Unsupervised Syntactic Structure Induction in Natural Language Processing

by

Anup Anand Deshmukh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis consists of the author's work done as a research assistant at the University of Alberta under the supervision of Prof. Lili Mou, which was accepted as a Findings paper at EMNLP 2021. At the time of submitting this thesis, the author was working towards a camera-ready version.

The work is first-authored by Anup Anand Deshmukh, who implemented the knowledge-transfer approach, including all student and teacher models, self-supervised objective, and chunking-to-parsing methods. One of the co-authors, Qianqiu Zhang, helped implement two baselines, namely Hidden Markov Model (HMM) and Pointwise Mutual Information (PMI) for unsupervised chunking.

Abstract

This work addresses unsupervised chunking as a task for syntactic structure induction, which could help understand the linguistic structures of human languages especially, low-resource languages. In chunking, words of a sentence are grouped together into different phrases (also known as chunks) in a non-hierarchical fashion. Understanding text fundamentally requires finding noun and verb phrases, which makes unsupervised chunking an important step in several real-world applications.

In this thesis, we establish several baselines and discuss our three-step knowledge transfer approach for unsupervised chunking. In the first step, we take advantage of state-of-the-art unsupervised parsers, and in the second, we heuristically induce chunk labels from them. We propose a simple heuristic that does not require any supervision of annotated grammar and generates reasonable (albeit noisy) chunks. In the third step, we design a hierarchical recurrent neural network (HRNN) that learns from these pseudo ground-truth labels. The HRNN explicitly models the composition of words into chunks and smooths out the noise from heuristically induced labels. Our HRNN a) maintains both word-level and phrase-level representations and b) explicitly handles the chunking decisions by providing autoregressiveness at each step. Furthermore, we make a case for exploring the self-supervised learning objectives for unsupervised chunking. Finally, we discuss our attempt to transfer knowledge from chunking back to parsing in an unsupervised setting.

We conduct comprehensive experiments on three datasets: CoNLL-2000 (English), CoNLL-2003 (German), and the English Web Treebank. Results show that our HRNN improves upon the teacher model (Compound PCFG) in terms of both phrase F1 and tag accuracy. Our HRNN can smooth out the noise from induced chunk labels and accurately capture the chunking patterns. We evaluate different chunking heuristics and show that maximal left-branching performs the best, reinforcing the fact that left-branching structures indicate closely related words. We also present rigorous analysis on the HRNN's architecture and discuss the performance of vanilla recurrent neural networks.

Acknowledgements

This work would not have been completed successfully in the absence of a long list of people. I would like to thank my supervisors Prof. Ming Li and Prof. Jimmy Lin, for their support and guidance. I would like to thank Prof. Lili Mou, who taught me the ways of conducting research and motivated me to be more ambitious. I would also like to thank all the researchers for their seminal work in Natural Language Processing, which sparked my interest in this field.

I would like to thank Prof. Charles Clarke and Prof. Jian Zhao, who agreed to serve as readers of my thesis.

Although I could not meet my friends because of the pandemic, they still helped me keep my mind off the pressure. I express my sincere gratitude to Aalekhya, Nihareeka, Smit and Anshul. A special thanks to Pratheeksha for her constant support, motivation, and brainstorming sessions.

Finally, I would like to express acknowledgment to my family: my brother Aniket, for constantly motivating me; my parents, who have sacrificed a lot for providing me with a good education. I thank them for their unconditional love.

Dedication

This thesis is dedicated to my parents and my brother.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

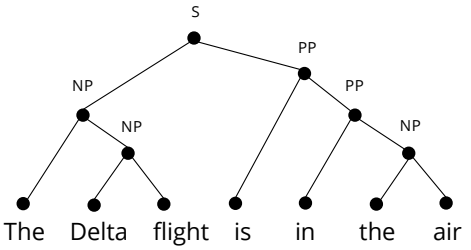
1.1 Syntactic Structure Induction

Understanding the linguistic structure of language (e.g., parsing and chunking) is an important research topic in natural language processing (NLP). The linguistic structure captures the grammar rules and acts as an intermediate representation for many downstream NLP tasks.

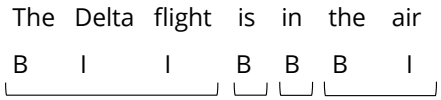
Constituency parsing is one of the most common means of defining the syntactic structure of text. In constituency parsing, the words in a sentence are hierarchically organized to yield a parse tree. In Figure 1.1a, for instance, words "the" and "ball" together form a noun phrase (NP). Then, this NP along with the word "hit" forms a verb phrase (VP). The process continues until the entire tree is built.

Chunking is another meaningful means of defining the syntactic structure of text, where words of a sentence are grouped into chunks (roughly speaking, phrases) in a non-hierarchical fashion [67, 38]. Thus, chunking can be viewed as a flattened version of parsing, since all the chunks in a sentence are non-overlapping. In Figure 1.1b, the two consecutive words "delta" and "flight" form one chunk indicating a noun phrase, while the words "is" and "in" form a chunk individually.

Most previous work employs supervised machine learning methods for predicting syntactic structures. While these methods achieve high performance, they require massive data labeled with linguistic structures, such as treebanks. For example, the Penn treebank [50] contains a manually labeled constituency parse tree for every sentence. Unfortunately, existing resources, including the Penn treebank, are mainly constructed for common languages like English and



(a) Constituency parsing



(b) Chunking and the BI schema

Figure 1.1: Two tasks for syntactic structure induction

Chinese. Creating these new treebanks for low-resource languages is cumbersome and expensive. Furthermore, the labeled treebanks are often restricted to a specific domain, limiting the generalization capability of machine learning models. For example, models trained on newswire treebanks tend to perform considerably worse when applied to new types of data [18]. Such issues highlight the importance of circumventing labeled treebanks.

Unsupervised linguistic structure discovery has been attracting increasing interest in recent years [32, 74, 76]. This task concerns discovering linguistic structures of text without using linguistically labeled data and is essential to NLP research because of its potential use for low-resource languages.

Apart from the limited generalization capability, supervised methods use labeled treebanks and introduce a structural bias into the learning model. Such bias stems from the labeled treebanks, which subscribe to a particular set of grammar rules. However, linguists often disagree on the existence of the grammar for a language [80]. Some linguists argue against its existence simply because there are equally or more plausible grammars that can associate structure and meaning to the same text [44]. Hence, the unsupervised setting is an important research direction since it allows for automatic linguistic knowledge discovery.

Moreover, grammar learned by these unsupervised methods can also be used to examine cognitive principles in a language. For example, the distributional principle says that morphemes present in similar contexts belong to the same category [80]. Morphemes are nothing but a meaningful unit of a language that cannot be further divided (e.g., in, come, -ing, which forms incoming). Syntactic structures learned by unsupervised methods can further validate such theories.

1.2 Problem Statement

Previous work mainly focuses on unsupervised constituency parsing for syntactic structure discovery [34, 63, 32, 33, 74]. In this work, we address *unsupervised chunking*, which aims to group the words of a sentence into chunks in a non-hierarchical fashion. Moreover, this setting would like to detect chunks without the supervision of annotated linguistic structures. Unsupervised chunking has real-world applications, as understanding text fundamentally requires finding spans like noun phrases and verb phrases. It would benefit various downstream tasks, such as keywords extraction [21] and open information extraction [52]. A wide range of non-trivial NLP tasks determine chunks as part of their pre-processing [69]. Such tasks include machine translation and question answering, making chunking an important component in NLP models.

Chunking is simpler than full constituency parsing because the former predicts a two-level flattened structure. Thus, it is more likely to obtain meaningful chunking structures than parse trees in an unsupervised setting. This idea of "starting small" can be traced back to the seminal work on training neural networks by J.L Elman [19]. This work suggests that breaking down a problem into simpler ones does not act as a limitation. Instead, it contributes as a necessary prerequisite for mastering a more complex task. For example, neural networks analyze complex sentences that involve clauses, phrases, and several inter-relations. This work argues that - the learning of such complex tasks (e.g., unsupervised parsing) would succeed when neural networks begin with limited resources and tasks (e.g., unsupervised chunking).

To this end, we propose a knowledge-transfer approach to unsupervised chunking by hierarchical recurrent neural networks (HRNN). We utilize the recent advances of unsupervised parsers and propose a maximal left-branching heuristic to induce chunk labels from them. Without any supervision of annotated grammars, such heuristic leads to reasonable (albeit noisy and imperfect) chunks. These unsupervised parsers, along with the chunking heuristics, act as teacher models. We further design a student model — an HRNN that learns from the heuristic chunk labels. Our HRNN involves a trainable chunking gate that switches between a lower word-level RNN and an upper phrase-level RNN. Such a gate explicitly models the composition of words into chunks and also provides autoregressiveness at each step. Results on the three datasets: CoNLL-2000 (English), CoNLL-2003 (German), and the English Web Treebank show that our HRNN can smooth out the noise of heuristically induced chunk labels, with a considerable improvement in terms of the phrase-F1 score. We also provide detailed analyses of our maximal left-branching chunking heuristic and student HRNN model to better understand their contribution.

1.3 Contributions

The main contributions of this thesis can be summarized as follows:

- We propose a novel three-step knowledge transfer approach to *unsupervised chunking*;
    1. Utilizing state-of-the-art unsupervised parsers: We propose to use the recent advances in unsupervised parsing, namely Compound Probabilistic Context-Free Grammar (Compound PCFG) [33]
    2. Inducing chunk labels from Compound PCFG: We present a simple yet effective heuristic that extracts maximal left-branching subtrees as chunks from Compound PCFG
    3. Training a learning model on the induced chunk labels: We design an HRNN that models the composition of words into chunks by treating the induced chunk labels as pseudo ground-truth
- We make a case for exploring the self-supervised learning objectives (such as sentence reconstruction) to further improve the chunking performance. *Unsupervised chunking* can be looked at as a new task for syntactic structure induction. To this end, we also discuss our attempt to induce knowledge from chunking back to parsing in an unsupervised setting.
- We implement several baselines for the task of *unsupervised chunking*. We show that our student HRNN model outperforms Compound PCFG by a significant margin and thus establishes state-of-the-art performance. We further show that our knowledge transfer approach with HRNN largely bridges the gap between supervised and unsupervised chunking.

1.4 Thesis Organization

Following this introduction, the remainder of this thesis is organized as follows:

- Chapter 2, Section 2.1.1 highlights the background of Neural Networks; Section 2.2 and 2.3 reviews related work on supervised and unsupervised syntactic strucure induction.
- In Chapter 3, we describe our three-step knowledge transfer approach for unsupervised chunking in detail.

- In Chapter 4, we first introduce the experimental setup. We then present our results on unsupervised chunking and provide detailed analyses.
- Chapter 5 concludes this thesis by summarizing the main contributions and discusses future work.

Chapter 2

Background and Related Work

2.1 Neural Networks

Neural networks are algorithms designed to learn complex patterns in data and they roughly model the human brain. These algorithms have the ability to group together unlabelled data based on their similarities. They can also be trained to classify data into different groups when given access to 'labels'. The general goal of a neural network is to approximate an unknown function $f(x) = y$ that maps an input x to an output y. It finds the right f by means of several training example pairs (x, y) such that it generalizes to unseen input x' and produces output y'.

Neural networks interpret the input data through multiple complex connections (*synapses*) between nodes (*neurons*) and transmit information (*signals*) in the form of real numbers. The connections between nodes are called edges, and they carry a weight that is adjusted as the learning proceeds. A non-linear 'activation function' acts on the output of each node in the network and this non-linearity increases the expressive power of neural networks. In other words, the activation function maps the weighted inputs to the output of the neurons. Normally, nodes are assembled in multiple layers where each layer performs transformations on its inputs. For instance, Multi-Layer Perceptrons (MLP) are a class of feed-forward neural networks with one input layer, one output layer, and at least one hidden layer. The output for each neuron is given by,

$$\boldsymbol{y} = \phi(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{2.1}$$

where $\boldsymbol{w}$ denotes the vector of weights, $\boldsymbol{x}$ is the vector of inputs and b is a bias term. ϕ denotes the non-linear activation function.

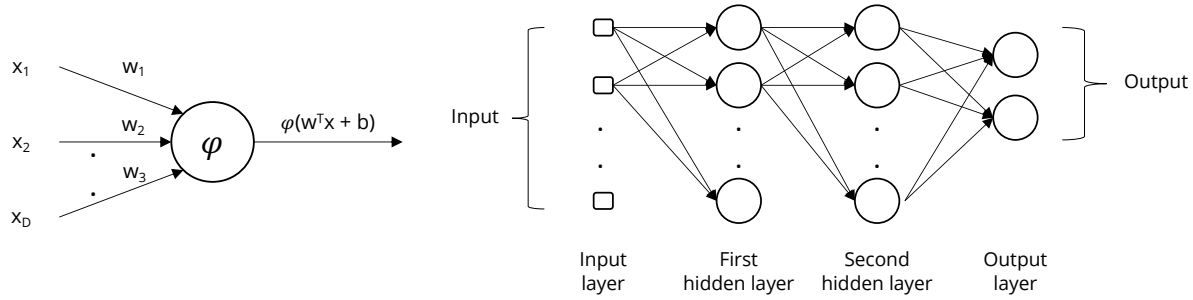


Figure 2.1: A perceptron (left) and a multi-layer perceptron (right)

With the forward pass in MLP using Equation 2.1, the decision of the output layer is recorded for all input examples in the training set. This output is then compared against the ground truth labels using a function to measure the error. In the backward pass, partial derivatives of the error function w.r.t. the various weights and biases are back-propagated through the MLP [66]. With these derivatives (gradients) in hand, the parameters (weights) are adjusted such that the loss function takes one step in the direction that minimizes the error. This can be done with any gradient-based optimisation algorithm such as stochastic gradient descent [65]. This algorithm (Backpropagation [66]) allows for the learning of the model parameters. MLPs are proven to be very expressive and can approximate any function that maps an interval of real numbers to some other interval of real numbers [26].

