Dependency Recurrent Neural Language Models for Sentence Completion

Piotr Mirowski
Google DeepMind
piotr.mirowski@computer.org

Andreas Vlachos
University College London
a.vlachos@cs.ucl.ac.uk

Abstract

Recent work on language modelling has shifted focus from count-based models to neural models. In these works, the words in each sentence are always considered in a left-to-right order. In this paper we show how we can improve the performance of the recurrent neural network (RNN) language model by incorporating the syntactic dependencies of a sentence, which have the effect of bringing relevant contexts closer to the word being predicted. We evaluate our approach on the Microsoft Research Sentence Completion Challenge and show that the dependency RNN proposed improves over the RNN by about 10 points in accuracy. Furthermore, we achieve results comparable with the state-of-the-art models on this task.

1 Introduction

Language Models (LM) are commonly used to score a sequence of tokens according to its probability of occurring in natural language. They are an essential building block in a variety of applications such as machine translation, speech recognition and grammatical error correction. The standard way of evaluating a language model has been to calculate its perplexity on a large corpus. However, this evaluation assumes the output of the language model to be probabilistic and it has been observed that perplexity does not always correlate with the downstream task performance.

For these reasons, Zweig and Burges (2012) proposed the Sentence Completion Challenge, in which the task is to pick the correct word to complete a sentence out of five candidates. Performance is evaluated by accuracy (how many sentences were completed correctly), thus both probabilistic and non-probabilistic models (e.g. Roark et al. (2007)) can be compared. Recent approaches for this task include both neural and count-based language models (Zweig et al., 2012; Gubbins and Vlachos, 2013; Mnih and Kavukcuoglu, 2013; Mikolov et al., 2013).

Most neural language models consider the tokens in a sentence in the order they appear, and the hidden state representation of the network is typically reset at the beginning of each sentence. In this work we propose a novel neural language model that learns a recurrent neural network (RNN) (Mikolov et al., 2010) on top of the syntactic dependency parse of a sentence. Syntactic dependencies bring relevant contexts closer to the word being predicted, thus enhancing performance as shown by Gubbins and Vlachos (2013) for count-based language models. Our Dependency RNN model is published simultaneously with another model, introduced in Tai et al. (2015), who extend the Long-Short Term Memory (LSTM) architecture to tree-structured network topologies and evaluate it at sentence-level sentiment classification and semantic relatedness tasks, but not as a language model.

Adapting the RNN to use the syntactic dependency structure required to reset and run the network on all the paths in the dependency parse tree of a given sentence, while maintaining a count of how often each token appears in those paths. Furthermore, we explain how we can incorporate the dependency labels as features.

Our results show that the dependency RNN language model proposed outperforms the RNN proposed by Mikolov et al. (2011) by about 10 points in accuracy. Furthermore, it improves upon the count-based dependency language model of Gubbins and Vlachos (2013), while achieving slightly worse than the recent state-of-the-art results by Mnih and Kavukcuoglu (2013). Finally, we make the code and preprocessed data available to facilitate comparisons with future work.

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Short Papers), pages 511–517,
Beijing, China, July 26-31, 2015.

2 Dependency Recurrent Neural Network

Count-based language models operate by assigning probabilities to sentences by factorizing their likelihood into n-grams. Neural language models further *embed* each word $w(t)$ into a low-dimensional vector representation (denoted by $\mathbf{s}(t)$)[1]. These word representations are learned as the language model is trained (Bengio et al., 2003) and enable to define a word in relation to other words in a metric space.

