Efficient Constituency Parsing by Pointing

Thanh-Tung Nguyen[†¶], **Xuan-Phi Nguyen**[†¶], **Shafiq Joty**[¶§], **Xiaoli Li**[†]
[¶]Nanyang Technological University
[§]Salesforce Research Asia
[†]Institute for Infocomm Research, A-STAR
Singapore
{ng0155ng@e.;nguyenxu002@e.;srjoty@}ntu.edu.sg
xlli@i2r.a-star.edu.sg

Abstract

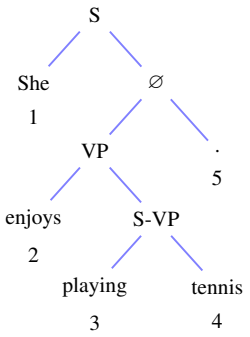
We propose a novel constituency parsing model that casts the parsing problem into a series of pointing tasks. Specifically, our model estimates the likelihood of a span being a legitimate tree constituent via the pointing score corresponding to the boundary words of the span. Our parsing model supports efficient top-down decoding and our learning objective is able to enforce structural consistency without resorting to the expensive CKY inference. The experiments on the standard English Penn Treebank parsing task show that our method achieves 92.78 F1 without using pre-trained models, which is higher than all the existing methods with similar time complexity. Using pre-trained BERT, our model achieves 95.48 F1, which is competitive with the state-of-the-art while being faster. Our approach also establishes new state-of-the-art in Basque and Swedish in the SPMRL shared tasks on multilingual constituency parsing.

1 Introduction

Constituency or phrase structure parsing is a core task in natural language processing (NLP) with myriad downstream applications. Therefore, devising effective and efficient algorithms for parsing has been a key focus in NLP.

With the advancements in neural approaches, various neural architectures have been proposed for constituency parsing as they are able to effectively encode the input tokens into dense vector representations while modeling the structural dependencies between tokens in a sentence. These include recurrent networks (Dyer et al., 2016; Stern et al., 2017b) and more recently self-attentive networks (Kitaev and Klein, 2018).

The parsing methods can be broadly distinguished based on whether they employ a greedy transition-based algorithm or a globally optimized chart parsing algorithm. The transition-based parsers (Dyer et al., 2016; Cross and Huang, 2016; Liu and Zhang, 2017) generate trees autoregressively as a form of shift-reduce decisions. Though computationally attractive, the local decisions made at each step may propagate errors to subsequent steps which would suffer from exposure bias.



Span Representation
$\mathcal{S}(T) = \{((1, 5), \text{S}), ((2, 5), \varnothing), ((2, 4), \text{VP}), ((3, 4), \text{S-VP})\}$

Pointing Representation
$\mathcal{P}(T) = \{(1 \rightarrow 5, \text{S}), (2 \rightarrow 5, \varnothing), (3 \rightarrow 4, \text{S-VP}), (4 \rightarrow 2, \text{VP}), (5 \rightarrow 1, \text{S})\}$

Figure 1: A binarized constituency tree for the sentence "She enjoys playing tennis.". The node *S-VP* is an example of a collapsed atomic label. We omit POS tags and singleton spans for simplicity. Below the tree, we show span and pointing representations of the tree.

Chart parsing methods, on the other hand, learn scoring functions for subtrees and perform global search over all possible trees to find the most probable tree for a sentence (Durrett and Klein, 2015; Gaddy et al., 2018; Kitaev and Klein, 2018; Kitaev et al., 2019). In this way, these methods can ensure consistency in predicting structured output. The limitation, however, is that they run slowly at $\mathcal{O}(n^3)$ or higher time complexity.

In this paper, we propose a novel parsing approach that casts constituency parsing into a series of pointing problems (Figure 1). Specifically,

our parsing model estimates the pointing score from one word to another in the input sentence, which represents the likelihood of the span covering those words being a legitimate phrase structure (*i.e.,* a subtree in the constituency tree). During training, the likelihoods of legitimate spans are maximized using the cross entropy loss. This enables our model to enforce structural consistency, while avoiding the use of structured loss that requires expensive $\mathcal{O}(n^3)$ CKY inference (Gaddy et al., 2018; Kitaev and Klein, 2018). The training in our model can be fully parallelized without requiring structured inference as in (Shen et al., 2018; Gómez and Vilares, 2018). Our pointing mechanism also allows efficient top-down decoding with a best and worse case running time of $\mathcal{O}(n \log n)$ and $\mathcal{O}(n^2)$, respectively.

