perform better when integrated into the statistical machine translation system: the combined metric (TER-BLEU)/2 decreases from 14.82 (architecture `1024-1024`) to 14.65 (architecture `768-768-768`), despite the fact that the deep architecture has 23% less parameters. Again, increasing the size of the layers has not significant impact. Using four hidden layers for the probability estimation task does not bring additional improvements. Note that we only use standard back-propagation training in these experiments – more elaborated techniques like layer-wise pre-training may be more appropriate for this type of architectures. The best CSLM uses a 320-dimensional projection layer and three hidden layers of dimension 768 for the probability estimation task. It achieves a reduction in perplexity by 25% relative with respect to the best back-off LM we were able to build (76.3→57.4). Note that we do not interpolate the CSLM with the back-off LM to achieve this value. When using this CSLM to rescore 1000-best lists, we achieve an BLEU score of 27.96 and an TER of 57.27 on the test

[3] We do not include the parameters needed for the projection layer since this is constant for all architectures. Also, these parameters can be updated very quickly since for each example only 320 values need to be changed.

Network architecture	Px	Nbr of parameters	Training time [min]	SMT performance BLEU	TER	TB2
Back-off 4-gram	76.3	n/a	n/a	26.64	58.38	15.87
Initial CSLM, 1024-384	61.0	14.9M	60m	27.22	57.94	15.36
One hidden layer CSLM						
512	62.7	9.4M	26m	27.58	57.61	15.01
1024	59.8	18.8M	46m	27.60	57.62	15.01
2048	60.0	37.5M	80m	27.74	57.92	15.09
Two hidden layer CSLM						
1024-1024	57.5	19.8M	46m	27.85	57.49	14.82
1536-1536	57.3	30.5M	58m	27.95	57.67	14.86
3072-1536	57.6	35.8M	68m	27.77	57.51	14.87
Three hidden layer CSLM						
768-768-768	**57.4**	**15.3M**	67m	**27.96**	**57.27**	**14.65**
1536-1024-768	57.3	17.9M	74m	27.96	57.57	14.80
1536-1536-1536	56.9	32.8M	63m	28.00	57.39	14.69
Four hidden layer CSLM						
1536-1024-768-768	57.5	18.5M	50m	27.95	57.29	14.67
1536-1536-1024-1024	57.0	24.7M	83m	27.78	57.43	14.82
2048-2048-1536-1536	57.2	38.8M	78m	27.86	57.77	14.96
3072-2048-1536-1536	57.0	42.9M	83m	27.78	57.51	14.86

Table 2: Performance summary for different architectures of the neural network language model. All networks use a short list of 16k. The projection layer is of dimension $6 \times 320 = 1920$. It is not counted in the number of "hidden layers".

set. This is substantially better than the neural network LM trained with "default parameters": the combined metric (TER-BLEU)/2 decreased from 15.36 to 14.65.

5 Using long contexts

To the best of our knowledge, most of the experiments with feed-forward neural network LMs are performed using "usual" language model orders, in the range of 4 to 6. Recurrent neural networks have the theoretical advantage that they can capture a long history and they have been successfully applied to language modeling [14]. However, there are theoretical arguments that gradients in recurrent neural networks tend to vanish over time [5]. It seems that in practice the recurrent neural networks tends to "forget" the context after less than ten words. Also, efficient training of recurrent neural networks is tricky since the examples need to be presented in sequential order.

In this section work, we report results on training standard feed-forward architecture with a large fixed-size context, up to 30-gram. Shorter n-grams are handled by using a special `NULL_WORD` token as described above. In comparison to recurrent neural networks, all the inputs have the same importance since they are seen simultaneously. There is no effect of "forgetting" over time until a context of 29 words. We will provide experimental result that increasing the length of the context up to 10 words does actually improve performance, but not beyond.

5.1 Task and resources

The results reported below were also obtained for a Arabic/English translation task, but using a different set of data. We switched to the news domain for which the sentences are in average longer (which makes it more appropriate to evaluate the impact of long context language models). The available resources for this task are summarized in Table 3.

We trained an 4-gram back-off LM and CSLMs of various orders on this data, i.e. a total of more than one billion words after data selection. The neural networks were trained for 30 epochs. At each epoch, we randomly sample about 30M words from the selected subset. The projection layer has a dimension of 320, followed by three 1024-dimensional tanh hidden layers and a softmax output

Training data	genre	words	subset
Parallel	mixed	248M	42M
Monolingual	news	5.1G	1.1G
Total		**5348M**	**1142M**

Table 3: Resources available for Arabic/English news translation task.

layer with a 32k shortlist. The learning rate was set to 0.06 and it was decreased over the iterations. As described in section 4.2, the networks weights were initialized to small random values which are scaled in function of the dimension of the layers. The size of the context window has only a minor impact on the training speed: it ranges from 1h13 for an 4-gram to 1h30 for an 30-gram (per epoch on a Nvidia Tesla K40 GPU). This means that feed-forward CSLMs can be very efficiently trained, even with large amounts of data and large context windows.

