GRAMMAR AS A FOREIGN LANGUAGE

Oriol Vinyals,* Lukasz Kaiser,* Terry Koo, Slav Petrov, Ilya Sutskever, Geoffrey Hinton
Google
{vinyals,lukaszkaiser,terrykoo,slav,ilyasu,geoffhinton}@google.com

ABSTRACT

Syntactic parsing is a fundamental problem in computational linguistics and natural language processing. Traditional approaches to parsing are highly complex and problem specific. Recently, Sutskever et al. (2014) presented a domain-independent method for learning to map input sequences to output sequences that achieved strong results on a large scale machine translation problem. In this work, we show that precisely the same sequence-to-sequence method achieves results that are close to state-of-the-art on syntactic constituency parsing, whilst making almost no assumptions about the structure of the problem.

1 INTRODUCTION

It has recently been shown that a recurrent neural network can learn complex sequence-to-sequence mappings directly from raw data. This was first demonstrated on the English-to-French translation task (Sutskever et al., 2014), but the same approach also works for evaluating short python programs (Zaremba & Sutskever, 2014). In this work, we use the same type of recurrent neural net, called Long Short-Term Memory (Hochreiter & Schmidhuber, 1997, LSTM). The LSTM model directly maps a variable-length input sequence to a large but fixed-sized vector, which is then mapped to a variable-length output. It can therefore be used as a general function learning mechanism: Given example inputs x and corresponding outputs y of any serializable type, to learn a function f such that $f(x) = y$, just serialize each x and y and apply the sequence-to-sequence learning model.

The procedure described above will not work for arbitrary functions f, as the sequence-to-sequence network has inherent limitations: it uses a memory of constant size and runs in linear time. It does, however, achieve high performance on a large scale machine translation task (Luong et al., 2014). In this paper we show that it also works well for syntactic constituency parsing, even though this task requires modeling complex relations between input words and producing trees as the output.

Our main results are as follows: We train a deep LSTM model with 34M parameters on a dataset consisting of 90K sentences obtained from various treebanks and 7M sentences from the web that are automatically parsed with the BerkeleyParser (Petrov et al., 2006). This model achieves an F1 score of 90.5 on section 23 of the Penn Treebank. For comparison, the BerkeleyParser achieves an F1 score of 90.2 when trained on the same treebank data. A model combination of 10 such LSTMs achieves an F1 score of 91.6.

As we will demonstrate in the experimental section, the automatically parsed data is crucial for our model. Since the LSTM lacks prior task-specific knowledge, it needs many examples to learn to parse accurately. However, in the presence of sufficient training examples, it is able to automatically learn the complex syntactic relationships between the input and output pairs, which are typically manually engineered into parsing models. In particular, we do not binarize the parse trees and do not need any special handling for unary productions or unknown words, which are simply mapped to a single unknown word token. Despite the simplicity of our approach, our final results are competitive and close to the state of the art.

A common criticism of the sequence-to-sequence approach of Sutskever et al. (2014) is that it is fundamentally incapable of dealing with long inputs and outputs, due to its need to store the entire input sequence in its short-term memory. Despite this potential concern, the deep LSTM model (which in our experiments used a 4,000-dimensional state) had little trouble with fairly long sequences. Indeed, the average sentence in the dataset has 22 words and the average parse tree annotation has 66 symbols, which did not pose a challenge to our sequence-to-sequence LSTM model.

*Equal contribution

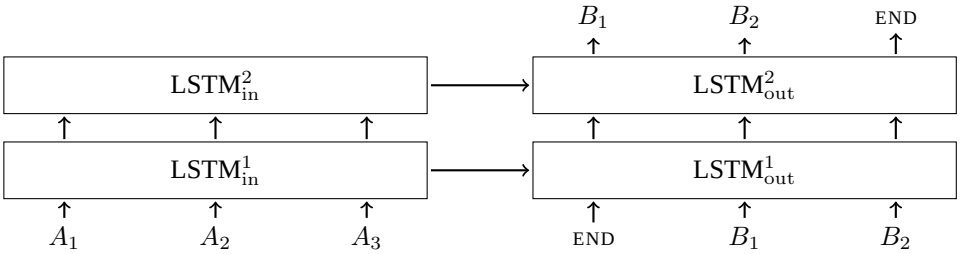


Figure 1: A schematic outline of the sequence-to-sequence model of Sutskever et al. (2014). A deep input LSTM reads the input sequence A_1, A_2, A_3 one token at a time and encodes it as its final hidden state vector. Then another deep LSTM starts from that hidden state vector and stochastically decodes it to produce the output sequence B_1, B_2. Every sequence must terminate with a special end-of-sequence token.