In the experiments with English Penn Treebank parsing, our model without any pre-training achieves 92.78 F1, outperforming all existing methods with similar time complexity. With pre-trained BERT (Devlin et al., 2019), our model pushes the F1 score to 95.48, which is on par with the state-of-the-art (Kitaev et al., 2019), while supporting faster decoding. Our model also performs competitively on the multilingual parsing tasks in the SPMRL 2013/2014 shared tasks and establishes new state-of-the-art in Basque and Swedish. We will release our code at https://ntunlpsg.github.io/project/parser/ptr-constituency-parser

2 Model

Similar to Stern et al. (2017a), we view constituency parsing as the problem of finding a set of labeled spans over the input sentence. Let $\mathcal{S}(T)$ denote the set of *labeled spans* for a parse tree T. Formally, $\mathcal{S}(T)$ can be expressed as

$$\mathcal{S}(T) := \{((i_t, j_t), l_t)\}_{t=1}^{|\mathcal{S}(T)|} \text{ for } i_t < j_t \quad (1)$$

where $|\mathcal{S}(T)|$ is the number of spans in the tree. Figure 1 shows an example constituency tree and its corresponding labeled span representation.

Following the standard practice in parsing (Gaddy et al., 2018; Shen et al., 2018), we convert the n-ary tree into a binary form and introduce a dummy label $\varnothing$ to spans that are not constituents in the original tree but created as a result of binarization. Similarly, the labels in unary chains corresponding to nested labeled spans are collapsed into unique atomic labels, such as S-VP in Fig. 1.

Although our method shares the same "span-based" view with that of Stern et al. (2017a), our approach diverges significantly from their framework in the way we treat the whole parsing problem, and the representation and modeling of the spans, as we describe below.

2.1 Parsing as Pointing

In contrast to previous approaches, we cast parsing as a series of pointing decisions. For each index i in the input sequence, the parsing model points it to another index p_i in order to identify the tree span (i, p_i), where $i \neq p_i$. Similar to Pointer Networks (Vinyals et al., 2015a), each pointing mechanism is modeled as a multinomial distribution over the indices of the input tokens (or encoder states). However, unlike the original pointer network where a decoder state points to an encoder state, in our approach, every encoder state $\boldsymbol{h}_i$ points to another encoder state $\boldsymbol{h}_{p_i}$.

In this paper, we generally use $x \curvearrowright y$ to mean x points to y. We will refer to the pointing operation either as a function of the encoder states (*e.g.,* $\boldsymbol{h}_i \curvearrowright \boldsymbol{h}_{p_i}$) or simply the corresponding indices (*e.g.,* $i \curvearrowright p_i$). They both mean the same operation where the pointing function takes the encoder state $\boldsymbol{h}_i$ as the query vector and points to $\boldsymbol{h}_{p_i}$ by computing an attention distribution over all the encoder states.

Let $\mathcal{P}(T)$ denote the set of pointing decisions derived from a tree T by a transformation $\mathcal{H}$, *i.e.,* $\mathcal{H} : T \rightarrow \mathcal{P}(T)$. For the parsing process to be valid, the transformation $\mathcal{H}$ and its inverse $\mathcal{H}'$ which transforms $\mathcal{P}(T)$ back to T, should both have a one-to-one mapping property. Otherwise, the parsing model may confuse two different parse trees with the same pointing representation. In this paper, we propose a novel transformation that satisfies this property, as defined by the following proposition (proof provided in the Appendix).

Proposition 1 *Given a binary constituency tree T for a sentence containing n tokens, the transformation $\mathcal{H}$ converts it into a set of pointing decisions $\mathcal{P}(T) = \{(i \curvearrowright p_i, l_i) : i = 1, \ldots, n-1; i \neq p_i\}$ such that $(\min(i, p_i), \max(i, p_i))$ is the **largest** span that starts or ends at i, and l_i is the label of the nonterminal associated with the span.*

To elaborate further, each pointing decision in $\mathcal{P}(T)$ represents a specific span in $\mathcal{S}(T)$. The pointing $i \curvearrowright p_i$ is directional, while the span that it represents (i', j') is non-directional. In other words, there may exist position i such that $i > p_i$,

Algorithm 1 Convert binary tree to Pointing

Input: Binary tree T and its span representation $\mathcal{S}(T)$
Output: Pointing representation $\mathcal{P}(T)$

```
P(T) = []                                  ▷Empty pointing list
for each leaf_i in T do
    node ← leaf_i
    (x, y) ← (i, i)                        ▷Initialize current span, x ≤ y
    l_i ← ∅                                ▷Initialize label of current span
    while x = i or y = i do
        p_i ← x + y − i
        l_i ← node.label                   ▷The span's label
        node ← node.parent
        (x, y) ← node.span                 ▷Span covered by node
    end while                              ▷Until i is no longer start/end point
    push(P(T), (i→p_i, l_i))
end for
return P(T)
```

while $i' < j'\ \forall i', j' \in [1, n]$. In fact, it is easy to see that if the token at index i is a left-child of a subtree, the largest span involving i starts at i, and in this case $i < p_i$ and $i' = i, j' = p_i$. On the other hand, if the token is a right-child of a subtree, the respective largest span ends at position i, in which case $i > p_i$ and $i' = p_i, j' = i$ (*e.g.,* see $4 \rightarrow 2$ in Figure 1). In addition, as the spans in $\mathcal{S}(T)$ are unique, it can be shown that the pointing decisions in $\mathcal{P}(T)$ are also distinct from one another (see Appendix for a proof by contradiction).

