Constituent Parsing as Sequence Labeling

Carlos Gómez-Rodríguez
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Campus de Elviña s/n, 15071
A Coruña, Spain
carlos.gomez@udc.es

David Vilares
Universidade da Coruña
FASTPARSE Lab, LyS Group
Departamento de Computación
Campus de Elviña s/n, 15071
A Coruña, Spain
david.vilares@udc.es

Abstract

We introduce a method to reduce constituent parsing to sequence labeling. For each word w_t, it generates a label that encodes: (1) the number of ancestors in the tree that the words w_t and w_{t+1} have in common, and (2) the nonterminal symbol at the lowest common ancestor. We first prove that the proposed encoding function is injective for any tree without unary branches. In practice, the approach is made extensible to all constituency trees by collapsing unary branches. We then use the PTB and CTB treebanks as testbeds and propose a set of fast baselines. We achieve 90.7% F-score on the PTB test set, outperforming the Vinyals et al. (2015) sequence-to-sequence parser. In addition, sacrificing some accuracy, our approach achieves the fastest constituent parsing speeds reported to date on PTB by a wide margin.[1]

1 Introduction

Constituent parsing is a core problem in NLP where the goal is to obtain the syntactic structure of sentences expressed as a phrase structure tree.

Traditionally, constituent-based parsers have been built relying on chart-based, statistical models (Collins, 1997; Charniak, 2000; Petrov et al., 2006), which are accurate but slow, with typical speeds well below 10 sentences per second on modern CPUs (Kummerfeld et al., 2012).

Several authors have proposed more efficient approaches which are helpful to gain speed while preserving (or even improving) accuracy. Sagae and Lavie (2005) present a classifier for constituency parsing that runs in linear time by relying on a shift-reduce stack-based algorithm, instead of a grammar. It is essentially an extension of transition-based dependency parsing (Nivre, 2003). This line of research has been polished through the years (Wang et al., 2006; Zhu et al., 2013; Dyer et al., 2016; Liu and Zhang, 2017; Fernández-González and Gómez-Rodríguez, 2018).

With an aim more related to our work, other authors have reduced constituency parsing to tasks that can be solved faster or in a more generic way. Fernández-González and Martins (2015) reduce phrase structure parsing to dependency parsing. They propose an intermediate representation where dependency labels from a head to its dependents encode the nonterminal symbol and an attachment order that is used to arrange nodes into constituents. Their approach makes it possible to use off-the-shelf dependency parsers for constituency parsing. In a different line, Vinyals et al. (2015) address the problem by relying on a sequence-to-sequence model where trees are linearized in a depth-first traversal order. Their solution can be seen as a machine translation model that maps a sequence of words into a parenthesized version of the tree. Choe and Charniak (2016) recast parsing as language modeling. They train a generative parser that obtains the phrasal structure of sentences by relying on the Vinyals et al. (2015) intuition and on the Zaremba et al. (2014) model to build the basic language modeling architecture.

More recently, Shen et al. (2018) propose an architecture to speed up the current state-of-the-art chart parsers trained with deep neural networks (Stern et al., 2017; Kitaev and Klein, 2018). They introduce the concept of *syntactic distances*, which specify the order in which the splitting points of a sentence will be selected. The model learns to predict such distances, to then recursively partition the input in a top-down fashion.

Contribution We propose a method to transform constituent parsing into sequence labeling.

[1]This is a revision with improved results of our paper originally published in EMNLP 2018. The previous version contained a bug where the script EVALB was not considering the COLLINS.prm parameter file.

This reduces it to the complexity of tasks such as part-of-speech (PoS) tagging, chunking or named-entity recognition. The contribution is two-fold.

First, we describe a method to linearize a tree into a sequence of labels (§2) of the same length of the sentence minus one.[2] The label generated for each word encodes the number of common ancestors in the constituent tree between that word and the next, and the nonterminal symbol associated with the lowest common ancestor. We prove that the encoding function is injective for any tree without unary branchings. After applying collapsing techniques, the method can parse unary chains.

Second, we use such encoding to present different baselines that can effectively predict the structure of sentences (§3). To do so, we rely on a recurrent sequence labeling model based on BILSTM's (Hochreiter and Schmidhuber, 1997; Yang and Zhang, 2018). We also test other models inspired in classic approaches for other tagging tasks (Schmid, 1994; Sha and Pereira, 2003). We use the Penn Treebank (PTB) and the Penn Chinese Treebank (CTB) as testbeds.

The comparison against Vinyals et al. (2015), the closest work to ours, shows that our method is able to train more accurate parsers. This is in spite of the fact that our approach addresses constituent parsing as a sequence labeling problem, which is simpler than a sequence-to-sequence problem, where the output sequence has variable/unknown length. Despite being the first sequence labeling method for constituent parsing, our baselines achieve decent accuracy results in comparison to models coming from mature lines of research, and their speeds are the fastest reported to our knowledge.

2 Linearization of *n*-ary trees

Notation and Preliminaries In what follows, we use bold style to refer to vectors and matrices (e.g $\mathbf{x}$ and $\mathbf{W}$). Let $\mathbf{w}$=$[w_1, w_2, ..., w_{|w|}]$ be an input sequence of words, where $w_i \in V$. Let $T_{|w|}$ be the set of constituent trees with $|w|$ leaf nodes that have no unary branches. For now, we will assume that the constituent parsing problem consists in mapping each sentence $\mathbf{w}$ to a tree in $T_{|w|}$, i.e., we assume that correct parses have no unary branches. We will deal with unary branches later.