2 LSTM PARSING MODEL

Let us first recall the sequence-to-sequence LSTM model. The Long Short-Term Memory model of Hochreiter & Schmidhuber (1997) is defined as follows. Let x_t, h_t, and m_t be the input, control state, and memory state at timestep t. Then, given a sequence of inputs $(x_1, \ldots, x_T)$, the LSTM computes the h-sequence $(h_1, \ldots, h_T)$ and the m-sequence $(m_1, \ldots, m_T)$ as follows

$$\begin{aligned} i_t &= \text{sigm}(W_1 x_t + W_2 h_{t-1}) \\ i'_t &= \tanh(W_3 x_t + W_4 h_{t-1}) \\ f_t &= \text{sigm}(W_5 x_t + W_6 h_{t-1}) \\ o_t &= \text{sigm}(W_7 x_t + W_8 h_{t-1}) \\ m_t &= m_{t-1} \odot f_t + i_t \odot i'_t \\ h_t &= m_t \odot o_t \end{aligned}$$

The operator $\odot$ denotes element-wise multiplication, the matrices $W_1, \ldots, W_8$ and the vector h_0 are the parameters of the model, and all the nonlinearities are computed element-wise. Note that the LSTM these equations and their derivatives need to be implemented only once.

In a deep LSTM, each subsequent layer uses the h-sequence of the previous layer for its input sequence x. The deep LSTM defines a distribution over output sequences given an input sequence:

$$\begin{aligned} P(B|A) &= \prod_{t=1}^{T_B} P(B_t | A_1, \ldots, A_{T_A}, B_1, \ldots, B_{t-1}) \\ &\equiv \prod_{t=1}^{T_B} \text{softmax}(W_{\text{o}} \cdot h_{t-1+T_A})^\top \delta_{B_t} \end{aligned}$$

The above equation assumes a deep LSTM whose input sequence is $x = (A_1, \ldots, A_{T_A}, B_1, \ldots, B_{T_B})$, so h_t denotes t-th element of the h-sequence of topmost LSTM, which is a function of $(A_1, \ldots, A_{T_A}, B_1, \ldots, B_{t-T_A})$. The matrix W_{o} consists of the vector representations of each output symbol and the symbol δ_b is a Kronecker delta with a dimension for each output symbol, so $\text{softmax}(W_{\text{o}} \cdot h_{t-1+T_A})^\top \delta_{B_t}$ is precisely the B_t'th element of the distribution defined by the softmax. Every output sequence terminates with a special end-of-sequence token which is necessary in order to define a distribution over sequences of variable lengths. We may sometimes use two different sets of LSTM parameters, one for the input sequence and one for the output sequence, as shown in Figure 1. When we use two sets of LSTM parameters, we say that the parameters are *untied*. Otherwise, when $\text{LSTM}_{\text{in}} = \text{LSTM}_{\text{out}}$, we say that they are *tied*. Stochastic gradient descent is used to maximize the training objective which is the average over the training set of the log probability of the correct output sequence given the input sequence.

2.1 ADAPTATIONS FOR PARSING

To apply the model described above to parsing, we need to design an invertible way of converting the parse tree into a sequence (linearization). We experimented with two ways of using the network

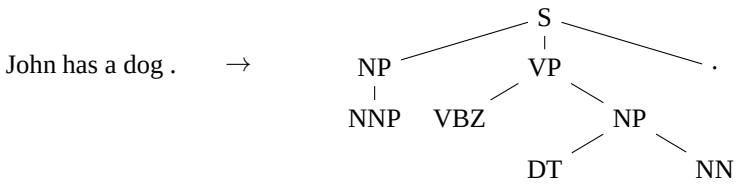
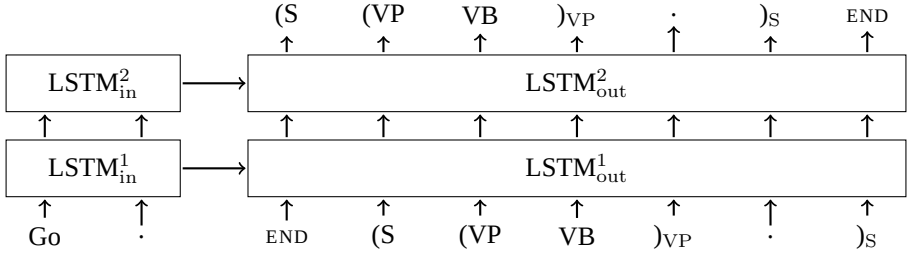


Figure 2: Example parsing task, a basic linearization, and one with stack control symbols.

