Sequential Parsing with In-order Tree Traversals

Anonymous NAACL submission

Abstract

We apply in-order tree traversals to the Seq2Seq parser (Vinyals et al., 2015), simplifying symbol prediction. Our simple greedy parser achieves 91.4 F1, which is the best among greedy parsers in the literature, on the Penn Treebank without any additional resources such as pre-trained embeddings, predicted part-of-speech tags, or hand-engineered features. Incorporating semi-supervised training methods, we achieve 93.0 F1 in linear time. With an ensemble of semi-supervised rerankers (Choe and Charniak, 2016) we obtain the state-of-the-art F1 of 94.7 (94.6% LAS, 96.4% UAS) on WSJ Section 23. Strong results on five other domains show the generality of our techniques.

1 Introduction

In recent years, many strong greedy constituency parsers have been proposed thanks to the advent of deep learning (Vinyals et al., 2015; Coavoux and Crabbé, 2016; Cross and Huang, 2016; Dyer et al., 2016). We present an extension of the sequence-to-sequence (Seq2Seq) parser of Vinyals et al. (2015), which is simple, scalable and fast. When our parser is trained on in-order tree traversal sequences, it performs competitively. With an ensemble of semi-supervised rerankers (Choe and Charniak, 2016), our semi-supervised parsers reach 94.7 F1.

1.1 Seq2Seq Parsing

Sequence-to-sequence models have been applied to machine translation (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2014). Vinyals et al. (2015) apply a variant of it to syntactic parsing by using word tokens as the input sequence and sequential (linearized) trees as the output (Figure 1b). Essentially, the parser learns to “translate” input tokens into an ASCII-based tree format. In this paper, we show that Seq2Seq parsers are more effective when in-order tree traversals are used instead of pre-order ones, described below.

1.2 Tree Traversals

The pre-order traversal Vinyals et al. (2015) use to train their Seq2Seq parser is one of many possible sequential representations of trees. The pre-order traversal is a reasonable default because constituency trees are often annotated in pre-order traversals (Marcus et al., 1993). To develop accurate parsers, we explore two simple alternatives: an in-order traversal and a reversed in-order traversal. Given a node, an in-order traversal first processes its left child, then the node, and finally the rest of the children (Figure 1c). A reversed in-order the right child, the node, and then the remaining children (Figure 1d).
BLSTMs have been proven to be very effective as the Seq2Seq parser with a bidirectional LSTM (Graves and Schmidhuber, 2005) encoder of word representations, the decoder (described below) uses the concatenation of these, \( v_i = f_i \oplus b_i \), which represents word at position \( i \) and its surrounding words. The encoder has an auxiliary objective over word representations to predict part-of-speech (POS) tags so that the encoder includes tagging information inside these word representations. The auxiliary objective improves parsing performance a bit. The objective is defined as

\[
P(t_i | v_s) = \text{softmax}(W_1 \cdot v_i + b_1)[\text{Idx}(t_i)], \quad (1)
\]

where \( t_i \) is a gold tag for word at position \( i \) and \( \text{Idx} \) is a function that returns the index of \( t_i \).

### 3.2 Decoder

As in shift-reduce parsing (Nivre, 2008), we maintain a buffer of word representations for the input sentence, \( (v_1, \ldots, v_n) \), from left to right and shift one word from the buffer whenever the decoder outputs a word. A key difference from shift-reducing parsing is that instead of representing parsing state (e.g., previous shifted symbols) with a discrete stack, we use an LSTM. At step \( j \), the decoder reads in an embedding of its previous output, \( p_{j-1} \), and the first word representation in the buffer, \( v_{x_j} \), where \( x_j \) is the position of the first word in the buffer at step \( j \). The decoder pushes \( p_{j-1} \oplus v_{x_j} \), the concatenation of the two, through its LSTM function and uses its hidden state, \( h_j \), to predict a symbol \( s_j \):

\[
P(s_j | h_j) = \text{softmax}(W_2 \cdot h_j + b_2)[\text{Idx}(s_j)], \quad (2)
\]

where \( s_j \) is a gold symbol at position \( j \). During training, we optimize the sum of Equations (1) and (2), but during inference, we only use Equation (2). The decoder predicts one of 26 phrase tags, ‘)’, WORD, END and PAD.\(^3\) When the decoder predicts WORD, it outputs the first word in the buffer rather than the \( \text{WORD} \) symbol.

