An Empirical Study of Building a Strong Baseline for Constituency Parsing

Jun Suzuki, Sho Takase, Hidetaka Kamigaito, Makoto Morishita, and Masaaki Nagata
NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0237 Japan
{suzuki.jun, takase.sho, kamigaito.hidetaka, morishita.makoto, nagata.masaaki}@lab.ntt.co.jp

Abstract

This paper investigates the construction of a strong baseline based on general purpose sequence-to-sequence models for constituency parsing. We incorporate several techniques that were mainly developed in natural language generation tasks, e.g., machine translation and summarization, and demonstrate that the sequence-to-sequence model achieves the current top-notch parsers’ performance without requiring explicit task-specific knowledge or architecture of constituent parsing.

1 Introduction

Sequence-to-sequence (Seq2seq) models have successfully improved many well-studied NLP tasks, especially for natural language generation (NLG) tasks, such as machine translation (MT) (Sutskever et al., 2014; Cho et al., 2014) and abstractive summarization (Rush et al., 2015). Seq2seq models have also been applied to constituency parsing (Vinyals et al., 2015) and provided a fairly good result. However one obvious, intuitive drawback of Seq2seq models when they are applied to constituency parsing is that they have no explicit architecture to model latent nested relationships among the words and phrases in constituency parse trees. Thus, models that directly model them, such as RNNG (Dyer et al., 2016), are an intuitively more promising approach. In fact, RNNG and its extensions (Kuncoro et al., 2017; Fried et al., 2017) provide the current state-of-the-art performance. Seq2seq models are currently considered a simple baseline of neural-based constituency parsing.

After the first proposal of an Seq2seq constituency parser, many task-independent techniques have been developed, mainly in the NLG research area. Our aim is to update the Seq2seq approach proposed in Vinyals et al. (2015) as a stronger baseline of constituency parsing. Our motivation is basically identical to that described in Denkowski and Neubig (2017). A strong baseline is crucial for reporting reliable experimental results. It offers a fair evaluation of promising new techniques if they solve new issues or simply resolve issues that have already been addressed by current generic technology. More specifically, it might become possible to analyze what types of implicit linguistic structures are easier or harder to capture for neural models by comparing the outputs of strong Seq2seq models and task-specific models, e.g., RNNG.

The contributions of this paper are summarized as follows: (1) a strong baseline for constituency parsing based on general purpose Seq2seq models, (2) an empirical investigation of several generic techniques that can (or cannot) contribute to improve the parser performance, (3) empirical evidence that Seq2seq models implicitly learn parse tree structures well without knowing task-specific and explicit tree structure information.

2 Constituency Parsing by Seq2seq

Our starting point is an RNN-based Seq2seq model with an attention mechanism that was applied to constituency parsing (Vinyals et al., 2015). We omit detailed descriptions due to space limitations, but note that our model architecture is identical to the one introduced in Luong et al. (2015a).

A key trick for applying Seq2seq models to constituency parsing is the linearization of parse

1 Our code and experimental configurations for reproducing our experiments are publicly available: https://github.com/nttcslab-nlp/strong_s2s_baseline_parser

2 More specifically, our Seq2seq model follows the one implemented in seq2seq-attn (https://github.com/harvardnlp/seq2seq-attn), which is the alpha-version of the OpenNMT tool (http://opennmt.net).
trees (Vinyals et al., 2015). Roughly speaking, a **linearized parse tree** consists of open, close bracketing and POS-tags that correspond to a given input raw sentence. Since a one-to-one mapping exists between a parse tree and its linearized form (if the linearized form is a valid tree), we can recover parse trees from the predicted linearized parse tree. Vinyals et al. (2015) also introduced the part-of-speech (POS) tag normalization technique. They substituted each POS tag in a linearized parse tree to a single XX-tag\(^3\), which allows Seq2seq models to achieve a more competitive performance range than the current state-of-the-art parses\(^4\).

Table 1: Examples of linearization and POS-tag normalization (Vinyals et al., 2015)

<table>
<thead>
<tr>
<th>Original input</th>
<th>S-exp.</th>
<th>Linearized form with POS normalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>John has a dog</td>
<td>S (NP NNP) (VP VBZ (NP DT NN) )</td>
<td>(s (t: NNP NNP) (t: VP VBZ (t: NP DT NN) ) )</td>
</tr>
</tbody>
</table>

Table 2: List of notations used in this paper.

