Improving Neural Language Models with a Continuous Cache

Edouard Grave, Armand Joulin, Nicolas Usunier
Facebook AI Research
`{egrave,ajoulin,usunier}@fb.com`

Abstract

We propose an extension to neural network language models to adapt their prediction to the recent history. Our model is a simplified version of memory augmented networks, which stores past hidden activations as memory and accesses them through a dot product with the current hidden activation. This mechanism is very efficient and scales to very large memory sizes. We also draw a link between the use of external memory in neural network and cache models used with count based language models. We demonstrate on several language model datasets that our approach performs significantly better than recent memory augmented networks.

1 Introduction

Language modeling is a core problem in natural language processing, with many applications such as machine translation (Brown et al., 1993), speech recognition (Bahl et al., 1983) or dialogue agents (Stolcke et al., 2000). While traditional neural networks language models have obtained state-of-the-art performance in this domain (Jozefowicz et al., 2016; Mikolov et al., 2010), they lack the capacity to adapt to their recent history, limiting their application to dynamic environments (Dodge et al., 2015). A recent approach to solve this problem is to augment these networks with an external memory (Graves et al., 2014; Grefenstette et al., 2015; Joulin & Mikolov, 2015; Sukhbaatar et al., 2015). These models can potentially use their external memory to store new information and adapt to a changing environment.

While these networks have obtained promising results on language modeling datasets (Sukhbaatar et al., 2015), they are quite computationally expensive. Typically, they have to learn a parametrizable mechanism to read or write to memory cells (Graves et al., 2014; Joulin & Mikolov, 2015). This may limit both the size of their usable memory as well as the quantity of data they can be trained on. In this work, we propose a very light-weight alternative that shares some of the properties of memory augmented networks, notably the capability to dynamically adapt over time. By minimizing the computation burden of the memory, we are able to use larger memory and scale to bigger datasets. We observe in practice that this allows us to surpass the perfomance of memory augmented networks on different language modeling tasks.

Our model share some similarities with a model proposed by Kuhn (1988), called the *cache model*. A cache model stores a simple representation of the recent past, often in the form of unigrams, and uses them for prediction (Kuhn & De Mori, 1990). This contextual information is quite cheap to store and can be accessed efficiently. It also does not need any training and can be appplied on top of any model. This makes this model particularly interesting for domain adaptation (Kneser & Steinbiss, 1993).

Our main contribution is to propose a continuous version of the cache model, called *Neural Cache Model*, that can be adapted to any neural network language model. We store recent hidden activations and use them as representation for the context. Using simply a dot-product with the current hidden activations, they turn out to be extremely informative for prediction. Our model requires *no training* and can be used on any pre-trained neural networks. It also scales effortlessly to thousands of memory cells. We demonstrate the quality of the Neural Cache models on several language model tasks and the LAMBADA dataset (Paperno et al., 2016).

2 LANGUAGE MODELING

A language model is a probability distribution over sequences of words. Let V be the size of the vocabulary; each word is represented by a one-hot encoding vector x in $\mathbb{R}^V = \mathcal{V}$, corresponding to its index in the vocabulary. Using the chain rule, the probability assigned to a sequence of words $x_1, \dots, x_T$ can be factorized as

$$p(x_1, ..., x_T) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, ..., x_1).$$

Language modeling is often framed as learning the conditional probability over words, given the history (Bahl et al., 1983).

This conditional probability is traditionally approximated with non-parameteric models based on counting statistics (Goodman, 2001). In particular, smoothed N-gram models (Katz, 1987; Kneser & Ney, 1995) achieve good performance in practice (Mikolov et al., 2011). Parametrized alternatives are either maximum entropy language models (Rosenfeld, 1996), feedforward networks (Bengio et al., 2003) or recurrent networks (Mikolov et al., 2010). In particular, recurrent networks are currently the best solution to approximate this conditional probability, achieving state-of-the-arts performance on standard language modeling benchmarks (Jozefowicz et al., 2016; Zilly et al., 2016).