Recurrent Neural Network Mikolov et al. (2010) suggested the use of Recurrent Neural Networks (RNN) to model long-range dependencies between words as they are not restricted to a fixed context length, like the feedforward neural network (Bengio et al., 2003). The hidden representation $\mathbf{s}(t)$ for the word in position t of the sentence in the RNN follows a first order auto-regressive dynamic (Eq. 1), where $\mathbf{W}$ is the matrix connecting the hidden representation of the previous word $\mathbf{s}(t-1)$ to the current one, $\mathbf{w}(t)$ is the one-hot index of the current word (in a vocabulary of size N words) and $\mathbf{U}$ is the matrix containing the embeddings for all the words in the vocabulary:

$$\mathbf{s}(t) = f\left(\mathbf{W}\mathbf{s}(t-1) + \mathbf{U}\mathbf{w}(t)\right) \quad (1)$$

The nonlinearity f is typically the logistic sigmoid function $f(x) = \frac{1}{1+\exp(-x)}$. At each time step, the RNN generates the word probability vector $\mathbf{y}(t)$ for the next word $\mathbf{w}(t+1)$, using the output word embedding matrix $\mathbf{V}$ and the softmax nonlinearity $g(x_i) = \frac{\exp(x_i)}{\sum_i \exp(x_i)}$:

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t)\right) \quad (2)$$

RNN with Maximum Entropy Model Mikolov et al. (2011) combined RNNs with a maximum entropy model, essentially adding a matrix that directly connects the input words' n-gram context $\mathbf{w}(t-n+1,\ldots,t)$ to the output word probabilities. In practice, because of the large vocabulary size N, designing such a matrix is computationally prohibitive. Instead, a hash-based implementation is used, where the word context is fed through a hash function h that computes the index $h(\mathbf{w}(t-n+1,\ldots,t))$ of the context words in a one-dimensional array $\mathbf{d}$ of size D (typically, $D = 10^9$). Array $\mathbf{d}$ is trained in the same way as the rest of the RNN model and contributes to the output word probabilities:

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t) + \mathbf{d}_{h(\mathbf{w}(t-n+1,\ldots,t))}\right) \quad (3)$$

As we show in our experiments, this additional matrix is crucial to a good performance on word completion tasks.

Training RNNs RNNs are trained using maximum likelihood through gradient-based optimization, such as Stochastic Gradient Descent (SGD) with an annealed learning rate λ. The Back-Propagation Through Time (BPTT) variant of SGD enables to sum-up gradients from consecutive time steps before updating the parameters of the RNN and to handle the long-range temporal dependencies in the hidden $\mathbf{s}$ and output $\mathbf{y}$ sequences. The loss function is the cross-entropy between the generated word distribution $\mathbf{y}(t)$ and the target *one-hot* word distribution $\mathbf{w}(t+1)$, and involves the log-likelihood terms $\log y_{w(t+1)}(t)$.

For speed-up, the estimation of the output word probabilities is done using hierarchical softmax outputs, i.e., class-based factorization (Mikolov and Zweig, 2012). Each word w^i is assigned to a class c^i and the corresponding log-likelihood is effectively $\log y_{w^i}(t) = \log y_{c^i}(t) + \log y_{w^j}(t)$, where j is the index of word w^i among words belonging to class c^i. In our experiments, we binned the words found in our training corpus into 250 classes according to frequency, roughly corresponding to the square root of the vocabulary size.

Dependency RNN RNNs are designed to process sequential data by iteratively presenting them with word $\mathbf{w}(t)$ and generating next word's probability distribution $\mathbf{y}(t)$ at each time step. They can be reset at the beginning of a sentence by setting all the values of hidden vector $\mathbf{s}(t)$ to zero.

Dependency parsing (Nivre, 2005) generates, for each sentence (which we note $\{w(t)\}_{t=0}^{T}$), a parse tree with a single root, many leaves and an unique path (also called *unroll*) from the root to each leaf, as illustrated on Figure 1. We now note $\{w_i\}_i$ the set of word tokens appearing in the parse tree of a sentence. The order in the notation derives from the breadth-first traversal of that tree (i.e., the root word is noted w_0). Each of the unrolls can be seen as a different sequence of words

[1] In our notation, we make a distinction between the word token $w(t)$ at position t in the sentence and its one-hot vector representation $\mathbf{w}(t)$. We note w_i the i-th word token on a breadth-first traversal of a dependency parse tree.

