PBML

The Prague Bulletin of Mathematical Linguistics
NUMBER 102 OCTOBER 2014 81–92

OxLM: A Neural Language Modelling Framework for Machine Translation

Paul Baltescu[a], Phil Blunsom[a], Hieu Hoang[b]

[a] University of Oxford, Department of Computer Science
[b] University of Edinburgh, School of Informatics

Abstract

This paper presents an open source implementation[1] of a neural language model for machine translation. Neural language models deal with the problem of data sparsity by learning distributed representations for words in a continuous vector space. The language modelling probabilities are estimated by projecting a word's context in the same space as the word representations and by assigning probabilities proportional to the distance between the words and the context's projection. Neural language models are notoriously slow to train and test. Our framework is designed with scalability in mind and provides two optional techniques for reducing the computational cost: the so-called class decomposition trick and a training algorithm based on noise contrastive estimation. Our models may be extended to incorporate direct n-gram features to learn weights for every n-gram in the training data. Our framework comes with wrappers for the `cdec` and `Moses` translation toolkits, allowing our language models to be incorporated as *normalized* features in their decoders (inside the beam search).

1. Introduction

Language models are statistical models used to score how likely a sequence of words is to occur in a certain language. They are central to a number of natural language applications, including machine translation. The goal of a language model in a translation system is to ensure the fluency of the output sentences.

[1] Our code is publicly accessible at: `https://github.com/pauldb89/oxlm`

 Corresponding author: `paul.baltescu@cs.ox.ac.uk`
Cite as: Paul Baltescu, Phil Blunsom, Hieu Hoang. OxLM: A Neural Language Modelling Framework for Machine Translation. The Prague Bulletin of Mathematical Linguistics No. 102, 2014, pp. 81–92.
doi: 10.2478/pralin-2014-0016.

Most machine translation systems today use highly efficient implementations of n-gram language models (Heafield, 2011; Stolcke, 2002). N-gram language models represent the target vocabulary as a discrete set of tokens and estimate the conditional probabilities $P(w_i|w_{i-1},\ldots,w_{i-n})$ via frequency counting. Kneser-Ney smoothing (Chen and Goodman, 1999) is typically used to ameliorate the effect of data sparsity. Querying n-gram language models is extremely fast as the only operations involved are hashtable or trie lookups, depending on the implementation.

Neural language models (Bengio et al., 2003) are a more recent class of language models which use neural networks to learn distributed representations for words. Neural language models project words and contexts into a continuous vector space. The conditional probabilities $P(w_i|w_{i-1},\ldots,w_{i-n})$ are defined to be proportional to the distance between the continuous representation of the word w_i and the context $w_{i-1},\ldots,w_{i-n}$. Neural language models learn to cluster word vectors according to their syntactic and semantic role. The strength of neural language models lies in their ability to generalize to unseen n-grams, because similar words will share the probability of following a context. Neural language models have been shown to outperform n-gram language models using intrinsic evaluation (Chelba et al., 2013; Mikolov et al., 2011a; Schwenk, 2007) or as part of other natural language systems such as speech recognizers (Mikolov et al., 2011a; Schwenk, 2007). In machine translation, it has been shown that neural language models improve translation quality if incorporated as an additional feature into a machine translation decoder (Botha and Blunsom, 2014; Vaswani et al., 2013) or if used for n-best list rescoring (Schwenk, 2010). Querying a neural language model involves an expensive normalization step linear in the size of the vocabulary and scaling this operation requires special attention in order for a translation system to maintain an acceptable decoding speed.

The goal of this paper is to introduce an open source implementation of a feed forward neural language model. As part of our implementation, we release wrappers which enable the integration of our models as *normalized* features in the `cdec` (Dyer et al., 2010) and `Moses` (Koehn et al., 2007) decoders. Our framework is designed with scalability in mind and provides two techniques for speeding up training: class-based factorization (Morin and Bengio, 2005) and noise contrastive estimation (Mnih and Teh, 2012). The class decomposition trick is also helpful for reducing the cost of querying the language model and allows a decoder incorporating our feature to maintain an acceptable decoding speed. In addition to this, our framework optionally extends neural language models by incorporating direct n-gram features (similar to Mikolov et al. (2011a)).

