A Minimal Span-Based Neural Constituency Parser

Mitchell Stern **Jacob Andreas** **Dan Klein**
Computer Science Division
University of California, Berkeley
{mitchell,jda,klein}@cs.berkeley.edu

Abstract

In this work, we present a minimal neural model for constituency parsing based on independent scoring of labels and spans. We show that this model is not only compatible with classical dynamic programming techniques, but also admits a novel greedy top-down inference algorithm based on recursive partitioning of the input. We demonstrate empirically that both prediction schemes are competitive with recent work, and when combined with basic extensions to the scoring model are capable of achieving state-of-the-art single-model performance on the Penn Treebank (91.79 F1) and strong performance on the French Treebank (82.23 F1).

1 Introduction

This paper presents a minimal but surprisingly effective span-based neural model for constituency parsing. Recent years have seen a great deal of interest in parsing architectures that make use of recurrent neural network (RNN) representations of input sentences (Vinyals et al., 2015). Despite evidence that linear RNN decoders are implicitly able to respect some nontrivial well-formedness constraints on structured outputs (Graves, 2013), researchers have consistently found that the best performance is achieved by systems that explicitly require the decoder to generate well-formed tree structures (Chen and Manning, 2014).

There are two general approaches to ensuring this structural consistency. The most common is to encode the output as a sequence of operations within a transition system which constructs trees incrementally. This transforms the parsing problem back into a sequence-to-sequence problem, while making it easy to force the decoder to take only actions guaranteed to produce well-formed outputs. However, transition-based models do not admit fast dynamic programs and require careful feature engineering to support exact search-based inference (Thang et al., 2015). Moreover, models with recurrent state require complex training procedures to benefit from anything other than greedy decoding (Wiseman and Rush, 2016).

An alternative line of work focuses on *chart parsers*, which use log-linear or neural scoring potentials to parameterize a tree-structured dynamic program for maximization or marginalization (Finkel et al., 2008; Durrett and Klein, 2015). These models enjoy a number of appealing formal properties, including support for exact inference and structured loss functions. However, previous chart-based approaches have required considerable scaffolding beyond a simple well-formedness potential, e.g. pre-specification of a complete context-free grammar for generating output structures and initial pruning of the output space with a weaker model (Hall et al., 2014). Additionally, we are unaware of any recent chart-based models that achieve results competitive with the best transition-based models.

In this work, we present an extremely simple chart-based neural parser based on independent scoring of labels and spans, and show how this model can be adapted to support a greedy top-down decoding procedure. Our goal is to preserve the basic algorithmic properties of span-oriented (rather than transition-oriented) parse representations, while exploring the extent to which neural representational machinery can replace the additional structure required by existing chart parsers. On the Penn Treebank, our approach outperforms a number of recent models for chart-based and transition-based parsing—including the state-of-the-art models of Cross and Huang (2016) and Liu and Zhang (2016)—achieving an F1 score of 91.79. We additionally obtain a strong F1 score of 82.23 on the French Treebank.

2 Model

A constituency tree can be regarded as a collection of labeled spans over a sentence. Taking this view as a guiding principle, we propose a model with two components, one which assigns scores to span labels and one which assigns scores directly to span existence. The former is used to determine the labeling of the output, and the latter provides its structure.

At the core of both of these components is the issue of span representation. Given that a span's correct label and its quality as a constituent depend heavily on the context in which it appears, we naturally turn to recurrent neural networks as a starting point, since they have previously been shown to capture contextual information suitable for use in a variety of natural language applications (Bahdanau et al., 2014; Wang et al., 2015)

In particular, we run a bidirectional LSTM over the input to obtain context-sensitive forward and backward encodings for each position i, denoted by $\mathbf{f}_i$ and $\mathbf{b}_i$, respectively. Our representation of the span (i, j) is then the concatenatation the vector differences $\mathbf{f}_j - \mathbf{f}_i$ and $\mathbf{b}_i - \mathbf{b}_j$. This corresponds to a bidirectional version of the LSTM-Minus features first proposed by Wang and Chang (2016).