<table>
<thead>
<tr>
<th>$D$</th>
<th>dimension of the embeddings</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H$</td>
<td>dimension of the hidden states</td>
</tr>
<tr>
<td>$i$</td>
<td>index of the (token) position in input sentence</td>
</tr>
<tr>
<td>$j$</td>
<td>index of the (token) position in output linearized format of parse tree</td>
</tr>
<tr>
<td>$\psi$</td>
<td>vocabulary of word for input (encoder side)</td>
</tr>
<tr>
<td>$\psi^{*}$</td>
<td>vocabulary of word for input (encoder side)</td>
</tr>
<tr>
<td>$E$</td>
<td>encoder embedding matrix for $\psi^{*}$, where $E \in \mathbb{R}^{D \times</td>
</tr>
<tr>
<td>$F$</td>
<td>encoder embedding matrix for $\psi$, where $F \in \mathbb{R}^{D \times</td>
</tr>
<tr>
<td>$u_i$</td>
<td>$i$-th word (token) in the input sentence, $u_i \in \psi^{*}$</td>
</tr>
<tr>
<td>$x_i$</td>
<td>one-hot vector representation of the $i$-th word in $\psi$</td>
</tr>
<tr>
<td>$e_i$</td>
<td>encoder embedding vector at position $i$ in encoder</td>
</tr>
<tr>
<td>$q_i$</td>
<td>output scores of auxiliary task at decoder position $j$</td>
</tr>
<tr>
<td>$a_j$</td>
<td>final decoder output scores at decoder position $j$</td>
</tr>
<tr>
<td>$s_k$</td>
<td>one-hot vector representation of the $k$-th subword in $\psi$</td>
</tr>
<tr>
<td>$e_i$</td>
<td>encoder embedding vector of unknown token</td>
</tr>
<tr>
<td>$\phi$</td>
<td>function that returns the index of given word in the vocabulary $\psi$</td>
</tr>
<tr>
<td>$\psi^{n}$</td>
<td>vocabulary of output with POS-tag normalization</td>
</tr>
<tr>
<td>$\psi^{e}$</td>
<td>vocabulary of output without POS-tag normalization</td>
</tr>
<tr>
<td>$W^{d}$</td>
<td>decoder output matrix for $\psi^{e}$, where $W^{d} \in \mathbb{R}^{D \times</td>
</tr>
<tr>
<td>$W^{n}$</td>
<td>decoder output matrix for $\psi^{n}$, where $W^{n} \in \mathbb{R}^{D \times</td>
</tr>
</tbody>
</table>

Table 2: List of notations used in this paper.

Where $k = \phi(w_i)$.

Note that the second term of RHS indicates our additional subword features, and the first represents the standard word embedding extraction procedure. Among several choices, we used the byte-pair encoding (BPE) approach proposed in Sennrich et al. (2016) applying 1,000 merge operations\(^6\).

### 3.2 Unknown token embedding as a bias

We generally replace rare words, e.g., those appearing less than five times in the training data, with unknown tokens in the Seq2seq approach. However, we suspect that embedding vectors, which correspond to unknown tokens, cannot be trained well for the following reasons: (1) the occurrence of unknown tokens remains relatively small in the training data since they are obvious replacements for rare words, and (2) Seq2seq is relatively ineffective for training infrequent words (Luong et al., 2015b). Based on these observations, we utilize the unknown embedding as a bias term $b$ of linear layer ($W x + b$) when obtaining every encoder embeddings for overcoming infrequent word problem. Then, we modify Eq. 2 as follows:

$$e_i = (E x_k + u) + \sum_{k' \in \psi(u_i)} (F s_{k'} + u).$$

Note that if $u_i$ is unknown token, then Eq. 2 becomes

$$e_i = 2u + \sum_{k' \in \psi(u_i)} (F s_{k'} + u).$$

---

\(^{3}\)We did not substitute POS-tags for punctuation symbols such as “.”, and “,”.

\(^{4}\)Several recently developed neural-based constituency parsers ignore POS tags since they are not evaluated in the standard evaluation metric of constituency parsing (Bracketing F-measure).

\(^{5}\)Figure in the supplementary material shows the brief sketch of the method explained in the following section.

\(^{6}\)https://github.com/rsennrich/subword-nmt
3.3 Multi-task learning

Several papers on the Seq2seq approach (Luong et al., 2016) have reported that the multi-task learning extension often improves the task performance if we can find effective auxiliary tasks related to the target task. From this general knowledge, we re-consider jointly estimating POS-tags by incorporating the linearized forms without the POS-tag normalization as an auxiliary task. In detail, the linearized forms with and without the POS-tag normalization are independently and simultaneously estimated as \( o_j \) and \( q_j \), respectively, in the decoder output layer by following equation:

\[
    o_j = W^{(o)}z_j, \quad \text{and} \quad q_j = W^{(q)}z_j. \tag{3}
\]

3.4 Output length controlling

As described in Vinyals et al. (2015), not all the outputs (predicted linearized parse trees) obtained from the Seq2seq parser are valid (well-formed) as a parse tree. Toward guaranteeing that every output is a valid tree, we introduce a simple extension of the method for controlling the Seq2seq output length (Kikuchi et al., 2016).