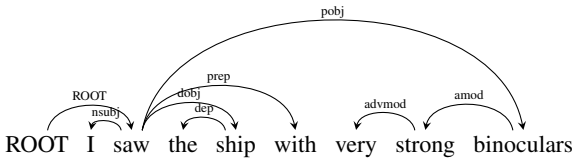

Figure 1: Example dependency tree

$\{w_i\}$, starting from the single root w_0, that are visited when one takes a specific path on the parse tree. We propose a simple transformation to the RNN algorithm so that it can process dependency parse trees. The RNN is reset and independently run on each such unroll. As detailed in the next paragraph, when evaluating the log-probability of the sentence, a word token w_i can appear in multiple unrolls but its log-likelihood is counted only once. During training, and to avoid over-training the network on word tokens that appear in more than one unroll (words near the root appear in more unrolls than those nearer the leaves), each word token w_i is given a weight discount $\alpha_i = \frac{1}{n_i}$, based on the number n_i of unrolls the token appears in. Since the RNN is optimized using SGD and updated at every time-step, the contribution of word token w_i can be discounted by multiplying the learning rate by the discount factor: $\alpha_i \lambda$.

Sentence Probability in Dependency RNN Given a word w_i, let us define the ancestor sequence $A(w_i)$ to be the subsequence of words, taken as a subset from $\{w_k\}_{k=0}^{i-1}$ and describing the path from the root node w_0 to the parent of w_i. For example, in Figure 1, the ancestors $A(\texttt{very})$ of word token `very` are `saw`, `binoculars` and `strong`. Assuming that each word w_i is conditionally independent of the words outside of its ancestor sequence, given its ancestor sequence $A(w_i)$, Gubbins and Vlachos (2013) showed that the probability of a sentence (i.e., the probability of a lexicalized tree S^T given an unlexicalized tree T) could be written as:

$$P[S^T|T] = \prod_{i=1}^{|S|} P[w_i|A(w_i)] \qquad (4)$$

This means that the conditional likelihood of a word given its ancestors needs to be counted only once in the calculation of the sentence likelihood, even though each word can appear in multiple unrolls. When modeling a sentence using an RNN, the state $\mathbf{s}_j$ that is used to generate the distribution of words $\mathbf{w}_i$ (where j is the parent of i in the tree), represents the vector embedding of the history of the ancestor words $A(w_i)$. Therefore, we count the term $P[\mathbf{w}_i|\mathbf{s}_j]$ only once when computing the likelihood of the sentence.

3 Labelled Dependency RNN

The model presented so far does not use dependency labels. For this purpose we adapted the context-dependent RNN (Mikolov and Zweig, 2012) to handle them as additional M-dimensional label input features $\mathbf{f}(t)$. These features require a matrix $\mathbf{F}$ that connects label features to word vectors, thus yielding a new dynamical model (Eq. 5) in the RNN, and a matrix $\mathbf{G}$ that connects label features to output word probabilities. The full model becomes as follows:

$$\mathbf{s}(t) = f\left(\mathbf{W}\mathbf{s}(t-1) + \mathbf{U}\mathbf{w}(t) + \mathbf{F}\mathbf{f}(t)\right) \qquad (5)$$

$$\mathbf{y}(t) = g\left(\mathbf{V}\mathbf{s}(t) + \mathbf{G}\mathbf{f}(t) + \mathbf{d}_{h(\mathbf{w}_{t-n+1}^{t})}\right) \qquad (6)$$

On our training dataset, the dependency parsing model found $M = 44$ distinct labels (e.g., *nsubj*, *det* or *prep*). At each time step t, the context word $\mathbf{w}(t)$ is associated a single dependency label $\mathbf{f}(t)$ (a one-hot vector of dimension M).