Given such pointing formulation, for every constituency tree, there exists a trivial case $(1 \rightarrow n, l_1)$ where $p_1 = n$ and l_1 is generally 'S'. Thus, to make our formulation more general with n inputs and n outputs and convenient for the method description discussed later on, we add another trivial case $(n \rightarrow 1, l_1)$. With this generalization, we can represent the pointing decisions of any binary constituency tree T as:

$$\mathcal{P}(T) = \{(i \rightarrow p_i, l_i) : i = 1, \ldots, n; i \neq p_i\} \quad (2)$$

The pointing representation of the tree in Figure 1 is given at the bottom of the figure. To illustrate, in the parse tree, the largest phrase that starts or ends at token 2 ('enjoys') is the subtree rooted at '$\varnothing$', which spans from 2 to 5. In this case, the span *starts* at token 2. Similarly, the largest phrase that starts or ends at token 4 ('tennis') is the span "enjoys playing tennis", which is rooted at 'VP'. In this case, the span *ends* at token 4.

Algorithm 1 describes the procedure to convert a binary tree to its corresponding pointing representation. Specifically, from each leaf token i, the algorithm traverses upward along the hierarchy until the non-terminal node that does not start or end with i. In this way, the largest span starting or ending with i can be identified.

2.2 Top-Down Tree Inference

In the previous section, we described how to convert a constituency tree T into a sequence of pointing decisions $\mathcal{P}(T)$. We use this transformation to train the parsing model (described in detail in Sections 2.3 - 2.4). During inference, given a sentence to parse, our decoder with the help of the parsing model predicts $\mathcal{P}(T)$, from which we can construct the tree T. However, not all sets of pointings $\mathcal{P}(T)$ guarantee the generation of a valid tree. For example, for a sentence with four (4) tokens, the pointing $\mathcal{P}(T) = \{(1 \rightarrow 4, l_1), (2 \rightarrow 3, l_2), (3 \rightarrow 4, l_3), (4 \rightarrow 1, l_1)\}$ does not generate a valid tree because token '3' cannot belong to both spans $(2, 3)$ and $(3, 4)$. In other words, simply taking the $\arg\max$ over the pointing distributions may not generate a valid tree.

Our approach to decoding is inspired by the span-based approach of Stern et al. (2017a). In particular, to reduce the search space, we score for span identification (given by the pointing function) and label assignment separately.

Span Identification. We adopt a top-down greedy approach formulated as follows.

$$k^* = \arg\max_k s_{\text{split}}(i, k, j) \quad (3)$$

where $s_{\text{split}}(i, k, j)$ is the score of having a split-point at position k ($i \leq k < j$), as defined by the following equation.

$$s_{\text{split}}(i, k, j) = \rho(k \rightarrow i) + \rho(k{+}1 \rightarrow j) \quad (4)$$

where $\rho(k \rightarrow i)$ and $\rho(k+1 \rightarrow j)$ are the pointing scores (probabilities) for spans (i, k) and $(k{+}1, j)$, respectively. Note that the pointing scores are *asymmetric*, meaning that $\rho(i \rightarrow j)$ may not be equal to $\rho(j \rightarrow i)$, because pointing from i to j is different from pointing from j to i. This is different from previous approaches, where the score of a span is defined to be symmetric. We build a tree for the input sentence by computing Eq. 3 recursively starting from the full sentence span $(1, n)$.

In the general case when $i < k < j - 1$, our pointing-based parsing model should learn to assign high scores to the two spans (i, k) and $(k + 1, j)$, or equivalently the pointing decisions $k \rightarrow i$ and $k{+}1 \rightarrow j$. However, the pointing formulation described so far omits the trivial *self-pointing*

decisions, which represent the *singleton spans*. A singleton span is only created when the splitting decision splits an n-size span into a single-token span (singleton span) and a sub-span of size $n-1$, *i.e.,* when $k = i$ or $k = j-1$. For instance, for the parsing process in Figure 2a, the splitting decision at the root span $(1, 5)$ results in a singleton span $(1, 1)$ and a general span $(2, 5)$. For this splitting decision, Eq. 3 requires the scores of $(1, 1)$ and $(2, 5)$. However, the set of pointing decisions $\mathcal{P}(T)$ does not cover the pointing for $(1, 1)$. This discrepancy can be resolved by modeling the singleton spans separately. To achieve that, we redefine Eq. 3 as follows:

$$s_{\text{split}}(i, k, j) = \begin{cases} sp(i \rightarrow i) + gp(i{+}1 \rightarrow j) & \text{if } k = i \\ gp(j{-}1 \rightarrow i) + sp(j \rightarrow j) & \text{if } k = j-1 \\ gp(k \rightarrow i) + gp(k{+}1 \rightarrow j) & \text{otherwise} \end{cases} \quad (5)$$

where sp and gp respectively represent the scores for the singleton and general pointing functions (to be defined formally in Section 2.3).