On top of this base, our label and span scoring functions are implemented as one-layer feedforward networks, taking as input the concatenated span difference and producing as output either a vector of label scores or a single span score. More formally, letting $\mathbf{s}_{ij}$ denote the vector representation of span (i, j), we define

$$s_{\text{labels}}(i, j) = \mathbf{V}_\ell g(\mathbf{W}_\ell \mathbf{s}_{ij} + \mathbf{b}_\ell),$$
$$s_{\text{span}}(i, j) = \mathbf{v}_s^\top g(\mathbf{W}_s \mathbf{s}_{ij} + \mathbf{b}_s),$$

where g denotes an elementwise nonlinearity. For notational convenience, we also let the score of an individual label ℓ be denoted by

$$s_{\text{label}}(i, j, \ell) = [s_{\text{labels}}(i, j)]_\ell,$$

where the right-hand side is the corresponding element of the label score vector.

One potential issue is the existence of unary chains, corresponding to nested labeled spans with the same endpoints. We take the common approach of treating these as additional atomic labels alongside all elementary nonterminals. To accommodate n-ary trees, our inventory additionally includes a special empty label $\varnothing$ used for spans that are not themselves full constituents but arise during the course of implicit binarization.

Our model shares several features in common with that of Cross and Huang (2016). In particular, our representation of spans and the form of our label scoring function were directly inspired by their work, as were our handling of unary chains and our use of an empty label. However, our approach differs in its treatment of structural decisions, and consequently, the inference algorithms we describe below diverge significantly from their transition-based framework.

3 Chart Parsing

Our basic model is compatible with traditional chart-based dynamic programming. Representing a constituency tree T by its labeled spans,

$$T := \{(\ell_t, (i_t, j_t)) : t = 1, \ldots, |T|\},$$

we define the score of a tree to be the sum of its constituent label and span scores,

$$s_{\text{tree}}(T) = \sum_{(\ell,(i,j))\in T} \left[s_{\text{label}}(i, j, \ell) + s_{\text{span}}(i, j)\right].$$

To find the tree with the highest score for a given sentence, we use a modified CKY recursion. As with classical chart parsing, the running time of our procedure is $O(n^3)$ for a sentence of length n.

3.1 Dynamic Program for Inference

The base case is a span $(i, i + 1)$ consisting of a single word. Since every valid tree must include all singleton spans, possibly with empty labels, we need not consider the span score in this case and perform only a single maximization over the choice of label:

$$s_{\text{best}}(i, i + 1) = \max_\ell \left[s_{\text{label}}(i, i + 1, \ell)\right].$$

For a general span (i, j), we define the score of the split (i, k, j) as the sum of its subspan scores,

$$s_{\text{split}}(i, k, j) = s_{\text{span}}(i, k) + s_{\text{span}}(k, j). \quad (1)$$

For convenience, we also define an augmented split score incorporating the scores of the corresponding subtrees,

$$\tilde{s}_{\text{split}}(i, k, j) = s_{\text{split}}(i, k, j) + s_{\text{best}}(i, k) + s_{\text{best}}(k, j).$$

Using these quantities, we can then write the general joint label and split decision as

$$s_{\text{best}}(i,j) = \max_{\ell,k} \left[s_{\text{label}}(i,j,\ell) + \tilde{s}_{\text{split}}(i,k,j) \right]. \tag{2}$$

Because our model assigns independent scores to labels and spans, this maximization decomposes into two disjoint subproblems, greatly reducing the size of the state space:

$$\begin{aligned} s_{\text{best}}(i,j) = &\max_{\ell} \left[s_{\text{label}}(i,j,\ell) \right] \\ &+ \max_{k} \left[\tilde{s}_{\text{split}}(i,k,j) \right]. \end{aligned}$$

We also note that the span scores $s_{\text{span}}(i,j)$ for each span (i,j) in the sentence can be computed once at the beginning of the procedure and shared across different subproblems with common left or right endpoints, allowing for a quadratic rather than cubic number of span score computations.

3.2 Margin Training

Training the model under this inference scheme is accomplished using a margin-based approach. When presented with an example sentence and its corresponding parse tree T^*, we compute the best prediction under the current model using the above dynamic program,

$$\widehat{T} = \operatorname*{argmax}_{T} \left[s_{\text{tree}}(T) \right].$$

If $\widehat{T} = T^*$, then our prediction was correct and no changes need to be made. Otherwise, we incur a hinge penalty of the form

$$\max\left(0,\ 1 - s_{\text{tree}}(T^*) + s_{\text{tree}}(\widehat{T})\right)$$

to encourage the model to keep a margin of at least 1 between the gold tree and the best alternative. The loss to be minimized is then the sum of penalties across all training examples.