Figure 2 shows the perplexities on the development data for the different models. The baseline 4-gram back-off achieves a perplexity of 75.0. An CSLM of the same order achieves 60.0 which is already an 20% relative improvement. In addition, the CSLM rapidly improves with increasing context size, reaching a perplexity of 47.6 for an 10-gram and around 45 for an 16-gram and beyond. This is an 40% relative improvement with respect to the 4-gram back-off LM. Similar tendencies were observed for other tasks. We are not aware of reports that back-off LMs achieve notable improvements in perplexity beyond an order or 5. On the other hand, our experiments clearly show that the CSLM can take advantage of this longer contexts.

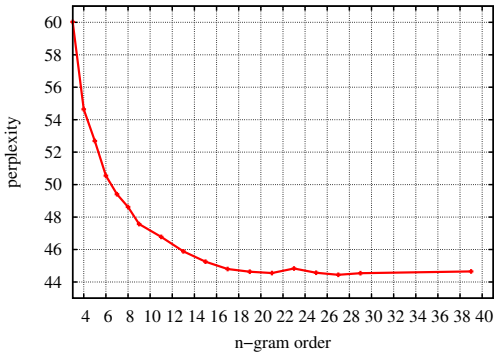

Figure 2: Perplexity of the CSLM in function of the LM order (Arabic/English news task). The 4-gram back-off LM has a perplexity of 75.0.

Figure 3 left shows the BLEU score as a function of the CSLM order. The 4-gram CSLM achieves an improvement of about 1 point BLEU in comparison to the 4-gram back-off LM (57.5→58.6). Further increasing the order of the CSLM substantially improves the BLEU score, reaching an overall gain of 2.7 BLEU for an 12-gram (BLEU 60.2). Even longer contexts do not change significantly the BLEU scores while there is still a (small) difference in perplexity (cf. Figure 2). A quite similar tendency was observed with an Arabic/French SMT system (see Figure 3 right). The overall gain brought by the CSLM is 2.5 BLEU (48.5→51.0).

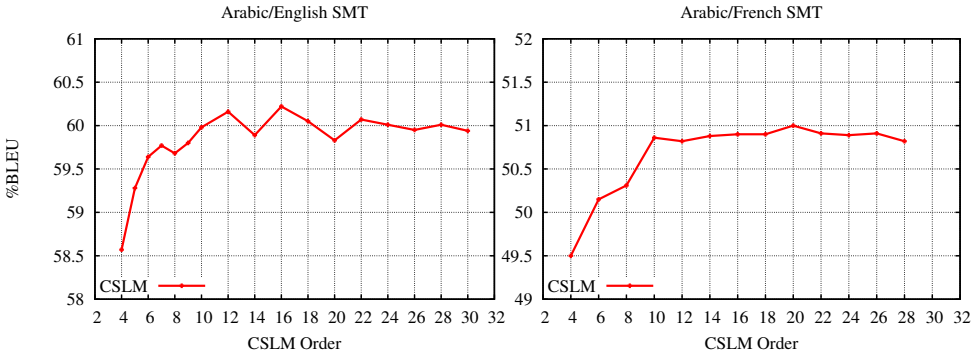

Figure 3: BLEU scores of the CSLM in function of the LM order.
Left: Arabic/English (baseline: 57.5), Right: Arabic/French (baseline: 48.50)

These results could be seen as evidence that it is possible to learn long distance dependencies in a very efficient manner with deep feed-forward architectures. Training on more than 1 billion examples (using resampling) took less than two days. Training recurrent or LSTM networks on as much data is more challenging. Also, applying the notion of a deep architectures to recurrent neural networks is still research question, see for instance [20]. This also confirms similar results by [10]. In that work, the authors report improvements when increasing the context size of a feed-forward CSLM up to 9 words. The authors were not able to beat this feed-forward network with a recurrent neural network LM.