To reduce the problem to a sequence labeling task, we define a set of labels L that allows us to encode each tree in $T_{|w|}$ as a unique sequence of labels in $L^{(|w|-1)}$, via an encoding function $\Phi_{|w|} : T_{|w|} \rightarrow L^{(|w|-1)}$. Then, we can reduce the constituent parsing problem to a sequence labeling task where the goal is to predict a function $F_{|w|,\theta} : V^{|w|} \rightarrow L^{|w|-1}$, where θ are the parameters to be learned. To parse a sentence, we label it and then decode the resulting label sequence into a constituent tree, i.e., we apply $F_{|w|,\theta} \circ \Phi^{-1}_{|w|}$.

For the method to be correct, we need the encoding of trees to be complete (every tree in $T_{|w|}$ must be expressible as a label sequence, i.e., $\Phi_{|w|}$ must be a function, so we have full coverage of constituent trees) and injective (so that the inverse function $\Phi^{-1}_{|w|}$ is well-defined). Surjectivity is also desirable, so that the inverse is a function on $L^{|w|-1}$, and the parser outputs a tree for any sequence of labels that the classifier can generate.

We now define our $\Phi_{|w|}$ and show that it is total and injective. Our encoding is not surjective per se. We handle ill-formed label sequences in §2.3.

2.1 The Encoding

Let w_i be a word located at position i in the sentence, for $1 \leq i \leq |w| - 1$. We will assign it a 2-tuple label $l_i = (n_i, c_i)$, where: n_i is an integer that encodes the number of common ancestors between w_i and w_{i+1}, and c_i is the nonterminal symbol at the lowest common ancestor.

Basic encodings The number of common ancestors may be encoded in several ways.

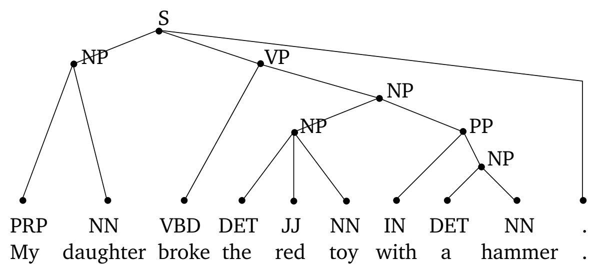
1. Absolute scale: The simplest encoding is to make n_i directly equal to the number of ancestors in common between w_i and w_{i+1}.

2. Relative scale: A second and better variant consists in making n_i represent the difference with respect to the number of ancestors encoded in n_{i-1}. Its main advantage is that the size of the label set is reduced considerably.

Figure 1 shows an example of a tree linearized according to both absolute and relative scales.

Encoding for trees with exactly k children For trees where all branchings have exactly k children, it is possible to obtain a even more efficient linearization in terms of number of labels. To do so, we take the relative scale encoding as our starting

[2]A last dummy label is generated to fulfill the properties of sequence labeling tasks.


S
NP
VP
NP
NP
PP
NP
PRP NN VBD DET JJ NN IN DET NN .
My daughter broke the red toy with a hammer .

Linearized tree (absolute scale):

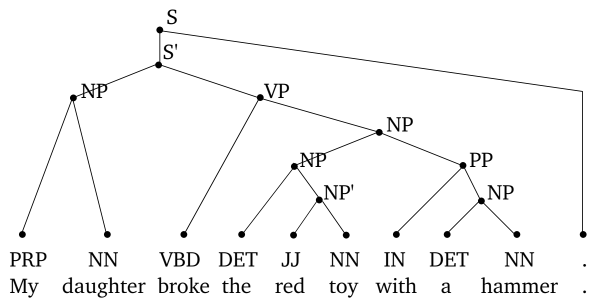
2NP 1S 2VP 4NP 4NP 3NP 4PP 5NP 1S

Linearized tree (relative scale):

2NP -1S 1VP 2NP 0NP -1NP 1PP 1NP -4S

Figure 1: An example of a constituency tree linearized applying both absolute and relative scales.

point. If we build the tree incrementally in a left-to-right manner from the labels, if we find a negative n_i, we will need to attach the word w_{i+1} (or a new subtree with that word as its leftmost leaf) to the $(-n_i + 2)$th node in the path going from w_i to the root. If every node must have exactly k children, there is only one valid negative value of n_i: the one pointing to the first node in said path that has not received its kth child yet. Any smaller value would leave this node without enough children (which cannot be fixed later due to the left-to-right order in which we build the tree), and any larger value would create a node with too many children. Thus, we can map negative values to a single label. Figure 2 shows an example for the case of binarized trees ($k = 2$).


S
S'
NP
VP
NP
NP
NP'
PP
NP
PRP NN VBD DET JJ NN IN DET NN .
My daughter broke the red toy with a hammer .

Linearized tree (absolute scale):

3NP 2S' 3VP 5NP 6NP' 4NP 5PP 6NP 1S

Linearized tree (relative scale):

3NP -1S' 1VP 2NP 1NP' -2NP 1PP 1NP -5S

Linearized tree (simplified relative scale):

3NP -S' 1VP 2NP 1NP' -NP 1PP 1NP -S

Figure 2: An example of a binarized constituency tree, linearized both applying absolute and relative scales.

Links to root Another variant emerged from the empirical observation that some tokens that are usually linked to the root node (such as the final punctuation in Figure 1) were particularly difficult to learn for the simpler baselines. To successfully deal with these cases in practice, it makes sense to consider a simplified annotation scheme where a node is assigned a special tag (ROOT, c_i) when it is directly linked to the root of the tree.