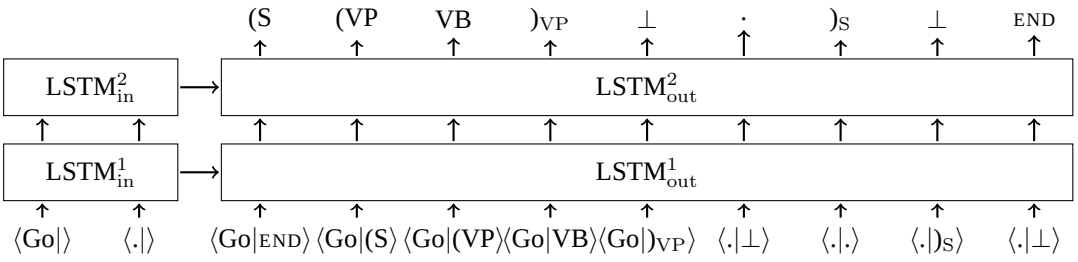
which both rely on linearizing the tree in a very simple way following a depth-first traversal order, as depicted in Figure 2.

The most basic way of using the above model for parsing works as follows. First, the network consumes the full sentence in a linear left-to-right sweep, creating a vector in memory. Then, it outputs the linearized parse tree using only the information in this memory vector. We call this "basic encoding" in the remainder of the paper. An example run of this model on the sentence "Go." with a 2-layer network looks as follows.



While the above method is the most straightforward application of the translation model, we experimented with ways of improving the input format for the network. Inspired by shift-reduce transition-based parsers (Nivre, 2008; Zhu et al., 2013), we introduce a stack where the words are provided to the LSTM as additional inputs during decoding.

First, as before, the network consumes the full sentence in a linear left-to-right sweep, creating a vector in memory. Then, while outputing the linearized parse tree, we maintain a stack of words. The word on the top of the stack is provided to the network as an additional feature at each time step. When the network produces the symbol $\perp$, we pop a word from the stack. We train the network to pop words after reaching the common ancestor of the previous and next word in the depth-first tree traversal (cf. Figure 2). Note that since we do not employ complex transition strategies our model only has access to the linear ordering of the words. Furthermore, we provide only the current word as an input. We call this "stack encoding" in the remainder of the paper. An example run of this model on the sentence "Go." looks as follows.



3 EXPERIMENTS

We performed a number of experiments with both the basic encoding and the stack encoding model, as described above. We experimented with tied and untied input and output LSTMs and measured the influence of using pre-trained word embeddings and fine-tuning on in-domain data.

3.1 TRAINING DATA

In our experiments we focus on English, but the model could easily be applied to other languages. Our goal is to build a robust and domain-independent parser, that can be used to process text from various genres. To this end, we train and evaluate on the union of a number of publicly available treebanks. We use the OntoNotes corpus version 5 (Hovy et al., 2006), the English Web Treebank (Petrov & McDonald, 2012) and the updated and corrected Question Treebank (Judge et al., 2006).[1] Note that the popular Wall Street Journal section of the Penn Treebank (Marcus et al., 1993) as part of the OntoNotes corpus. In total, we train on $\sim$90K sentences and evaluate on $\sim$13K sentences. We term the 90K set the Treebank union. We chose this setup since it allows us to train and evaluate on a more diverse set of sentences, rather than overfitting the WSJ evaluation set which has been in use for 20 years and is not representative for text encountered on the web (Petrov & McDonald, 2012). To compare to an established baseline parser, we also train and test the publicly available BerkeleyParser (Petrov et al., 2006) on the Treebank union.

Additionally, we use a corpus of $\sim$7 million unlabeled sentences sampled from the web. These sentences are parsed with an in-house reimplementation of the BerkeleyParser trained on the Treebank union. We include our reimplementation as an additional baseline, and also add a self-training experiment where it is trained on its own output, similar to Huang & Harper (2009). These automatically parsed sentences are used as additional training data in our experiments and result in large performance gains (see Table 1). We will release this data to facilitate replication of our experiments.