2. Related work

In this section, we briefly analyze three open source neural language modelling toolkits. We discuss how each implementation is different from our own and show where our approach has additional strengths.

CSLM (Schwenk, 2010) is an open source toolkit implementing a continuous space language model which has a similar architecture to our own. CSLM employs a *short-list* to reduce the computational cost of the normalization step. A short-list contains the most frequent words in the training corpus. Schwenk (2010) reports setting the size of the short-list to 8192 or 12288 words. The continuous space language model is used to predict only the words in the short-list, while the remaining words are scored using a back-off n-gram language model. We believe this optimization hurts the model where the potential benefit is the greatest, as the strength of neural language models relies in predicting rare words. In addition to this, CSLM may only be used to rescore n-best lists and cannot be incorporated as a feature in a decoder.

NPLM (Vaswani et al., 2013) is another open source implementation of a neural language model. In contrast to our implementation, Vaswani et al. (2013) do not explicitly normalize the values produced by their model and claim that these scores can be roughly interpreted as probabilities. In practice, we observed that unnormalized scores do not sum up to values close to 1 when the predicted word is marginalized over the vocabulary. Our approach trades decoding speed for the guarantee of using properly scaled feature values.

RNNLM (Mikolov et al., 2011b) is an open source implementation of a recurrent neural language model. Recurrent neural language models have a somewhat different architecture where the hidden layer at step i is provided as input to the network at step $i + 1$. RNNLM uses the class decomposition trick to speed up queries. The toolkit also allows extending the language models with direct n-gram features. RNNLM has been successfully used in speech recognition tasks (Mikolov et al., 2011a), and Auli and Gao (2014) show that recurrent neural language models considerably improve translation quality when integrated as an unnormalized feature into a decoder.

3. Model description

Our implementation follows the basic architecture of a log-bilinear language model (Mnih and Hinton, 2007). We define two vector representations $\mathbf{q}_w, \mathbf{r}_w \in \mathbb{R}^D$ for every word w in the vocabulary V. $\mathbf{q}_w$ represents w's syntactic and semantic role when the word is part of the conditioning context, while $\mathbf{r}_w$ is used to represent w's role as a prediction. For some word w_i in a given corpus, let h_i denote the conditioning context $w_{i-1}, \ldots, w_{i-n}$. To find the conditional probability $P(w_i|h_i)$, our model first computes a context projection vector:

$$\mathbf{p} = \sum_{j=1}^{n-1} C_j \mathbf{q}_{h_{ij}}, \tag{1}$$

where $C_j \in \mathbb{R}^{D \times D}$ are position-specific transformation matrices. Our implementation provides an optional flag which applies a component-wise sigmoid non-linearity to the projection layer, transforming the model into one similar to Bengio et al. (2003).

The model computes a set of similarity scores indicating how well each word $w \in V$ matches the context projection of h_i. The similarity score is defined as:

$$\phi(w, h_i) = \mathbf{r}_w^T \mathbf{p} + b_w, \tag{2}$$

where b_w is a bias term incorporating the prior probability of w. The similarity scores are transformed into a probability distribution using the *softmax* function:

$$P(w_i|h_i) = \frac{\exp(\phi(w_i, h_i))}{\sum_{w \in V} \exp(\phi(w, h_i))} \tag{3}$$

The complete set of parameters is $(C_j, Q, R, \mathbf{b})$, where $Q, R \in \mathbb{R}^{D \times |V|}$ and $\mathbf{b} \in \mathbb{R}^{|V|}$. The model is trained using minibatch stochastic gradient descent to minimize the negative log-likelihood of the training data. L_2 regularization is used to prevent overfitting.