2.1.1 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of deep neural network designed for sequential data and is commonly used for temporal problems like machine translation, image captioning, and speech recognition. They can be distinguished from MLPs by their ability to memorize; RNNs use information from previous time steps to influence the current time step and the output.

In order to understand a language, it is essential to understand the position of each word and its role in a context. For example, let's consider the idiom, "Kill two birds with one stone", which is commonly used to refer to the completion of two tasks with a single action. For this idiom to make sense, the ordering of words is important. Hence, RNNs take into account the position of each word and use this information of the previous word to predict the next word in the sequence.

Figure 2.2 shows both the rolled and unrolled versions of RNN. Rolled visual represents

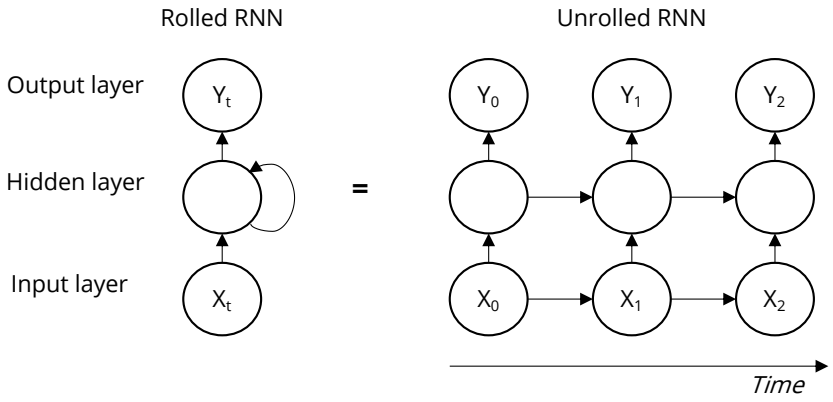


Figure 2.2: Recurrent Neural Network

the entire sequence, and unrolled visual represents the individual timesteps of the neural network. Each timestep indicates a single word, and its representation depends on all the previous timesteps. For example, the representation of the word *birds* depends on *kill* and *two* which are represented as a hidden state in the third timestep.

RNNs use backpropagation through time (BPTT) to determine the gradients in the backward pass. The core idea behind BPTT is the same as the original backpropagation algorithm [66] where errors are calculated from the output layer to its input layer. These error calculations are then used to update the parameters of the model. The only difference in BPTT is that it sums up the errors at each timestep which is not the case with MLPs as they do not share parameters across each layer.

2.1.2 Sequence to Sequence Models

The sequence-to-sequence models are a class of neural architectures that map the input sequence to the output sequence. For example, in machine translation, the goal could be to translate English sentences into French. Regular RNNs cannot be employed to map each word from one language to another. On the other hand, sequence-to-sequence models make use of two different neural networks to learn this mapping.

Broadly, sequence-to-sequence models have two parts - an encoder and a decoder. The task of an encoder network is to understand the input sequence and generate a compact representation. With such representation in hand, the decoder can generate a target sequence. The target

sequence depends on the task in focus. For example, for the task of image captioning, the output sequence is the description of an image, while for sentence reconstruction, it is simply the input sequence.

The encoder is a stack of multiple RNN units or, more often, LSTM units to avoid the problem of vanishing gradient [24]. At each timestep, the output of a node (i.e., unit) is determined by previous nodes and input for that particular timestep. The equation below gives the formula for the hidden state at tth timestep which takes h_{t-1} (previous hidden state) and x_t (current input vector) as an input. These transitions are captured in Figure 2.2.

$$h_t = f_e(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \tag{2.2}$$

Where $W^{(hh)}$, $W^{(hx)}$ are the respective weight matrices for hidden states and the input vectors. The hidden representation from the final timestep of an Encoder (h_T) is treated as the compact representation which captures the entire input sequence. This representation acts as an initial hidden state for the decoder and helps make accurate predictions at each step.

Similar to Encoders, a stack of multiple RNNs are used as Decoders which not only produce hidden state but also predict an output y_t at a each time step t. This is achieved by using the softmax function to create a probability vector over the output space. The word associated to the index with the maximum probability is chosen as the prediction.

$$h_t = f_d(W^{(hh)}h_{t-1}) \tag{2.3}$$

$$y_t = \text{softmax}(W^S h_t) \tag{2.4}$$

The above discussion covers the sequence-to-sequence model in its simplest form. There are many enhancements that further improve the neural framework's performance, making it suitable for much complex NLP tasks.

2.1.3 Transformers and Language Models

In sequence-to-sequence models, the idea behind using context vectors is to get the representation of a sentence in its entirety. In the case of long sentences, it becomes difficult to incorporate their whole meaning into fixed size context vectors. It has been shown that using a single vector of a fixed dimension for encoding entire input sequences is not capable of capturing the whole information. Bahdanau et al. [3] propose a refined technique called "attention" which improves the performance of sequence-to-sequence models on machine translation systems [3, 3].

Attention mechanism allows the sequence-to-sequence model, specifically the decoder, to focus on the relevant parts of the input sequence. There are two key differences between attention

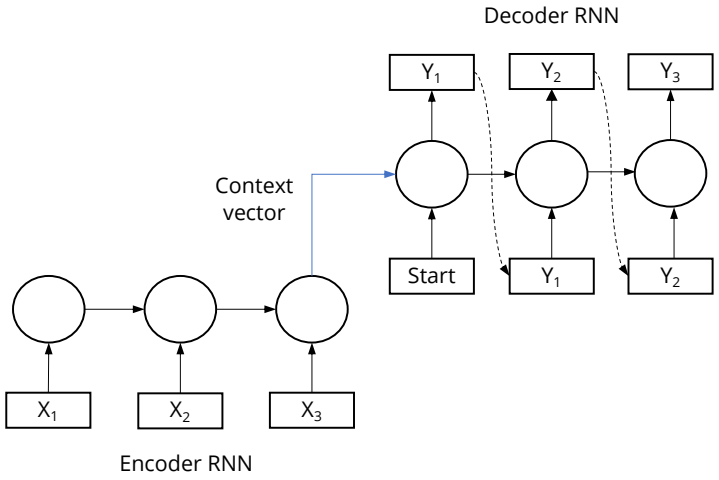


Figure 2.3: Sequence-to-sequence model

model and the vanilla sequence-to-sequence model. First, instead of passing only the last hidden state, the encoder passes all of the hidden states to the decoder. Second, at each time step, the decoder looks at all the hidden states and assigns a score. This score in turn is used to generate a context vector for that time step which is then used to make a prediction.

Let $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$ and $\boldsymbol{y} = [y_1, y_2, \cdots, y_m]$ be an input and output sequence. The hidden states of the encoder and decoder are represented by h_i and d_t respectively. The context vector is $\boldsymbol{c}_t$. The top-down process of attention mechanism as proposed by Bahdanau et al. [3] is captured in the equations below,

$$\boldsymbol{c}_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{2.5}$$

$$\alpha_{t,i} = \frac{exp(\text{score}(\boldsymbol{d}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} exp(\text{score}(\boldsymbol{d}_{t-1}, \boldsymbol{h}'_i))} \tag{2.6}$$

$$y'_t = f_1([\boldsymbol{d}_t; \boldsymbol{c}_t]) \tag{2.7}$$

The context vector is the sum of the encoder hidden states weighted by 'alignment scores'. Alignment scores are essentially importance scores assigned to each of the hidden states of the encoder, by the decoder while generating an output (Equations 2.5, 2.6). At time step t, once the context vector $\boldsymbol{c}_t$ is generated, it is simply concatenated with the decoder's hidden representation $\boldsymbol{d}_t$ and fed to a feed forward MLP to get the prediction (Equation 2.7). The attention mechanism

then parameterizes the scoring function with a feed-forward neural network [3].

$$\text{score}(\boldsymbol{d}_{t-1}, \boldsymbol{h}_i) = f_2([\boldsymbol{d}_{t-1}; \boldsymbol{h}_i]) \tag{2.8}$$

Self-attention is a straightforward extension of attention mechanism where different positions in a sequence are given varying importance while computing its representation. For this reason, it is also called 'intra-attention' and has been shown to be very useful in machine reading, abstractive summarization, and image description generation. For instance, consider the sentence *"The bird hit the windmill because it was lost"*. When the model is processing the word *it*, self-attention directs the model to focus more on the word *bird*. Refer to Cheng et al. [12] for more details on self-attention.

Transformer is the key component in recent language models [82]. It uses self-attention layers and circumvents the need for recurrent units by allowing the model to look at other positions in the input sequence while encoding a particular word. This provides more information that can help generate better word representations. Transfomers have improved language modelling on two fronts. First, using attentions boosts the training speed of the model and second, it gives state-of-the-art performance on several NLP tasks including machine translation.

The encoding component of Transformers has a stack of encoders (six in the original paper) and each of them contain two sub-layers: a self-attention layer and a feed forward layer, both followed by a normalization layer. The input to the encoder first goes through the self attention layer where it encodes specific words while attending to other words in the input sentence. To remember the order of words, the transformer also uses positional embeddings. These embeddings help the model determine the position of each word, or the distance between different words in the sequence.

The decoding component has both self-attention and feed forward layers, with an attention layer in between that helps the decoder focus on different parts of the input sentence. This layer is similar to the attention mechanism in sequence-to-sequence models. The encoder starts by processing the input sequence and the output of the top encoder is transformed into a set of attention vectors. These are to be used in each of the "encoder-decoder attention" layers.

The output of the decoder is then fed to a linear layer which converts the vector produced by the stack of decoders, into a higher dimensional vector called a logits vector. The softmax layer takes the logits vector as an input and gives out probabilities over the dictionary of words from the training corpus. The index with the highest probability and the word associated with it is produced as the output for this time step.

Language models like BERT (Bidirectional Encoder Representations from Transformers) [31] take advantage of the Transformers architecture. The key idea behind BERT is using the bidirectional training of Transformers for language modelling. Previous works parsed the sequence

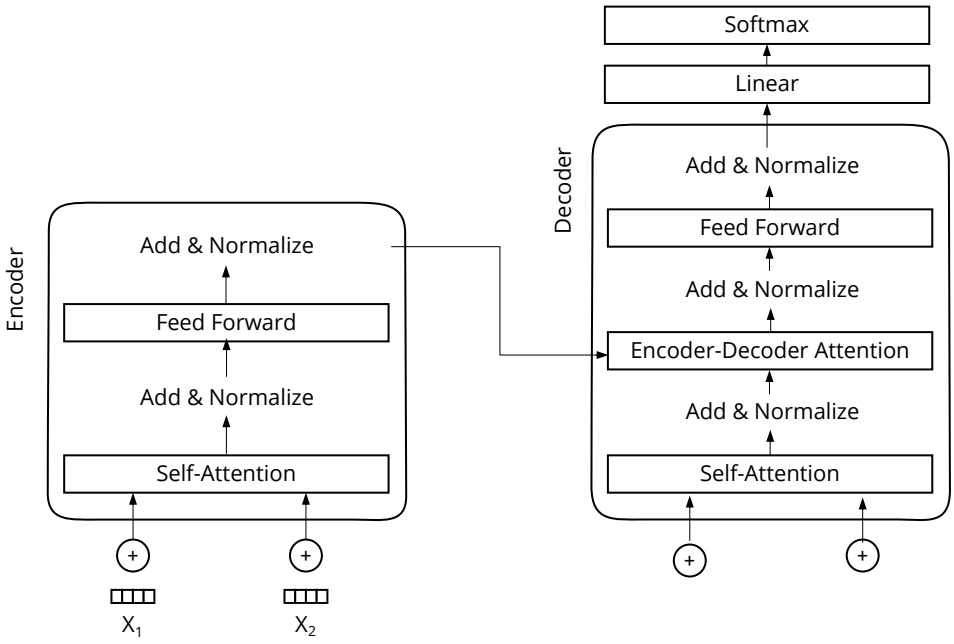


Figure 2.4: Transformer architecture with one encoder and one decoder

in a left-to-right manner or combined both left-to-right and right-to-left training. BERT shows that if the language model is trained in a bidirectional manner then it can have a deeper sense of language and context compared to single-direction language models.

Since BERT aims to improve language modelling, it makes use of only the encoder component of Transformers. It learns the context of a word based on all of its surroundings. Training BERT using only an encoder in a bidirectional manner poses a significant challenge as there is no defined prediction goal. To overcome this, BERT employs two training strategies: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP).

In MLM, 15% of the words in the input sequence are replaced by a [MASK] token before feeding them into the model. The goal of the model is to predict the masked tokens based on the context provided by the non-masked words in the sequence. To carry out such training, first, BERT adds a classification layer on top of Transformer's encoder output. Second, it multiplies the output vectors with the word embedding matrix learned so far and third, calculates the probability of each word in the vocabulary with softmax. Finally, the loss is calculated only for the prediction of the masked tokens.