From now on, unless otherwise specified, we use the relative scale without the simplification for exactly k children. This will be the encoding used in the experiments (§4), because the size of the label set is significantly lower than the one obtained by relying on the absolute one. Also, it works directly with non-binarized trees, in contrast to the encoding that we introduce for trees with exactly k children, which is described only for completeness and possible interest for future work. For the experiments (§4), we also use the special tag (ROOT, c_i) to further reduce the size of the label set and to simplify the classification of tokens connected to the root, where $|n_i|$ is expected to be large.

2.2 Theoretical correctness

We now prove that $\Phi_{|w|}$ is a total function and injective for any tree in $T_{|w|}$. We remind that trees in this set have no unary branches. Later (in §2.3) we describe how we deal with unary branches. To prove correctness, we use the relative scale. Correctness for the other scales follows trivially.

Completeness Every pair of nodes in a rooted tree has at least one common ancestor, and a unique lowest common ancestor. Hence, for any tree in $T_{|w|}$, the label $l_i = (n_i, c_i)$ defined in Section 2.1 is well-defined and unique for each word w_i, $1 \leq i \leq |w| - 1$; and thus $\Phi_{|w|}$ is a total function from $T_{|w|}$ to $L^{(|w|-1)}$.

Injectivity The encoding method must ensure that any given sequence of labels corresponds to exactly one tree. Otherwise, we have to deal with ambiguity, which is not desirable.

For simplicity, we will prove injectivity in two steps. First, we will show that the encoding is injective if we ignore nonterminals (i.e., equivalently, that the encoding is injective for the set of trees resulting from replacing all the nonterminals in trees in $T_{|w|}$ with a generic nonterminal X). Then, we will show that it remains injective when we take nonterminals into account.

For the first part, let $\tau \in T_{|w|}$ be a tree where nonterminals take a generic value X. We represent the label of the ith leaf node as $\bullet_i$. Consider the representation of τ as a bracketed string, where a single-node tree with a node labeled A is represented by (A), and a tree rooted at R with child subtrees $C_1 \dots C_n$ is represented as $(R(C_1 \dots C_n))$.

Each leaf node will appear in this string as a substring $(\bullet_i)$. Thus, the parenthesized string has the form $\alpha_0(\bullet_1)\alpha_1(\bullet_2)\dots\alpha_{|w|-1}(\bullet_{|w|})\alpha_w$, where the α_is are strings that can only contain brackets and nonterminals, as by construction there can be no leaf nodes between $(\bullet_i)$ and $(\bullet_{i+1})$.

We now observe some properties of this parenthesized string. First, note that each of the substrings α_i must necessarily be composed of zero or more closing parentheses followed by zero or more opening parentheses with their corresponding nonterminal, i.e., it must be of the form $[)]^*[(X]^*$. This is because an opening parenthesis followed by a closing parenthesis would represent a leaf node, and there are no leaf nodes between $(\bullet_i)$ and $(\bullet_{i+1})$ in the tree.

Thus, we can write α_i as $\alpha_{i)}\alpha_{i(}$, where $\alpha_{i)}$ is a string matching the expression $[)]^*$ and $\alpha_{i(}$ a string matching the expression $[(X]^*$. With this, we can write the parenthesized string for τ as

$$\alpha_{0)}\alpha_{0(}(\bullet_1)\alpha_{1)}\alpha_{1(}(\bullet_2)\alpha_{2)}\alpha_{2(}\dots(\bullet_{|w|})\alpha_{|w|)}\alpha_{|w|(}.$$

Let us now denote by β_i the string $\alpha_{i-1(}(\bullet_i)\alpha_{i)}$. Then, and taking into account that $\alpha_{0)}$ and $\alpha_{w(}$ are trivially empty in the previous expression due to bracket balancing, the expression for the tree becomes simply $\beta_1\beta_2\dots\beta_{|w|}$, where we know, by construction, that each β_i is of the form $[(X]^*(\bullet_i)[)]^*$.

Since we have shown that each tree in $T_{|w|}$ uniquely corresponds to a string $\beta_1\beta_2\dots\beta_{|w|}$, to show injectivity of the encoding, it suffices to show that different values for a β_i generate different label sequences.

To show this, we can say more about the form of β_i: it must be either of the form $[(X]^*(\bullet_i)$ or of the form $(\bullet_i)[)]^*$, i.e., it is not possible that β_i contains both opening parenthesis before the leaf node and closing parentheses after the leaf node. This could only happen if the tree had a subtree of the form $(X(\bullet_i))$, but this is not possible since we are forbidding unary branches.

Hence, we can identify each β_i with an integer number $\delta(\beta_i)$: 0 if β_i has neither opening nor closing parentheses outside the leaf node, $+k$ if it has k opening parentheses, and $-k$ if it has k closing parentheses. It is easy to see that $\delta(\beta_1)\delta(\beta_2)\dots\delta(\beta_{|w|-1})$ corresponds to the values n_i in the relative-scale label encoding of the tree τ. To see this, note that the number of unclosed parentheses at the point right after β_i in the string exactly corresponds to the number of common ancestors between the ith and $(i+1)$th leaf nodes. A positive $\delta(\beta_i) = k$ corresponds to opening k parentheses before β_i, so the number of common ancestors of w_i and w_{i+1} will be k more than that of w_{i-1} and w_i. A negative $\delta(\beta_i) = -k$ corresponds to closing k parentheses after β_i, so the number of common ancestors will conversely decrease by k. A value of zero means no opening or closing parentheses, and no change in the number of common ancestors.

Thus, different parenthesized strings $\beta_1\beta_2\dots\beta_{|w|}$ generate different label sequences, which proves injectivity ignoring nonterminals (note that $\delta(\beta_{|w|})$ does not affect injectivity as it is uniquely determined by the other values: it corresponds to closing all the parentheses that remain unclosed at that point).