Prior work has found that it can be beneficial in a variety of applications to incorporate a structured loss function into this margin objective, replacing the hinge penalty above with one of the form

$$\max\left(0,\ \Delta(\widehat{T}, T^*) - s_{\text{tree}}(T^*) + s_{\text{tree}}(\widehat{T})\right)$$

for a loss function Δ that measures the similarity between the prediction $\widehat{T}$ and the reference T^*. Here we take Δ to be a Hamming loss on labeled spans. To incorporate this loss into the training objective, we modify the dynamic program of Section 3.1 to support loss-augmented decoding (Taskar et al., 2005). Since the label decisions are isolated from the structural decisions, it suffices to replace every occurrence of the label scoring function $s_{\text{label}}(i,j,\ell)$ by

$$s_{\text{label}}(i,j,\ell) + \mathbf{1}(\ell \neq \ell^*_{ij}),$$

where ℓ^*_{ij} is the label of span (i,j) in the gold tree T^*. This has the effect of requiring larger margins between the gold tree and predictions that contain more mistakes, offering a greater degree of robustness and better generalization.

4 Top-Down Parsing

While we have so far motivated our model from the perspective of classical chart parsing, it also allows for a novel inference algorithm in which trees are constructed greedily from the top down. At a high level, given a span, we independently assign it a label and pick a split point, then repeat this process for the left and right subspans; the recursion bottoms out with length-one spans that can no longer be split. Figure 1 gives an illustration of the process, which we describe in more detail below.

The base case is again a singleton span $(i, i+1)$, and follows the same form as the base case for the chart parser. In particular, we select the label $\widehat{\ell}$ that satisfies

$$\widehat{\ell} = \operatorname*{argmax}_{\ell} \left[s_{\text{label}}(i, i+1, \ell) \right],$$

omitting span scores from consideration since singleton spans cannot be split.

To construct a tree over a general span (i,j), we aim to solve the maximization problem

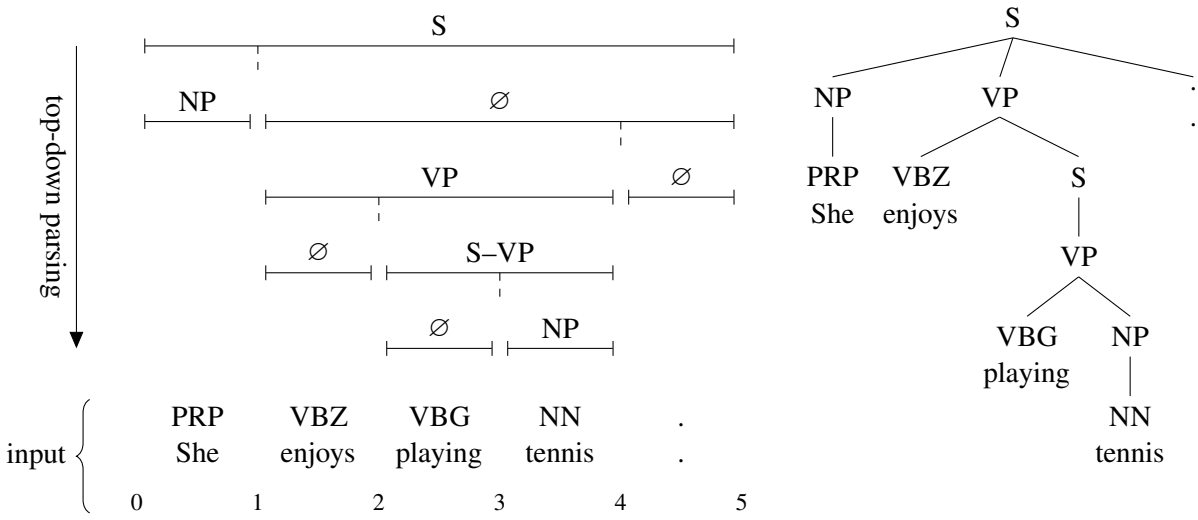
$$(\widehat{\ell}, \widehat{k}) = \operatorname*{argmax}_{\ell,k} \left[s_{\text{label}}(i,j,\ell) + s_{\text{split}}(i,k,j) \right],$$

where $s_{\text{split}}(i,k,j)$ is defined as in Equation (1). The independence of our label and span scoring functions again yields the decomposed form

$$\begin{aligned} \widehat{\ell} &= \operatorname*{argmax}_{\ell} \left[s_{\text{label}}(i,j,\ell) \right], \\ \widehat{k} &= \operatorname*{argmax}_{k} \left[s_{\text{split}}(i,k,j) \right], \end{aligned} \tag{3}$$

leading to a significant reduction in the size of the state space.