We use EVALB for evaluation and report F1 scores on three data sets: (1) WSJ 22: section 22 of the Penn Treebank, (2) Questions: 1000 sentences from the Question Treebank, (3) Web: the first half of each domain from Web Treebank. We also test our best model on section 23 of the Penn Treebank. More details on the experimental setup can be found in the Appendix.

We do not apply any special preprocessing to the data. In particular, we do not binarize the parse trees or handle unaries in any specific way. We also treat unknown words in a naive way: we map all words beyond our 50K vocabulary to a single UNK token. This potentially underestimates our final results, but keeps our framework task-independent.

3.2 MODELS AND PARAMETERS

In our experiments we used a deep LSTM model with either 3 layers with 640 units, or 4 layers with 512 units per layer. All our architectures have about 4000 dimensions for representing the input sentence dimensions. The LSTM has an input vocabulary of 50K and an output vocabulary of 100 symbols, which results in 34M parameters. The exact architecture and optimization parameters are further discussed in the Appendix. We used a beam of size 20 during decoding, but a beam size of 2 achieved almost identical results (see below). We also found it useful to reverse the input sentences but not their parse trees, similarly to Sutskever et al. (2014), and we did it in all of our experiments. Not reversing the input had a negative impact on our development set of up to 2% absolute F1.

A basic encoding LSTM produces malformed trees in 10% of the test cases. However, a simple change to the decoder, in which we do not consider outputs that do not consume all the input words, eliminates all the malformed trees. The stack encoding LSTM implicitly enforces the consumption of all the input words, so no further modification of the decoder is needed. In the few cases where the LSTM outputs a malformed tree, we simply add brackets to either the beginning or the end of the tree in order to make it balanced.

Table 1 presents some baseline numbers for the BerkeleyParser and our reimplementation, as well as several LSTM versions. The main observation is that, despite the lack of prior knowledge encoded

[1] All treebanks are available through the Linguistic Data Consortium (LDC): OntoNotes (LDC2013T19), English Web Treebank (LDC2012T13) and Question Treebank (LDC2012R121).

Method	WSJ 22	Questions	Web
LSTM stack encoding, 3x640, tied, Treebank union only	76.6	91.7	69.9
LSTM basic encoding, 4x512, tied	89.3	95.2	79.8
LSTM basic encoding, 4x512, untied	91.1	95.7	81.2
LSTM stack encoding, 3x640, tied, fine-tuned	91.3	95.7	82.2
Ensemble of 10 basic encoding LSTM	92.1	96.0	83.4
BerkeleyParser Treebank union only	91.1	95.9	84.1
In-house BerkeleyParser Treebank union only	91.5	96.2	84.4
In-house BerkeleyParser self-trained	91.2	96.2	84.8

Table 1: F1 scores of various models on our three evaluation sets. See text for discussion.

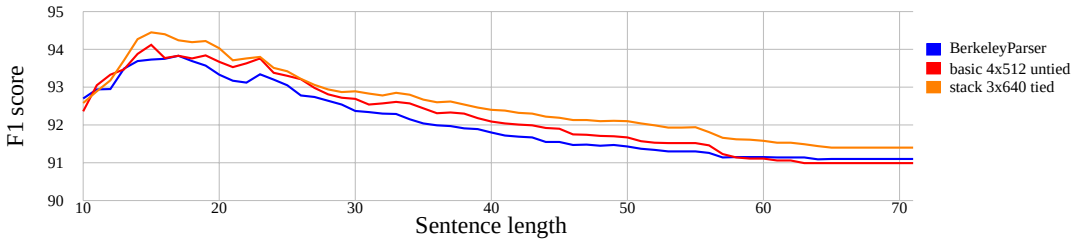


Figure 3: Effect of sentence length on the F1 score on Newswire.

in the model, both LSTM with basic encoding and stack encoding perform as well as the BerkeleyParser on WSJ section 22 (which was used as a development set), as well as in the questions dataset. On the more challenging web data, the basic encoding suffers – likely as a result of early mistakes propagating forward through the left to right naive decoder. Using stack encoding, these early mistakes are likely less of an issue. We trained a stack encoding LSTM only on Treebank data as well. This model (cf. first line of Table 1) does not achieve good results, so a large dataset parsed with BerkeleyParser is indeed crucial for this method. We also report an ensemble of 10 basic encoding LSTMs, which is an effective way to improve the performance of neural networks. Indeed, all the scores substantially improve over the in-house BerkeleyParser – used to generate the training set – on WSJ 22 and questions, and closes most of the gap on the web dataset. Lastly, untying the parameters was effective for the basic encoding scheme, but unnecessary with stack encoding.