First, we introduce an additional bias term \( b \) in the decoder output layer to prevent the selection of certain output words:

\[
    p_j = \text{softmax}(o_j + b). \tag{4}
\]

If we set a large negative value at the \( m \)-th element in \( b \), namely \( b_m \approx -\infty \), then the \( m \)-th element in \( p_j \) becomes approximately 0, namely \( p_{jm} \approx 0 \), regardless of the value of the \( k \)-th element in \( o_j \). We refer to this operation to set value \(-\infty \) in \( b \) as a mask. Since this naive masking approach is harmless to GPU-friendly processing, we can still exploit GPU parallelization.

We set \( b \) to always mask the EOS-tag and change \( b \) when at least one of the following conditions is satisfied: (1) if the number of open and closed brackets generated so far is the same, then we mask the XX-tags (or the POS-tags) and all the closed brackets. (2) if the number of predicted XX-tags (or POS-tags) is equivalent to that of the words in a given input sentence, then we mask the XX-tags (or all the POS-tags) and all the open brackets. If both conditions (1) and (2) are satisfied, then the decoding process is finished. The additional cost for masking the control is to count the number of XX-tags and the open and closed brackets so far generated in the decoding process.

3.5 Pre-trained word embeddings

The pre-trained word embeddings obtained from a large external corpora often boost the final task performance even if they only initialize the input embedding layer. In constituency parsing, several systems also incorporate pre-trained word embeddings, such as Vinyals et al. (2015); Durrett and Klein (2015). To maintain as much reproducibility of our experiments as possible, we simply applied publicly available pre-trained word embeddings, i.e., \texttt{glove.840B.300d}³, as initial values of the encoder embedding layer.

3.6 Model ensemble

Ensembling several independently trained models together significantly improves many NLP tasks. In the ensembling process, we predict the output tokens using the arithmetic mean of predicted probabilities computed by each model:

\[
    p_j = \frac{1}{A} \sum_{a=1}^{A} p_j^{(a)}, \tag{5}
\]

where \( p_j^{(a)} \) represents the probability distribution at position \( j \) predicted by the \( a \)-th model.

3.7 Language model (LM) reranking

Choe and Charniak (2016) demonstrated that reranking the predicted parser output candidates with an RNN language model (LM) significantly improves performance. We refer to this reranking process as \( \text{LM-rerank} \). Following their success, we also trained RNN-LMs on the PTB dataset with their published preprocessing code⁸ to reproduce the experiments in Choe and Charniak (2016) for our LM-reranker. We selected the current state-of-the-art LM (Yang et al., 2018)⁹ as our LM-reranker, which is a much stronger LM than was used in Choe and Charniak (2016).

³https://nlp.stanford.edu/projects/glove/
⁹We used the identical hyper-parameters introduced in their site: https://github.com/zhangdai/nos.

---

<table>
<thead>
<tr>
<th>Dim. of embedding ( D )</th>
<th>300</th>
<th>Dim. of hidden state ( H )</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder RNN unit</td>
<td>bi-LSTM</td>
<td>Num. of layers ( L )</td>
<td>2</td>
</tr>
<tr>
<td>Decoder RNN unit</td>
<td>LSTM with attention</td>
<td>Dropout rate</td>
<td>0.3</td>
</tr>
<tr>
<td>Learning rate decay</td>
<td>0.9 (after 50 epoch)</td>
<td>Gradient clipping ( \zeta )</td>
<td>1.0</td>
</tr>
<tr>
<td>Mini-batch size ( M )</td>
<td>16 (shuffled at each epoch)</td>
<td>Initial learning rate</td>
<td>1.0</td>
</tr>
<tr>
<td>Stopping criterion</td>
<td>100 epochs (w/o early stopping)</td>
<td>Beam size at Test ( H )</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3: List of model and optimization configurations (hyper-parameters) in our experiments.
Table 4: Results on English PTB data: Results were average (ave), worst (min), and best (max) performance of ten models independently trained with distinct random initial values. Test data was only evaluated on baseline and our best setting ((a), (e), (f) and (j)) to prevent over-tuning to the test data. We confirmed that all our results contained no malformed parse trees.

Table 5: Ensembling and reranking results

Table 6: Impact of hyper-parameter selections. We only evaluated the development data (PTB Sec. 22) to prevent over-tuning to the test data.