Recurrent networks. Assuming that we have a vector $h_t \in \mathbb{R}^d$ encoding the history $x_t, ..., x_1$, the conditional probability of a word w can be parametrized as

$$p_{vocab}(w \mid x_t, ..., x_1) \propto \exp(h_t^\top o_w).$$

The history vector h_t is computed by a recurrent network by recursively applying an equation of the form

$$h_t = \Phi\left(x_t, h_{t-1}\right),$$

where Φ is a function depending on the architecture of the network. Several architecture for recurrent networks have been proposed, such as the Elman network (Elman, 1990), the long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) or the gated recurrent unit (GRU) (Chung et al., 2014). One of the simplest recurrent networks is the Elman network (Elman, 1990), where

$$h_t = \sigma\left(Lx_t + Rh_{t-1}\right),$$

where σ is a non-linearity such as the logistic or tanh functions, $L \in \mathbb{R}^{d \times V}$ is a word embedding matrix and $R \in \mathbb{R}^{d \times d}$ is the recurrent matrix. The LSTM architecture is particularly interesting in the context of language modelling (Jozefowicz et al., 2016) and we refer the reader to Graves et al. (2013) for details on this architecture.

The parameters of recurrent neural network language models are learned by minimizing the negative log-likelihood of the training data. This objective function is usually minimized by using the stochastic gradient descent algorithm, or variants such as Adagrad (Duchi et al., 2011). The gradient is computed using the truncated backpropagation through time algorithm (Werbos, 1990; Williams & Peng, 1990).

Cache model. After a word appears once in a document, it is much more likely to appear again. As an example, the frequency of the word *tiger* on the Wikipedia page of the same name is 2.8%, compared to 0.0037% over the whole Wikipedia. Cache models exploit this simple observation to improve n-gram language models by capturing long-range dependencies in documents. More precisely, these models have a cache component, which contains the words that appeared in the recent history (either the document or a fixed number of words). A simple language model, such as a unigram or smoothed bigram model, is fitted on the words of the cache and interpolated with the static language model (trained over a larger dataset). This technique has many advantages. First, this is a very efficient way to adapt a language model to a new domain. Second, such models can predict out-of-vocabulary words (OOV words), after seeing them once. Finally, this helps capture long-range dependencies in documents, in order to generate more coherent text.

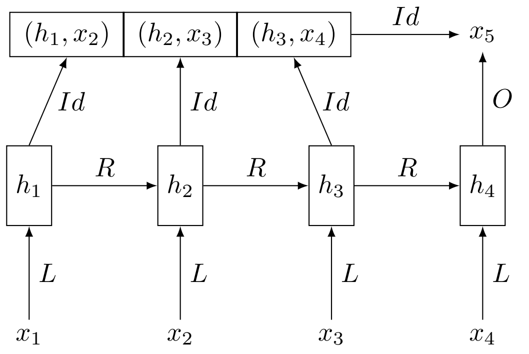

Figure 1: The neural cache stores the previous hidden states in memory cells. They are then used as keys to retrieve their corresponding word, that is the next word. There is no transformation applied to the storage during writing and reading.

3 Neural Cache Model

The Neural Cache Model adds a cache-like memory to neural network language models. It exploits the hidden representations h_t to define a probability distribution over the words in the cache. As illustrated Figure 1, the cache stores pairs (h_i, x_{i+1}) of a hidden representation, and the word which was generated based on this representation (we remind the reader that the vector h_i encodes the history $x_i, ..., x_1$). At time t, we then define a probability distribution over words stored in the cache based on the stored hidden representations and the current one h_t as

$$p_{cache}(w \mid h_{1..t},\ x_{1..t}) \propto \sum_{i=1}^{t-1} \mathbb{1}_{\{w=x_{i+1}\}} \exp(\theta h_t^\top h_i)$$

where the scalar θ is a parameter which controls the flatness of the distribution. When θ is equal to zero, the probability distribution over the history is uniform, and our model is equivalent to a unigram cache model (Kuhn & De Mori, 1990).