Remark on structural consistency. It is important to note that since the pointing functions are defined to have a global structural property (*i.e.,* the largest span that starts/ends with i), our model inherently enforces structural consistency. The pointing formulation of the parsing problem also makes the training process simple and efficient; it allows us to train the model effectively with simple cross entropy loss (see Section 2.4).

Label Assignment. Label assignment of spans is performed after every split decision. Specifically, as we split a span (i, j) into two sub-spans (i, k) and $(k{+}1, j)$ which corresponds to the pointing functions of $k \rightarrow i$ and $k{+}1 \rightarrow j$, we perform the label assignments for the two new sub-spans as

$$l_k = \arg\max_{l \in L} gc(l|k)$$
$$l_{k+1} = \arg\max_{l \in L} gc(l|k+1) \quad (6)$$

where gc is the label classifier for any general (non-unary) span and L is the set of possible non-terminal labels. Following Shen et al. (2018), we use a separate classifier uc for determining the labels of the unary spans, *e.g.,* the first layer of labels NP, $\varnothing$, ..., NP, $\varnothing$) in Figure 2. Also, note that the label assignment is done based on only the query vector (the encoder state that is used to point).

Algorithm 2 Pointing parsing algorithm

Input: Sentence length n; pointing scores: $gp(i, j)$, $sp(i, j)$; label scores: $gc(l|i)$, $uc(l|i)$, $1 \le i \le j \le n, l \in L_g/L_u$
Output: Parse tree $\mathcal{T}$

```
Q = [(1, n)]                                   ▷queue of spans
S = [(1, n, argmax_l gc(l|1)]                  ▷general spans, labels
U = {((t, t), argmax_l uc(l|t))}_{t=1}^{n}     ▷unary spans, labels
while Q ≠ ∅ do
    (i, j) = pop(Q)
    if j ≤ i + 1 then
        Continue
    end if
    k* = argmax_{i≤k<j} s_split(i, k, j)       ▷using gp, sp
    if k = i then
        push(Q, (i + 1, j))
        push(S, (i + 1, j, argmax_l gc(l|i + 1)))
    else if k = j − 1 then
        push(Q, (i, j − 1))
        push(S, (i, j − 1, argmax_l gc(l|j − 1)))
    else
        push(Q, (i, k))
        push(Q, (k + 1, j))
        push(S, (i, k, argmax_l gc(l|k)))
        push(S, (k + 1, j, argmax_l gc(l|k + 1)))
    end if
end while
T = S ∪ U
```

Figure 2 illustrates the top-down parsing process for our running example. It consists of a sequence of pointing decisions (Figure 2a, top to bottom), which are then trivially converted to the parse tree (Figure 2b). We also provide the pseudocode in Algorithm 2. Specifically, the algorithm finds the best split for the current span (i, j) using the pointing scores and pushes the newly created sub-spans into the FIFO queue $\mathcal{Q}$. The process terminates when there are no more spans to be split. Similar to Stern et al. (2017a), our parsing algorithm has the worst and best case time complexities of $\mathcal{O}(n^2)$ and $\mathcal{O}(n \log n)$, respectively.

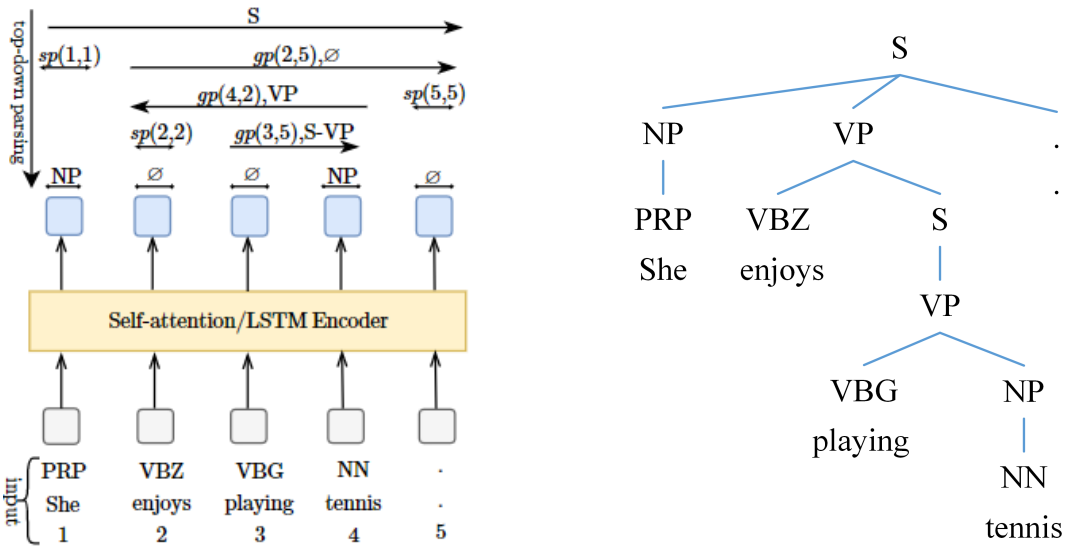
2.3 Model Architecture

We now describe the architecture of our parsing model: the sentence encoder, the pointing model and the labeling model.