It remains to show that injectivity still holds when nonterminals are taken into account. Since we have already proven that trees with different structure produce different values of n_i in the labels, it suffices to show that trees with the same structure, but different nonterminals, produce different values of c_i. Essentially, this reduces to showing that every nonterminal in the tree is mapped into a concrete c_i. That said, consider a tree $\tau \in T_{|w|}$, and some nonterminal X in τ. Since trees in T_w do not have unary branches, X has at least two children. Consider the rightmost word in the first child subtree, and call it w_i. Then, w_{i+1} is the leftmost word in the second child subtree, and X is the lowest common ancestor of w_i and w_{i+1}. Thus, $c_i = X$, and a tree with identical structure but a different nonterminal at that position will generate a label sequence with a different value of c_i. This concludes the proof of injectivity.

2.3 Limitations

We have shown that our proposed encoding is a total, injective function from trees without unary branches with yield of length $|w|$ to sequences of $|w|-1$ labels. This will serve as the basis for our reduction of constituent parsing to sequence labeling. However, to go from theory to practice, we need to overcome two limitations of the theoretical encoding: non-surjectivity and the inability to encode unary branches. Fortunately, both can be overcome with simple techniques.

Handling of unary branches The encoding function $\Phi_{|w|}$ cannot directly assign the nonterminal symbols of unary branches, as there is not any pair of words (w_i, w_{i+1}) that have those in common. Figure 3 illustrates it with an example.

It is worth remarking that this is not a limitation of our encoding, but of any encoding that would facilitate constituent parsing as sequence labeling, as the number of nonterminal nodes in a tree with unary branches is not bounded by any function of $|w|$. The fact that our encoding works for trees without unary branches owes to the fact that such a tree cannot have more than $|w|-1$ non-leaf nodes, and therefore it is always possible to encode all of them in labels associated with $|w|-1$ leaf nodes.

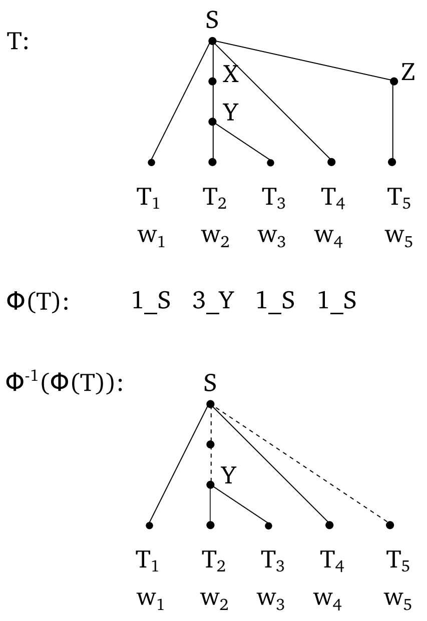


Figure 3: An example of a tree that cannot be directly linearized with our approach. w_i and T_i abstract over words and PoS tags. Dotted lines represent incorrect branches after applying and inverting our encoding naively without any adaptation for unaries. The nonterminal symbol of the second ancestor of w_2 (X) cannot be decoded, as no pair of words have X as their lowest common ancestor. A similar situation can be observed for the closest ancestor of w_5 (Z).

To overcome this issue, we follow a collapsing approach, as is common in parsers that need special treatment of unary chains (Finkel et al., 2008; Narayan and Cohen, 2016; Shen et al., 2018). For clarity, we use the name *intermediate unary chains* to refer to unary chains that end up into a nonterminal symbol (e.g. X $\rightarrow$ Y in Figure 3) and *leaf unary chains* to name those that yield a PoS tag (e.g. Z $\rightarrow$ T_5). Intermediate unary chains are collapsed into a chained single symbol, which can be encoded by $\Phi_{|w|}$ as any other nonterminal symbol. On the other hand, leaf unary chains are collapsed together with the PoS tag, but these cannot be encoded and decoded by relying on $\Phi_{|w|}$, as our encoding assumes a fixed sequence of leaf nodes and does not encode them explicitly. To overcome this, we propose two methods:

1. To use an extra function to enrich the PoS tags before applying our main sequence labeling function. This function is of the form $\Psi_{|w|} : V^{|w|} \rightarrow U^{|w|}$, where U is the set of labels of the leaf unary chains (without including the PoS tags) plus a dummy label $\varnothing$. $\Psi_{|w|}$ maps w_i to $\varnothing$ if there is no leaf unary chain at w_i, or to the collapsed label otherwise.

2. To extend our encoding function to predict them as a part of our labels l_i, by transforming them into 3-tuples (n_i, c_i, u_i) where u_i encodes the leaf unary chain collapsed label for w_i, if there is any, or none otherwise. We call this extended encoding function $\Phi'_{|w|}$.

The former requires to run two passes of sequence labeling to deal with leaf unary chains. The latter avoids this, but the number of labels is larger and sparser. In §4 we discuss how these two approaches behave in terms of accuracy and speed.

Non-surjectivity Our encoding, as defined formally in Section 2.1, is injective but not surjective, i.e., not every sequence of $|w|-1$ labels of the form (n_i, c_i) corresponds to a tree in $T_{|w|}$. In particular, there are two situations where a label sequence formally has no tree, and thus $\Phi^{-1}_{|w|}$ is not formally defined and we have to use extra heuristics or processing to define it:

- Sequences with conflicting nonterminals. A nonterminal can be the lowest common ancestor of more than two pairs of contiguous words when branches are non-binary. For example, in the tree in Figure 1, the lowest common ancestor of both "the" and "red" and of "red" and "toy" is the same NP node. This translates into $c_4 = NP$, $c_5 = NP$ in the label sequence. If we take that sequence and set $c_5 = VP$, we obtain a label sequence that does not strictly correspond to the encoding of any tree, as it contains a contradiction: two elements referencing the same node indicate different nonterminal labels. In practice, this problem is trivial to solve: when a label sequence encodes several conflicting nonterminals at a given position in the tree, we compute $\Phi^{-1}_{|w|}$ using the first such nonterminal and ignoring the rest.