To generate a tree for the whole sentence, we call this procedure on the full sentence span $(0,n)$ and return the result. As there are $O(n)$ spans each



(a) Execution of the top-down parsing algorithm.

(b) Output parse tree.

Figure 1: An execution of our top-down parsing algorithm (a) and the resulting parse tree (b) for the sentence "She enjoys playing tennis." Part-of-speech tags, shown here together with the words, are predicted externally and are included as part of the input to our system. Beginning with the full sentence span $(0, 5)$, the label S and the split point 1 are predicted, and recursive calls are made on the child spans $(0, 1)$ and $(1, 5)$. The left child span $(0, 1)$ is assigned the label NP, and with no further splits to make, recursion terminates on this branch. The right child span $(1, 5)$ is assigned the empty label $\varnothing$, indicating that it does not represent a constituent in the tree. A split point of 4 is selected, and further recursive calls are made on the grandchild spans $(1, 4)$ and $(4, 5)$. This process of labeling and splitting continues until every branch of recursion bottoms out in singleton spans, at which point the full parse tree can be returned. Note that the unary chain S–VP is produced in a single labeling step.

requiring one label evaluation and at most $n - 1$ split point evaluations, the running time of the procedure is $O(n^2)$.

The algorithm outlined here bears a strong resemblance to the chart parsing dynamic program discussed in Section 3, but differs in one key aspect. When performing inference from the bottom up, we have already computed the scores of all of the subtrees below the current span, and we can take this knowledge into consideration when selecting a split point. In contrast, when producing a tree from the top down, we can only select a split point based on top-level evaluations of span quality, without knowing anything about the subtrees that will be generated below them. This difference is manifested in the augmented split score $\tilde{s}_{\text{split}}$ used in the definition of s_{best} in Equation (2), where the scores of the subtrees associated with a split point are included in the chart recursion but necessarily excluded from the top-down recursion.

While this apparent deficiency may be a cause for concern, we demonstrate the surprising empirical result in Section 6 that there is no loss in performance when moving from the globally-optimal chart parser to the greedy top-down procedure.

4.1 Margin Training

As with the chart parsing formulation, we also use a margin-based method for learning under the top-down model. However, rather than requiring separation between the scores of full trees, we instead enforce a local margin at every decision point.

For a span (i, j) occurring in the gold tree, let ℓ^* and k^* represent the correct label and split point, and let $\widehat{\ell}$ and $\widehat{k}$ be the predictions made by computing the maximizations in Equation (3). If $\widehat{\ell} \neq \ell^*$, meaning the prediction is incorrect, we incur a hinge penalty of the form

$$\max\left(0, 1 - s_{\text{label}}(i, j, \ell^*) + s_{\text{label}}(i, j, \widehat{\ell})\right).$$

Similarly, if $\widehat{k} \neq k^*$, we incur a hinge penalty of the form

$$\max\left(0, 1 - s_{\text{split}}(i, k^*, j) + s_{\text{split}}(i, \widehat{k}, j)\right).$$

To obtain the loss for a given training example, we trace out the actions corresponding to the gold tree and accumulate the above penalties over all decision points. As before, the total loss to be minimized is the sum of losses across all training examples.

Loss augmentation is also beneficial for the local decisions made by the top-down model, and can be implemented in a manner akin to the one discussed in Section 3.2.

4.2 Training with Exploration

The hinge penalties given above are only defined for spans (i, j) that appear in the example tree. The model must therefore be constrained at training time to follow decisions that exactly reproduce the gold tree, since supervision cannot be provided otherwise. As a result, the model is never exposed to its mistakes, which can lead to a lack of calibration and poor performance at test time.

To circumvent this issue, a *dynamic oracle* can be defined to inform the model about correct behavior even after it has deviated from the gold tree. Cross and Huang (2016) propose such an oracle for a related transition-based parsing system, and prove its optimality for the F1 metric on labeled spans. We adapt their result here to obtain a dynamic oracle for the present model with similar guarantees.

The oracle for labeling decisions carries over without modification: the correct label for a span is the label assigned to that span if it is part of the gold tree, or the empty label $\varnothing$ otherwise.