3.1. Class based factorization

The difficulty of scaling neural language models lies in optimizing the normalization step illustrated in Equation 3. Our implementation relies on class based decomposition (Morin and Bengio, 2005; Goodman, 2001) to reduce the cost of normalization. We partition our vocabulary in K classes $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ using Brown clustering (Liang, 2005; Brown et al., 1992) such that $V = \bigcup_{i=1}^{K} \mathcal{C}_i$ and $\mathcal{C}_i \cap \mathcal{C}_j = \varnothing, \forall 1 \leq i < j \leq K$. We define the conditional probability as:

$$P(w_i|h_i) = P(c_i|h_i) P(w_i|c_i, h_i), \tag{4}$$

where c_i is the index of the class w_i is assigned to, i.e. $w_i \in \mathcal{C}_{c_i}$. We associate a vector representation $\mathbf{s}_c$ and a bias term t_c for each class c. The class conditional probability is computed reusing the prediction vector $\mathbf{p}$ by means of a scoring function $\psi(c, h_i) = \mathbf{s}_c^T p + t_c$. Each conditional distribution is now normalized separately:

$$P(c_i|h_i) = \frac{\exp(\psi(c_i, h_i))}{\sum_{j=1}^{K} \exp(\psi(c_j, h_i))} \tag{5}$$

$$P(w_i|c_i, h_i) = \frac{\exp(\phi(w_i, h_i))}{\sum_{w \in C_{c_i}} \exp(\phi(w, h_i))} \tag{6}$$

The best performance is achieved when $K \approx \sqrt{|V|}$ and the word classes have roughly equal sizes. In that case, the normalization cost for predicting a word is reduced from $O(|V|)$ to $O(\sqrt{|V|})$.

3.2. Noise contrastive estimation

Training neural language models using stochastic gradient descent is slow because the entire matrix $R \in \mathbb{R}^{D \times |V|}$ is modified with every gradient update. The class based factorization reduces the cost of computing the gradient of R to $O(D \times \sqrt{|V|})$. In our implementation, we provide an optimization for computing the gradient updates based on noise contrastive estimation, a technique which does not involve normalized probabilities (Mnih and Teh, 2012). Noise contrastive training can be used with or without class based decomposition.

The key idea behind noise contrastive estimation is to reduce a density estimation problem to a classification problem, by training a binary classifier to discriminate between samples from the data distribution and samples from a known noise distribution. In our implementation, we draw the noise samples n_i from the unigram distribution denoted by $P_n(w)$. Following Mnih and Teh (2012), we use k times more noise samples than data samples, where k is specified via an input argument. The posterior probability that a word is generated from the data distribution given its context is:

$$P(C = 1|w_i, h_i) = \frac{P(w_i|h_i)}{P(w_i|h_i) + kP_n(w_i)} \tag{7}$$

Mnih and Teh (2012) show that the gradient of the classification objective:

$$J(\theta) = \sum_{i=1}^{m} \log p(C = 1|\theta, w_i, h_i) + \sum_{i=1}^{km} \log p(C = 0|\theta, n_i, h_i) \tag{8}$$

is an approximation which converges to the maximum likelihood gradient as $k \to \infty$. Noise contrastive estimation allows us to replace the normalization terms with model parameters. Mnih and Teh (2012) showed that setting these parameters to 1 results in no perplexity loss. In our implementation of noise contrastive training, we simply ignore the normalization terms, but this optimization is not applicable at test time.

3.3. Direct n-gram features

Direct features (or connections) for unigrams were originally introduced in neural language models by Bengio et al. (2003). Mikolov et al. (2011a) extend these features to n-grams and show they are useful for reducing perplexity and improving word error rate in speech recognizers. Direct n-gram features are reminiscent of maximum entropy language models (Berger et al., 1996) and are sometimes called maximum entropy features (e.g. in Mikolov et al. (2011a)).

The basic idea behind direct features is to define a set of binary feature functions $\mathbf{f}(w, h)$ and to assign each function a weight from a real valued vector $\mathbf{u}$. In our implementation, we define a feature function $f(w, h)$ for every n-gram in the training

data, up to some order specified by an input argument. To account for word classes, we also define a set of n-gram features $\mathbf{g}_c(w, h)$ and a vector of weights $\mathbf{v}_c$ for each word cluster c. An n-gram (w, h) has a corresponding feature function $g_c(w, h)$ only if $w \in \mathcal{C}_c$. To incorporate the features into our model, we update the scoring functions as follows:

$$\psi(c_i, h_i) = \mathbf{s}_{c_i}^\top \mathbf{p} + t_{c_i} + \mathbf{u}^\top \mathbf{f}(w_i, h_i) \tag{9}$$

$$\phi(w_i, h_i) = \mathbf{r}_{w_i}^\top \mathbf{p} + b_{w_i} + \mathbf{v}_{c_i}^\top \mathbf{g}_{c_i}(w_i, h_i) \tag{10}$$