- Sequences that produce unary structures. There are sequences of values n_i that do not correspond to a tree in $T_{|w|}$ because the only tree structure satisfying the common ancestor conditions of their values (the one built by generating the string of β_is in the injectivity proof) contains unary branchings, causing the problem described above where we do not have a specification for every nonterminal. An example of this is the sequence $(1, S), (3, Y), (1, S), (1, S)$ in absolute scaling, that was introduced in Figure 3. In practice, as unary chains have been previously collapsed, any generated unary node is considered as not valid and removed.

3 Sequence Labeling

Sequence labeling is an structured prediction task that generates an output label for every token in an input sequence (Rei and Søgaard, 2018). Examples of practical tasks that can be formulated under this framework in natural language processing are PoS tagging, chunking or named-entity recognition, which are in general fast. However, to our knowledge, there is no previous work on sequence labeling methods for constituent parsing, as an encoding allowing it was lacking so far.

In this work, we consider a range of methods ranging from traditional models to state-of-the-art neural models for sequence labeling, to test whether they are valid to train constituency-based parsers following our approach. We give the essential details needed to comprehend the core of each approach, but will mainly treat them as black boxes, referring the reader to the references for a careful and detailed mathematical analysis of each method. Appendix A specifies additional hyperparameters for the tested models.

Preprocessing We add to every sentence both beginning and end tokens.

3.1 Traditional Sequence Labeling Methods

We consider two baselines to train our prediction function $F_{|w|,\theta}$, based on popular sequence labeling methods used in NLP problems, such as PoS tagging or shallow parsing (Schmid, 1994; Sha and Pereira, 2003).

Conditional Random Fields (Lafferty et al., 2001) Let $\text{CRF}_{|w|,\theta}$ be its prediction function, a CRF model computes conditional probability distributions of the form $p(\mathbf{l}, \mathbf{w})$ such that $\text{CRF}_\theta(\mathbf{w})$ = $\mathbf{l}$ = $\arg\max_{\mathbf{l}'} p(\mathbf{l}', \mathbf{w})$. In our work, the inputs to the CRF are words and PoS tags. To represent a word w_i, we are using information of the word itself and also contextual information from $\mathbf{w}_{[i-1:i+1]}$.[3] In particular:

- We extract the word form (lowercased), the PoS tag and its prefix of length 2, from $\mathbf{w}_{[i-1:i+1]}$. For these words we also include binary features: whether it is the first word, the last word, a number, whether the word is capitalized or uppercased.
- Additionally, for w_i we look at the suffixes of both length 3 and 2 (i.e. $w_{i[-3:]}$ and $w_{i[-2:]}$).

To build our CRF models, we relied on the `sklearn-crfsuite` library[4].

MultiLayer Perceptron (Rosenblatt, 1958) We use one hidden layer. Let $\text{MLP}_{|w|,\theta}$ be its prediction function, it treats sequence labeling as a set of independent predictions, one per word. The prediction for a word is computed as $softmax(\mathbf{W_2} \cdot relu(\mathbf{W_1} \cdot \mathbf{x} + \mathbf{b_1}) + \mathbf{b_2})$, where $\mathbf{x}$ is the input vector and W_i and b_i the weights and biases to be learned at layer i. We consider both a discrete (MLP_d) and an embedded (MLP_e) perceptron. For the former, we use as inputs the same set of features as for the CRF. For the latter, the vector $\mathbf{x}$ for w_i is defined as a concatenation of word and PoS tag embeddings from $\mathbf{w}_{[i-2:i+2]}$.[5]

To build our MLPs, we relied on `keras`.[6]

3.2 Sequence Labeling Neural Models

We are using NCRFpp[7], a sequence labeling framework based on recurrent neural networks (RNN)

[3]We tried contextual information beyond the immediate previous and next word, but the performance was similar.

[4]https://sklearn-crfsuite.readthedocs.io/en/latest/

[5]In contrast to the discrete input, larger contextual information was useful.

[6]https://keras.io/

[7]https://github.com/jiesutd/NCRFpp, with PyTorch.

(Yang and Zhang, 2018), and more specifically on bidirectional short-term memory networks (Hochreiter and Schmidhuber, 1997), which have been successfully applied to problems such as PoS tagging or dependency parsing (Plank et al., 2016; Kiperwasser and Goldberg, 2016). Let $\text{LSTM}(\mathbf{x})$ be an abstraction of a standard long short-term memory network that processes the sequence $\mathbf{x} = [\mathbf{x}_1, ..., \mathbf{x}_{|\mathbf{x}|}]$, then a BILSTM encoding of its ith element, $\text{BILSTM}(\mathbf{x}, i)$ is defined as:

$$\text{BILSTM}(\mathbf{x}, i) = \mathbf{h}_i = \mathbf{h}_i^l \circ \mathbf{h}_i^r = \text{LSTM}^l(\mathbf{x}_{[1:i]}) \circ \text{LSTM}^r(\mathbf{x}_{[|\mathbf{x}|:i]})$$

In the case of multilayer BILSTM'S, the time-step outputs of the BILSTM_m are fed as input to the BILSTM_{m+1}. The output label for each w_i is finally predicted as $softmax(W \cdot \mathbf{h}_i + b)$.