Let $G(w)$ be the sequence of grammatical relations (dependency tree labels) between successive elements of $(A(w), w)$. The factorization of the sentence likelihood from Eq. 4 becomes:

$$P[S^T|T] = \prod_{i=1}^{|S|} P[w_i|A(w_i), G(w_i)] \qquad (7)$$

4 Implementation and Dataset

We modified the Feature-Augmented RNN toolkit[2] and adapted it to handle tree-structured data. Specifically, and instead of being run sequentially on the entire training corpus, the RNN is run on all the word tokens in all unrolls of all the sentences in all the books of the corpus. The RNN is reset at the beginning of each unroll of a sentence. When calculating the log-probability of a sentence, the contribution of each word token is counted only once (and stored in a hash-table specific for that sentence). Once all the unrolls of a sentence are processed, the log-probability of the sentence is the sum of the per-token log-probabilities in that hash-table. We also further

[2] http://research.microsoft.com/en-us/projects/rnn/

enhanced the RNN library by replacing some large matrix multiplication routines by calls to the CBLAS library, thus yielding a two- to three-fold speed-up in the test and training time.[3]

The training corpus consists of 522 19th century novels from Project Gutenberg (Zweig and Burges, 2012). All processing (sentence-splitting, PoS tagging, syntactic parsing) was performed using the Stanford CoreNLP toolkit (Manning et al., 2014). The test set contains 1040 sentences to be completed. Each sentence consists of one ground truth and 4 impostor sentences where a specific word has been replaced with a syntactically correct but semantically incorrect *impostor* word. Dependency trees are generated for each sentence candidate. We split that set into two, using the first 520 sentences in the validation (development) set and the latter 520 sentences in the test set. During training, we start annealing the learning rate λ with decay factor 0.66 as soon as the classification error on the validation set starts to increase.

5 Results

Table 1 shows the accuracy (validation and test sets) obtained using a simple RNN with 50, 100, 200 and 300-dimensional hidden word representation and 250 frequency-based word classes (vocabulary size $N = 72846$ words appearing at least 5 times in the training corpus). One notices that adding the direct word context to target word connections (using the additional matrix described in section 2), enables to jump from a poor performance of about 30% accuracy to about 40% test accuracy, essentially matching the 39% accuracy reported for Good-Turing n-gram language models in Zweig et al. (2012). Modelling 4-grams yields even better results, closer to the 45% accuracy reported for RNNs in (Zweig et al., 2012).[4]

As Table 2 shows, dependency RNNs (depRNN) enable about 10 point word accuracy improvement over sequential RNNs.

The best accuracy achieved by the depRNN on the combined development and test sets used to report results in previous work was 53.5%. The best reported results in the MSR sentence completion challenge have been achieved by Log-BiLinear Models (LBLs) (Mnih and Hinton, 2007), a variant of neural language models with 54.7% to 55.5% accuracy (Mnih and Teh, 2012; Mnih and Kavukcuoglu, 2013). We conjecture that their superior performance might stem from the fact that LBLs, just like n-grams, take into account the order of the words in the context and can thus model higher-order Markovian dynamics than the simple first-order autoregressive dynamics in RNNs. The depRNN proposed ignores the left-to-right word order, thus it is likely that a combination of these approaches will result in even higher accuracies. Gubbins and Vlachos (2013) developed a count-based dependency language model achieving 50% accuracy. Finally, Mikolov et al. (2013) report that they achieved 55.4% accuracy with an ensemble of RNNs, without giving any other details.

Architecture	**50h**	**100h**	**200h**	**300h**
RNN (dev)	*29.6*	*30.0*	*30.0*	*30.6*
RNN (test)	28.1	30.0	30.4	28.5
RNN+2g (dev)	*29.6*	*28.7*	*29.4*	*29.8*
RNN+2g (test)	29.6	28.7	28.1	30.2
RNN+3g (dev)	*39.2*	*39.4*	*38.8*	*36.5*
RNN+3g (test)	40.8	40.6	40.2	39.8
RNN+4g (dev)	*40.2*	*40.6*	*40.0*	*40.2*
RNN+4g (test)	42.3	41.2	40.4	39.2

Table 1: Accuracy of sequential RNN on the MSR Sentence Completion Challenge.