Otherwise, our model definition remains unchanged. The weight vectors $\mathbf{u}$ and $\mathbf{v}_c$ are learned together with the rest of the parameters using gradient descent. From the perspective of the machine translation system, the language model is extended to learn weights for every n-gram in the training data, weights which bear a similar role to the frequency counts used by traditional n-gram language models.

4. Implementation details

4.1. Training language models

Our language modelling framework is implemented in `C++`. Compiling it will result in a number of binaries. `train_sgd`, `train_factored_sgd` and `train_maxent_sgd` are used for training language models, while the other binaries are useful for evaluation and debugging. Due to lack of space, we will only discuss the most important arguments provided in the training scripts. For a complete list of available options and a short description of each, any binary may be run with the `--help` argument. Examples of intended usage and recommended configurations are released together with our code.

`train_sgd` is used to train neural language models without class factorization or direct features. The binary reads the training data from the file specified via the `--input` parameter. The optional `--test-set` parameter is used to specify the file containing the test corpus. If specified, the training script computes the test set perplexity every 1000 minibatches and at the end of every training epoch. The `--model-out` argument specifies the path where the language model is saved. The language model is written to disk every time the test set perplexity reaches a new minimum. The `--order` parameter specifies the order of the model, the `--word-width` parameter specifies the size of the distributed representations and the `--lambda-lbl` parameter is the inverse of the variance for the L_2 regularizer. If `--noise-samples` is set to 0, the model is trained using stochastic gradient descent. Otherwise, the parameter specifies the number of noise samples drawn from the unigram distribution for each training instance during noise contrastive training.

Factored models are trained with the `train_factored_sgd` binary. In addition to the previous arguments, this script includes the `--class-file` option which points

to the files containing the Brown clusters. The expected format matches the output format of Liang (2005)'s agglomerative clustering tool[2]. If the `--class-file` argument is not specified, the user is required to set the `--classes` argument. In this case, the word clusters are obtained using frequency binning.

Factored models incorporating direct features are trained with `train_maxent_sgd`. We implement two types of feature stores for storing the weights of the n-gram features. Sparse feature stores use identity mapping to map every n-gram with its corresponding weight. Collision stores hash the n-grams to a lower dimensional space instead, leading to potential collisions. If configured correctly using the `--hash-space` parameter, collision stores require less memory than sparse feature stores, without any perplexity loss. If the argument is set to 0, sparse stores are used instead. The `--min-ngram-freq` argument may be used to ignore n-grams below a frequency threshold, while `--max-ngrams` may be used to restrict the number of direct features to the most frequent n-grams in the training data.

4.2. Feature wrappers for `cdec` and `Moses`

Our language models may be incorporated into the `cdec` and `Moses` decoders as normalized features to score partial translation hypotheses during beam search. The decoders often keep the conditioning context unchanged and create new translation hypotheses by adding new words at the end of the conditioning context. We significantly speed up decoding by caching the normalization terms to avoid recomputing them every time a new word is added after the same context. The normalization cache is reset every time the decoders receive a new sentence as input.

Compiling our framework results in the `libcdec_ff_lbl.so` shared library which is used to dynamically load our language models as a feature in the `cdec` decoder. To load the feature, a single line must be added to the decoder configuration file specifying the path to the shared library, the file containing the language model and the type of the language model (standard, factored or factored with direct features). A complete `cdec` integration example is provided in the documentation released with our code.

The feature wrapper for `Moses` is included in the `Moses` repository[3]. To include our language models in the decoder, `Moses` must be compiled with the `--with-lbllm` argument pointing to the location of the `oxlm` repository. The decoder configuration file must be updated to include the feature definition, the path to the file containing the language model and the initial feature weight. A complete example on how to integrate our language models in `Moses` is provided in the documentation released with our code.