In NSP, BERT is given a pair of sentences and it learns to predict whether the second sentence is subsequent to the first sentence in the original document. For this task, BERT processes the input in a very specific way. First, it inserts a [CLS] token at the beginning of the first sentence and a [SEP] token at the end of every sentence. Second, it uses sentence embeddings to distinguish between the first and the second sentence. Lastly, similar to the Transformers model, it adds a positional embedding to each token to indicate its position in the sequence. To make a prediction in the NSP task, the entire sequence (concatenation of first and the second sequence) is fed to the BERT. BERT then uses a simple classification layer to transform the output of [CLS] token into a 2×1 vector. This vector contains context information from both sentences and is used to predict the probability of *IsNextSequence*, with the help of softmax function.

BERT is then jointly trained on both MLM and NSP tasks where the goal is to minimize the combined loss function of the two training strategies.

2.2 Supervised Syntactic Structure Induction

In a natural language, formal grammar is defined by a set of production rules that cover all possible words and phrases. Context-free grammar has production rules which follow a specific form, $A \rightarrow a$ where A is a nonterminal symbol and a is a string consisting of terminal and/or nonterminal symbols. It is called context-free because a right-hand side can always substitute nonterminal on a left-hand side with any context.

Since the advent of formal grammar, linguists have described the grammars of languages in their block/chunk structures. They have also described how sentences in a language are recursively built from smaller units [25]. Consider a sentence, *The little bear saw the fat rabbit in the woods* whose logical units form a recursive structure:

[[The [little bear]] [[saw [the [fine [fat rabbit]]] [in] [the] [woods]]]]

Such logical units are also called constituents or phrases. For example, *little bear* and *fine fat rabbit* are two such smallest constituents.

A context-free grammar precisely captures the mechanism by which smaller constituents in a sentence are combined to form a larger constituent. First, they capture this block structure of a language with mathematically precise production rules. For example, the rule, ($\mathrm{NP} \rightarrow$ *the woods*) suggests that NP can be split into two terminals, namely, *the* and *woods*. Second, they can exactly describe the recursive structure of sentences with the rules like, ($\mathrm{NP} \rightarrow \mathrm{NP\ VP}$). Such recursive structures correspond to parse trees for a sentence. Below, we formally define the context-free grammar.

Definition 1. *A context-free grammar:* $G = (S, N, \Sigma, R)$ *where,*

- S *is a start symbol which represents an entire sentence.*
- N *is a finite set of non terminal symbols. Each* $n \in N$ *defines a different kind of constituent in a sentence.*
- Σ *is a finite set of terminals which is disjoint from N. This set makes up for the actual words in sentences.*
- R *is a finite set of productions (i.e., rules) contained by* $N \times (N \cup \Sigma)^*$*, where the* $*$ *denotes the kleen star operation.*

In following subsections we will discuss the use of context-free grammars for supervised parsing.

2.2.1 Cocke–Younger–Kasami Parsing

A naive approach to generate a parse tree is to search through all possibilities until we find one that obeys all the grammar rules and yields an input sentence. The sentence of length n has $n-1$ internal nodes (i.e., nodes with children). A recursion can give the number of binary parse trees with n internal nodes,

$$C_n = \sum_{i=0}^{n-1} C_{n-i} C_i = \frac{(2n)!}{(n+1)!n!} \tag{2.9}$$

Consider the example, *He was jumping over the long and tall fence* which has $C_9 = 4862$ possible untyped parse trees. Each of the internal nodes (non-terminals) in all possible C_9 parse trees must contain one of many phrase types (p), and each of the preterminal must contain one of many POS tags (s). With $p = 5$ and $s = 7$, there are $4862 \times 5^8 \times 7^9$ typed parse trees. It is not ideal to employ exhaustive search to find all valid possible parse trees for a sentence with such a large search space. Nonetheless, the Cocke–Younger–Kasami (CYK) algorithm is the first polynomial-time parsing algorithm that allows multiple derivations for the same sentence.

Cocke–Younger–Kasami (CYK) is one of the earliest parsing algorithms. Given context-free grammar and an input sentence, CYK retrieves all possible parse trees that follow the given production rules. However, this algorithm requires context-free grammar to be in a Chomsky Normal Form (CNF), which explores all the options for splitting the current sequence into two

Algorithm 1 CYK Algorithm [16, 91, 30]

```
1:  d = [d_1, d_2, ··· , d_{n−1}]                ▷ distance between every two adjacent words
2:  procedure CYK-PARSE((words,grammar))
3:      for j ← from 1 to length(words) do
4:          for all {A|A words[j] ∈ grammar do
5:              table[j − 1, j] ← table[j − 1, j] ∪ A
6:          for i ← from j − 2 downto 0 do
7:              for k ← i + 1 to j − 1 do
8:                  for all all {A — A words[j] ∈ grammar do
9:                      node ← Node(child_l, child_r)
10:     return node
```

smaller sequences. The context-free grammar is in a CNF form only if it follows the following rules,

$$\mathrm{A} \rightarrow \mathrm{B}\ \mathrm{C} \tag{2.10}$$

$$\mathrm{A} \rightarrow \alpha \tag{2.11}$$

$$\mathrm{S} \rightarrow \epsilon \tag{2.12}$$

A, B and C are non-terminals, α is a terminal and ϵ represents the empty string.

Furthermore, any context-free grammar can be converted into a CNF form with three main steps. First, terminals within binary rules are converted to dummy non-terminals resulting in rules given by Equation 2.10 and 2.11 ($A \rightarrow \alpha\, C$ is changed to $A \rightarrow X\, C, X \rightarrow \alpha$). Second, unit non-terminal to non-terminal productions are converted to a rule in Equation 2.11 ($A \rightarrow B$ is changed to $A \rightarrow \alpha, B \rightarrow \alpha$). Last, all rules are converted into binary ($A \rightarrow B\, C\, D$ is changed to $A \rightarrow Y\, D, Y \rightarrow B\, C$).

Steps one and three introduce new non-terminals into the grammar. The number of such new non-terminals depends on the children chosen to combine in step three. It is shown that in the worst case, there is a quadratic increase in the number of rules when context-free grammar is converted into an equivalent CNF.

CYK circumvents going through all binary trees by exploiting the fact that a particular binary tree is valid if its subtrees are also valid. This algorithm is a bottom-up tree-building procedure that stores all valid sub-trees. It uses a dynamic programming algorithm to hierarchically build larger sub-trees from smaller sub-trees without any re-calculations 1. Specifically, for a sentence with length n, CKY builds the upper triangular matrix of $(n+1)\times(n+1)$ whose diagonal contains POS tags. Figure 2.5 shows the entries for the first two cells and then the last cell. Each cell $[i, j]$

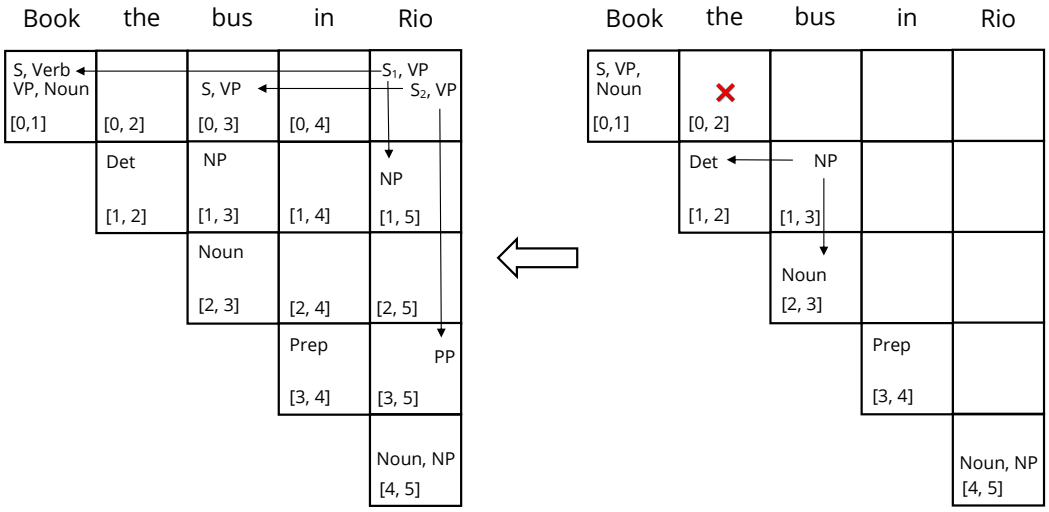


Figure 2.5: Cocke–Younger–Kasami parsing (First and second step on the right and final step on the left)

spans positions i through j of the input sentence where $i \in [0, \cdots, n-1], j \in [1, \cdots, n]$. CKY fills this table in a bottom-to-up and left-to-right fashion. At each cell, CKY checks if the contents of two other cells can be combined in a way that is approved by a grammar rule. If such a rule exists, the non-terminal on its left-hand side is entered into the table. For example, once all the diagonal entries are made, CYK looks at the cell $[0, 1]$ in the first step. For this cell there are no rules with one of $\{\mathrm{S}, \mathrm{Verb}, \mathrm{VP}, \mathrm{Noun}\}$ and *Det* on the right hand side. On the contrary, in the second step, for the cell $[1, 3]$ there exists a rule $\mathrm{NP} \rightarrow$ Det Noun and hence NP is populated.

CYK algorithm addresses the ambiguities and produces all possible parse trees. For example, in the final step there are two possible parses. In the first one, *Book* as a verb can be combined with the noun-phrase *the bus in Rio*. Second one combines *Book the bus* as a verb-phrase with *in Rio* as a prepositional-phrase. Figure 2.6 shows another example of such structural ambiguity among parse trees.

To retrieve all valid parse trees for a sentence, CYK simply starts from the start symbol and works back down through the tree. During this traversal, if multiple constituents at any cell are found then all the combinations are considered.

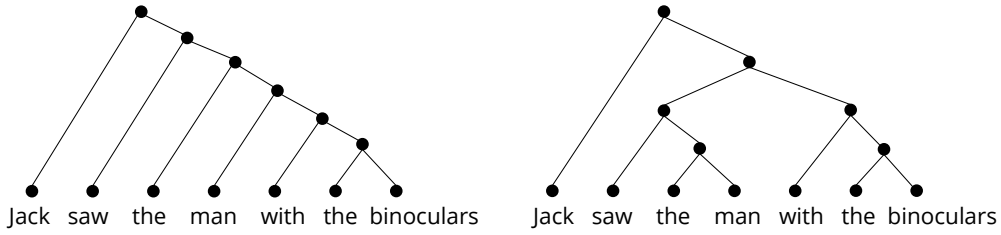


Figure 2.6: Structural ambiguity in constituency parsing

2.2.2 Probabilistic Context Free Grammars for Parsing

CKY algorithm can capture ambiguities and produce all possible parse trees but cannot resolve them, i.e., it cannot find the most likely parse tree for a sentence. On the other hand, probabilistic context-free grammars (PCFGs) can associate probabilities to all possible parse trees. Then, single best parse tree with the maximum probability is chosen. PCFG is defined below.

Definition 2. *A PCFG is:*

- *A context-free grammar* $G = (S, N, \Sigma, R)$
- *and a parameter* $q(r)$ *for each rule* $r \in R$*.*

This parameter $q(r)$ is nothing but the conditional probability of choosing a rule r given a fixed non-terminal being expanded. For example, $q(\text{ NP} \rightarrow$ *the bank*$)$ is a probability of choosing a rule, $\text{NP} \rightarrow$ *the bank* given LHS of the rule is NP. This parameter $q(r)$ is the only addition to the previous definition of context-free grammar and allows us to have a probability distribution over all possible derivations. Due to the new parameter, for any $X \in N$, we have the constraint,

$$\sum_{A \rightarrow B \in R \,:\, A = X} q(A \rightarrow B) = 1 \tag{2.13}$$

Also, $q(A \rightarrow B) \geq 0 \; \forall A \rightarrow B \in R$. Meaning, for any non-terminal X, the rule probabilities with X on the left-hand side must sum to one (and each of them must be non-negative). Moreover, given a parse tree, $t \in \mathcal{T}_G$, the probability associated with that tree is simply the product of all the rules used in its derivation, $A_1 \rightarrow B_1, \cdots, A_i \rightarrow B_i$.

$$p(t) = \prod_{i=1}^{n} q(A_i \rightarrow B_i) \tag{2.14}$$

where $\mathcal{T}_G$ is the set of all possible parse trees under the grammar G. Given a sentence s and $p(t) \, \forall \, t \in \mathcal{T}_G$ we can retrieve the best possible parse tree,

$$t_{best} = \underset{t \in \mathcal{T}_G(s)}{\arg\max} \, p(t) \tag{2.15}$$

There are two important details that are still unanswered. First, how do we get hold of $q(r) \, \forall \, r \in R$?

Assuming an access to the training set $T = \{(t_1, s_1), \cdots, (t_N, s_N)\}$ we can simply count the number of occurrences of different rules to get $q(r)$ as given in Equation 2.16.

$$q(A \rightarrow B) = \frac{\text{Count}(A \rightarrow B)}{\text{Count}(A)} \tag{2.16}$$

Second, how do we get the most likely tree given in Equation 2.15?

PCFG uses an updated CKY algorithm, saving the probabilities for every cell in the chart. Specifically, constituents in each cell has a third dimension of maximum length $|N|$ where N denote the non-terminals. Similar to CKY, each cell $[i, j, X]$ in this $(n+1) * (n+1) * |N|$ matrix carries the probability of a constituent of type X that spans positions i through j of the input.