For split point decisions, the oracle can be broken down into two cases. If a span (i, j) appears as a constituent in the gold tree T, we let $b(i, j)$ denote the collection of its interior boundary points. For example, if the constituent over $(1, 7)$ has children spanning $(1, 3)$, $(3, 6)$, and $(6, 7)$, then we would have the two interior boundary points, $b(1, 7) = \{3, 6\}$. The oracle for a span appearing in the gold tree is then precisely the output of this function. Otherwise, for spans (i, j) not corresponding to gold constituents, we must instead identify the smallest enclosing gold constituent:

$$(i^*, j^*) = \min\{(i', j') \in T : i' \leq i < j \leq j'\},$$

where the minimum is taken with respect to the partial ordering induced by span length. The output of the oracle is then the set of interior boundary points of this enclosing span that also lie inside the original, $\{k \in b(i^*, j^*) : i < k < j\}$. The proof of correctness is similar to the proof in Cross and Huang (2016); we refer to the Dynamic Oracle section in their paper for a more detailed discussion.

As presented, the dynamic oracle for split point decisions returns a collection of one or more splits rather than a single correct answer. Any of these is a valid choice, with different splits corresponding to different binarizations of the original n-ary tree. We choose to use the leftmost split point for consistency in our implementation, but remark that the oracle split with the highest score could also be chosen at training time to allow for additional flexibility.

Having defined the dynamic oracle for our system, we note that training with exploration can be implemented by a single modification to the procedure described in Section 4.1. Local penalties are accumulated as before, but instead of tracing out the decisions required to produce the gold tree, we instead follow the decisions predicted by the model. In this way, supervision is provided at states within the prediction procedure that are more likely to arise at test time when greedy inference is performed.

5 Scoring and Loss Alternatives

The model presented in Section 2 is designed to be as simple as possible. However, there are many variations of the label and span scoring functions that could be explored; we discuss some of the options here.

5.1 Top-Middle-Bottom Label Scoring

Our basic model treats the empty label, elementary nonterminals, and unary chains each as atomic units, obscuring similarities between unary chains and their component nonterminals or between different unary chains with common prefixes or suffixes. To address this lack of structure, we consider an alternative scoring scheme in which labels are predicted in three parts: a top nonterminal, a middle unary chain, and a bottom nonterminal (each of which is possibly empty).[1] This not only allows for parameter sharing across labels with common subcomponents, but also has the added benefit of allowing the model to produce novel unary chains at test time.

[1] In more detail, $\varnothing$ decomposes as $(\varnothing, \varnothing, \varnothing)$, X decomposes as $(X, \varnothing, \varnothing)$, X–Y decomposes as $(X, \varnothing, Y)$, and X–Z_1–$\cdots$–Z_k–Y decomposes as $(X, Z_1$–$\cdots$–$Z_k, Y)$.

More precisely, we introduce the decomposition

$$s_{\text{label}}(i, j, (\ell_t, \ell_m, \ell_b)) = s_{\text{top}}(i, j, \ell_t) + s_{\text{middle}}(i, j, \ell_m) + s_{\text{bottom}}(i, j, \ell_b),$$

where s_{top}, s_{middle}, and s_{bottom} are independent one-layer feedforward networks of the same form as s_{label} that output scores for all label tops, label middle chains, and label bottoms encountered in the training corpus, respectively. The best label for a span (i, j) is then computed by solving the maximization problem

$$\max_{\ell_t, \ell_m, \ell_b} \left[s_{\text{label}}(i, j, (\ell_t, \ell_m, \ell_b)) \right],$$

which decomposes into three independent subproblems corresponding to the three label components. The final label is obtained by concatenating ℓ_t, ℓ_m, and ℓ_b, with empty components being omitted from the concatenation.

5.2 Left and Right Span Scoring

The basic model uses the same span scoring function s_{span} to assign a score to the left and right subspans of a given span. One simple extension is to replace this by a pair of distinct left and right feedforward networks of the same form, giving the decomposition

$$s_{\text{split}}(i, k, j) = s_{\text{left}}(i, k) + s_{\text{right}}(k, j).$$

5.3 Span Concatenation Scoring

Since span scores are only used to score splits in our model, we also consider directly scoring a split by feeding the concatenation of the span representations of the left and right subspans through a single feedforward network, giving

$$s_{\text{split}}(i, k, j) = \mathbf{v}_s^\top g\left(\mathbf{W}_s[\mathbf{s}_{ik}; \mathbf{s}_{kj}] + \mathbf{b}_s\right).$$

This is similar to the structural scoring function used by Cross and Huang (2016), although whereas they additionally include features for the outside spans $(0, i)$ and (j, n) in their concatenation, we omit these from our implementation, finding that they do not improve performance.