4 Experiments

Our experiments used the English Penn Treebank data (Marcus et al., 1994), which are the most widely used benchmark data in the literature. We used the standard split of training (Sec.02–21), development (Sec.22), and test data (Sec.23) and strictly followed the instructions for the evaluation settings explained in Vinyals et al. (2015). For data pre-processing, all the parse trees were transformed into linearized forms, which include standard UNK replacement for OOV words and POS-tag normalization by XX-tags. As explained in Vinyals et al. (2015), we did not apply any parse tree binarization or special unary treatment, which were used as common techniques in the literature.

Table 3 summarizes the model configurations and the optimization settings used in our experiments unless otherwise specified.

4.1 Results

Table 4 shows the main results of our experiments. We reported the Bracketing F-measures (Bra.F) and the complete match scores (CM) evaluated by the EVALB tool.10 The averages (ave), standard deviations (stdev), lowest (min), and highest (max) scores were calculated from ten independent runs of each setting trained with different random initialization values. This table empirically reveals the effectiveness of individual techniques. Each technique gradually improved the performance, and the best result (j) achieved approx. 

10http://nlp.cs.nyu.edu/evalb/
proximately 3 point gain from the baseline conventional Seq2seq model (a) on test data Bra.F.

One drawback of Seq2seq approach is that it seems sensitive to initialization. Comparing only with a single result for each setting may produce inaccurate conclusions. Therefore, we should evaluate the performances over several trials to improve the evaluation reliability.

The baseline Seq2seq models, (a) and (f), produced the malformed parse trees. We post-processed such malformed parse trees by simple rules introduced in (Vinyals et al., 2015). On the other hand, we confirmed that all the results applying the technique explained in Sec. 3.4 produced no malformed parse trees.

**Ensembling and Reranking:** Table 5 shows the results of our models with model ensembling and LM-reranking. For ensemble, we randomly selected eight of the ten Seq2seq models reported in Table 4. For LM-reranking, we first generated 80 candidates by the above eight ensemble models and selected the best parse tree for each input in terms of the LM-reranker. The results in Table 5 were taken from a single-shot evaluation, unlike the averages of ten independent runs in Table 4.

**Hyper-parameter selection:** We empirically investigated the impact of the hyper-parameter selections. Table 6 shows the results. The following observations appear informative for building strong baseline systems: (1) Smaller mini-batch size \(M\) and gradient clipping \(G\) provided the better performance. Such settings lead to slower and longer training, but higher performance. (2) Larger layer size, hidden state dimension, and beam size have little impact on the performance; our setting, \(L = 2\), \(H = 200\), and \(B = 5\) looks adequate in terms of speed/performance trade-off.

**Input unit selection:** As often demonstrated in the NMT literature, using subword split as input token unit instead of standard tokenized word unit has potential to improve the performance. Table 6 (e) shows the results of utilizing subword splits. Clearly, \(8K\) and \(16K\) subword splits as input token units significantly degraded the performance. It seems that the numbers of XX-tags in output and tokens in input should keep consistent for better performance since Seq2seq models look to somehow learn such relationship, and used it during the decoding. Thus, using subword information as features is one promising approach for leveraging subword information into constituency parsing.

### 4.2 Comparison to current top systems

Table 7 lists the reported constituency parsing scores on PTB that were recently published in the literature. We split the results into three categories. The first category (top row) contains the results of the methods that were trained only from the pre-defined training data (PTB Sec.02–21), without any additional resources. The second category (middle row) consists of the results of methods that were trained from the pre-defined PTB training data as well as those listed in the top row, but incorporating word embeddings obtained from a large-scale external corpus to initialize the encoder embedding layer. The third category (bottom row) shows the performance of the methods that were trained using high-confidence, auto-parsed trees in addition to the pre-defined PTB training data.

Our Seq2seq approach successfully achieved the competitive level as the current top-notch methods: RNNG and its variants. Note here that, as described in Dyer et al. (2016), RNNG uses Berkeley parser’s mapping rules for effectively handling singleton words in the training corpus. In contrast, we demonstrated that Seq2seq models have enough power to achieve a competitive state-of-the-art performance without leveraging such task-dependent knowledge. Moreover, they need no explicit information of parse tree structures, transition states, stacks, (Stanford or Berkeley) mapping rules, or external silver training data during the model training except general purpose word embeddings as initial values. These observations from our experiments imply that recently developed Seq2seq models have enough ability to implicitly learn parsing structures from linearized parse trees. Our results argue that Seq2seq models can be a strong baseline for constituency parsing.

### 5 Conclusion

This paper investigated how well general purpose Seq2seq models can achieve the higher performance of constituency parsing as a strong baseline method. We incorporated several generic techniques to enhance Seq2seq models, such as incorporating subword features, and output length controlling. We experimentally demonstrated that by applying ensemble and LM-reranking techniques, a general purpose Seq2seq model achieved almost the same performance level as the state-of-the-art constituency parser without any task-specific or explicit tree structure information.
References