3.3 Effect of Sentence Length and Beam Size

An important concern with the sequence-to-sequence LSTM is that it may not be able to handle long sentences well. We determine the extent of this problem by partitioning the validation set by length, and evaluating the LSTM and the BerkeleyParser on sentences of each length. The results, which are presented in Figure 3, clearly show that the LSTM's performance does not deteriorate on long sentences, as compared to the performance of the BerkeleyParser.

As for the effects of beam size in the decoder, using no beam search at all (i.e., beam size of 1) lowers the score significantly. The numbers below show the F1 scores on WSJ 22 for different beam sizes for the stack encoding 3x640 tied and fine-tuned model.

Beam size	1	2	3	6	9	20
F1 score	90.7	91.0	91.0	91.1	91.2	91.3

3.4 Effect of Pre-training and Fine-tuning

In addition to the results above, where we train only on the joint set of 7M+90K sentences and with a vocabulary of size 50K, we experimented with two additional variations.

For one, instead of feeding the network with tokens from a 50K vocabulary and learning an embedding for them, we tried to provide the network with already embedded words. These skip-gram

Experiments from this paper	F1
LSTM basic encoding, 4x512, untied	90.5
Ensemble of 10 basic encoding LSTMs	91.6
Petrov et al. (2006) Treebank union only	90.4

Experiments from other work	F1
Petrov et al. (2006) WSJ only	90.4
Petrov (2010) WSJ only ensemble	91.8
Huang & Harper (2009) semi-supervised	91.3
Huang et al. (2010) semi-supervised ensemble	92.4
Zhu et al. (2013) WSJ only	90.4
Zhu et al. (2013) semi-supervised	91.3

Table 2: F1 scores on Section 23 from the Penn Treebank.

embeddings of size 512 were pre-trained using word2vec (Mikolov et al., 2013) on a 10B-word corpus, and kept fix while training the other parameters of the network.

In addition to pre-training, we also experimented with fine-tuning the model only on the 40K set of WSJ training sentences. This gives the network a chance to correct certain errors, but it must be stopped early to prevent overfitting.

We measured the influence of these factors on a stack-driven model with 3 LSTM layers of size 640. The version with no pre-training but with fine-tuning corresponds to a row in the table above. The influence of pre-training and fine-tuning on the F1 score on our development set (section 22 of WSJ) is summarized below. We write PT to stand for "pre-training" and FT to stand for "fine-tuning".

Model	No PT, No FT	PT, No FT	No PT, FT	PT, FT
F1 score	90.9	90.5	91.3	91.2

One can see that fine-tuning brings moderate gains while pre-training has almost no influence. It is worth noting that pre-training significantly reduces the number of trainable parameters of the model.

3.5 Final Results

It is difficult to directly compare our results to those reported in previous work since our training setup differs. To compare to a publicly available state-of-the-art parser, we trained the Berkeley-Parser (Petrov et al., 2006) on our experimental setup. Table 2 shows performance on section 23 from the Penn Treebank when training on our setup on the left, and results from other papers on the right. Additionally, we compare to variants of the BerkeleyParser that use self-training on unlabeled data (Huang & Harper, 2009), or built an ensemble of multiple parsers (Petrov, 2010), or combine both techniques. It is interesting to see that additional treebank data does not help much on Section 23 of WSJ, but it helps a lot for parsing out-of-domain text. Finally, we include the best linear-time parser in the literature, the transition-based parser of Zhu et al. (2013).

It is encouraging to see that our LSTM models are competitive with these highly optimized parsers that have received a lot of task specific tuning. Moreover, when running on GPUs the LSTM model has a significant speed advantage. Using batches of 128 sentences on a generic, unoptimized decoder on a GPU we decode about 160 sentences from WSJ per second. This is better than the speed reported in Figure 4 of (Hall et al., 2014) even though we run on sentences of all lengths (not only under 40), our model achieves better accuracy, and their code is highly optimized.

4 Analysis

To shed some light on what the LSTM is learning, we examined some of the induced latent representations. We looked at individual words, non-terminal labels and also entire sentences.