5.4 Deep Biaffine Span Scoring

Inspired by the success of deep biaffine scoring in recent work by Dozat and Manning (2016) for dependency parsing, we also consider a split scoring function of a similar form for our model. Specifically, we let $\mathbf{h}_{ik} = f_{\text{left}}(\mathbf{s}_{ik})$ and $\mathbf{h}_{kj} = f_{\text{right}}(\mathbf{s}_{kj})$ be deep left and right span representations obtained by passing the child vectors through corresponding left and right feedforward networks. We then define the biaffine split scoring function

$$s_{\text{split}}(i, k, j) = \mathbf{h}_{ik}^\top \mathbf{W}_s \mathbf{h}_{kj} + \mathbf{v}_{\text{left}}^\top \mathbf{h}_{ik} + \mathbf{v}_{\text{right}}^\top \mathbf{h}_{kj},$$

which consists of the sum of a bilinear form between the two hidden representations together with two inner products.

5.5 Structured Label Loss

The three-way label scoring scheme described in Section 5.1 offers one path towards the incorporation of label structure into the model. We additionally consider a structured Hamming loss on labels. More specifically, given two labels ℓ_1 and ℓ_2 consisting of zero or more nonterminals, we define the loss as $|\ell_1 \setminus \ell_2| + |\ell_2 \setminus \ell_1|$, treating each label as a multiset of nonterminals. This structured loss can be incorporated into the training process using the methods described in Sections 3.2 and 4.1.

6 Experiments

We first describe the general setup used for our experiments. We use the Penn Treebank (Marcus et al., 1993) for our English experiments, with standard splits of sections 2-21 for training, section 22 for development, and section 23 for testing. We use the French Treebank from the SPMRL 2014 shared task (Seddah et al., 2014) with its provided splits for our French experiments. No token preprocessing is performed, and only a single `<UNK>` token is used for unknown words at test time. The inputs to our system are concatenations of 100-dimensional word embeddings and 50-dimensional part-of-speech embeddings. In the case of the French Treebank, we also include 50-dimensional embeddings of each morphological tag. We use automatically predicted tags for training and testing, obtaining predicted part-of-speech tags for the Penn Treebank using the Stanford tagger (Toutanova et al., 2003) with 10-way jackknifing, and using the provided predicted part-of-speech and morphological tags for the French Treebank. Words are replaced by `<UNK>` with probability $1/(1+\text{freq}(w))$ during training, where $\text{freq}(w)$ is the frequency of w in the training data.

We use a two-layer bidirectional LSTM for our base span features. Dropout with a ratio selected from $\{0.2, 0.3, 0.4\}$ is applied to all non-recurrent

WSJ Dev, Atomic Labels, Basic 0-1 Label Loss

Parser	Minimal	Left-Right	Concat.	Biaffine
Chart	91.95	92.09	92.15	91.96
Top-Down	92.16	92.25	92.24	92.14

(a)

WSJ Dev, Atomic Labels, Structured Label Loss

Parser	Minimal	Left-Right	Concat.	Biaffine
Chart	91.86	92.12	92.09	91.95
Top-Down	92.12	92.31	92.26	92.20

(b)

WSJ Dev, 3-Part Labels, Basic 0-1 Label Loss

Parser	Minimal	Left-Right	Concat.	Biaffine
Chart	92.08	92.05	91.94	91.79
Top-Down	92.12	92.18	92.14	92.02

(c)

WSJ Dev, 3-Part Labels, Structured Label Loss

Parser	Minimal	Left-Right	Concat.	Biaffine
Chart	91.92	91.96	91.97	91.78
Top-Down	91.98	92.27	92.17	92.06

(d)

Table 1: Development F1 scores on the Penn Treebank. Each table corresponds to a particular choice of label loss (either the basic 0-1 loss or the structured Hamming label loss of Section 5.5) and labeling scheme (either the basic atomic scheme or the top-middle-bottom labeling scheme of Section 5.1). The columns within each table correspond to different split scoring schemes: basic minimal scoring, the left-right scoring of Section 5.2, the concatenation scoring of Section 5.3, and the deep biaffine scoring of Section 5.4.

connections of the LSTM, including its inputs and outputs. We tie the hidden dimension of the LSTM and all feedforward networks, selecting a size from $\{150, 200, 250\}$. All parameters (including word and tag embeddings) are randomly initialized using Glorot initialization (Glorot and Bengio, 2010), and are tuned on development set performance. We use the Adam optimizer (Kingma and Ba, 2014) with its default settings for optimization, with a batch size of 10. Our system is implemented in C++ using the DyNet neural network library (Neubig et al., 2017).