While filling the table, at each step, the most likely subtree is recorded. Consider a rule, $\mathrm{A} \rightarrow \mathrm{BC}$ which is satisfied at a particular cell $[i, k]$. Then there must be a subtree rooted at $[i, j]$ (labeled B) and $[j, k]$ (labeled C). Let's denote the probability of the most likely subtree rooted at $[i, k]$ as $p((i, k))$ by the following equation,

$$p_A(i, k) = \max_{j : i < j < k} q(\mathrm{A} \rightarrow \mathrm{BC}) \, p_B(i, j) \, p_C(j, k) \tag{2.17}$$

Where $q(\mathrm{A} \rightarrow \mathrm{BC})$ is simply a transition probability associated with that particular rule. The probability on the left-hand side is nothing but the product of two sub-components times the probability of the rule that joints them to form a constituent A. Here, j acts as a valid split point and recording all such js results in different structures for a parse tree.

The probability of a most likely parse ($p_A(0, |w|)$) is given by the last cell. Finding the most likely parse is straightforward once each entry $p_A(i, k)$ is associated with "back pointers" to the $p_B(i, j)$ and $p_C(j, k)$ that maximize Equation 2.17.

2.2.3 Supervised Chunking

Chunking aims to find non-hierarchical and non-overlapping phrases in a text. Ramshaw and Marcus [61] proposed to solve the task of chunking using machine learning methods. Since

then, there have been various machine learning techniques [39, 28, 37]. In 2002, chunking was extended to the CoNLL-2000 task with standard dataset and evaluation metrics [67].

Supervised Machine Learning approaches for chunking utilize chunk labels from datasets like CoNLL-2000 [67] and give relatively high performance. These methods can be divided into two groups. First, the generative modeling, which commonly uses probabilistic graphical models. These models assign a joint probability $p(X, Y)$ to input sentence (X) and the labels (Y). Such approaches provide a direct relationship between words and their role on the phrase level. Hidden Markov Model (HMM) is one such approach that is widely used for chunking [51] and other parallel tasks like NER, POS tagging etc [94, 22]. Second, the conditional modeling, where chunking is looked at as a sequence of classification problems. Conditional Random Fields (CRFs) fall into this group which aims to learn the probability of label sequence given an input sentence — $p(Y|X)$ [40]. Such modeling gives a better performance for chunking since CRFs can handle arbitrary and more global features [43].

HMMs impose some independence assumptions which restrict each word to depend only on its label and each label to depend only on the previous label. Ideally, predicting chunks would benefit from the long-range dependencies and the context beyond the surrounding words [40]. While dependencies in HMMs are local, CRFs use more global features. See Figure 2.7 for their graphical structure.

There are many works which exploit the advantages offered by CRFs for the task of chunking. With recent advances in language modeling, using their contextual word embeddings have become more prominent for various tasks [57, 2, 31]. These embeddings encompass the sentence-level information, and using them with chain CRFs have shown to give a state-of-the-art performance on sequence labeling tasks [88]. Another straightforward extension is to employ neural networks instead of linear potential functions in CRFs. Such approach gives a further improvement on supervised chunking [73]. Likewise, Shah et al. [73] proposes to use locally-contextual and nonlinear CRFs for the task of chunking.

There exist works which demonstrate the effectiveness of concatenating different kinds of word embeddings (contextualized/non-contextualized, word-level/character-level) [81]. Although effective, finding the best mix of embeddings is non-trivial and difficult. To solve this issue, recent work by Wang et al. [84] frames this problem as Neural Architecture Search (NAS). Furthermore, by taking advantage of CRFs, this work establishes a new state-of-art-performance for supervised chunking on CoNLL-2000 task [67].

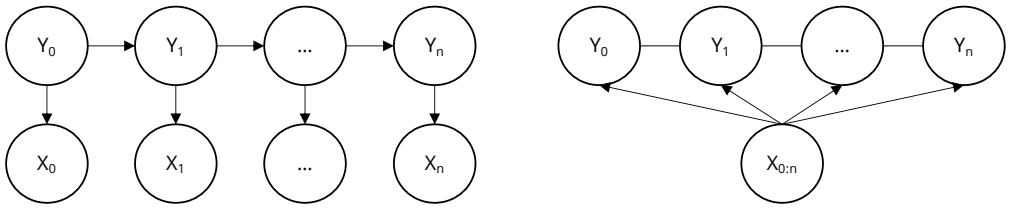


Figure 2.7: Graphical structure of HMMs (left) and chain CRFs (right)

2.3 Unsupervised Syntactic Structure Induction

Unsupervised syntactic structure detection has attracted much attention in early NLP research because of its use in low resource scenarios [15, 34]. Klein and Manning [35] propose to model constituency and context for each spans with an Expectation–Maximization (EM) algorithm. Recent work also focused on unsupervised dependency parsing for syntactic structure induction [72, 54]. Klein and Manning [36] combine constituency and dependency models via co-training to further boost their performance.

To learn the syntactic structures, Haghighi and Klein [23] propose a probabilistic context-free grammar (PCFG), augmented with manually designed features. Reichart and Rappoport [63] perform label clustering using syntactic features to obtain labeled parse trees. Clark [15] clusters sequences of tags based on their local mutual information to build parse trees. Such early studies typically used heuristics, linguistic knowledge, and manually designed features for unsupervised syntactic structure induction [87, 35, 15].

Unsupervised parsing has been attracting increasing attention in recent years due to advances in deep learning based NLP. Most of these works are trained on a downstream task that provides indirect feedback to the parsing actions. Next word prediction, sentence reconstruction are some of the most commonly employed tasks for such indirect supervision. Recent works have also exploited visual knowledge to understand the subject-object relations. In this section, we briefly discuss such recent advances in unsupervised syntactic structure induction.

Socher et al. [79] propose the recursive autoencoder, where a binary tree is built by greedily minimizing the reconstruction loss. Such recursive tree structures can also be learned in an unsupervised way by CYK-style marginalization [49] and Gumbel-softmax [13]. Yogatama et al. [90] learn a shift-reduce parser by reinforcement learning towards a downstream task. However, evidence shows the above approaches do not yield linguistically plausible trees [85]. Shen et al. [74] propose to model the syntactic distance or syntactic ordering and recently also propose to

induce constituency and dependency structures from the text jointly [77]. Kim et al. [33] propose a Compound PCFG for unsupervised parsing [1]. The trees given by these approaches are more correlated with constituency trees.

Parsing Reading Predict Network (PRPN) [74] is trained on a language modeling objective, precisely, to predict a next word based on all previous words. It is one of the early works that uses syntactic distances to build parse trees and had three main components. First, a differentiable parsing network uses a convolutional neural network to generate syntactic distances between words. Second, a reading network computes a representation for every token based on syntactically closer previous tokens. This is achieved through the attention mechanism. Third, a predict network generates the next token based on the representation from the reading network.

Although PRPN gives a state-of-the-art performance on unsupervised parsing, it relies on ad-hoc heuristics and does not explicitly handle tree-building operations. To this end, Li et al. [45] propose to transfer knowledge from PRPN to a Tree-LSTM with discrete parsing actions. Such a student-teacher model whose policy is updated via straight-through Gumbel-Softmax obtains better performance. Our work is inspired by such knowledge transfer, but we propose insightful heuristics to induce chunk labels from unsupervised parsers.

Natural language is thought of as a set of nested constituents forming a tree structure. In these tree structures, when a larger constituent ends, all nested smaller constituents also end. Shen et al. [76] propose Ordered Neurons LSTM (ON-LSTM), which explicitly injects such inductive bias into a recurrent neural network. Specifically, the inductive bias is to have distinct life cycles for the memory of each neuron. That is, when a higher-ranking neuron is erased, all lower-ranking neurons are also erased. Shen et al. [76] also propose a new activation function called cumulative softmax to have this distinction between high-ranking and low-ranking neurons. ON-LSTM is trained on the language modeling objective, and then the best model is simply taken to perform the unsupervised constituency parsing using the distance-to-tree Algorithm 2.

Drozdov et al. [18] propose a Deep Inside Outside Recursive Auto-Encoder (DIORA) that incorporates the inside-outside algorithm [4] into auto-encoder and CKY parser framework. The key motivation behind exploiting the auto-encoder framework is the substitution principle that says, *"model can best reconstruct the input by discovering and exploiting syntactic regularities of the text"*. The inside pass is used to calculate the representations of all possible constituents. While doing so, DIORA only looks at the descendant constituents within that sub-tree. For hierarchically grouping words together, it is also important to look at the surrounding words in the sentence. Keeping this in mind, in an outside pass, DIORA considers the outside context for every constituent. The training objective is to reconstruct the corresponding input word by

[1] The details of Compound PCFG are reserved for the section 3.2.

using the outside context of that particular word. Once the inside and outside representations are learned, the single most likely tree is recovered using CKY.

There exist recent works which are aimed at gaining indirect supervision from the natural images and corresponding captions. Visually Grounded Neural Syntax Learner (VG-NSL) is one such approach [78]. VG-NSL recursively composes representation for constituents and then matches it with the visual representation. The main hypothesis behind VG-NSL is that better alignment between vision and language would lead to better syntactic structure. Zhao and Titov [93] propose multi-modal Compound PCFG, which effectively aggregates features from different modalities like scene, audio, face, speech, etc. Such aggregation gives a significant advantage over Compound PCFG [33].

As discussed in this section so far, previous work mainly focuses on unsupervised constituency parsing for syntactic structure discovery. There exist few studies which address unsupervised chunking as an important task in speech processing; they use acoustic information to determine the chunks [55, 6]. The work by Pate and Goldwater [55] considers cues like prosody, which is the structure of speech conveyed through rhythm and intonation. The idea behind this stems from the previous work on *prosodic bootstrapping hypothesis* which suggest the utility of prosodic cues for syntactic structure induction in the text [2]. Barrett et al. [6] use the records of eye-tracking, speech, and keystroke logs to aid the task of unsupervised POS tagging. On the other hand, our work only considers textual information and views unsupervised chunking as a new task of syntactic structure induction.

[2]Prosodic cues essentially define the larger units of speech like intonation, rhythm and are not strictly related to the phonetic segments.

Chapter 3

A Knowledge-Transfer Approach to Unsupervised Chunking

In this section, we first outline the task of unsupervised chunking and then discuss our three-step knowledge transfer approach in detail. We define Compound PCFG (our teacher model) in Section 3.2. We introduce our maximal left-branching chunking heuristic in Section 3.3 where goal is to induce chunk labels from the parse trees generated by Compound PCFG. Section 3.4 discusses an HRNN (our student model) that learns from these induced labels. Figure 3.1 gives an overall picture of our knowledge-transfer approach.

Our further exploration of two ideas is discussed in next sections. Section 3.5 covers the use of employing self-supervised learning objectives for the task of unsupervised chunking. Section 3.6 introduces our approach for transferring the chunking knowledge back to parsing in an unsupervised setting.

3.1 Problem Formulation

Chunking groups the words of a sentence into chunks/phrases in a non-hierarchical and non-overlapping fashion. The unsupervised setting detects chunk labels without the supervision of annotated linguistic structures. Chunk labels then follow a BI schema [61] where "B" refers to the beginning of a chunk and "I" refers to the inside of a chunk.

The CoNLL-2000 dataset [67] is labeled with the BIO schema, where "O" indicates outside of a chunk. Tokens outside any chunk are mostly punctuation signs, and in rare cases, they are conjunctions in common coordinated phrases. For example, in *"He was named president and*

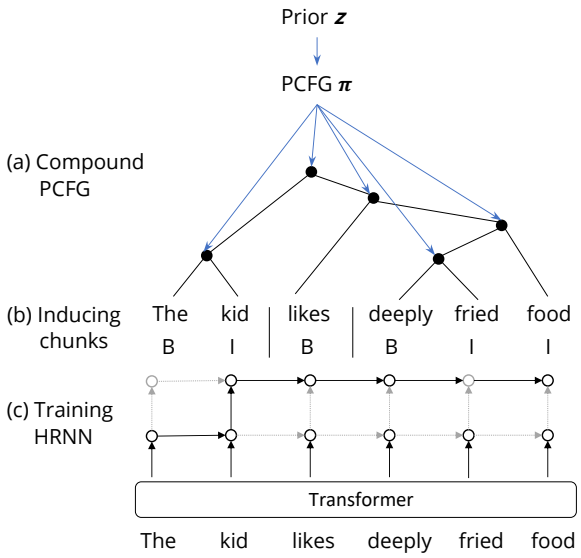


Figure 3.1: Overview of our approach: (a) describes our Compound PCFG teacher model. (b) denotes the use of chunking heuristic to induce the chunk labels from parse trees. (c) is our Hierarchical RNN model that switches between word-level and phrase-level RNN.

chief operating officer," the word "and" will have "O" as its chunk label. We follow the BI schema and ignore the "O" tokens in our task formulation. Consequently, our model will predict whether the word is inside of a chunk or a beginning of a chunk.

3.2 Teacher Model: Compound PCFG

We propose to induce chunk labels from state-of-the-art unsupervised parsers. The intuition is that the chunking structure can be thought of as a flattened parse tree and thus should agree with the parsing structure to some extent. Our knowledge transfer approach then takes advantage of recent advances in unsupervised parsing; especially, we adopt Compound Probabilistic Context-Free Grammar (Compound PCFG) [33].