Table 3 shows some selected examples that exhibit interesting patterns for words and their closest neighbors by cosine similarity. The table contains the closest three neighbors for the embeddings learnt by the LSTM and contrasts them to the embeddings learnt by a skip-gram model by Mikolov et al. (2013). It is interesting to observe that the neighbors differ quite significantly between the two methods. While the skip-gram neighbors are very topical and semantic in nature, the LSTM neighbors are a lot more syntactic. For example, the neighbors of *give* and *wait* are different forms of the same verbs for the skip-grams, but are different verbs that take the same number of arguments for the LSTM. Similarly, *book* is grouped with other words that can be both nouns and

word	LSTM neighbors	skipgram neighbors
give	bring, send, tell	giving, gave, gives
wait	stay, fly, walk	waited, waiting, waits
book	deal, bank, film	boks, memoir, autobiography
Turkey	Congress, Europe, Spain	Ankara, Turkish, Azerbaijan
Angeles	York, Jersey, Francisco	Los, LA, Seattle
he	she, they, we	his, He, she
further	better, faster, harder	review, edits, No
do	did, does, Do	should, not, modify
Monday	Tuesday, Thursday, Friday	Thursday, Tuesday, Wednesday

Table 3: Selected words and their three closest neighbors by cosine similarity.

verbs by the LSTM, while the skip-gram embedded it with its noun synonyms. It is also interesting to see that *Angeles* is grouped with the second part of other noun noun compound city names by the LSTM, but with *LA* and *Los* by the skip-gram.

Since the non-terminals of the parse trees are also embedded we can examine their neighbors as well. In general, the distance between the various non-terminal labels is much larger than between different words. Only a few non-terminals are close to each other, for example *(S* is close to *(SQ*, *(SINV* and *(SBARQ*, but most others are pretty far apart. For the part of speech tags, only *NNP* has close neighbors, namely *NN*, *NNPS*, and *CD*.

Finally, we can examine full sentence embeddings and their neighbors. For the sentences in the development set, we find that the LSTM groups sentences of similar lengths. Furthermore there is some syntactic and semantic resemblance, but since the sentences are quite diverse there are few interesting neighbors. We therefore supplemented the development set with a few manually generated sentences. In particular, we generated sentences with identical syntactic structure and only lexical differences, e.g. *I have a* $\{$*brother, sister*$\}$*, who has a* $\{$*brother, sister*$\}$. The LSTM grouped these sentences in close proximity and there was very little difference due to the different lexical choices. Some of the other manually generated sentences included prepositional phrase attachment ambiguities: *I ate the pasta with the* $\{$*tomatoes, cheese, pesto, fork, spoon*$\}$. It was exciting to see that the sentences where the prepositional phrase modifies the noun *pasta* were grouped closer to each other than the ones where the prepositional phrase modifies the verb *ate*. Unfortunately, the predicted parse trees all had the prepositional phrase attaching low and modifying the noun.

To see how well the LSTM can handle embedded clauses and nested structures, we tried parsing sentences of the form *I have a (brother who has a)*^{n} *brother*. The LSTM can parse such sentences for n up to 12 without forgetting to close all clauses, which suggests that it has learned to model context-free structures for a reasonably deep stack. This further confirms the finding that LSTMs are capable of handling long and complex sentences in a robust way.

5 Related Work

The task of syntactic constituency parsing has received a tremendous amount of attention in the last 20 years. Traditional approaches to constituency parsing rely on probabilistic context-free grammars (CFGs). The focus in these approaches is on devising appropriate smoothing techniques for highly lexicalized and thus rare events (Collins, 1997) or carefully crafting the model structure (Klein & Manning, 2003). Petrov et al. (2006) partially alleviate the heavy reliance on manual modeling of linguistic structure by using latent variables to learn a more articulated model. However, their model still depends on a CFG backbone and is thereby potentially restricted in its capacity.

Early neural network approaches to parsing, for example by Henderson (2003; 2004) also relied on strong linguistic insights. Titov & Henderson (2007) and Henderson & Titov (2010) introduced Incremental Sigmoid Belief Networks for syntactic parsing. By constructing the model structure incrementally, they are able to avoid making strong independence assumptions but inference becomes intractable. To avoid complex inference methods, Collobert (2011) propose a recurrent neural network where parse trees are decomposed into a stack of independent levels. Unfortunately, this decomposition breaks for long sentences and their accuracy on longer sentences falls quite significantly behind the state of the art. Socher et al. (2011) used a tree-structured neural network to score

candidate parse trees. Their model however relies again on the CFG assumption and furthermore can only be used to score candidate trees rather than for full inference.