We begin by training the minimal version of our proposed chart and top-down parsers on the Penn Treebank. Out of the box, we obtain test F1 scores of 91.69 for the chart parser and 91.58 for the top-down parser. The higher of these matches the recent state-of-the-art score of 91.7 reported by Liu and Zhang (2016), demonstrating that our simple neural parsing system is already capable of achieving strong results.

Building on this, we explore the effects of different split scoring functions when using either the basic 0-1 label loss or the structured label loss discussed in Section 5.5. Our results are presented in Tables 1a and 1b.

We observe that regardless of the label loss, the minimal and deep biaffine split scoring schemes perform a notch below the left-right and concatenation scoring schemes. That the minimal scoring scheme performs worse than the left-right scheme is unsurprising, since the latter is a strict generalization of the former. It is evident, however, that joint scoring of left and right subspans is not required for strong results—in fact, the left-right scheme which scores child subspans in isolation slightly outperforms the concatenation scheme in all but one case, and is stronger than the deep biaffine scoring function across the board.

Comparing results across the choice of label loss, however, we find that fewer trends are apparent. The scores obtained by training with a 0-1 loss are all within 0.1 of those obtained using a structured Hamming loss, being slightly higher in four out of eight cases and slightly lower in the other half. This leads us to conclude that the more elementary approach is sufficient when selecting atomic labels from a fixed inventory.

We also perform the same set of experiments under the setting where the top-middle-bottom label scoring function described in Section 5.1 is used in place of an atomic label scoring function. These results are shown in Tables 1c and 1d.

A priori, we might expect that exposing additional structure would allow the model to make better predictions, but on the whole we find that the scores in this set of experiments are worse than those in the previous set. Trends similar to before hold across the different choices of scoring functions, though in this case the minimal setting has scores closer to those of the left-right setting, even exceeding its performance in the case of a chart parser with a 0-1 label loss.

Our final test results are given in Table 2, along with the results of other recent single-model parsers trained without external parse data. We

Final Parsing Results on Penn Treebank

Parser	LR	LP	F1
Durrett and Klein (2015)	–	–	91.1
Vinyals et al. (2015)	–	–	88.3
Dyer et al. (2016)	–	–	89.8
Cross and Huang (2016)	90.5	92.1	91.3
Liu and Zhang (2016)	91.3	92.1	91.7
Best Chart Parser	90.63	92.98	91.79
Best Top-Down Parser	90.35	93.23	91.77

Table 2: Comparison of final test F1 scores on the Penn Treebank. Here we only include scores from single-model parsers trained without external parse data.

Final Parsing Results on French Treebank

Parser	LR	LP	F1
Björkelund et al. (2014)	–	–	82.53
Durrett and Klein (2015)	–	–	81.25
Cross and Huang (2016)	81.90	84.77	83.11
Best Chart Parser	80.26	84.12	82.14
Best Top-Down Parser	79.60	85.05	82.23

Table 3: Comparison of final test F1 scores on the French Treebank.

achieve a new state-of-the-art F1 score of 91.79 with our best model. Interestingly, we observe that our parsers have a noticeably higher gap between precision and recall than do other top parsers, perhaps in part owing to the structured label loss which penalizes mismatching nonterminals more heavily than it does a nonterminal and empty label mismatch. In addition, there is little difference between the best top-down model and the best chart model, indicating that global normalization is not required to achieve strong results. Processing one sentence at a time on a `c4.4xlarge` Amazon EC2 instance, our best chart and top-down parsers operate at speeds of 20.3 sentences per second and 75.5 sentences per second, respectively, as measured on the test set.

We additionally train parsers on the French Treebank using the same settings from our English experiments, selecting the best model of each type based on development performance. We list our test results along with those of several other recent papers in Table 3. Although we fall short of the scores obtained by Cross and Huang (2016), we achieve competitive performance relative to the neural CRF parser of Durrett and Klein (2015).

7 Related Work

Many early successful approaches to constituency parsing focused on rich modeling of correlations in the *output space*, typically by engineering probabilistic context-free grammars with state spaces enriched to capture long-distance dependencies and lexical phenomena (Collins, 2003; Klein and Manning, 2003; Petrov and Klein, 2007). By contrast, the approach we have described here continues a recent line of work on direct modeling of correlations in the *input space*, by using rich feature representations to parameterize local potentials that interact with a comparatively unconstrained structured decoder. As noted in the introduction, this class of feature-based tree scoring functions can be implemented with either a linear transition system (Chen and Manning, 2014) or a global decoder (Finkel et al., 2008). Kiperwasser and Goldberg (2016) describe an approach closely related to ours but targeted at dependency formalisms, and which easily accommodates both sparse log-linear scoring models (Hall et al., 2014) and deep neural potentials (Henderson, 2004; Ballesteros et al., 2016).