From the point of view of memory-augmented neural networks, the probability $p_{cache}(w \mid h_{1..t},\ x_{1..t})$ given by the neural cache model can be interpreted as the probability to retrieve the word w from the memory given the query h_t, where the desired answer is the next word x_{t+1}. Using previous hidden states as keys for the words in the memory, the memory lookup operator can be implemented with simple dot products between the keys and the query. In contrast to existing memory-augmented neural networks, the neural cache model avoids the need to learn the memory lookup operator. Such a cache can thus be added to a pre-trained recurrent neural language model without fine tuning of the parameters, and large cache size can be used with negligible impact on the computational cost of a prediction.

Neural cache language model. Following the standard practice in n-gram cache-based language models, the final probability of a word is given by the linear interpolation of the cache language model with the regular language model, obtaining:

$$p(w \mid h_{1..t},\ x_{1..t}) = (1-\lambda) p_{vocab}(w \mid h_t) + \lambda p_{cache}(w \mid h_{1..t}, x_{1..t})\,.$$

Instead of taking a linear interpolation between the two distribution with a fixed λ, we also consider a global normalization over the two distribution:

$$p(w \mid h_{1..t},\ x_{1..t}) \propto \left(\exp(h_t^\top o_w) + \sum_{i=1}^{t-1} \mathbb{1}_{\{w=x_{i+1}\}} \exp(\theta h_t^\top h_i + \alpha) \right)\,.$$

This corresponds to taking a softmax over the vocabulary and the words in the cache. The parameter α controls the weight of the cache component, and is the counterpart of the λ parameter for linear interpolation.

The addition of the neural cache to a recurrent neural language model inherits the advantages of n-gram caches in usual cache-based models: The probability distribution over words is updated online depending on the context, and out-of-vocabulary words can be predicted as soon as they have been seen at least once in the recent history. The neural cache also inherits the ability of the hidden states of recurrent neural networks to model longer-term contexts than small n-grams, and thus allows for a finer modeling of the current context than e.g., unigram caches.

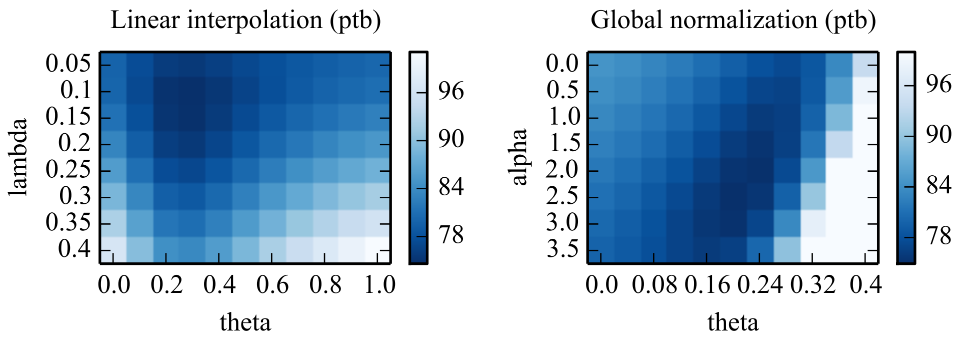

Figure 2: Perplexity on the validation set of `Penn Tree Bank` for linear interpolation (left) and global normalization (right), for various values of hyperparameters θ, λ and α. We use a cache model of size 500. The base model has a validation perplexity of 86.9. The best linear interpolation has a perplexity of 74.6, while the best global normalization has a perplexity of 74.9.

Model	Test PPL
RNN+LSA+KN5+cache (Mikolov & Zweig, 2012)	90.3
LSTM (Zaremba et al., 2014)	78.4
Variational LSTM (Gal & Ghahramani, 2015)	73.4
Recurrent Highway Network (Zilly et al., 2016)	66.0
Pointer Sentinel LSTM (Merity et al., 2016)	70.9
LSTM (our implem.)	82.3
Neural cache model	72.1

Table 1: Test perplexity on the `Penn Tree Bank`.