Given a sentence $[w_1, w_2, ..., w_{|w|}]$, the input to the sequence model is a sequence of embeddings $[\mathbf{w_1}, \mathbf{w_2}, ..., \mathbf{w}_{|\mathbf{w}|}]$ where each $\mathbf{w_i} = \mathbf{w}_i \circ \mathbf{p}_i \circ \mathbf{ch}_i$, such that $\mathbf{w}_i$ and $\mathbf{p}_i$ are a word and a PoS tag embedding, and $\mathbf{ch}_i$ is a word embedding obtained from an initial character embedding layer, also based on a BILSTM. Figure 4 shows the architecture of the network.

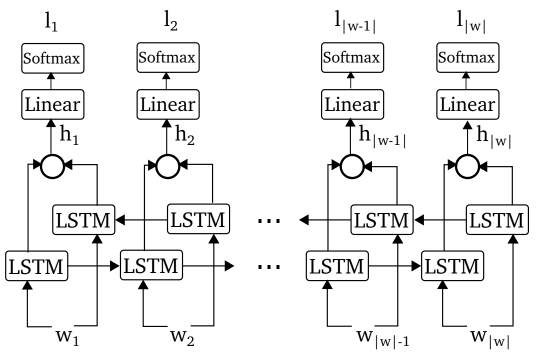


Figure 4: Architecture of the neural model

4 Experiments

We report results on models trained using the relative scale encoding and the special tag (ROOT,c_i). As a reminder, to deal also with leaf unary chains, we proposed two methods in §2.3: to predict them relying both on the encoding functions $\Phi_{|w|}$ and $\Psi_{|w|}$, or to predict them as a part of an enriched label predicted by the function $\Phi'_{|w|}$. For clarity, we are naming these models with the superscripts $^{\Psi,\Phi}$ and $^{\Phi'}$, respectively.

Datasets We use the Penn Treebank (Marcus et al., 1994) and its official splits: Sections 2 to 21 for training, 22 for development and 23 for testing. For the Chinese Penn Treebank (Xue et al., 2005): articles 001- 270 and 440-1151 are used for training, articles 301-325 for development, and articles 271-300 for testing. We use the version of the corpus with the predicted PoS tags of Dyer et al. (2016). We train the Φ models based on the predicted output by the corresponding Ψ model.

Metrics We use the F-score from the EVALB script using COLLINS.prm as the parameter file. Speed is measured in sentences per second. We briefly comment on the accuracy (percentage of correctly predicted labels, no symbol excluded here) of our baselines.

Source code It can be found at `https://github.com/aghie/tree2labels`

Hardware The models are run on a single thread of a CPU[8] and on a consumer-grade GPU[9]. In sequence-to-sequence work (Vinyals et al., 2015) the authors use a multi-core CPU (the number of threads was not specified), while we provide results on a single core for easier comparability. Parsing sentences on a CPU can be framed as an "embarrassingly parallel" problem (Hall et al., 2014), so speed can be made to scale linearly with the number of cores. We use the same batch size as Vinyals et al. (2015) for testing (128).[10]

4.1 Results

Table 1 shows the performance of our baselines on the PTB development set. It is worth noting that since we are using different libraries to train the models, these might show some differences in terms of performance/speed beyond those expected in theory. For the BILSTM model we test:

- $\text{BILSTM}_{m=1}$: It does not use pretrained word embeddings nor character embeddings. The number of layers m is set to 1.
- $\text{BILSTM}_{m=1,e}$: It adds pretrained word embeddings from GloVe (Pennington et al., 2014) for English and from the Gigaword corpus for Chinese (Liu and Zhang, 2017).
- $\text{BILSTM}_{m=1,e,ch}$: It includes character embeddings processed through a BILSTM.

[8]An Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz.
[9]A GeForce GTX 1080.
[10]A larger batch will likely result in faster parsing when executing the model on a GPU, but not necessarily on a CPU.

- BILSTM$_{m=2,e}$: m is set to 2. No character embeddings.
- BILSTM$_{m=2,e,ch}$: m is set to 2.

Model	**F-score**	**Acc.**	**Sent/s (CPU)**	**Sent/s (GPU)**
CRF$^{\Psi,\Phi}$	62.5	63.9	83	-
MLP$_d^{\Psi,\Phi}$	74.8	78.0	16	49
MLP$_e^{\Psi,\Phi}$	76.6	79.3	503	666
CRF$^{\Phi'}$	62.3	65.4	6	-
MLP$_d^{\Phi'}$	74.2	78.0	31	95
MLP$_e^{\Phi'}$	77.2	79.7	342	890
BILSTM$_{m=1}^{\Psi,\Phi}$	88.1	88.9	144	541
BILSTM$_{m=1,e}^{\Psi,\Phi}$	89.2	89.8	144	543
BILSTM$_{m=1,e,ch}^{\Psi,\Phi}$	89.5	90.0	120	456
BILSTM$_{m=2,e}^{\Psi,\Phi}$	90.7	90.7	72	476
BILSTM$_{m=2,e,ch}^{\Psi,\Phi}$	90.7	90.9	65	405
BILSTM$_{m=1}^{\Phi'}$	88.3	89.3	206	941
BILSTM$_{m=1,e}^{\Phi'}$	89.5	90.1	209	957
BILSTM$_{m=1,e,ch}^{\Phi'}$	89.0	90.0	180	808
BILSTM$_{m=2,e}^{\Phi'}$	90.8	90.9	119	842
BILSTM$_{m=2,e,ch}^{\Phi'}$	90.6	90.9	109	716

Table 1: Performance of the proposed sequence labeling methods on the development set of the PTB. For the CRF models the complexity is quadratic with respect to the number of labels, which causes CRF$^{\Phi'}$ to be particularly slow.