**Table 1:** Statistics of sequential trees in WSJ training. The first row shows the average number of valid symbols in three kinds of gold sequential trees. The second one the average distance between pairs of matching parentheses.

<table>
<thead>
<tr>
<th></th>
<th>PRE</th>
<th>IN</th>
<th>R-IN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. # Valid Symbols</td>
<td>27.2</td>
<td>19.7</td>
<td>19.7</td>
</tr>
<tr>
<td>Avg. Distance</td>
<td>17.9</td>
<td>14.2</td>
<td>6.8</td>
</tr>
</tbody>
</table>

---

\(^1\)28 symbols in total: 26 start phrase tags, \text{WORD} \ and ‘)’.

\(^2\)In pilot experiments, we found that processing the previous output and first word together worked better than processing the output only and later passing the hidden state along with word representation to softmax layers.

\(^3\)END denotes the end of parsing and \text{PAD} \ gets added to short sentences for batching.
Unlike Vinyals et al. (2015), we don’t use attention over word representations because in parsing we know what word to process next. Compared to theirs, ours can be thought of as a Seq2Seq with explicit attention.

### 3.3 Network Architecture

We use 300 dimensions for the BLSTM encoder, 900 for the LSTM decoder. For both encoder and decoder, we stack two LSTMs and have skip connections from input layers to second LSTM layers (He et al., 2016).\(^4\) We apply dropout (Srivastava et al., 2014) to embeddings and vertical LSTM connections (Pham et al., 2014).

### 4 Experiments

We use the standard split of Wall Street Journal (WSJ) in the Penn Treebank for training (Sections 2–21), validating (22) and testing (23) our parsers. We replace singleton tokens in the training with UNK and remove spurious unary chains, e.g., ‘(NP (NP a dog))’ becomes ‘(NP a dog)’, during training but keep them during evaluation for comparison. For semi-supervised experiments (McClosky et al., 2006), we train the parsers on auto-parsed New York Times (NYT) trees (Choe and Charniak, 2016) in addition to WSJ.

#### 4.1 Supervised Training

We train models for 50 epochs and save parameters that perform best on Section 22. During training, models attempt to maximize the probability of symbols in gold sequential trees. During inference, they greedily choose the most likely valid symbols. Training takes 5.3 hours on an Nvidia 1080 GPU and parsing Section 22 takes a little less than 13 seconds (130 sentences per second). We report the hyperparameters in the Appendix.

The performance of our models with different sequential trees is reported in Table 2. Both in-order (IN) and reversed in-order (R-IN) perform substantially better than pre-order (PRE). During decoding, IN (WSJ) and R-IN (WSJ) can get confused and produce infinite chains of unaries. In these cases, we stop decoding and output trees with single S nodes. Sampling a few trees and selecting one for a failed sentence easily solves the problem. Semi-supervised training also improves the model, solving this problem as described below.

#### 4.2 Semi-supervised Training

One advantage of sequential models is that they are very scalable and we train our models on millions of NYT trees. Training data consists of WSJ training and 1.2 million auto-parsed NYT trees (resampled every epoch). As in supervised training, we train the models for 50 epochs and choose best settings on the validation set. The semi-supervised training takes about 83.3 hours on a single GPU. See results in Table 2.