Sentence Encoder. Given an input sequence of n words $\boldsymbol{X} = (x_1, \ldots, x_n)$, we first embed each word x_i to its respective vector representation e_i as:

$$\boldsymbol{e}_i = \boldsymbol{e}_i^{\text{char}} + \boldsymbol{e}_i^{\text{word}} + \boldsymbol{e}_i^{\text{pos}} \quad (7)$$

where $\boldsymbol{e}_i^{\text{char}}$, $\boldsymbol{e}_i^{\text{word}}$, $\boldsymbol{e}_i^{\text{pos}}$ are respectively the character, word, and part-of-speech (POS) embeddings of the word x_i. Following Kitaev and Klein (2018), we use a Character LSTM to compute the character embedding of a word. We experiment with both randomly initialized and pre-



(a) Execution of pointing parsing algorithm

(b) Output parse tree.

Figure 2: Inferring the parse tree for a given sentence and its part-of-speech (POS) tags (predicted by an external POS tagger). Starting with the full sentence span $(1, 5)$ and its label S, we predict split point 1 using the base (sp) and general (gp) pointing scores as per Eqn. 3-5. The left singleton span $(1, 1)$ is assigned with a label NP and the right span $(2, 5)$ is assigned with a label $\varnothing$ using the label classifier gc as per Eqn. 6. The recursion of splitting and labeling continues until the process reaches a terminal node. The label assignment for the unary spans is done by the uc classifier.

trained word embeddings. If pretrained embeddings are used, the word embedding $\boldsymbol{e}_i^{\text{word}}$ is the summation of the word's randomly-initialized embedding and the pretrained embedding. The POS embeddings ($\boldsymbol{e}_i^{\text{pos}}$) are randomly initialized.

The word representations ($\boldsymbol{e}_i$) are then passed to a neural network based sequence encoder to obtain their hidden representations. Since our method does not require any specific encoder, one may use any encoder model, such as Bi-LSTM (Hochreiter and Schmidhuber, 1997) or self-attentive encoder (Kitaev and Klein, 2018). In this paper, unless otherwise specified, we use the self-attentive encoder model as our main sequence encoder because of its efficiency with parallel computation. The model is factorized into content and position information in both the self-attention sub-layer and the feed-forward layer. Details about this factorization process is provided in Kitaev and Klein (2018).

Pointing and Labeling Models. The results of the aforementioned sequence encoding process are used to compute the pointing and labeling scores. More formally, the encoder network produces a sequence of n latent vectors $\boldsymbol{H} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n)$ for the input sequence $\boldsymbol{X} = (x_1, \ldots, x_n)$. After that, we apply four (4) separate position-wise two-layer Feed-Forward Networks (FFN), formulated as $\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$, to transform $\boldsymbol{H}$ into task-specific latent representations for the respective pointing and labeling tasks.

$$\boldsymbol{h}_i^{gp} = \text{FFN}_{gp}(\boldsymbol{h}_i); \quad \boldsymbol{h}_i^{sp} = \text{FFN}_{sp}(\boldsymbol{h}_i) \tag{8}$$

$$\boldsymbol{h}_i^{gc} = \text{FFN}_{gc}(\boldsymbol{h}_i); \quad \boldsymbol{h}_i^{uc} = \text{FFN}_{uc}(\boldsymbol{h}_i) \tag{9}$$

Note that there is no parameter sharing between FFN_{gp}, FFN_{sp}, FFN_{gc} and FFN_{uc}. The pointing functions are then modeled as the multinomial (or attention) distributions over the input indices for each input position i as follows.

$$gp(i, k) = \frac{\exp(\boldsymbol{h}_i^{gp}(\boldsymbol{h}_k^{gp})^T)}{\sum_{k=1}^{n} \exp(\boldsymbol{h}_i^{gp}(\boldsymbol{h}_k^{gp})^T)} \tag{10}$$

$$sp(i, k) = \frac{\exp(\boldsymbol{h}_i^{sp}(\boldsymbol{h}_k^{sp})^T)}{\sum_{k=1}^{n} \exp(\boldsymbol{h}_i^{sp}(\boldsymbol{h}_k^{sp})^T)} \tag{11}$$

For label assignment functions, we simply feed the label representations $\boldsymbol{H}^{gc} = (\boldsymbol{h}_1^{gc}, \ldots, \boldsymbol{h}_n^{gc})$ and $\boldsymbol{H}^{uc} = (\boldsymbol{h}_1^{uc}, \ldots, \boldsymbol{h}_n^{uc})$ into the respective softmax classification layers as follows.