---

[2]The tool is publicly available at: `https://github.com/percyliang/brown-cluster`

[3]The feature wrapper is accessible here: `https://github.com/moses-smt/mosesdecoder/tree/master/moses/LM/oxlm`

4.3. Optimizations

Our framework includes several smaller optimizations designed to speed up training and testing. We provide an optional `--diagonal-contexts` argument which informs the framework to learn a model with diagonal context matrices. This optimization significantly speeds up querying the language model and helps the training algorithm converge after fewer iterations without any loss in perplexity.

Our implementation leverages the benefits of a multithreaded environment to distribute the gradient calculation during training. The training instances in a minibatch are shared evenly across the number of available threads (specified by the user via the `--threads` parameter). In our gradient descent implementation, we use adaptive learning (Duchi et al., 2011) to converge to a better set of parameters.

We rely on `Eigen`, a high-level `C++` library for linear algebra, to speed up the matrix and vector operations involved in training and querying our models. Where possible, we group together multiple similar operations to further speed up the computations.

Finally, we speed up the process of tuning the translation system feature weights by taking advantage of the fact that the development corpus is decoded for several iterations with different weights. As a result, the n-grams scored by the language model often repeat themselves over a number of iterations. We maintain sentence-specific caches mapping n-grams to language model probabilities which are persistent between consecutive iterations of the tuning algorithm. A persistent cache is loaded from disk when a sentence is received as input and saved back to disk when the system has finished decoding the sentence. This optimization massively speeds up the process of tuning the translation system and can be enabled via the `--persistent-cache` flag in the decoder configuration file.

5. Experiments

In this section, we provide experimental results to illustrate the strengths of our language modelling framework. We report perplexities and improvements in the BLEU score when the language model is used as an *additional* feature in the decoder. We also report the training times for stochastic gradient descent and noise contrastive estimation. Finally, we compare the average time needed to decode a sentence with our language modelling feature against a standard system using only an efficient implementation of a backoff n-gram model.

In our experiments, we used the `europarl-v7` and the `news-commentary-v9` French-English data to train a hierarchical phrase-based translation system (`cdec`). The corpus was tokenized, lowercased and filtered to exclude sentences longer than 80 words or having substantially different lengths using the preprocessing scripts available in `cdec`[4]. After preprocessing, the training corpus consisted of 2,008,627 pairs of sen-

[4]We followed the indications provided here: `http://www.cdec-decoder.org/guide/tutorial.html`

Model	Training algorithm	Training time (hours)	Perplexity	BLEU
KenLM	-	0.1	267.814	24.75
FactoredLM	SGD	34.1	226.44	25.2
FactoredLM	NCE 1	9.4	258.623	25.25
FactoredLM	NCE 10	12.5	245.748	25.06
FactoredLM	NCE 50	18.1	241.481	25.23
DirectFactoredLM	SGD	42.0	210.275	25.46

Table 1. Training time, perplexities and BLEU scores for various models.

tences. The corpus was aligned using `fast_align` (Dyer et al., 2013) and the alignments were symmetrized using the `grow-diag-final-and` heuristic. We split the `newstest2012`[5] data evenly into a development set and a test set, by assigning sentences to each dataset alternatively. The translation system is tuned on the development set using MIRA. We report average BLEU scores over 3 MIRA runs.

The baseline system includes an efficient implementation (Heafield, 2011) of a 5-gram language model (`KenLM`). The language models are trained on the target side of the parallel corpus, on a total of 55,061,862 tokens. Before training the neural language models, singletons are replaced with a special `<unk>` token. The neural language model vocabulary consists of 57,782 words and is factored into 240 classes using Brown clustering. In our experiments, we set the order of the neural language models to 5, the dimensionality of the word representations to 200 and make use of diagonal contexts. The standard factored language model is labelled with `FactoredLM`. `DirectFactoredLM` is an extension incorporating direct n-gram features. In our experiments, we define a feature function for every n-gram ($n \leq 5$) observed at least 3 times in the training corpus. The feature weights are hashed into a collision store with a capacity of 5 million features.