Training procedure. For now, we first train the (recurrent) neural network language model, without the cache component. We only apply the cache model at test time, and choose the hyperparameters θ and λ (or α) on the validation set. A big advantage of our method is that it is very easy and cheap to apply, with already trained neural models. There is no need to perform backpropagation over large contexts, and we can thus apply our method with large cache sizes (larger than one thousand).

4 Related work

Cache model. Adding a cache to a language model was intoducted in the context of speech recognition(Kuhn, 1988; Kupiec, 1989; Kuhn & De Mori, 1990). These models were further extended by Jelinek et al. (1991) into a smoothed trigram language model, reporting reduction in both perplexity and word error rates. Della Pietra et al. (1992) adapt the cache to a general n-gram model such that it satisfies marginal constraints obtained from the current document.

Adaptive language models. Other adaptive language models have been proposed in the past: Kneser & Steinbiss (1993) and Iyer & Ostendorf (1999) dynamically adapt the parameters of their model to the recent history using different weight interpolation schemes. Bellegarda (2000) and Coccaro & Jurafsky (1998) use latent semantic analysis to adapt their models to the current context. Similarly, topic features have been used with either maximum entropy models (Khudanpur & Wu, 2000) or recurrent networks (Mikolov & Zweig, 2012; Wang & Cho, 2015). Finally, Lau et al. (1993) proposes to use pairs of distant of words to capture long-range dependencies.

Memory augmented neural networks. In the context of sequence prediction, several memory augmented neural networks have obtained promising results (Sukhbaatar et al., 2015; Graves et al., 2014; Grefenstette et al., 2015; Joulin & Mikolov, 2015). In particular, Sukhbaatar et al. (2015) stores a representation of the recent past and accesses it using an attention mechanism Bahdanau et al. (2014). Sukhbaatar et al. (2015) shows that this reduces the perplexity for language modeling.

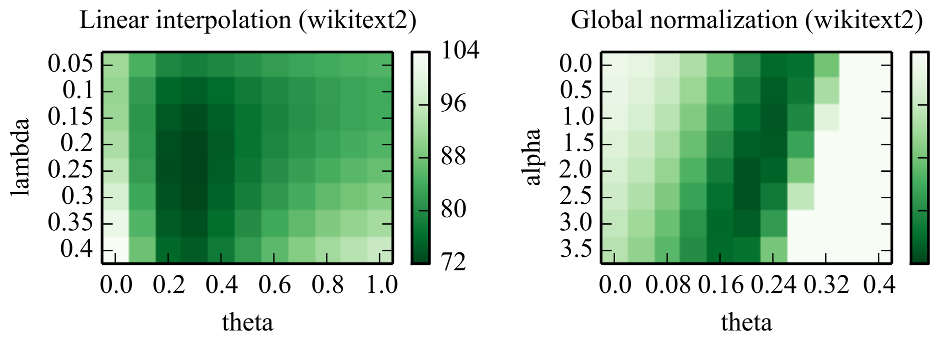

Figure 3: Perplexity on the validation set of `wikitext2` for linear interpolation (left) and global normalization (right), for various values of hyperparameters θ, λ and α. We use a cache model of size 2000. The base model has a validation perplexity of 104.2. The best linear interpolation has a perplexity of 72.1, while the best global normalization has a perplexity of 73.5.

Model	`wikitext2`	`wikitext103`
Zoneout + Variational LSTM (Merity et al., 2016)	100.9	-
Pointer Sentinel LSTM (Merity et al., 2016)	80.8	-
LSTM (our implementation)	99.3	48.7
Neural cache model (size = 100)	81.6	44.8
Neural cache model (size = 2,000)	68.9	40.8

Table 2: Test perplexity on the `wikitext` datasets. The two datasets share the same validation and test sets, making all the results comparable.

This approach has been successfully applied to question answering, when the answer is contained in a given paragraph (Chen et al., 2016; Hermann et al., 2015; Kadlec et al., 2016; Sukhbaatar et al., 2015). Similarly, Vinyals et al. (2015) explores the use of this mechanism to reorder sequences of tokens. Their network uses an attention (or “pointer”) over the input sequence to predict which element should be selected as the next output. Gulcehre et al. (2016) have shown that a similar mechanism called *pointer softmax* could be used in the context of machine translation, to decide which word to copy from the source to target.