$$gc(l|i) = \frac{\exp(\boldsymbol{h}_i^{gc}\boldsymbol{w}_l^{gc})}{\sum_{l=1}^{|L_g|} \exp(\boldsymbol{h}_i^{gc}\boldsymbol{w}_l^{gc})} \tag{12}$$

$$uc(l|i) = \frac{\exp(\boldsymbol{h}_i^{uc}\boldsymbol{w}_l^{uc})}{\sum_{l=1}^{|L_u|} \exp(\boldsymbol{h}_i^{uc}\boldsymbol{w}_l^{uc})} \tag{13}$$

where L_g and L_u are the set of possible labels for the general and unary spans respectively, $\boldsymbol{w}_l^{gc}$ and $\boldsymbol{w}_l^{uc}$ are the class-specific trainable weight vectors.

2.4 Training Objective

We train our parsing model by minimizing the total loss $\mathcal{L}_{total}(\theta)$ defined as:

$$\mathcal{L}_{\text{total}}(\theta) = \mathcal{L}_{gp}(\theta_e, \theta_{gp}) + \mathcal{L}_{sp}(\theta_e, \theta_{sp}) + \mathcal{L}_{gc}(\theta_e, \theta_{gc}) + \mathcal{L}_{uc}(\theta_e, \theta_{uc}) \quad (14)$$

where each individual loss is a cross entropy loss computed for the corresponding labeling or pointing task, and $\theta = \{\theta_e, \theta_{gp}, \theta_{sp}, \theta_{gc}, \theta_{uc}\}$ represents the overall model parameters; specifically, θ_e denotes the encoder parameters shared by all components, while $\theta_{gp}, \theta_{sp}, \theta_{gc}$ and θ_{uc} denote the separate parameters catering for the four pointing and labeling functions, gp, sp, gc and uc, respectively.

3 Experiments

To show the effectiveness of our approach, we conduct experiments on English and Multilingual parsing tasks. For English, we use the standard Wall Street Journal (WSJ) part of the Penn Treebank (PTB) (Marcus et al., 1993), whereas for multilingual, we experiment with seven (7) different languages from the SPMRL 2013-2014 shared task (Seddah et al., 2013): Basque, French, German, Hungarian, Korean, Polish and Swedish.

For evaluation on PTB, we report the standard labeled precision (LP), labeled recall (LR), and labelled F1 computed by `evalb`[1]. For the SPMRL datasets, we report labeled F1 and use the same setup in `evalb` as Kitaev and Klein (2018).

3.1 English (PTB) Experiments

Setup. We follow the standard train/valid/test split, which uses sections 2-21 for training, section 22 for development and section 23 for evaluation. This gives 45K sentences for training, 1,700 sentences for development, and 2,416 sentences for testing. Following previous studies, our model uses POS tags predicted by the Stanford tagger (Toutanova et al., 2003).

For our model, we adopt the self-attention encoder with similar hyperparameter details proposed by Kitaev and Klein (2018). The character embeddings are of 64 dimensions. For general and unary label classifiers (gc and uc), the hidden dimension of the specific position-wise feed-forward networks is 250, while those for pointing functions (gp and sp) have hidden dimensions of 1024. Our model is trained using the Adam optimizer (Kingma and Ba, 2015) with a batch size of 100 sentences. Additionally, we use 100 warm-up steps, within which we linearly increase the learning rate from 0 to the base learning rate of 0.008. Model selection for testing is performed based on the labeled F1 score on the validation set.

[1] http://nlp.cs.nyu.edu/evalb/

Model	LR	LP	F1
Top-Down Inference			
Stern et al. (2017a)	93.20	90.30	91.80
Shen et al. (2018)	92.00	91.70	91.80
Our Model	**92.81**	**92.75**	**92.78**
CKY/Chart Inference			
Gaddy et al. (2018)	-	-	92.10
Kitaev and Klein (2018)	93.20	93.90	93.55
Other Approaches			
Gómez and Vilares (2018)	-	-	90.7
Liu and Zhang (2017)	-	-	91.8
Stern et al. (2017b)	92.57	92.56	92.56
Zhou and Zhao (2019)	93.64	93.92	93.78

Table 1: Results for single models (no pre-training) on the PTB WSJ test set, Section 23.

Results for Single Models. The experimental results on PTB for the models without pre-training are shown in Table 1. As it can be seen, our model achieves an F1 of 92.78, the highest among the models using *top-down inference* strategies. Specifically, our method outperforms Stern et al. (2017a) and Shen et al. (2018) by about 1.0 point in F1.