Our LSTM model significantly differs from all these models, as it makes no assumptions about the task. As a sequence-to-sequence prediction model it is somewhat related to the incremental parsing models, pioneered by Ratnaparkhi (1997) and extended by Collins & Roark (2004). Such linear time parsers however typically need some task-specific constraints and might build up the parse in multiple passes. Relatedly, Zhu et al. (2013) present excellent parsing results with a single left-to-right pass, but require a stack to explicitly delay making decisions and a parsing-specific transition strategy in order to achieve good parsing accuracies. The LSTM in contrast uses its short term memory to model the complex underlying structure that connects the input-output pairs.

LSTMs might not be the only models capable of such modeling. Recently, researchers have developed a number of neural network models that can be applied to general sequence-to-sequence problems. Graves (2013) was the first to propose a differentiable attention mechanism for the general problem of handwritten text synthesis, although his approach assumed a monotonic alignment between the input and the output sequences. Later, Bahdanau et al. (2014) introduced a more general attention model that does not assume a monotonic alignment, and applied it to machine translation, and Chorowski et al. (2014) applied the same model to speech recognition. Kalchbrenner & Blunsom (2013) used a convolutional neural network to encode a variable-sized input sentence into a vector of a fixed dimension and used an RNN to produce the output sentence. Essentially the same model has been used by Vinyals et al. (2014) to successfully learn to generate image captions. Even though most of these models could be applied to parsing, we chose the model of Sutskever et al. (2014) because it is the simplest architecture that can solve general sequence-to-sequence problems and because it achieves the best performance on a large scale machine translation task (Luong et al., 2014). It is also able to embed entire sentences in a continuous vector space. Finally, Ghahramani (1990) applied a similar recurrent neural network to the problem of syntactic parsing 20 years ago.

6 CONCLUSIONS

In this work, we have shown that the generic sequence-to-sequence approach of Sutskever et al. (2014) can achieve competitive results on syntactic constituent parsing with relatively little effort or tuning. Our results highlight the importance of large datasets when using large deep neural networks that do not contain domain-specific, hand-engineered knowledge. Lacking prior knowledge, our system was unable to learn an accurate parser from the treebank union alone (cf. first line of Table 1). Fortunately, there is a very simple way to benefit from the hand-engineering that goes into more conventional parsers: We use these parsers to create additional training data. This allows us to benefit from the prior knowledge in other parsing systems without putting any constraints on the form of the internal representations used by the LSTM. The fact that the LSTM was able to match and even outperform the BerkeleyParser that was used to annotate the 7M sentences suggests that this simple way of stealing prior knowledge is very effective, though computationally expensive. In the long run, however, generic learning algorithms for deep recurrent neural networks would be much more useful if they could learn from smaller datasets.

Acknowledgement. We would like to thank Amin Ahmad, Dan Bikel and Jonni Kanerva.

REFERENCES

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

Chorowski, Jan, Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint arXiv:1412.1602*, 2014.

Collins, Michael. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pp. 16–23, Madrid, Spain, July 1997. Association for Computational Linguistics. doi: 10.3115/976909.979620.

Collins, Michael and Roark, Brian. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 111–118, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1218970.

Collobert, Ronan. Deep learning for efficient discriminative parsing. In *International Conference on Artificial Intelligence and Statistics*, 2011.

Ghahramani, Zoubin. A neural network for learning how to parse tree adjoining grammar. B.S.Eng Thesis, University of Pennsylvania, 1990.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Hall, David, Berg-Kirkpatrick, Taylor, Canny, John, and Klein, Dan. Sparser, better, faster gpu parsing. In *ACL*, 2014.

Henderson, James. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 24–31, Edmonton, Canada, May 2003.

Henderson, James. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp. 95–102, Barcelona, Spain, July 2004. doi: 10.3115/1218955.1218968.

Henderson, James and Titov, Ivan. Incremental sigmoid belief networks for grammar learning. *Journal of Machine Learning Research*, 11:3541–3570, 2010.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hovy, Eduard, Marcus, Mitchell, Palmer, Martha, Ramshaw, Lance, and Weischedel, Ralph. Ontonotes: The 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Short Papers*, pp. 57–60, New York City, USA, June 2006. Association for Computational Linguistics.

Huang, Zhongqiang and Harper, Mary. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pp. 832–841, Singapore, August 2009. Association for Computational Linguistics.

Huang, Zhongqiang, Harper, Mary, and Petrov, Slav. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 12–22, Cambridge, MA, October 2010. Association for Computational Linguistics.