Architecture	**50h**	**100h**	**200h**
depRNN+3g (dev)	53.3	**54.2**	54.2
depRNN+3g (test)	51.9	**52.7**	51.9
ldepRNN+3g (dev)	48.8	51.5	49.0
ldepRNN+3g (test)	44.8	45.4	47.7
depRNN+4g (dev)	52.7	54.0	52.7
depRNN+4g (test)	48.9	51.3	50.8
ldepRNN+4g (dev)	49.4	50.0	(48.5)
ldepRNN+4g (test)	47.7	51.4	(47.7)

Table 2: Accuracy of (un-)labeled dependency RNN (depRNN and ldepRNN respectively).

6 Discussion

Related work Mirowski et al. (2010) incorporated syntactic information into neural language models using PoS tags as additional input to LBLs but obtained only a small reduction of the word error rate in a speech recognition task. Similarly, Bian et al. (2014) enriched the Continuous Bag-of-

[3]Our code and our preprocessed datasets are available from: `https://github.com/piotrmirowski/DependencyTreeRnn`

[4]The paper did not provide details on the maximum entropy features or on class-based hierarchical softmax).

Words (CBOW) model of Mikolov et al. (2013) by incorporating morphology, PoS tags and entity categories into 600-dimensional word embeddings trained on the Gutenberg dataset, increasing sentence completion accuracy from 41% to 44%. Other work on incorporating syntax into language modeling include Chelba et al. (1997) and Pauls and Klein (2012), however none of these approaches considered neural language models, only count-based ones. Levy and Goldberg (2014) and Zhao et al. (2014) proposed to train neural word embeddings using skip-grams and CBOWs on dependency parse trees, but did not extend their approach to actual language models such as LBL and RNN and did not evaluate the word embeddings on word completion tasks.

Note that we assume that the dependency tree is supplied prior to running the RNN which limits the scope of the Dependency RNN to the scoring of complete sentences, not to next word prediction (unless a dependency tree parse for the sentence to be generated is provided). Nevertheless, it is common in speech recognition and machine translation to use a conventional decoder to produce an N-best list of the most likely candidate sentences and then re-score them with the language model. (Chelba et al., 1997; Pauls and Klein, 2011)

Tai et al. (2015) propose a similar approach to ours, learning Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997; Graves, 2012) RNNs on dependency parse tree network topologies. Their architectures is not designed to predict next-word probability distributions, as in a language model, but to classify the input words (sentiment analysis task) or to measure the similarity in hidden representations (semantic relatedness task). Their relative improvement in performance (tree LSTMs vs standard LSTMs) on these two tasks is smaller than ours, probably because the LSTMs are better than RNNs at storing long-term dependencies and thus do not benefit form the word ordering from dependency trees as much as RNNs. In a similar vein to ours, Miceli-Barone and Attardi (2015) simply propose to enhance RNN-based machine translation by permuting the order of the words in the source sentence to match the order of the words in the target sentence, using a source-side dependency parsing.

Limitations of RNNs for word completion Zweig et al. (2012) reported that RNNs achieve lower perplexity than n-grams but do not always outperform them on word completion tasks. As illustrated in Fig. 2, the validation set perplexity (comprising all 5 choices for each sentence) of the RNN keeps decreasing monotonically (once we start annealing the learning rate), whereas the validation accuracy rapidly reaches a plateau and oscillates. Our observation confirms that, once an RNN went through a few training epochs, change in perplexity is no longer a good predictor of change in word accuracy. We presume that the log-likelihood of word distribution is not a training objective crafted for $precision@1$, and that further perplexity reduction happens in the middle and tail of the word distribution.