Independently of our work, Merity et al. (2016) apply the same mechanism to recurrent network. Unlike our work, they uses the current hidden activation as a representation of the current input (while we use it to represent the output). This requires additional learning of a transformation between the current representation and those in the past. The advantage of our approach is that we can scale to very large caches effortlessly.

5 EXPERIMENTS

In this section, we evaluate our method on various language modeling datasets, which have different sizes and characteristics. On all datasets, we train a static recurrent neural network language model with LSTM units. We then use the hidden representations from this model to obtain our cache, which is interpolated with the static LSTM model. We also evaluate a unigram cache model interpolated with the static model as another baseline.

5.1 SMALL SCALE EXPERIMENTS

Datasets. In this section, we describe experiments performed on two small datasets: the `Penn Tree Bank` (Marcus et al., 1993) and the `wikitext2` (Merity et al., 2016) datasets. The `Penn Tree Bank` dataset is made of articles from the Wall Street Journal, contains 929k training tokens and has a vocabulary size of 10k. The `wikitext2` dataset is derived from Wikipedia articles, contains 2M training tokens and has a vocabulary size of 33k. These datasets contain non-shuffled documents, therefore requiring models to capture inter-sentences dependencies to perform well.

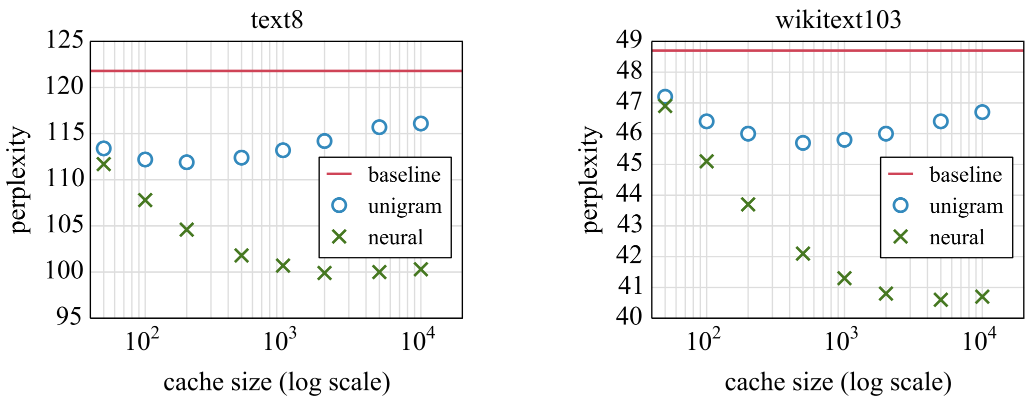

Figure 4: Test perplexity as a function of the number of words in the cache, for our method and a unigram cache baseline. We observe that our approach can uses larger caches than the baseline.

Implementation details. We train recurrent neural network language models with 1024 LSTM units, regularized with dropout (probability of dropping out units equals to 0.65). We use the Adagrad algorithm, with a learning rate of 0.2, a batchsize of 20 and initial weight uniformly sampled in the range $[-0.05, 0.05]$. We clip the norm of the gradient to 0.1 and unroll the network for 30 steps. We consider cache sizes on a logarithmic scale, from 50 to $10,000$, and fit the cache hyperparameters on the validation set.

Results. We report the perplexity on the validation sets in Figures 2 and 3, for various values of hyperparameters, for linear interpolation and global normalization. First, we observe that on both datasets, the linear interpolation method performs slightly better than the global normalization approach. It is also easier to apply in practice, and we thus use this method in the remainder of this paper. In Tables 1 and 2, we report the test perplexity of our approach and state-of-the-art models. Our approach is competitive with previous models, in particular with the pointer sentinel LSTM model of Merity et al. (2016). On `Penn Tree Bank`, we note that the improvement over the base model is similar for both methods. On the `wikitext2` dataset, both methods obtain similar results when using the same cache size (100 words). Since our method is computationally cheap, it is easy to increase the cache to larger values ($2,000$ words), leading to dramatic improvements (30% over the baseline, 12% over a small cache of 100 words).