#### 4.3 Sampling and Reranking

One caveat of greedy parsing is the model makes a series of local decisions, which obviously aren’t globally optimal. To overcome this, as in Dyer et al. (2016) we sample \(N\) trees with our parser for each sentence. For each tree, we sample a symbol at a time from a renormalized distribution with exponentiation \(\alpha\). We feed the samples through the CC reranker\(^3\) of Choe and Charniak (2016) and evaluate the reranked trees. The results of varying \(\alpha\) and \(N\) are reported in Tables 3 and 4. First note that the oracle scores are remarkable and there is a much room for developing better rerankers. \(\alpha\) seems to matter a bit but \(N\) doesn’t (at least once sufficiently large). For supervised evaluation, we sample 100 trees for each sentence to compare our model to reranking parsers of Dyer et al. (2016) and Kuncoro et al. (2017). For semi-supervised evaluation, we sample 200 trees.

#### 4.4 Supervised Evaluation

We first evaluate our parsers and compare them to strong neural greedy parsers and then compare our parsers with CC (WSJ) to other reranking parsers on Section 23. As shown in Table 5, the reduction in search space seems helpful for parsing but the

---

\(^4\)For example, the input to the second layer’s forward LSTM at position \(i\) is \(e_i + f_i^1\) where \(f_i^1\) is the hidden state of the first layer’s forward LSTM at position \(i\).

\(^3\)Models from github.com/cdg720/emnlp2016.
Table 3: Oracle and reranked performance of IN (semi) and CC (semi), varying $\alpha$ on WSJ Section 22. 1000 trees are sampled per sentence.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Oracle</th>
<th># Unique</th>
<th>Reranked</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>99.1</td>
<td>321.1</td>
<td>94.6</td>
</tr>
<tr>
<td>0.7</td>
<td>99.0</td>
<td>98.0</td>
<td>94.8</td>
</tr>
<tr>
<td>0.9</td>
<td>98.6</td>
<td>43.0</td>
<td>94.8</td>
</tr>
<tr>
<td>1.1</td>
<td>98.3</td>
<td>24.7</td>
<td>94.7</td>
</tr>
</tbody>
</table>

Table 4: Oracle and reranked performance of IN (semi) and CC (semi), varying the number of samples on WSJ Section 22. $\alpha = 0.7$ is used.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Oracle</th>
<th># Unique</th>
<th>Reranked</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>98.2</td>
<td>18.9</td>
<td>94.8</td>
</tr>
<tr>
<td>200</td>
<td>98.5</td>
<td>30.7</td>
<td>94.8</td>
</tr>
<tr>
<td>500</td>
<td>98.8</td>
<td>59.1</td>
<td>94.8</td>
</tr>
<tr>
<td>1000</td>
<td>99.0</td>
<td>98.0</td>
<td>94.8</td>
</tr>
</tbody>
</table>

4.5 Semi-supervised Evaluation

IN (semi) and R-IN (semi) perform comparably to state-of-the-art semi-supervised parsers (Table 6). With the CC (semi) reranker, IN (semi) and R-IN (semi) exhibit strong performance: 94.2 and 94.3. We can combine the samples of IN (semi) and R-IN (semi) and rerank them with a reranker (94.5) and five rerankers (94.7). Converting to dependenc,

models, we get 94.6 LAS and 96.4 UAS, which is within the range of human interannotator agreement.7

4.6 Out-of-domain Evaluation

The system of two parsers and five rerankers achieves 88.9 on BNC (Foster and van Genabith, 2008), 90.6 on Brown (Francis and Kučera, 1989), 80.8 on GENIA (Kim et al., 2003), 81.7 on Switchboard (Godfrey et al., 1992) and 93.1 on QuestionBank (Judge et al., 2006). This matches or substantially improves upon the best results for these datasets (see the Appendix for more details).

5 Conclusion

We have shown simple sequential models achieve the state-of-the-art parsing performance on the Penn Treebank. Using in-order traversals dramatically reduces the search space allowing the parser to parse accurately in linear-time without pre-training, an external POS tagger, or hand-engineered features. Using semi-supervised training and reranking, our accuracies are state-of-the-art on six different domains.

---

6Stanford Dependencies, version 3.3.0
7research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html
References