The Ψ, Φ and the Φ' models obtain similar F-scores. When it comes to speed, the BILSTMs$^{\Phi'}$ are notably faster than the BILSTMs$^{\Psi,\Phi}$. Φ' models are expected to be more efficient, as leaf unary chains are handled implicitly. In practice, Φ' is a more expensive function to compute than the original Φ, since the number of output labels is significantly larger, which reduces the expected gains with respect to the Ψ, Φ models. It is worth noting that our encoding is useful to train an MLP$_e$ with a decent sense of phrase structure, while being very fast. Paying attention to the differences between F-score and Accuracy for each baseline, we notice the gap between them is larger for CRFs and MLPs. This shows the difficulties that these methods have, in comparison to the BILSTM approaches, to predict the correct label when a word w_{i+1} has few common ancestors with w_i. For example, let -10X be the right (relative scale) label between w_i and w_{i+1}, and let l_1=-1X and l_2=-9X be two possible wrong labels. In terms of accuracy it is the same that a model predicts $l1$ or $l2$, but in terms of constituent F-score, the first will be much worse, as many closed parentheses will remain unmatched.

Tables 2 and 3 compare our best models against the state of the art on the PTB and CTB test sets. The performance corresponds to models without reranking strategies, unless otherwise specified.

5 Discussion

We are not aware of work that reduces constituency parsing to sequence labeling. The work that can be considered as the closest to ours is that of Vinyals et al. (2015), who address it as a sequence-to-sequence problem, where the output sequence has variable/unknown length. In this context, even a one hidden layer perceptron outperforms their 3-layer LSTM model without attention, while parsing hundreds of sentences per second. Our best models also outperformed their 3-layer LSTM model with attention and even a simple BILSTM model with pretrained GloVe embeddings obtains a similar performance. In terms of F-score, the proposed sequence labeling baselines still lag behind mature shift-reduce and chart parsers. In terms of speed, they are clearly faster than both CPU and GPU chart parsers and are at least on par with the fastest shift-reduce ones. Although with significant loss of accuracy, if phrase-representation is needed in large-scale tasks where the speed of current systems makes parsing infeasible (Gómez-Rodríguez, 2017; Gómez-Rodríguez et al., 2017), we can use the simpler, less accurate models to get speeds well above any parser reported to date.

It is also worth noting that in their recent work, published while this manuscript was under review, Shen et al. (2018) developed a mapping of binary trees with n leaves to sequences of $n-1$ integers (Shen et al., 2018, Algorithm 1). This encoding is different from the ones presented here, as it is based on the height of lowest common ancestors in the tree, rather than their depth. While their purpose is also different from ours, as they use this mapping to generate training data for a parsing algorithm based on recursive partitioning using real-valued distances, their encoding could also be applied with our sequence labeling approach. However, it has the drawback that it only supports binarized trees, and some of its theoretical properties are worse for our goal, as the way to define the inverse of an arbitrary label sequence can be highly ambiguous: for example, a sequence of $n-1$ equal labels in this encoding can represent *any* binary tree with n leaves.

Model	Testbed	CPU Run		GPU Run		F-score
		#Cores	Sents/s	#GPU	Sents/s	
Sequence labeling						
$\text{MLP}_e^{\Psi,\Phi}$	WSJ23	1	501	1	669	75.8
$\text{MLP}_e^{\Phi'}$	WSJ23	1	349	1	929	76.7
$\text{BILSTM}_{m=1,e}^{\Psi,\Phi}$	WSJ23	1	148	1	581	88.9
$\text{BILSTM}_{m=1,e}^{\Phi'}$	WSJ23	1	221	1	1016	89.1
$\text{BILSTM}_{m=2,e,ch}^{\Psi,\Phi}$	WSJ23	1	66	1	434	90.6
$\text{BILSTM}_{m=2,e,ch}^{\Phi'}$	WSJ23	1	115	1	780	90.7
$\text{BILSTM}_{m=2,e}^{\Psi,\Phi}$	WSJ23	1	74	1	506	90.7
$\text{BILSTM}_{m=2,e}^{\Phi'}$	WSJ23	1	126	1	898	90.7
Sequence-to-sequence						
3-layer LSTM	WSJ 23					<70
3-layer LSTM + Attention$^\diamond$ (Vinyals et al., 2015)	WSJ 23	Multi-core (number not specified)	120			88.3
Constituency parsing as dependency parsing						
Fernández-González and Martins (2015)$^\diamond$	WSJ23	1	41			90.2
Chart-based parsers						
Charniak (2000)*	WSJ23	1	6			89.5
Petrov and Klein (2007)*	WSJ23	1	6			90.1
Stern et al. (2017)$^\diamond$	WSJ23	16*	20			91.8
Kitaev and Klein (2018) +ELMo (Peters et al., 2018)$^\diamond$	WSJ23			2	70	95.1
Chart-based parsers with GPU-specific implementation						
Canny et al. (2013)$^\diamond$	WSJ(<30)			1	250	
Hall et al. (2014)$^\diamond$	WSJ(<40)			1	404	
Transition-based and other greedy constituent parsers						
Zhu et al. (2013)$^\diamond$	WSJ23	1	101			89.9
Zhu et al. (2013)+Padding$^\diamond$	WSJ23	1	90			90.4
Dyer et al. (2016)$^\triangleright$	WSJ23	1	17			91.2
Fernández and Gómez-Rodríguez (2018)$^\diamond$	WSJ23	1	18			91.7
Stern et al. (2017)$^\diamond$	WSJ23	16*	76			91.8
Liu and Zhang (2017)	WSJ23					91.8
Shen et al. (2018)	WSJ23			1	111	91.8

Table 2: Comparison against the state of the art.*Stern et al. (2017) report that they use a 16-core machine, but sentences are processed one-at-a-time. Hence, they do not exploit inter-sentence parallelism, but they may gain some speed from intra-sentence parallelism. $\diamond$ indicates the that the speed was reported in the paper itself. $*$ and $\triangleright$ indicate that the speeds were extracted from Zhu et al. (2013) and Fernández and Gómez-Rodríguez (2018).