On the other hand, while Kitaev and Klein (2018) and Zhou and Zhao (2019) achieve higher F1 score, their inference speed is significantly slower than ours because of the use of CKY based algorithms, which run at $\mathcal{O}(n^3)$ time complexity for Kitaev and Klein (2018) and $\mathcal{O}(n^5)$ for Zhou and Zhao (2019). Furthermore, their training objectives involve the use of structural hinge loss, which requires online CKY inference during training. This makes their training time considerably slower than that of our method, which is trained directly with span-wise cross entropy loss. In addition, Zhou and Zhao (2019) uses external supervision (*head* information) from the dependency parsing task. Dependency parsing models, in fact, have a strong resemblance to the pointing mechanism that our model employs (Ma et al., 2018). As

Model	F1
Our model $\text{BERT}_{\text{BASE-uncased}}$	95.34
Our model $\text{BERT}_{\text{LARGE-cased}}$	95.48
Kitaev and Klein (2018) ELMO	95.13
Kitaev et al. (2019) $\text{BERT}_{\text{LARGE-cased}}$	95.59

Table 2: Restuls on PTB WSJ test set with pretraining.

Model	# sents/sec
Petrov and Klein (2007)	6.2
Zhu et al. (2013)	89.5
Liu and Zhang (2017)	79.2
Stern et al. (2017a)	75.5
Kitaev and Klein (2018)	94.40
Shen et al. (2018)	111.1
Our model	130.2

Table 3: Parsing speed for different models computed on the PTB WSJ test set.

such, integrating dependency parsing information into our model may also be beneficial. We leave this for future work.

Results with Pre-training Similar to Kitaev and Klein (2018) and Kitaev et al. (2019), we also evaluate our models with BERT (Devlin et al., 2019) embeddings . Following them in the inclusion of contextualized token representations, we adjust the number of self-attentive layers to 2 and the base learning rate to 0.00005.

As shown in Table 2, our model achieves an F1 score of 95.48, which is on par with the state-of-the-art models. However, the advantage of our method is that it is faster than those methods. Specifically, our model runs at $\mathcal{O}(n^2)$ worst-case time complexity, while that of Kitaev et al. (2019) is $\mathcal{O}(n^3)$. Comparison on parsing speed is discussed in the following section.

Parsing Speed Comparison. In addition to parsing performance in F1 scores, we also compare our parser against the previous neural approaches in terms of parsing speed. We record the parsing timing over 2416 sentences of the PTB test set with batch size of 1, on a machine with NVIDIA GeForce GTX 1080Ti GPU and Intel(R) Xeon(R) Gold 6152 CPU. This setup is comparable to the setup of Shen et al. (2018).

As shown in Table 3, our parser outperforms Shen et al. (2018) by 19 more sentences per second, despite the fact that our parsing algorithm runs at $\mathcal{O}(n^2)$ worse-case time complexity while the one used by Shen et al. (2018) can theoretically run at $\mathcal{O}(n \log n)$ time complexity. To elaborate further, the algorithm presented in Shen et al. (2018) can only run at $\mathcal{O}(n^2)$ complexity. To achieve $\mathcal{O}(n \log n)$ complexity, it needs to sort the list of syntactic distances, which the provided code[2] does not implement. In addition, the speed up for our method can be attributed to the fact that our algorithm (see Algorithm 2) uses a *while loop*, while the algorithm of Shen et al. (2018) has many recursive function calls. Recursive algorithms tend to be less empirically efficient than their equivalent while/for loops in handling low-level memory allocations and function call stacks.

3.2 SPMRL Multilingual Experiments

Setup. Similar to the English PTB experiments, we use the predicted POS tags from external taggers (provided in the SPMRL datasets). The train/valid/test split is report in Table 6. For single model evaluation, we use the identical hyper-parameters and optimizer setups as in English PTB. For experiments with pre-trained models, we use the multilingual BERT (Devlin et al., 2019), which was trained jointly on 104 languages.

Results. The results for the single models are reported in Table 4. We see that our model achieves the highest F1 score in Basque and Swedish, which are higher than the baselines by 0.52 and 1.37 respective in F1. Our method also performs competitively with the previous state-of-the-art methods on other languages.

Table 5 reports the performance of the models using pre-trained BERT. Evidently, our method achieves state-of-the-art results in Basque and Swedish, and performs on par with the previous best method by Kitaev et al. (2019) in the other five languages. Again, note that our method is considerably faster and easier to train than the method of Kitaev et al. (2019).

4 Related Work

Prior to the neural tsunami in NLP, parsing methods typically model correlations in the *output space* through probabilistic context-free grammars (PCFGs) on top of sparse (and discrete) input representations either in a generative regime (Klein

[2] https://github.com/hantek/distance-parser

Model	Basque	French	German	Hungarian	Korean	Polish	Swedish
(Anders Bjorkelund and Szanto, 2014)	88.24	82.53	81.66	91.72	83.81	90.50	85.50
(Coavoux and Crabbé, 2017)	88.81	82.49	85.34	92.34	86.04	93.64	84.0
(Kitaev and Klein, 2018)	89.71	84.06	87.69	92.69	86.59	93.69	84.45
Our Model	90.23	82.20	84.91	91.07	85.36	93.99	86.87

Table 4: SPMRL experiment single model test.