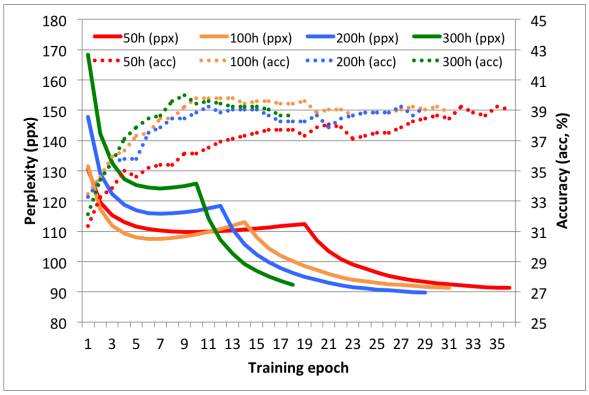

Figure 2: Perplexity vs. accuracy of RNNs

7 Conclusions

In this paper we proposed a novel language model, dependency RNN, which incorporates syntactic dependencies into the RNN formulation. We evaluated its performance on the MSR sentence completion task and showed that it improves over RNN by 10 points in accuracy, while achieving results comparable with the state-of-the-art. Further work will include extending the dependency tree language modeling to Long Short-Term Memory RNNs to handle longer syntactic dependencies.

Acknowledgements

We thank our anonymous reviewers for their valuable feedback. PM also thanks Geoffrey Zweig, Daniel Voinea, Francesco Nidito and Davide di Gennaro for sharing the original Feature-Augmented RNN toolkit on the Microsoft Research website and for insights about that code, as well as Bhaskar Mitra, Milad Shokouhi and Andriy Mnih for enlighting discussions about word embedding and sentence completion.

References

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

Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Knowledge-powered deep learning for word embedding. In *Machine Learning and Knowledge Discovery in Databases, Lecture Notes in Computer Science*, volume 8724, pages 132–148.

Ciprian Chelba, David Engle, Frederick Jelinek, Victor Jimenez, Sanjeev Khudanpur, Lidia Mangu, Harry Printz, Eric Ristad, Ronald Rosenfeld, Andreas Stolcke, et al. 1997. Structure and performance of a dependency language model. In *Proceedings of Eurospeech*, volume 5, pages 2775–2778.

Alex Graves. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer.

Joseph Gubbins and Andreas Vlachos. 2013. Dependency language models for sentence completion. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:17351780.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Antonio Valerio Miceli-Barone and Giuseppe Attardi. 2015. Non-projective dependency-based pre-reordering with recurrent neural network for machine translation. In *The 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference of the Asian Federation of Natural Language Processing*.

Tomas Mikolov and Geoff Zweig. 2012. Context dependent recurrent neural network language model. In *Speech Language Technologies (SLT), 2012 IEEE Workshop on*. IEEE.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocky. 2011. Strategies for training large scale neural network language models. In *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pages 196–201. IEEE.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Piotr Mirowski, Sumit Chopra, Suhrid Balakrishnan, and Srinivas Bangalore. 2010. Feature-rich continuous language models for speech recognition. In *Spoken Language Technology Workshop (SLT), 2010 IEEE*, pages 241–246. IEEE.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*, page 641648.

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2265–2273. Curran Associates, Inc.

Andriy Mnih and Yee W Teh. 2012. 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.

Joakim Nivre. 2005. Dependency grammar and dependency parsing. *MSI report*, 5133(1959):1–32.

Adam Pauls and Dan Klein. 2011. Faster and Smaller N-Gram Language Models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 258–267. Association for Computational Linguistics.

Adam Pauls and Dan Klein. 2012. Large-scale syntactic language modeling with treelets. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 959–968. Association for Computational Linguistics.

Brian Roark, Murat Saraclar, and Michael Collins. 2007. Discriminative n-gram language modeling. *Computer Speech & Language*, 21(2):373 – 392.

Kai Sheng Tai, Richard Socher, and Christopher Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *The 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference of the Asian Federation of Natural Language Processing*.

Yinggong Zhao, Shujian Huang, Xinyu Dai, Jianbing Zhang, and Jiajun Chen. 2014. Learning word embeddings from dependency relations. In *In Proceedings of Asian Language Processing (IALP)*.

Geoffrey Zweig and Christopher J. C. Burges. 2012. A challenge set for advancing language modeling. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 29–36. Association for Computational Linguistics.

Geoffrey Zweig, John C Platt, Christopher Meek, Christopher J. C. Burges, Ainur Yessenalina, and Qiang Liu. 2012. Computational approaches to sentence completion. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 601–610.