Table 1 summarizes the results of our experiments. We indicate the algorithm used to train each neural language model. Stochastic gradient descent is denoted by SGD, while noise contrastive estimation is denoted by NCE and followed by the number of noise samples used for estimating the gradient for each data point. In both cases, the gradient optimization was distributed over 8 threads and the minibatch size was set to 10,000 data points. We note that noise contrastive estimation leads to models with higher perplexities. However, that has no effect on the overall quality of the translation system, while massively reducing the training time of the neural language models. Overall, we observe a BLEU score improvement of 0.7 when a factored language model with direct n-gram features is used in addition to a standard 5-gram language model.

[5]The corpus is available here: `http://www.statmt.org/wmt14/translation-task.html`

Model	Decoding time (seconds)
`KenLM`	0.447
`FactoredLM`	3.356
`DirectFactoredLM`	6.633

Table 2. Average decoding speed.

Table 2 shows the average decoding speed with our neural language modelling features. The average decoding time is reported on the first 100 sentences of the development set. Overall, the neural language models slow down the decoder by roughly an order of magnitude.

In conclusion, this paper presents an open source implementation of a neural language modelling toolkit. The toolkit provides techniques for speeding up training and querying language models and incorporates direct n-gram features for better translation quality. The toolkit facilitates the integration of the neural language models as a feature in the beam search of the `cdec` and `Moses` decoders. Although our language modelling features slow down the decoders somewhat, they guarantee that the probabilities used to score partial translation hypotheses are properly normalized.

Bibliography

Auli, Michael and Jianfeng Gao. Decoder integration and expected bleu training for recurrent neural network language models. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL '14)*, pages 136–142, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

Berger, Adam L., Vincent J. Della Pietra, and Stephen A. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

Botha, Jan A. and Phil Blunsom. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on Machine Learning (ICML '14)*, Beijing, China, 2014.

Brown, Peter F., Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

Chelba, Ciprian, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, 2013.

Chen, Stanley F. and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.

Duchi, John, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Dyer, Chris, Adam Lopez, Juri Ganitkevitch, Jonathan Weese, Ferhan Ture, Phil Blunsom, Hendra Setiawan, Vladimir Eidelman, and Philip Resnik. cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the ACL 2010 System Demonstrations*, pages 7–12, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

Dyer, Chris, Victor Chahuneau, and Noah A. Smith. A simple, fast, and effective reparameterization of ibm model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL '13)*, pages 644–648, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

Goodman, Joshua. Classes for fast maximum entropy training. *CoRR*, 2001.

Heafield, Kenneth. Kenlm: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation (WMT '11)*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

Koehn, Philipp, 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 *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL '07)*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Liang, P. Semi-supervised learning for natural language. Master's thesis, Massachusetts Institute of Technology, 2005.

Mikolov, Tomas, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocky. Strategies for training large scale neural network language models. In *Proceedings of the 2011 Automatic Speech Recognition and Understanding Workshop*, pages 196–201. IEEE Signal Processing Society, 2011a.

Mikolov, Tomas, Stefan Kombrink, Anoop Deoras, Lukas Burget, and Jan Cernocky. Rnnlm - recurrent neural network language modeling toolkit. In *Proceedings of the 2011 Automatic Speech Recognition and Understanding Workshop*, pages 1–4. IEEE Signal Processing Society, 2011b.

Mnih, Andriy and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*, pages 641–648, Corvallis, OR, USA, 2007.

Mnih, Andriy and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning (ICML '12)*, pages 1751–1758, Edinburgh, Scotland, 2012.

Morin, Frederic and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS '05)*, pages 246–252. Society for Artificial Intelligence and Statistics, 2005.

Schwenk, Holger. Continuous space language models. *Computer Speech & Language*, 21(3):492–518, 2007.

Schwenk, Holger. Continuous-space language models for statistical machine translation. *Prague Bulletin of Mathematical Linguistics*, 93:137–146, 2010.

Stolcke, Andreas. Srilm - an extensible language modeling toolkit. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 901–904, 2002.

Vaswani, Ashish, Yinggong Zhao, Victoria Fossum, and David Chiang. Decoding with large-scale neural language models improves translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1387–1392, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

Address for correspondence:
Paul Baltescu
paul.baltescu@cs.ox.ac.uk
Department of Computer Science
University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD,
United Kingdom