Model	F-score
$\text{MLP}_e^{\Psi,\Phi}$	63.1
$\text{MLP}_e^{\Phi'}$	64.4
$\text{BILSTM}_{m=2,e,ch}^{\Psi,\Phi}$	84.4
$\text{BILSTM}_{m=2,e,ch}^{\Phi'}$	84.1
$\text{BILSTM}_{m=2,e}^{\Psi,\Phi}$	84.4
$\text{BILSTM}_{m=2,e}^{\Phi'}$	83.1
Zhu et al. (2013)	82.6
Zhu et al. (2013)+P	83.2
Dyer et al. (2016)	84.6
Liu and Zhang (2017)	86.1
Shen et al. (2018)	86.5
Fernández and Gómez-Rodríguez (2018)	86.8

Table 3: Performance on the CTB test set

6 Conclusion

We presented a new parsing paradigm, based on a reduction of constituency parsing to sequence labeling. We first described a linearization function to transform a constituent tree (with n leaves) into a sequence of $n-1$ labels that encodes it. We proved that this encoding function is total and injective for any tree without unary branches. We also discussed its limitations: how to deal with unary branches and non-surjectivity, and showed how these can be solved. We finally proposed a set of fast and strong baselines.

Acknowledgments

This work has received funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150), from the TELEPARES-UDC project (FFI2014-51978-C2-2-R) and the ANSWER-ASAP project (TIN2017-85160-C2-1-R) from MINECO, and from Xunta de Galicia

(ED431B 2017/01). We gratefully acknowledge NVIDIA Corporation for the donation of a GTX Titan X GPU.

References

John Canny, David Hall, and Dan Klein. 2013. A multi-teraflop constituency parser using GPUs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1898–1907.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 132–139. Association for Computational Linguistics.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 16–23. 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. Association for Computational Linguistics.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2018. Faster Shift-Reduce Constituent Parsing with a Non-Binary, Bottom-Up Strategy. *ArXiv e-prints*.

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. Association for Computational Linguistics.

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

Carlos Gómez-Rodríguez. 2017. Towards fast natural language parsing: FASTPARSE ERC Starting Grant. *Procesamiento del Lenguaje Natural*, 59.

Carlos Gómez-Rodríguez, Iago Alonso-Alonso, and David Vilares. 2017. How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis. *Artificial Intelligence Review*.

David Hall, Taylor Berg-Kirkpatrick, and Dan Klein. 2014. Sparser, better, faster GPU parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 208–217.

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

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

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)*, Melbourne, Australia. Association for Computational Linguistics.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the Wall Street corral: An empirical investigation of error types in parser output. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1048–1059, Jeju Island, Korea. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

J. Liu and Y. Zhang. 2017. In-Order Transition-based Constituent Parsing. *ArXiv e-prints*.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA. Association for Computational Linguistics.

Shashi Narayan and Shay B. Cohen. 2016. Optimizing spectral learning for parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1546–1556, Berlin, Germany. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. 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 2227–2237. Association for Computational Linguistics.

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

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.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. 2016. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 412–418, Berlin, Germany. Association for Computational Linguistics.

Marek Rei and Anders Søgaard. 2018. Zero-shot sequence labeling: Transferring knowledge from sentences to tokens. 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 293–302. Association for Computational Linguistics.

Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics.

Helmut Schmid. 1994. Part-of-speech tagging with neural networks. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 172–176, Stroudsburg, PA, USA. Association for Computational Linguistics.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. 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. Association for Computational Linguistics.

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

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.

Mengqiu Wang, Kenji Sagae, and Teruko Mitamura. 2006. A fast, accurate deterministic parser for chinese. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, ACL-44, pages 425–432, Stroudsburg, PA, USA. Association for Computational Linguistics.

Naiwen Xue, Fei Xia, Fu-Dong Chiou, and Marta Palmer. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural language engineering*, 11(2):207–238.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia. Association for Computational Linguistics.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

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)*, volume 1, pages 434–443.

A Setup configuration used to train our sequence labeling methods

Conditional Random Fields We use the default configuration provided together with the `sklearn-crfsuite` library.

MultiLayer Perceptron Both the discrete and distributed perceptrons are implemented in `keras`.

- *Training hyperparameters* The model is trained up to 30 epochs, with early stopping (patience=4). We use Stochastic Gradient Descent (SGD) to optimize the objective function. The initial learning rate is set to 0.1.

- *Layer and embedding sizes.* The dimension of the hidden layer is set to 100. For the perceptron fed with embeddings, we use 100 and 20 dimensions to represent a word and its PoS tag, respectively.

Bidirectional Long Short-Term Memory We relied on the `NCRFpp` framework (Yang and Zhang, 2018).

- *Training hyperparameters* We use mini-batching (the batch size during training is set to 8). As optimizer, we use SGD, setting the initial learning rate to 0.2, momentum to 0.9 and a linear decay of 0.05. We train the model up to 100 epochs and keep the best performing model in the development set.

- *Layer and embedding sizes*: We use 100, 30 and 20 dimensions to represent a word, a postag and a character embedding. The output hidden layer from the character embeddings layer is set to 50. The left-to-right and right-to-left LSTMs generate each a hidden vector of size 400.