It is challenging to learn probabilistic context-free grammars from natural language data through direct methods like optimizing the log-likelihood with the EM algorithm [41]. Compound PCFGs solve this problem by parameterizing PCFG's rule probabilities with neural networks. Moreover, they incorporate sentence-level latent vector into rule probabilities, offering

a context of longer-range dependencies. This makes Compound PCFGs more expressive than PCFGs and helps them to induce linguistically meaningful grammar accurately. We now define Compound PCFG,

Definition 3. *Compound PCFG is a 5-tuple grammar* $\mathcal{G} = (\mathcal{S}, \mathcal{N}, \mathcal{P}, \Sigma, \mathcal{R})$ *where,*

- $\mathcal{S}$ *is a start symbol*
- $\mathcal{N}$*,* $\mathcal{P}$*,* Σ *are finite set of nonterminal, preterminal and terminal symbols respectively*
- *sentence-level rule probabilities are given by* $\boldsymbol{\pi} = \{\pi_{\boldsymbol{z},r}\}_{r \in \mathcal{R}}$ *where* $\boldsymbol{z}$ *is a sentence-level latent vector and* $\mathcal{R}$ *is a finite set of rules taking one of the following forms:*
  - $\mathcal{S} \rightarrow A \qquad A \in \mathcal{N}$
  - $A \rightarrow B\ C \qquad B, C \in \mathcal{N} \cup \mathcal{P}$
  - $T \rightarrow w \qquad T \in \mathcal{P}, w \in \Sigma$
- *It is parameterized by* $\theta \in \{\boldsymbol{z}, \boldsymbol{\lambda}, \mathcal{E}_{\mathcal{G}}\}$ *where* $\boldsymbol{\lambda}$ *denotes the parameters of its neural networks and* $\mathcal{E}_{\mathcal{G}}$ *is the set of all input and output symbol embeddings.*

$S \rightarrow A$ is the start of a sentence and $T \rightarrow w$ indicates the generation of a word. $A \rightarrow B\ C$ models the bifurcations of a binary constituency tree, where a constituent node is not explicitly associated with a type (e.g., noun phrase).

The rule probabilities in Compound PCFG are parameterized by a neural network $f_{\lambda \in \boldsymbol{\lambda}}$. To achieve this, first, the embeddings are associated to both input and output symbols. For input symbol N on the left hand side of the rule, the embeddings are given by $\boldsymbol{w}_N$ where $N \in S \cup \mathcal{N} \cup \mathcal{P}$. Similarly, output symbol embeddings are given by $\boldsymbol{u}_M$ where both $\boldsymbol{w}_S, \boldsymbol{u}_A \in \mathcal{E}_{\mathcal{G}}$. Second, the input symbol embeddings are concatenated with $\boldsymbol{z}$ and then given as an input to the neural network. For instance, the rule probability for a rule $\mathcal{S} \rightarrow A$ is given by Equation 3.1.

$$\pi_{\boldsymbol{z}, \mathcal{S} \rightarrow A} = \frac{\exp(\boldsymbol{u}_A^T\, f_{\lambda \in \boldsymbol{\lambda}}([\boldsymbol{w}_S; \boldsymbol{z}]) + b_A)}{\sum_{A' \in \mathcal{N}} \exp(\boldsymbol{u}_{A'}^T\, f_{\lambda \in \boldsymbol{\lambda}}([\boldsymbol{w}_S; \boldsymbol{z}]) + b_{A'})} \tag{3.1}$$

where $[\boldsymbol{w}_S; \boldsymbol{z}]$ denotes vector concatenation. To produce a rule probability, the output of a neural network is multiplied with the output symbol embeddings $\boldsymbol{u}_A^T$ and then fed to a softmax function. Similar to Equation 3.1, $\pi_{\boldsymbol{z}, A \rightarrow B\ C}$ and $\pi_{\boldsymbol{z}, \mathcal{T} \rightarrow w}$ can be defined for two other respective rule types, $A \rightarrow B\ C$ and $T \rightarrow w$.

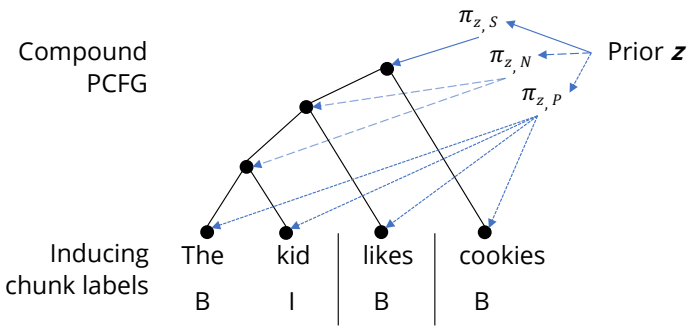


Figure 3.2: Teacher model: Compound PCFG

During training, Compund PCFG maximizes the log marginal likelihood of the observed sentence $\log p_\theta(\boldsymbol{x})$[1]. The log marginal likelihood is given by the Equation 3.2 where $\boldsymbol{z} \sim p_\gamma(\boldsymbol{z})$. Such prior defines the Compound PCFG's distribution over trees since $\boldsymbol{z}$ affects the rule probabilities as shown in Equation 3.1. While achieving this optimization goal Compound PCFG learns the sentence-level rule probabilities ($\boldsymbol{\pi}$).

$$\log p_\theta(\boldsymbol{x}) = \log(\int p_\theta(\boldsymbol{x}|\boldsymbol{z}) p_\gamma(\boldsymbol{z}) d\boldsymbol{z}) \tag{3.2}$$

For every sentence, these learnt rule probabilities are then fed to Viterbi-like CYK algorithm to obtain the corresponding parse tree. With parse trees in hand, we now can induce chunk labels from Compound PCFG.

3.3 Parsing-to-Chunking Heuristics

We propose a simple yet effective heuristic that extracts maximal left-branching subtrees as chunks. As known, the English language is strongly biased to right-branching structures [85, 45]. For instance, the phrases like ***laugh*** *loudly* and ***talk*** *softly* which are head-initial are more common than phrases like *the* ***house*** and *very* ***sad*** which are head-final. We observe, on the other hand, that a left-branching structure typically indicates closely related words.

Specifically, a *left-branching* subtree means that the words are grouped in the form of $((\cdots((\mathrm{x}_i\mathrm{x}_{i+1})\mathrm{x}_{i+2})\cdots)\mathrm{x}_{i+n-1})$. A left-branching subtree for words $\mathrm{x}_i \cdots \mathrm{x}_{i+n-1}$ is *maximal* if

[1]Compound PCFG relies on an amortized variational inference to tractably perform the computation of $\log p_\theta(\boldsymbol{x})$. We refer readers to Kim et al. [33] for details.

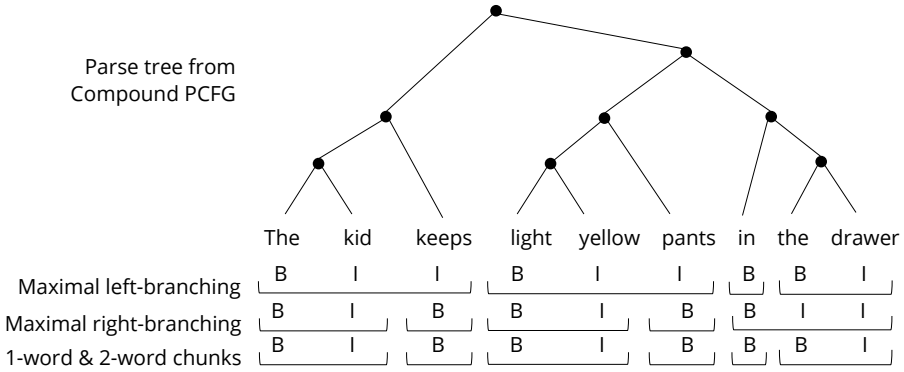


Figure 3.3: Comparison between different chunking heuristics

neither $\mathrm{x}_{i-1}\mathrm{x}_i \cdots \mathrm{x}_{i+n-1}$ nor $\mathrm{x}_i \cdots \mathrm{x}_{i+n-1}\mathrm{x}_{i+n}$ is left-branching. We extract all maximal left-branching subtrees as chunks. In Figure 3.3, for example, "light yellow pants" is a three-word maximal left-branching subtree $\curlywedge$, whereas "the kid" and "likes" are also maximal left-branching subtrees (although degenerated). Our heuristic treats them as chunks. For better understanding, Figure 3.3 also shows the contrast between the maximal left-branching heuristic and the maximal right-branching heuristic. The following theorem shows that our heuristic can unambiguously give chunk labels for any sentence with any parse tree.

Theorem 1. *Given any binary parse tree, every word will belong to one and only one chunk by the maximal left-branching heuristic.*

Proof. [Existence] A single word itself is a left-branching subtree, which belongs to some maximal left-branching subtree.

[Uniqueness] We will show that two different maximal left-branching subtrees $\mathbf{s}_1$ and $\mathbf{s}_2$ cannot overlap. Assume by way of contradiction that there exists a word x_i in both $\mathbf{s}_1$ and $\mathbf{s}_2$. Then, $\mathbf{s}_1$ must be a substructure of $\mathbf{s}_2$ or vice versa; otherwise, the paths $\mathrm{root}\text{–}\mathrm{s}_1\text{–}\mathrm{x}_i$ and $\mathrm{root}\text{–}\mathrm{s}_2\text{–}\mathrm{x}_i$ violate the acyclic nature of a tree. But $\mathbf{s}_1$ being a subtree of $\mathbf{s}_2$ (or vice versa) contradicts with the maximality of $\mathbf{s}_1$ and $\mathbf{s}_2$. □

This easy theorem shows our maximal left-branching heuristic can unambiguously give chunk labels for any sentence with any binary parse tree. It should also be mentioned that our simple heuristic achieves reasonable chunking performance, although it is noisy. Nevertheless, our HRNN learning (discussed in the next part) can smooth out such noise and yield more meaningful chunks.

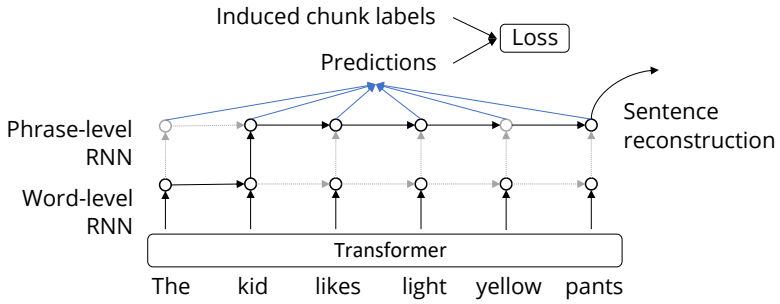


Figure 3.4: Student Model: Hierarchical RNN

3.4 Student Model: Hierarchical RNN

We would like to train a machine learning model to learn from the Compound PCFG-induced chunk labels. Our intuition is that a learning machine pools the knowledge of different samples into a parametric model, and thus may smooth out the noise of our heuristics.

Specifically, we train a neural network to predict the chunks. This is accomplished by running Compound PCFG on an unlabeled corpus to obtain chunk labels in the BI schema [61]. Then, a machine learning model (e.g., a neural network) will learn from the pseudo-ground-truth labels.

We observe that a classic RNN or Transformer may not be suitable for the chunking task, because the prediction at a time step is unaware of previous predicted chunks, thus lacking autoregressiveness. Feeding predicted chunk labels like a sequence-to-sequence model is not adequate, because a BI label only contains one bit information and cannot provide full autoregressive information either.

To this end, we design a hierarchical RNN to model the autoregressivenss of predicted chunks by altering the neural structure. Our HRNN contains a lower word-level RNN and an upper chunk-level RNN. We also design a gating mechanism that switches between the two RNNs in a soft manner, also serving as the chunk label.

Let $\mathrm{x}^{(1)}, \cdots, \mathrm{x}^{(n)}$ be the words in a sentence. We first apply the pretrained language model BERT [31] to obtain the contextual representations of the words, denoted by $\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n)}$. This helps our model to understand the global context of the sentence. For a step t, we first predict a switching gate $m^{(t)} \in (0, 1)$ as the chunking decision.[2]

$$m^{(t)} = \sigma(W[\underline{\boldsymbol{h}}^{(t-1)}; \overline{\boldsymbol{h}}^{(t-1)}; \boldsymbol{x}^{(t)}]) \tag{3.3}$$

[2] $m^{(t)} = 1$ corresponds to "B," i.e., a new chunk.

where $\underline{\boldsymbol{h}}^{(t-1)}$ is the hidden state of the lower RNN and $\overline{\boldsymbol{h}}^{(t-1)}$ is that of the upper RNN. Semicolon represents vector concatenation, and σ represents the sigmoid function.

To provide autoregressiveness, this gate is also used to control the information flow by altering the network architecture, shown in Figure 3.4.

Suppose our model predicts that the tth word is the beginning of a chunk. This essentially "cuts" the sequence into two parts at this step. The lower RNN and upper RNN are updated by

$$\underline{\boldsymbol{h}}_{\text{cut}}^{(t)} = \underline{f}\left(\boldsymbol{x}^{(t)}, \underline{\boldsymbol{h}}^{(\text{sos})}\right) \tag{3.4}$$

$$\overline{\boldsymbol{h}}_{\text{cut}}^{(t)} = \overline{f}\left(\underline{\boldsymbol{h}}^{(t-1)}, \overline{\boldsymbol{h}}^{(t-1)}\right) \tag{3.5}$$

where $\underline{f}$ and $\overline{f}$ are the transition functions of the two RNNs, respectively.