6 Conclusion

This paper provided an extensive study of techniques to improve the training of large neural network language models. The investigated meta-parameters include the size of the mini batch, the learning rate, the network initialization and the network architecture. We also propose to handle short n-grams by the neural network. We hope that this study will help other researchers to optimally train the neural network on their data. All described experiments were performed with the open-source CSLM toolkit.[4] To the best of our knowledge, this is the first study showing that deep architectures for language modeling do significantly outperform shallow ones, even with huge amounts of training data. We were able to train neural networks with about 15 million parameters on more than 5 billion words in 24 hours on a GPU card using the back-propagation algorithm. Key ingredients are: data selection and weighted resampling to focus on the most relevant data, the use of mini batch mode to process several examples at once, careful network initialization, processing of short n-grams by the neural network and the use of four hidden layers. Our neural network language model achieves a perplexity reduction of 25% with respect to the best back-off language model we were able to build. We confirmed that deep architectures generalize better than shallows one, and that they need less parameters. This improved generalization behavior is mainly observed when integrating the neural network language models into a state-of-the-art statistical machine translation system, and less when just analyzing their perplexity. These results suggest that comparing perplexities on some artificial small task may be insufficient to compare different language models.

In a second large scale experiment, we showed that deep feed-forward architectures are able to learn long distance dependencies in a very efficient way. We observed significant improvements in perplexity when using a context window of up to 30 words, reaching gains of 40% relative with respect to the best back-off LM we were able to build. The increased context size also leads to better translation quality when these networks are integrated into an SMT system. One could therefore argue that such long-distance feed-forward neural network LM are likely to have the same modeling power than recurrent archtectures. It rather seems that the main challenge is to efficiently train each model on all available data.

Acknowledgments

This work was partially financed by the DARPA Bolt project and the European Commission (project MateCat ICT-2011.4.2 – 287688).

References

[1] Ebri Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. Deep neural network language models. In *NAACL-HLT workshop on the Future of Language Modeling for HLT*, pages 20–28, 2012.

[2] Yoshua Bengio. Learning deep architectures for ai. Technical report, Univesité de Montréal, 2007.

[3] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade, Second Edition*. Springer, 2012.

[4] Yoshua Bengio and Rejean Ducharme. A neural probabilistic language model. In *NIPS*, volume 13, pages 932–938, 2001.

[4] http://www-lium.univ-lemans.fr/˜cslm/

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on NN*, 5(2):157–166, 1994.

[6] Stanley F. Chen and Joshua T. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward networks. In *AISTATS*, pages 249–256, 2012.

[8] G. Hinton, L. Deng, D. Yu, G. Dahl, A.Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, , and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 20, 2012.

[9] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *ACL, demonstration session*, 2007.

[10] Hai-Son Le, Alexandre Allauzen, and François Yvon. Measuring the influence of long range dependencies with neural network language models. In *NAACL-HLT workshop on the Future of Language Modeling for HLT*, 2012.

[11] Hai-Son Le, I. Oparin, A. Allauzen, J-L. Gauvain, and F. Yvon. Structured output layer neural network language model. In *ICASSP*, pages 5524–5527, 2011.

[12] Yann LeCun, Léon Bouttou, Geneviève B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the trade*. Springer, 1998.

[13] Tomáš Mikolov, A. Deoras, S. Kombrink, and L. Burget anad J. Černocký. Empirical evaluation and combination of advanced language modeling techniques. In *Interspeech*, pages 605–608, 2011.

[14] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, pages 1045–1048, 2010.

[15] Tomáš Mikolov, S. Kombrink, L. Burget, J.H. Cernocky, and S. Khudanpur. Extensions of recurrent neural network language model. In *ICASSP*, pages 5528–5531, 2011.

[16] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *NIPS*, 2013.

[17] Robert C. Moore and William Lewis. Intelligent selection of language model training data. In *ACL*, pages 220–224, 2010.

[18] Franz Josef Och. Minimum error rate training in statistical machine translation. In *ACL*, pages 160–167, 2003.

[19] K. Papineni, S. Roukos, T. Ward, and W.J. Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[20] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. In *ICLR*, 2014.

[21] Holger Schwenk. Continuous space language models. *Computer Speech and Language*, 21:492–518, 2007.

[22] Holger Schwenk, Anthony Rousseau, and Mohammed 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*, pages 11–19, 2012.

[23] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *ACL*, 2006.

[24] Andreas Stolcke. SRILM - an extensible language modeling toolkit. In *ICSLP*, pages II: 901–904, 2002.

[25] Martin Sundermeyer, Ilya Oparin, Jean-Luc Gauvain, B. Freiberg, Ralf Schlüter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *ICASSP*, 2013.

[26] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM neural networks for language modeling. In *Interspeech*, 2012.