Judge, John, Cahill, Aoife, and van Genabith, Josef. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pp. 497–504, Sydney, Australia, July 2006. Association for Computational Linguistics. doi: 10.3115/1220175.1220238.

Kalchbrenner, Nal and Blunsom, Phil. Recurrent continuous translation models. In *EMNLP*, pp. 1700–1709, 2013.

Klein, Dan and Manning, Christopher D. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pp. 423–430, Sapporo, Japan, July 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075150.

Luong, Thang, Sutskever, Ilya, Le, Quoc V, Vinyals, Oriol, and Zaremba, Wojciech. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

Marcus, Mitchell P., Santorini, Beatrice, and Marcinkiewicz, Mary Ann. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

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

Nivre, Joakim. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553, December 2008. ISSN 0891-2017. doi: 10.1162/coli.07-056-R1-07-027.

Petrov, Slav. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 19–27, Los Angeles, California, June 2010. Association for Computational Linguistics.

Petrov, Slav and McDonald, Ryan. Overview of the 2012 shared task on parsing the web. Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL), 2012.

Petrov, Slav, Barrett, Leon, Thibaux, Romain, and Klein, Dan. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pp. 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics.

Ratnaparkhi, Adwait. A linear observed time statistical parser based on maximum entropy models. In *Second Conference on Empirical Methods in Natural Language Processing*, 1997.

Socher, Richard, Lin, Cliff C, Manning, Chris, and Ng, Andrew Y. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 129–136, 2011.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc VV. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

Titov, Ivan and Henderson, James. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 632–639, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014.

Zaremba, Wojciech and Sutskever, Ilya. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

Zhu, Muhua, Zhang, Yue, Chen, Wenliang, Zhang, Min, and Zhu, Jingbo. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 1: Long Papers)*, pp. 434–443, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Corpus	Train	Dev	Test
OntoNotes WSJ	Section 2-21	Section 22	Section 23
OntoNotes BN, MZ, NW, WB	1-8/10 Sentences	*9/10 Sentences*	*10/10 Sentences*
Question Treebank	Sentences 1-1000, Sentences 2000-3000	Sentences 1000-1500, Sentences 3000-3500	*Sentences 1500-2000, Sentences 3500-4000*
Web Treebank	Second 50% of each genre	First 50% of each genre	-

Table 4: Details regarding the experimental setup. Data sets in italics were not used.

APPENDIX

7 DETAILED EXPERIMENTAL SETUP

In this section we present details regarding our experimental setup. We use the OntoNotes corpus version 5 (Hovy et al., 2006), the English Web Treebank (Petrov & McDonald, 2012) and the updated and corrected Question Treebank (Judge et al., 2006). All treebanks are available through the Linguistic Data Consortium (LDC): OntoNotes (LDC2013T19), English Web Treebank (LDC2012T13) and Question Treebank (LDC2012R121). Table 7 presents the splits into training, development and test data that we used for each treebank. We followed standard splits whenever possible, but had to devise also our own splits when no prior split was established. This was the case for the non-WSJ portions of the OntoNotes corpus, where we divided the data into shards of 10 sentences and selected the first 8 for training, while reserving the 9th for development and the 10th for test (neither of which we used in our experiments).

We trained a deep LSTM model with Stochastic Gradient Descent (without momentum). We initialize all the weights following a random uniform distribution between -0.08 and 0.08, and use a learning rate of 0.4 with an exponential decay which drops the learning rate by half every 1.5 epochs, and learning typically stops after 5 epochs. We also constrain our gradients by clipping them to be inside a sphere of radius 5. Our model has 512 cells, and a stack of 4 LSTMs, which with a 50K input vocabulary, and 100 output softmax, yields 34M parameters when the parameters of the encoder and decoder have tied parameters, and 42M when they are untied. We also experimented with a different architecture (which was used for the stack encoding LSTM model), which has 640 cells and a stack of 3 LSTMs. The two architectures make no difference in terms of performance.

For evaluation we used EVALB with new.prm as the configuration for the WSJ and Question Treebank. For the Web Treebank we followed the SANCL shared task evaluation and used sancl.prm, which removes some of the special handling around punctuation part-of-speech tags.

This figure "lenchart.png" is available in "png" format from:

http://arxiv.org/ps/1412.7449v1

This figure "lstm.png" is available in "png" format from:

http://arxiv.org/ps/1412.7449v1