Model	Basque	French	German	Hungarian	Korean	Polish	Swedish
(Kitaev et al., 2019)	91.63	87.43	90.20	94.90	88.80	96.36	88.86
Our model	92.02	86.69	90.28	94.24	88.71	96.14	89.10

Table 5: SPMRL experiment pre-trained model test (with pretraining).

Language	Train	Valid	Test
Basque	7,577	948	946
French	14,759	1,235	2,541
German	40,472	5,000	5,000
Hungarian	8,146	1,051	1,009
Korean	23,010	2,066	2,287
Polish	6,578	821	822
Swedish	5,000	494	666

Table 6: SPMRL Multilingual dataset split.

and Manning, 2003) or a discriminative regime (Finkel et al., 2008) or a combination of both (Charniak and Johnson, 2005).

Recently, however, with the advent of powerful neural encoders such as LSTMs (Hochreiter and Schmidhuber, 1997), the focus has been switched more towards effective modeling of correlations in the *input's latent space,* as the output structures are nothing but a function of the input (Gaddy et al., 2018). Various neural network models have been proposed to effectively encode the dense input representations and correlations, and have achieved state-of-the-art parsing results. To enforce the structural consistency, existing neural parsing methods either employ a transition-based algorithm (Dyer et al., 2016; Liu and Zhang, 2017) or a globally optimized chart-parsing algorithm (Gaddy et al., 2018; Kitaev and Klein, 2018).

Meanwhile, researchers also attempt to convert the constituency parsing problem into tasks that can be solved in alternative ways. For instance, Fernández-González and Martins (2015) transform the phrase structure into a special form of dependency structure. Such a dependency structure, however, requires certain corrections while converting back to the corresponding constituency tree. Gómez and Vilares (2018) and Shen et al. (2018) propose to map the constituency tree for a sentence of n tokens into a sequence of $n-1$ labels or scalars based on the depth or height of the lowest common ancestors between pairs of consecutive tokens. In addition, methods like (Vinyals et al., 2015b; Vaswani et al., 2017) apply the sequence-to-sequence framework to "translate" a sentence into the linearized form of its constituency tree. While being trivial and simple, parsers of this type do not guarantee structural correctness, because the syntax of the linearized form is not constrained during tree decoding.

Our approach differs from previous work in that it represents the constituency structure as a series of pointing representations and has a relatively simpler cross entropy based learning objective. The pointing representations can be computed in parallel, and can be efficiently converted into a full constituency tree using a top-down algorithm. Our pointing mechanism shares certain similarities with the Pointer Network (Vinyals et al., 2015a), but is distinct from it in that our method points a word to another word within the same encoded sequence.

5 Conclusion

We have presented a novel constituency parsing method that is based on a pointing mechanism. Our method utilizes an efficient top-down decoding algorithm that uses pointing functions for scoring possible spans. The pointing formulation inherently captures global structural properties and allows efficient training with cross entropy loss. With experiments we have shown that our method outperforms all existing top-down methods on the English Penn Treebank parsing task. Our method with pre-training rivals the state-of-

the-art method, while being faster than it. On multilingual constituency parsing, it also establishes new state-of-the-art in Basque and Swedish.

References

Agnieszka Falenska Richard Farkas Thomas Mueller Wolfgang Seeker Anders Bjorkelund, Ozlem Cetinoglu and Zsolt Szanto. 2014. The ims-wrocław-szeged-cis entry at the spmrl 2014 shared task: Reranking and morphosyntax meet unlabeled data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of NonCanonical Languages*, pages 97–102.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan. Association for Computational Linguistics.

Maximin Coavoux and Benoît Crabbé. 2017. Multilingual lexicalized constituency parsing with word-level auxiliary tasks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 331–336, Valencia, Spain. Association for Computational Linguistics.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1–11, Austin, Texas. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Daniel Fernández-González and André F. T. Martins. 2015. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China. Association for Computational Linguistics.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, Columbus, Ohio. Association for Computational Linguistics.

David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's going on in neural constituency parsers? an analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010, New Orleans, Louisiana. Association for Computational Linguistics.

Carlos Gómez, Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium. Association for Computational Linguistics.

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

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017. Shift-reduce constituent parsing with neural lookahead features. *Transactions of the Association for Computational Linguistics*, 5:45–58.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

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

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 404–411, Rochester, New York. Association for Computational Linguistics.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho D. Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clergerie. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics.

Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. 2018. Straight to the tree: Constituency parsing with neural syntactic distance. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1180, Melbourne, Australia. Association for Computational Linguistics.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017a. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 818–827.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017b. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1695–1700, Copenhagen, Denmark. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015a. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015b. Grammar as a foreign language. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2773–2781. Curran Associates, Inc.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on penn treebank. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2396–2408.

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