The best-performing constituency parsers in the last two years have largely been transition-based rather than global; examples include the models of Dyer et al. (2016), Cross and Huang (2016) and Liu and Zhang (2016). The present work takes many of the insights developed in these models (e.g. the recurrent representation of spans (Kiperwasser and Goldberg, 2016), and the use of a dynamic oracle and exploration policy during training (Goldberg and Nivre, 2013)) and extends these insights to span-oriented models, which support a wider range of decoding procedures. Our approach differs from other recent chart-based neural models (e.g. Durrett and Klein (2015)) in the use of a recurrent input representation, structured loss function, and comparatively simple parameterization of the scoring function. In addition to the globally optimal decoding procedures for which these models were designed, and in contrast to the left-to-right decoder typically employed by transition-based models, our model admits an additional greedy top-to-bottom inference procedure.

8 Conclusion

We have presented a minimal span-oriented parser that uses a recurrent input representation to score

trees with a sum of independent potentials on their constituent spans and labels. Our model supports both exact chart-based decoding and a novel top-down inference procedure. Both approaches achieve state-of-the-art performance on the Penn Treebank, and our best model achieves competitive performance on the French Treebank. Our experiments show that many of the key insights from recent neural transition-based approaches to parsing can be easily ported to the chart parsing setting, resulting in a pair of extremely simple models that nonetheless achieve excellent performance.

Acknowledgments

We would like to thank Nick Altieri and the anonymous reviewers for their valuable comments and suggestions. MS is supported by an NSF Graduate Research Fellowship. JA is supported by a Facebook graduate fellowship and a Berkeley AI / Huawei fellowship.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473. http://arxiv.org/abs/1409.0473.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. 2016. Training with exploration improves a greedy stack-lstm parser. *arXiv preprint arXiv:1603.03793* .

Anders Björkelund, Ozlem Cetinoglu, Agnieszka Falenska, Richárd Farkas, Thomas Müller, Wolfgang Seeker, and Zsolt Szántó. 2014. The ims-wrocław-szeged-cis entry at the spmrl 2014 shared task: Reranking and morphosyntax meet unlabeled data. *Notes of the SPMRL* .

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*. pages 740–750.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics* 29(4):589–637.

James Cross and Liang Huang. 2016. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *EMNLP*.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR* abs/1611.01734. http://arxiv.org/abs/1611.01734.

Greg Durrett and Dan Klein. 2015. Neural crf parsing. *arXiv preprint arXiv:1507.03641* .

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776* .

Jenny Rose Finkel, Alex Kleeman, and Christopher D Manning. 2008. Efficient, feature-based, conditional random field parsing. In *ACL*. volume 46, pages 959–967.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics* 1:403–414.

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

David Leo Wright Hall, Greg Durrett, and Dan Klein. 2014. Less grammar, more features. In *ACL (1)*. pages 228–237.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, page 95.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *arXiv preprint arXiv:1603.04351* .

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

Jiangming Liu and Yue Zhang. 2016. Shift-reduce constituent parsing with neural lookahead features. *CoRR* abs/1612.00567. http://arxiv.org/abs/1612.00567.

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. http://dl.acm.org/citation.cfm?id=972470.972475.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji,

Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980* .

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. Assocation for Computational Linguistics.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. Dublin City University, Dublin, Ireland, pages 103–109. http://www.aclweb.org/anthology/W14-6111.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of the 22Nd International Conference on Machine Learning*. ACM, New York, NY, USA, ICML '05, pages 896–903. https://doi.org/10.1145/1102351.1102464.

Le Quang Thang, Hiroshi Noji, and Yusuke Miyao. 2015. Optimal shift-reduce constituent parsing with structured perceptron. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. volume 1, pages 1534–1544.

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*. Association for Computational Linguistics, Stroudsburg, PA, USA, NAACL '03, pages 173–180. https://doi.org/10.3115/1073445.1073478.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*. pages 2773–2781.

Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. 2015. A unified tagging solution: Bidirectional LSTM recurrent neural network with word embedding. *CoRR* abs/1511.00215. http://arxiv.org/abs/1511.00215.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. http://aclweb.org/anthology/P/P16/P16-1218.pdf.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960* .