Here, the lower RNN ignores its previous hidden state but restarts from a learnable initial state $\underline{\boldsymbol{h}}^{(\text{sos})}$, due to the prediction of a new phrase. The upper RNN picks the newly formed phrase with representation $\underline{\boldsymbol{h}}^{(t-1)}$ captured by the lower RNN, and fuses it with the previous upper state $\overline{\boldsymbol{h}}^{(t-1)}$.

Suppose our model predicts that the tth word is not the beginning of a chunk, i.e., "no cut" is performed at this step. The RNNs are updated by

$$\underline{\boldsymbol{h}}_{\text{nocut}}^{(t)} = \underline{f}\left(\boldsymbol{x}^{(t)}, \underline{\boldsymbol{h}}^{(t-1)}\right) \tag{3.6}$$

$$\overline{\boldsymbol{h}}_{\text{nocut}}^{(t)} = \overline{\boldsymbol{h}}^{(t-1)} \tag{3.7}$$

Here, the lower RNN updates its hidden state with the input $\boldsymbol{x}^{(t)}$ as a normal RNN, whereas the upper RNN is idle because no phrase is formed.

The "cut" and "nocut" cases can be unified by

$$\overline{\boldsymbol{h}}^{(t)} = m^{(t)}\, \overline{\boldsymbol{h}}_{\text{cut}}^{(t)} + \left(1 - m^{(t)}\right) \overline{\boldsymbol{h}}_{\text{nocut}}^{(t)} \tag{3.8}$$

$$\underline{\boldsymbol{h}}^{(t)} = m^{(t)}\, \underline{\boldsymbol{h}}_{\text{cut}}^{(t)} + \left(1 - m^{(t)}\right) \underline{\boldsymbol{h}}_{\text{nocut}}^{(t)} \tag{3.9}$$

In fact, we keep $m^{(t)}$ as a real number and fuse the lower RNN and upper RNN in a soft manner. This is because chunking by its nature may be ambiguous, and our soft gating mechanism is able to better preserve the information.

3.5 Self-Supervised Fine-Tuning Objectives

We propose to employ our HRNN chunker in a downstream task. We hypothesize that sentence reconstruction may aid the task of chunking since the former discovers the syntactic regularities

of the text. This hypothesis is roughly based on the substitution principle of the language [42] which says that finding and exploiting syntactic regularities of the text is the best way to reconstruct the input. In this section, we discuss such a self-supervised fine-tuning objective for the task of chunking.

We employ an autoencoder framework for sentence reconstruction where HRNN acts as both a chunker and an encoder. We use one-layer LSTM as the decoder. For a step t, the hidden representation of both word-level and phrase-level RNN is used to predict the chunk labels (as given in Equation 3.3). The representation from a final state of the phrase-level RNN is given as an input to the decoder; we condition the reconstruction of a sentence on a single $z = \overline{\boldsymbol{h}}^{(N)}$ where N corresponds to the length of a sentence. We explore different training objectives for our auto-encoding framework.

Cross Entropy. The model is trained with the objective of sentence reconstruction. The representation, z, is then decompressed and trained to reconstruct the original sentence. To approximate the reconstruction, we use the cross-entropy loss. The overall loss for this framework is given by,

$$\mathcal{L} = \alpha \, \mathcal{L}_{\text{chunk}} + (1 - \alpha) \, \mathcal{L}_{AE} \tag{3.10}$$

Where $\mathcal{L}_{chunk}$ is a cross-entropy loss between predicted chunk labels from HRNN (student model) and the pseudo ground truth labels from the Compound PCFG (teacher model), $\mathcal{L}_{AE}$ is a cross-entropy loss for the auto-encoding, and α controls the trade-off between two losses.

Reinforcement Learning (RL). Using cross-entropy loss to maximize the alignment between the predicted sentence and the input sentence is a straightforward approach. However, the performance of such auto-encoding is evaluated by discrete metrics like BLEU, which measure the n-gram overlap between the two sentences [53]. Although training sequence-to-sequence models to directly optimize metrics like BLEU is desirable, it is difficult due to its non-differentiable nature. We use REINFORCE algorithm, which overcomes the issues associated with the discrete nature of the optimization by not requiring rewards to be differentiable [86].

In our framework, HRNN can be looked at as an *agent* which interacts with the *environment*. The neural parameters of our *agent* would determine a *policy*: a conditional probability $p(y|x)$ where y denotes chunk labels for a sentence x. At every time step, the *agent* will pick an *action* i.e., choosing a chunk label for a word. A reward will be observed when our agent predicts chunk labels for all words and the sentence is fully reconstructed by the decoder. The *reward* for our sentence reconstruction is the BLEU score between generated sentence and the input sentence.

The goal of this RL training is to maximize the expected reward,

$$\mathcal{L}_{RL} = \sum_{i=1}^{M} E_{\hat{y}_i \sim p(\hat{y}_i|x_i)} R(\hat{x}_i, x_i) \tag{3.11}$$

where $\hat{x}_i$ is ith reconstructed sentence and M is the number of sentences. There exist few implementation tricks for solving some of the issues with the RL objective. First, REINFORCE is used to approximate the expectation give in Equation 3.11 by sampling $\hat{y}$ from the policy $p(y|x)$. Second, reward shaping is used to make up for the sparsity of rewards during RL training. It solves this problem by creating K copies of an input sentence and sampling actions from the policy for all of them. This results in different rewards for each copy of the input sentence. Third, to reduce the high variance in gradient estimation, for every ith sentence, we subtract an average reward $\bar{r}_i$ from all K rewards. Above three implementation tricks lead to the RL objective as maximizing,

$$\hat{\mathcal{L}}_{RL} = \sum_{i=1}^{M} \sum_{j=1}^{K} (R(\hat{x}_{ij}, x_{ij}) - \bar{r}_i) \, \text{CE}(\hat{y_{ij}}, p(\hat{y}_i|x_i)) \tag{3.12}$$

where CE denotes the cross-entropy loss. The last important strategy that we employ is to combine the chunking loss with RL objective to further stabilize the training process. We refer readers to Ranzato et al. [62] for the detailed derivation of Equation 3.12.

$$\mathcal{L} = \beta \, \mathcal{L}_{\text{chunk}} + (1 - \beta) \, \hat{\mathcal{L}}_{RL} \tag{3.13}$$

where β is the hyperparameter controlling the weight given to each of MLE and RL objectives.

Gumbel-softmax. REINFORCE suffers from high variance and has bad empirical performance in certain NLP tasks [48]. In chunking, the reward needs to influence the discrete chunking decisions directly. Gumbel-softmax is a simple alternative to REINFORCE, which provides a differentiable approximation to the discrete sample [48]. It also allows backpropagating the loss directly from the reward to the model's parameters instead of using REINFORCE.

Let $\boldsymbol{z}$ be a categorical variable with categorical distribution (p_1, p_2), which corresponds to the probability of having no chunk and chunk, respectively. The Gumbel-Max trick provides the following formula for sampling a chunk label,

$$\boldsymbol{z} = \text{onehot}(\text{argmax}_i(G_i + \log(p_i))) \tag{3.14}$$

where $G_i \sim \text{Gumbel}(0, 1)$ are independent and identically distributed samples from the Gumbel distribution [48]. This process is simply refactoring a sampling process into a deterministic function and the independent noise. Such refactoring allows for backpropagation since we simply

have to compute the gradient w.r.t. parameters of a deterministic function. On the other hand, argmax still exists in the above reparameterization technique, and the sampling process remains non-differentiable. To solve the non-differentiability issue, softmax is used in place of argmax,

$$\hat{\boldsymbol{z}} = \frac{e^{\frac{G_i + log(p_i)}{\tau}}}{\sum_j e^{\frac{G_j + log(p_j)}{\tau}}} \tag{3.15}$$

where for both $i = 1$ and $i = 2$, the τ (temperature paramater) controls how closely new samples approximate the original discrete, one-hot vectors. For instance, as $\tau \to 0$, the formulation in Equation 3.15 approaches the argmax in a soft manner (e.g., $[0.99, 0.01]$) and when $\tau \to \infty$, sample vectors become uniform (e.g., $[0.5, 0.5]$).

The distribution with the sampling formula in Equation 3.15 is called the Gumbel-softmax distribution [48]. We employ the Gumbel-softmax trick to influence the chunking decisions with the objective of sentence reconstruction. In this case, the loss function would be the same as Equation 3.10. Note that continuous vectors are used during training, but the sample vectors are discretized to one-hot vectors during evaluation since we need hard chunking decisions.

3.6 Knowledge Transfer from Chunking back to Parsing

In our knowledge transfer approach, we first induce chunking labels from Compound PCFG [33]. Then, we train a a hierarchical RNN to learn from induced chunking labels to smooth out the noise. In this section, we discuss our attempt to transfer the knowledge from chunking back to parsing.

Our HRNN explicitly handles the chunking and makes corrections on the induced chunking labels from Compound PCFG. Since chunking is a shallow version of parsing, linguistic structures of both agree with each other to some extent. Hence, with the improvement on chunking, HRNN can potentially improve upon the parse trees given by Compound PCFG. To this end, we explore two simple heuristics to evaluate our HRNNs performance on unsupervised parsing. However, it is important to note that inducing parsing labels from chunking decisions is difficult mainly because of the complexity of full constituency parsing.

Distance-to-Tree In our first heuristic, we leverage the idea of the *syntactic distance* [74] to produce constituency parse trees from sentences. Given a sentence, $\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \cdots, \mathrm{x}^{(n)}$, we compute $\boldsymbol{d} = [d_1, d_2, \cdots, d_{n-1}]$ where d_i corresponds to the distance between $\mathrm{x}^{(i)}$ and $\mathrm{x}^{(i+1)}$. Once $\boldsymbol{d}$ is derived, it can be easily converted into the target constituency parse tree using Algorithm 2.

Algorithm 2 Distance-to-Tree Algorithm [75]

$S = [\mathrm{x}^1, \mathrm{x}^2, \cdots, \mathrm{x}^n]$ ▷ a sequence of words in a sentence of length n
$d = [d_1, d_2, \cdots, d_{n-1}]$ ▷ distance between every two adjacent words
procedure TREE((S, d))
    if $d = []$ **then**
        node ← Leaf([0])
    else
        $i \leftarrow \text{argmax}_i(d)$
        $\text{child}_l \leftarrow \text{TREE}(S_{\leq i}, d_{<i})$
        $\text{child}_r \leftarrow \text{TREE}(S_{>i}, d_{>i})$
        node ← Node(child_l, child_r)
    return node

The HRNN has a gating mechanism that switches between the word-level and phrase-level RNNs in a soft manner. This gating mechanism $m^{(t)} \in (0, 1)$ also serves as a chunk label for a word $\mathrm{x}^{(t)}$ as given by Equation 3.3. Following a BI schema [61], a higher value of $m^{(t)}$ would indicate tth word as a beginning of a new chunk, and a lower value would indicate tth word as part of a previous chunk. At time step t, this soft gate gives a probability of the word $\mathrm{x}^{(t)}$ as a start of the new chunk. Hence, higher probability $m^{(t)}$ essentially implies higher syntactic distance between the words $\mathrm{x}^{(t-1)}$ and $\mathrm{x}^{(t)}$. We simply consider the vector, $\boldsymbol{m} = [m^{(2)}, m^{(3)}, \cdots, m^{(n)}]$ as the syntactic distance vector ($\boldsymbol{d}$) and feed it to the distance-to-tree algorithm 2.

Employing such a simple distance-based heuristic makes it easy to inject inductive bias into the framework. Similar to Kim et al. [32], we further refine parse trees with the right skewness bias. The right skewness of gold-standard parse trees results from the fact that English is a head-initial language. Our goal is to influence the distance ($\boldsymbol{d}$) such that resulting parse trees are right-skewed, thus following the nature of gold-standard parse trees. Formally, we add the linear bias term to every d_i to compute $\hat{d}_i$:

$$\hat{d}_i = d_i + \lambda \times \text{AVG}(\boldsymbol{d}) \times (1 - \frac{(i-1)}{(n-2)}) \tag{3.16}$$

where λ is a hyperparameter and $i \in [1, n-1]$.

Refining Top-k parse trees With learned rule probabilities from Compound PCFG [33], the trees are marginalized by Viterbi-like CYK algorithm to get the best parse tree for a sentence. Instead, we extract top-k parse trees in our following simple heuristic and re-rank them based on our chunking knowledge.

Every constituency parse tree has a unique span representation. For example, the span representation for the right parse tree in Figure 2.6 is $s_p = [(1,2),(0,2),(5,6),(4,6),(3,6),(0,6)]$. Each tuple (i,j) corresponds to the span formed by ith and jth word in the sentence. Similarly, span representation can be easily retrieved from the chunking labels. The idea is to extract chunks of more than one word as spans. Such span representation would yield a list of non-overlapping spans on the word level. For instance, chunking structure for the same sentence can be given by $s_c = [(0,2),(5,6)]$. Although they may not fully describe a parse tree, these more accurate chunking structures from our HRNN model can be used to refine the top-k parse trees obtained from Compound PCFG.

For every sentence in the test set, we extract top-k parse trees, $T_i = [t_1, t_2, \cdots, t_k]$ from Compound PCFG. Let their span representation be given by $S_{p,i} = [s_{p,1}, s_{p,2}, \cdots, s_{p,k}]$. We also have chunking structure from HRNN given by a span representation $s_{c,i}$. For every ith sentence, we then calculate the agreement score between all the span representations in $S_{p,i}$ with $s_{c,i}$. The agreement score is nothing but the count of matching spans. The tree with the highest score is then chosen as the best parse tree.