5.2 Medium scale experiments

Datasets and implementation details. In this section, we describe experiments performed over two medium scale datasets: `text8` and `wikitext103`. Both datasets are derived from Wikipedia, but different pre-processing were applied. The `text8` dataset contains 17M training tokens and has a vocabulary size of 44k words, while the `wikitext103` dataset has a training set of size 103M, and a vocabulary size of 267k words. We use the same setting as in the previous section, except for the batchsize (we use 128) and dropout parameters (we use 0.45 for `text8` and 0.25 for `wikitext103`). Since both datasets have large vocabularies, we use the adaptive softmax (Grave et al., 2016) for faster training.

Results. We report the test perplexity as a function of the cache size in Figure 4, for the neural cache model and a unigram cache baseline. We observe that our approach can exploits larger cache sizes, compared to the baseline. In Table 2, we observe that the improvement in perplexity of our method over the LSTM baseline on `wikitext103` is smaller than for `wikitext2` (approx. 16% v.s. 30%). The fact that improvements obtained with more advanced techniques decrease when the size of training data increases has already been observed by Goodman (2001). Both `wikitext` datasets sharing the same test set, we also observe that the LSTM baseline, trained on 103M tokens (`wikitext103`), strongly outperforms more sophisticated methods, trained on 2M tokens (`wikitext2`). For these two reasons, we believe that it is important to evaluate and compare methods on relatively large datasets.

Model	Test
LSTM-500 (Mikolov et al., 2014)	156
SCRNN (Mikolov et al., 2014)	161
MemNN (Sukhbaatar et al., 2015)	147
LSTM-1024 (our implem.)	121.8
Neural cache model	99.9

(a) `text8`

Model	Dev	Ctrl
WB5 (Paperno et al., 2016)	3125	285
WB5+cache (Paperno et al., 2016)	768	270
LSTM-512 (Paperno et al., 2016)	5357	149
LSTM-1024 (our implem.)	4088	94
Neural cache model	138	129

(b) `lambada`

Table 3: Perplexity on the `text8` and `lambada` datasets. WB5 stands for 5-gram language model with Witten-Bell smoothing.

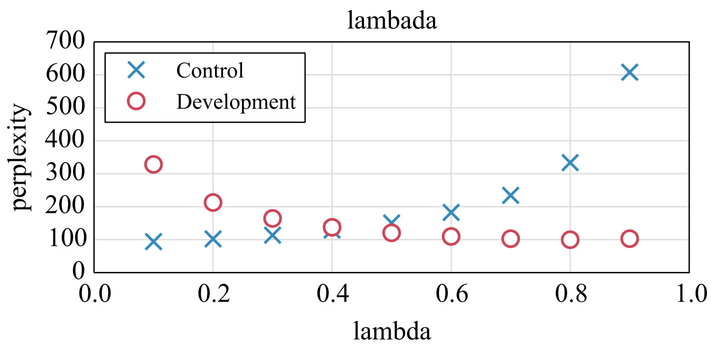

Figure 5: Perplexity on the development and control sets of `lambada`, as a function of the interpolation parameters λ.

5.3 Experiments on the lambada dataset

Finally, we report experiments carried on the `lambada` dataset, introduced by Paperno et al. (2016). This is a dataset of short passages extracted from novels. The goal is to predict the last word of the excerpt. This dataset was built so that human subjects solve the task perfectly when given the full context (approx. 4.6 sentences), but fail to do so when only given the sentence with the target word. Thus, most state-of-the-art language models fail on this dataset. The `lambada` training set contains approximately 200M tokens and has a vocabulary size of $93,215$. We report results for our method in Table 3, as well the performance of baselines from Paperno et al. (2016). Adding a neural cache model to the LSTM baseline strongly improves the performance on the `lambada` dataset. We also observe in Figure 5 that the best interpolation parameter between the static model and the cache is not the same for the development and control sets. This is due to the fact that more than 83% of passages of the development set include the target word, while this is true for only 14% of the control set. Ideally, a model should have strong results on both sets. One possible generalization of our model would be to adapt the interpolation parameter based on the current vector representation of the history h_t.

6 Conclusion

We presented the neural cache model to augment neural language models with a longer-term memory that dynamically updates the word probablilities based on the long-term context. A neural cache can be added on top of a pre-trained language model at negligible cost. Our experiments on both language modeling tasks and the challenging LAMBADA dataset shows that significant performance gains can be expected by adding this external memory component.

Technically, the neural cache models is similar to some recent memory-augmented neural networks such as pointer networks. However, its specific design makes it possible to avoid learning the memory lookup component. This makes the neural cache appealing since it can use larger cache sizes than memory-augment networks and can be applied as easily as traditional count-based caches.

REFERENCES

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. A maximum likelihood approach to continuous speech recognition. *PAMI*, 1983.

Jerome R Bellegarda. Exploiting latent semantic information in statistical language modeling. *Proceedings of the IEEE*, 2000.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *JMLR*, 2003.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 1993.

Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*, 2016.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Noah Coccaro and Daniel Jurafsky. Towards better integration of semantic predictors in statistical language modeling. In *ICSLP*. Citeseer, 1998.

Stephen Della Pietra, Vincent Della Pietra, Robert L Mercer, and Salim Roukos. Adaptive language modeling using minimum discriminant estimation. In *Proceedings of the workshop on Speech and Natural Language*, 1992.

Jesse Dodge, Andreea Gane, Xiang Zhang, Antoine Bordes, Sumit Chopra, Alexander Miller, Arthur Szlam, and Jason Weston. Evaluating prerequisite qualities for learning end-to-end dialog systems. *arXiv preprint arXiv:1511.06931*, 2015.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287*, 2015.

Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 2001.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for gpus. *arXiv preprint arXiv:1609.04309*, 2016.

A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pp. 1828–1836, 2015.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*, 2016.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Rukmini M Iyer and Mari Ostendorf. Modeling long distance dependence in language: Topic mixtures versus dynamic cache models. *IEEE Transactions on speech and audio processing*, 1999.

Frederick Jelinek, Bernard Merialdo, Salim Roukos, and Martin Strauss. A dynamic language model for speech recognition. In *HLT*, 1991.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in Neural Information Processing Systems*, pp. 190–198, 2015.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. *arXiv preprint arXiv:1603.01547*, 2016.

Slava M Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *ICASSP*, 1987.

Sanjeev Khudanpur and Jun Wu. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer Speech & Language*, 2000.

Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In *ICASSP*, 1995.

Reinhard Kneser and Volker Steinbiss. On the dynamic adaptation of stochastic language models. In *ICASSP*, 1993.

Roland Kuhn. Speech recognition and the frequency of recently used words: A modified markov model for natural language. In *Proceedings of the 12th conference on Computational linguistics-Volume 1*, 1988.

Roland Kuhn and Renato De Mori. A cache-based natural language model for speech recognition. *PAMI*, 1990.

Julien Kupiec. Probabilistic models of short and long distance word dependencies in running text. In *Proceedings of the workshop on Speech and Natural Language*, 1989.

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. Trigger-based language models: A maximum entropy approach. In *ICASSP*, 1993.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 1993.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *SLT*, 2012.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukas Burget, and Jan Cernockỳ. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, 2011.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Ronald Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 1996.

Andreas Stolcke, Noah Coccaro, Rebecca Bates, Paul Taylor, Carol Van Ess-Dykema, Klaus Ries, Elizabeth Shriberg, Daniel Jurafsky, Rachel Martin, and Marie Meteer. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational linguistics*, 2000.

Sainbayar Sukhbaatar, Szlam Arthur, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, 2015.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, 2015.

Tian Wang and Kyunghyun Cho. Larger-context language modelling. *arXiv preprint arXiv:1511.03729*, 2015.

Paul J Werbos. Backpropagation through time: what it does and how to do it. 1990.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 1990.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.