Chapter 4

Experiments and Analyses

4.1 Experimental Setup

4.1.1 Dataset

Setup. We used the CoNLL-2000 [67], CoNLL-2003 [68] and English Web Treebank [7] for evaluation. CoNLL-2000 is widely used for the task of chunking and contains groundtruth chunk labels. CoNLL-2003 (German) dataset was developed for language-independent named entity recognition [68]. Both CoNLL2000 and CoNLL2003 contain sentences from the newswire domain. To evaluate the performance on a different domain, we make use of the English Web Treebank [7]. It consists of online review sentences and their manually annotated parse trees. We use state-of-the-art supervised chunker [NLTK-tagger, 8] to generate chunk labels for these sentences. Please refer to Table 4.1 for details on the dataset statistics.

Our work is for unsupervised chunking, and thus we did not use the chunk labels of the training set. Instead, the training sentences were used to perform knowledge transfer, i.e., we predicted pseudo-chunk labels by Compound PCFG to train the Hierarchical RNN.

All the datasets are labeled with the BIO schema, where "O" indicates outside a chunk (mainly punctuation). We followed the BI schema and ignored the "O" tokens. We compare the model output with groundtruth chunks in terms of phrase F1 and tag accuracy. We adopted the standard evaluation script from the CoNLL-2000 shared task to evaluate our chunk labels [67]. It calculates the phrase F1 score and the tag accuracy of the predicted chunks against groundtruth labels from the dataset.

Dataset	#Train	#Val	#Test
CoNLL-2000 (English)	7929	950	2003
CoNLL-2003 (German)	7000	2000	1000
English Web Treebank	6496	1856	936

Table 4.1: Dataset statistics

Model Settings. We employ the pretrained BERT [31] to capture global contextual sentence information. The HRNN uses vanilla transition with 100 hidden dimensions. In our preliminary experiments, we also try 300 dimensions and achieve very close performance, suggesting that the model capacity is already enough for chunking. We use the Adam optimizer to train the student model during knowledge transfer. We picked the best model by validation for early stopping, following most work on unsupervised parsing [18, 45]. Roughly, such fine-tuning does not exceed 15 epochs.

4.1.2 Baselines

To the best of our knowledge, this work is the first to propose unsupervised chunking as a task for syntactic structure discovery. Here, we discuss the baselines that we developed for unsupervised chunking. Although there are no prior works on unsupervised chunking, there exist straightforward extensions such as point-wise mutual information (PMI) [17] and the unsupervised Hidden Markov Model (HMM) [59].

PMI. The task of chunking aims at finding constituents like noun phrases, verb phrases, etc. It is analogous to finding "concepts" that are formed by putting two or more words together. For example, the words *social* and *media* have independent meaning, but when they are put together, they express a unique concept. Considering such mutual information between words is relevant to the task of chunking.

The mutual information between two discrete random variables (i.e., words) X and Y is given by,

$$I(X;Y) = \sum_{X,Y} p(X=x, Y=y) \log \frac{p(X=x, Y=y)}{p(X=x)\, p(Y=y)} \tag{4.1}$$

Equation 4.1 tells us how closely words in a dataset are tied together. In NLP, pointwise mutual information (PMI) is often used to evaluate how meaningful the co-occurrences of words are.

$$\text{PMI}(x;y) = \log \frac{p(X=x, Y=y)}{p(X=x)\, p(Y=y)} \tag{4.2}$$

Algorithm 3 Baum-Welch Algorithm [58]

1: Randomly initialize distributions (θ)
2: **repeat**
3: Compute forward messages $\forall_{i,t}\alpha_i(t)$
4: Compute backward messages $\forall_{i,t}\beta_i(t)$
5: Compute posterior: $p(z_t = i|\boldsymbol{x}, \theta) \propto \alpha_i(t)\ \beta_i(t)$
6: Compute posterior: $p(z_t = i, z_{t+1} = j|\boldsymbol{x}, \theta) \propto \alpha_i(t)\ p(z_{t+1} = j|z_t = i) \times \beta_j(t + 1)\ p(x_{t+1}|z_{t+1} = j)$
7: Update θ
8: **until** Converged

PMI quantifies the discrepancy between the probability of their co-occurrence given by their joint distribution and their individual distributions. Suppose one of the words has a high probability of occurrence, then it is likely for it to co-occur with another word by chance (i.e., they may not necessarily form a unique concept). On the other hand, if either one of the words has a low probability of occurrence, but its joint probability with the other word is high, it means that the two are likely to express a unique concept. In this baseline, we calculate the PMI of all the consecutive pairs of words to determine if they should be chunked together.

HMM. Probabilistic Graphical Models are one of the important set of methods for the NLP community. Hidden Markov Models (HMMs) is one such model in which the observation X (sentences) is assumed to be a Markov process with unobservable states (chunk labels) [59].

Specifically, while modelling a sentence, HMM assumes that every word x_t is generated by a current latent class z_t where z_t depends on the previous latent class z_{t-1}. For such representation we can define emission and transition probabilities as $p(x_t|z_t)$ and $p(z_t|z_{t-1})$ respectively. The probability of an entire sequence is given by simply multiplying all emission and transition probabilities across all steps 4.3.

$$p(\boldsymbol{x}, \boldsymbol{z}) = \prod_{t=1}^{n+1} p(z_t|z_{t-1}) \prod_{t=1}^{n} p(x_t|z_t) \tag{4.3}$$

Although we are exploring in an unsupervised regime and we do not have a priori access to these distributions, they can be estimated by Baum-Welch algorithm [58]. See Algorithm 3 for the details [1]

[1] The Baum-Welch algorithm is an iterative EM algorithm that updates parameters θ at each step, such that $p(\theta|\boldsymbol{x})$ is maximized. E-step computes the posterior $p(\boldsymbol{z}|\boldsymbol{x}, \theta)$ (line 5 and 6) and M-step updates the parameters (line 7). Posterior in the E-step depends on the forward and backward probabilities (line 3 and 4).

#	Function (f)	Definition
	Functions f^h that can be applied on g^h	
1	COS($\boldsymbol{r}, \boldsymbol{s}$)	$(\boldsymbol{r}^T\boldsymbol{s}/((\sum_{i=1}^{d} r_i^2)^{1/2} \cdot (\sum_{i=1}^{d} s_i^2)^{1/2}) + 1)/2$
2	L1($\boldsymbol{r}, \boldsymbol{s}$)	$\sum_{i=1}^{d} \lvert r_i - s_i \rvert$
3	L2($\boldsymbol{r}, \boldsymbol{s}$)	$\sum_{i=1}^{d} (r_i - s_i)^2)^{1/2}$
	Function f^a that can be applied on g^a	
4	HEL($\boldsymbol{P}, \boldsymbol{Q}$)	$\frac{1}{\sqrt{2}}(\sum (\sqrt{p_i} - \sqrt{q_i})^2)^{1/2}$

Table 4.2: The distance measure functions and their definitions where $\boldsymbol{r} = g^h(w_i)$, $\boldsymbol{s} = g^h(w_{i+1})$, $\boldsymbol{P} = g^a(w_i)$ and $\boldsymbol{Q} = g^a(w_{i+1})$,

LM Chunker. In the NLP community, pretrained language models (LMs) are leveraged as a means of acquiring contextualized word representations [57, 2, 31, 89]. Such representations are proven to be effective in recent advancements for many NLP tasks [27, 64, 1]. The multi-head self-attention is a key component in Transformer-based LMs, and it helps LMs to capture semantic and syntactic knowledge. There exist a work that shows that several attention heads of the language models exhibit syntactic structure similar to constituency grammar [32]. Similar to the work of Kim et al. [32], this baseline investigates the performance of pre-trained language models for the task of unsupervised chunking.

From the work of Shen et al. [74], we adopt the idea of *Syntactic Distance* to generate chunk labels. Given a sequence of words $\mathrm{x}^{(1)}, \mathrm{x}^{(2)}, \cdots, \mathrm{x}^{(n)}$, we compute $\boldsymbol{d} = [d_1, d_2, \cdots, d_{n-1}]$ where d_i corresponds to the distance between $\mathrm{x}^{(i)}$ and $\mathrm{x}^{(i+1)}$. The distances between words are given by:

$$d_i = f(g(\mathrm{x}^{(i)}), g(\mathrm{x}^{(i+1)})) \tag{4.4}$$

where $f(\cdot)$ is a distance measure function and $g(\cdot)$ is a representation extractor function. The function g makes use of a language model to get the vector representation for each word, while f calculates the syntactic distance between these vector representations. Following the work of [32], we explore several language models like BERT [31], RoBERTa [47] and GPT [60] for g. The same set of functions are considered for f [32]. See Table 4.2.

LM chunker makes use of two categories of representation extractor functions, G^h and G^a. Specifically, $G^h = \{g_j^h | j = 1, \cdots, l\}$ refers to the set of functions which output the intermediate hidden representation of a given word on the jth layer of the LM. Similarly, each of the functions in $G^a = \{g_{j,k}^a | j = 1, \cdots, l \text{ and } k = 1, \cdots, a+1\}$ output the attention distribution of an input word by the kth attention head on the jth layer of the LM. We then use corresponding function f^h and f^a as shown in Table 4.2. It is important to note that k ranges from 1 to $a+1$ where a is the number of attention heads. That is because we also consider the average of all attention

Notation	Definition
C_{gt}	ground-truth chunks
T_{gt}	ground-truth tags
C_{pred}	predicted chunks
T_{pred}	predicted tags
$C_{correct}$	correctly identified chunks
$T_{correct}$	correctly identified tags

Table 4.3: Notations for defining evaluation metrics

distributions on the same layer in addition to the individual ones. This special distance measure function is denoted by AVG_HEL and can be looked at as an ensemble of HEL function (function # 4) acting on different attention heads [32].

With Equation 4.4, we generate distances between words for all the M sentences in the CoNLL-2000 test set. To generate the chunk labels, we propose a simple heuristic where we use statistics of the test set. For instance, if there are p phrases in the test set, it implies there are the same number of "B" tags. We then select the largest $(p - M)$ distances and assign a "B" tag for the corresponding word. We subtract N because of the trivial assignment of a "B" tag for the first word of N sentences. The rest of the words are assigned with the "I" tag, indicating the inside of the chunk.

4.1.3 Evaluation Metric

For evaluating the chunking structures, we make use of the ground-truth labels and the standard evaluation script of CoNLL-2000 [67]. Phrase-level F_1 and Tag-level accuracy are two evaluation metrics where the former looks at predictions for chunks and the latter for the words. Refer to the Table 4.3 for the notations and their definitions.

Precision is the fraction of accurate instances from the total retrieved instances, and recall is the fraction of relevant instances identified. In the context of chunking, precision would be a ratio of correctly identified chunks and a total number of predicted chunks. Similarly, Recall would be a ratio of correctly identified chunks and a total number of chunks in the CoNLL-2000

Language Model - g	f	**L**	**Phrase** F_1	**Tag Acc.**
GPT2	AVG_HEL	10	30.68	58.5
GPT2-medium	AVG_HEL	10	39.46	67.43
RoBERTa-base	AVG_HEL	9	30.45	59.07
RoBERTa-large	AVG_HEL	12	36.82	59.07
BERT-base	AVG_HEL	5	40.63	67.18
BERT-large	AVG_HEL	16	**42.05**	**68.74**

Table 4.4: Analysis of LM chunker with different representation extractor (g) and distance measure functions (f). **L** represents the layer from which the hidden representations are extracted. HRNN is not applied in this comparison.

corpus. Formally they are defined as,

$$\text{precision} = \frac{C_{correct}}{C_{pred}} \tag{4.5}$$

$$\text{recall} = \frac{C_{correct}}{C_{gt}} \tag{4.6}$$

$$\text{Phrasal } F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4.7}$$

$$\text{Tag Accuracy} = \frac{T_{correct}}{T_{gt}} \tag{4.8}$$

Also, while counting the number of correct chunks $C_{correct}$, we consider only those which have accurate chunk labels for all the words in it (i.e., partially accurate chunks are regarded as inaccurate).

4.2 Results

4.2.1 Overall Performance

Analysis of LM Chunker. Unsupervised parsing based on the features of a pretrained language model (LM) is shown to have a reasonable performance [32]. This performance is further increased by injecting an inductive bias into the framework. Specially, right-skewness bias is exploited to get a competitive performance compared to other state-of-the-art unsupervised parsers like ON-LSTM [76] and PRPN [74].

For unsupervised chunking, we evaluate the performance of different language models and examine their effectiveness in learning the syntactic structure of the language. We use several

Method	CoNLL-2000 English (Newswire)		CoNLL-2003 German (Newswire)		English Web Treebank (Reviews)	
	Phrase F1	Tag Acc.	Phrase F1	Tag Acc.	Phrase F1	Tag Acc.
Supervised Methods						
NLTK-tagger-chunker	83.71	89.51	87.82	93.59	-	-
Supervised HMM	87.68	93.99	90.16	94.77	98.62	99.44
Unsupervised Methods						
PMI Chunker	35.64	64.5	42.19	64.42	32.28	65.34
Baum–Welch HMM	25.04	58.93	27.01	58.52	24.17	58.02
LM Chunker	42.05	68.74	45.06	68.62	31.23	62.55
Compound PCFG Chunker	62.89	81.64	55.94	75.54	58.17	79.33
LM $\rightarrow$ HRNN	47.99	73.10	48.40	70.10	39.43	70.5
Compound PCFG $\rightarrow$ HRNN	**68.12**	**83.90**	**57.14**	**75.81**	**64.32**	**83.25**

Table 4.5: Chunking performance on the CoNLL-2000, CoNLL-2003 and English Web Treebank. For both CoNLL datasets, the Phrasal F1 and Tag accuracy scores are calculated against groundtruth chunk labels. For the English Web Treebank, we treat the chunks generated by [NLTK-tagger, 8]) as groundtruth labels. $\rightarrow$ refers to our knowledge-transfer approaches.

pairs of representation extractor, and distance measure functions (f, g). We list the best distance measure function for every representation extractor function in Table 4.4.

We make three key observations. First, we observe that among all LM models, BERT demonstrates its effectiveness for unsupervised chunking. Particularly, both BERT models serve as a robust baseline achieving the top performance among all other LM models. Second, we see an interesting trend that using f^a (AVG_HEL) leads to the best chunking performance for all language models. This shows us that deriving syntactic structure is more efficient and accurate if attention distributions are extracted from LMs instead of their intermediate hidden representations. Third, large parameterization of the LM always increases the chunking performance. For instance, BERT large shows two-point improvement over BERT base on the phrase-F_1 score (GPT and RoBERTa show a similar trend). We use the best performing LM chunker for comparison with other unsupervised chunkers in Table 4.5.

Comparison between teacher models. We induce chunk labels from different state-of-the-art unsupervised parsers. Specifically, Compound PCFG [33] and parser based on the features of a pretrained language model (LM) [32]. We observe that the LM chunker is worse than the Compound PCFG. LM chunker simply explores the recent advances in language modeling and their effectiveness in learning the syntactic structures. Unlike LM chunker, Compound PCFG explicitly learns the grammar by handling the rule probabilities. Such a model and our strong maximal left-branching heuristic gives a $20, 10$, and 30 point improvement compared with the LM chunker in the phrase F_1 score on CoNLL-2000, CoNLL-2003 and English Web Treebank

Method	Phrase F1	Tag Acc.	Time (Sec.)
CoNLL-2000 (English)			
Compund PCFG	62.89	81.64	1803.90
Our HRNN model	**68.12**	**83.90**	**364.71**
CoNLL-2003 (German)			
Compund PCFG	55.94	75.54	163.04
Our HRNN model	**57.14**	**75.81**	**71.38**
English Web Treebank			
Compund PCFG	58.17	79.33	311.38
Our HRNN model	**64.32**	**83.25**	**167.29**

Table 4.6: Comparing the chunking quality and inference efficiency of the teacher Compound PCFG and our student HRNN. The inference time (in second) is obtained on NVIDIA Quadro RTX 6000 GPU with 25 GB RAM.

respectively. Therefore, we use Compound PCFG as our "teacher" model, i.e., the source of knowledge transfer.

Effectiveness of the knowledge transfer approach. Table 4.5 summarizes our main results. We then train our student HRNN model to learn from the heuristically induced chunk labels. Results show that we achieve an improvement of more than five percentage points based on either the LM-based chunker or Compound PCFG (42.05 vs. 47.99; 62.89 vs. 68.12). The large margins imply that our HRNN can indeed smooth out the noise of heuristics and capture the chunking patterns. The results suggest our knowledge transfer is a promising approach, as the student HRNN largely outperforms its teacher for unsupervised chunking.

We evaluate our knowledge-transfer approach on a different language (German). The results show that our student HRNN improves by 3 and 1 phrase F1 percentage points over LM chunker and Compound PCFG, respectively. Results on the text from a new domain, namely online reviews show that our student HRNN model beats LM chunker and Compound PCFG with a large margin of 8 and 6 phrase F1 score. This highlights the robustness of our knowledge-transfer approach and the HRNNs ability to learn the chunking structures in the text from different languages and domains.

We tested traditional unsupervised methods for chunking, such as thresholding point-wise mutual information (PMI) [17] and the Baum–Welch algorithm for the hidden Markov model [59]. These methods perform significantly worse than recent advances in unsupervised syntactic structure discovery. In Table 4.5, we also list the performance of some of the widely used supervised chunkers. There exist a large gap between supervised and traditional unsupervised chunking methods. With an improvement of roughly 33 phrase F_1 points over traditional unsupervised methods, our knowledge transfer approach with HRNN largely bridges the gap between super-

#	Chunking Heuristics	Phrase F_1	Tag Acc.
	Compound PCFG		
1	+ 1-word & 2-word chunks	55.72	75.14
2	+ Maximal right branching	40.83	69.28
3	+ Maximal left branching	**62.89**	**81.64**

Table 4.7: Analysis of chunking heuristics. HRNN is not applied in this comparison.

vised and unsupervised chunking.

We compare the inference efficiency of our student HRNN and the teacher Compound PCFG in Table 4.6. We observe that Compound PCFG is slow in inference, as it requires Monte Carlo sampling to marginalize the latent variable and dynamic programming to marginalize the PCFG. Our HRNN not only yields higher-quality chunks with 5 phrase-F1 points improvement, but also is 5x faster than Compound PCFG on the CoNLL-2000 dataset. Furthermore, HRNN is also roughly 2x faster than Compound PCFG on both CoNLL-2003, and English Web Treebank. The differences in terms of time efficiency among three datasets are due to sentence lengths. The average sentence lengths in CoNLL-2000 are much higher that of English Web Treebank. The larger gap between Compound PCFG and HRNN on CoNLL-2000 shows that HRNN is more effiecient on large datasets.

4.2.2 Analysis of Chunking Heuristics

We conduct all the ablation studies only on the CoNLL-2000 dataset. Table 4.7 compares the heuristics that induce chunks from parse trees obtained from Compound PCFG. Similar to our proposed maximal left-branching, we evaluate maximal right-branching heuristic. Figure 3.3 shows the difference between maximal-left branching and maximal-right branching heuristics. We also evaluate a thresholding approach that extracts one-word and two-word chunks only, since we observe most ground-truth chunks contain one or two words.

We observe that the performance of one-word and two-word chunk heuristic is higher than maximal right-branching, but worse than our maximal left-branching. Moreover, our maximal left-branching heuristic outperforms right-branching by 22 points in Phrase F_1. On a chunk level, this result agrees with a common knowledge that English is a head-initial language. Further, with regards to a parsing structure, the results are consistent with our conjecture that right-branching is a common structure of English and does not suggest meaningful chunks. On the contrary, left-branching indicates closely related words and is an effective heuristic for inducing chunks from parse trees.

#	Method	Phrase F_1	Tag Acc.
1	Teacher: Compound PCFG	62.89	81.64
2	$\rightarrow$ HRNN only	65.01	82.22
3	$\rightarrow$ BERT+1-layer RNN	67.19	83.86
4	$\rightarrow$ BERT+2-layer RNN	66.53	83.34
5	$\rightarrow$ BERT+HRNN (hard)	67.90	83.80
6	$\rightarrow$ BERT+HRNN	**68.12**	**83.90**

Table 4.8: Ablation study of the student model.

4.2.3 Analysis of Student Model

We present an ablation study on the student model in Table 4.8. As seen, all student models outperform the teacher model, showing that the imperfection of chunk heuristics can indeed be smoothed out by a machine learning model. However, a classic RNN or the Transformer predicts chunk labels individually for each word and thus does not provide autoregressive information. On the other hand, our HRNN shares chunking knowledge (via a gating mechanism) from one word to the next. The performance of BERT+RNN is worse than BERT+HRNN even if the number of layers is controlled (Rows 3, 4 vs. 6). This highlights the importance of both the explicitly handling of chunk decisions and the autoregressive information.

We also evaluate the performance of the hard gating mechanism in our HRNN where mask $m^{(t)}$ for a word $\mathrm{x}^{(t)}$ is simply a binary variable. The HRNN using soft gates outperforms a hard HRNN (Rows 5 vs. 6). This verifies that our soft HRNN can better handle the ambiguity of chunks and provide better autoregressive information. Building HRNN on top of BERT is also helpful (Rows 2 vs. 6), as BERT can capture global contextual information.

4.2.4 Additional results

Self-Supervised objectives. As discussed in section 3.5, we hypothesize that a self-supervised objective may aid unsupervised chunking. To this end, we proposed to explore the objective of sentence reconstruction via the Auto-encoder framework. To achieve this, we try different strategies like simple cross-entropy loss [5], reinforcement learning [86] and Gumbel softmax [48]. We evaluate this approah on the CoNLL-2000 dataset.

We explore two settings that differ in their training process. In our first setting, we train our HRNN encoder with a chunking objective ($\alpha = 0$ in Equation 3.10). Once the encoder (i.e., chunker) is trained, we train our encoder-decoder framework with both chunking and sentence reconstruction objectives. Our experiments show no further improvement in the chunking

#	Model	Phrase F_1	Tag Accuracy
1	Teacher: Compound PCFG	62.89	81.64
2	$\rightarrow$ HRNN	**68.12**	**83.90**
3	$\rightarrow$ HRNN AE	67.50	83.50
4	$\rightarrow$ HRNN AE-RL	66.42	82.37
5	$\rightarrow$ HRNN AE-Gumbel	66.67	82.80

Table 4.9: Chunking performance of using self-supervised objective in our knowledge transfer framework.

performance when we employ the sentence reconstruction objective via either cross-entropy, reinforcement learning, or Gumbel-softmax.

In our second setting, we directly train the encoder-decoder framework from scratch with a joint objective. As seen in Table 4.9, we observe that such a self-supervised training objective does not help the chunking performance. All three variants (Row 3, 4, and 5 vs. row 2) fail to beat the vanilla HRNN. One possible explanation for this behavior can be attributed to the fact that chunking is much simpler than parsing. Our knowledge transfer framework with Compound PCFG already gives a good performance, which is difficult to improve upon by a self-supervised objective.

Chunking to Parsing. Table 4.10 shows the unlabeled F_1 scores of our method and several other baselines on unsupervised parsing. Pre-trained LM without inductive bias performs worse than right-skewed trees. On the other hand, there is a big jump in the F_1 score when right-skewness bias is injected. Both the neural parameterization of PCFG and the Compound PCFG beat LM parser by a significant margin since both former models explicitly learn the grammar by handling the rule probabilities.

In our first attempt at unsupervised parsing, we generate distances between words from our HRNN model and then employ a distance-to-tree algorithm (Algorithm 2). We observe that our model, both with and without injecting a right skewness bias, outperforms the respective LM parser by a large margin. For example, our model with inductive bias achieves a Sent-F_1 score of 53.07, roughly 9 points higher than that of the LM parser with inductive bias. A similar trend is followed for unsupervised chunking, which shows that the linguistic structures of both the tasks agree with each other to some extent. Also, our model with inductive bias beats the neural PCFG but fails to outperform Compound PCFG. With such a simple method, improvement over Compound PCFG in chunking does not lead to improvement in parsing. This behavior implies the demand for more sophisticated methods to induce knowledge from chunking to parsing.

Our second attempt heavily relies on the teacher model since it simply reranks the top-k parse trees obtained from Compound PCFG. The reranking is done to find a parse tree that agrees the

Model	Sent-F_1	Corpus-F_1
Random Trees	18.1	16.4
Balance Trees	18.5	-
Right Skewed Trees	39.5	36.1
LM BERT-large* (w/o bias)	34.2	-
LM BERT-large* (w bias)	44.4	-
Neural PCFG	52.6	48.7
Compound PCFG	**60.1**	**58.03**
CPCFG $\rightarrow$ HRNN* (w/o bias)	39.74	37.15
CPCFG $\rightarrow$ HRNN* (w bias)	**53.07**	**51.07**
CPCFG $\rightarrow$ HRNN Reranking	53.92	52.18

Table 4.10: Unsupervised Parsing performance on PTB test sets. Maximum Sentence-level and Corpus-level F_1 scores are reported. * denote the use of distance-to-tree algorithm for generating parse trees (Algorithm 2)

most with chunking labels learned by our HRNN. This simple approach fails to beat Compound PCFG, and we suspect it is because top-k trees do not differ much in their structure. This limited diversity in the top-k trees leaves less room for our reranking method to improve.

4.2.5 Case study

In Figure 4.1, we present a few examples of chunking structures generated by both HRNN (student model) and Compound PCFG (teacher model) along with the ground-truth. The sentences are directly taken from the CoNLL-2000 dataset.

Our method (HRNN) is able to detect longer noun phrases, such as *small cable-television systems* (Example 1) and *white house press secretary marlin fitzwater* (Example 3), which agree more with the ground-truth chunks.

HRNN is also able to correct nonsensical chunks produced by Compound PCFG. While Compound PCFG combines *of* and *survival*, HRNN separates them, following the ground-truth labels (Example 2). Similarly, in Example 3, Compound PCFG predicts (*bush*) as one chunk and (*aids lawmakers*) as another. HRNN makes correction over Compound PCFG by associating *aids* with *bush* and thus having (*bush aids*) as one chunk and (*lawmakers*) as another. This way, the chunking structures given by HRNN are more aligned with the ground-truth labels.

In our knowledge-transfer approach, Compound PCFG acts as a teacher model and HRNN as a student model. HRNN retains the accurate chunking structure given by Compound PCFG. For example, HRNN follows the same chunking pattern as that of Compound PCFG by treating

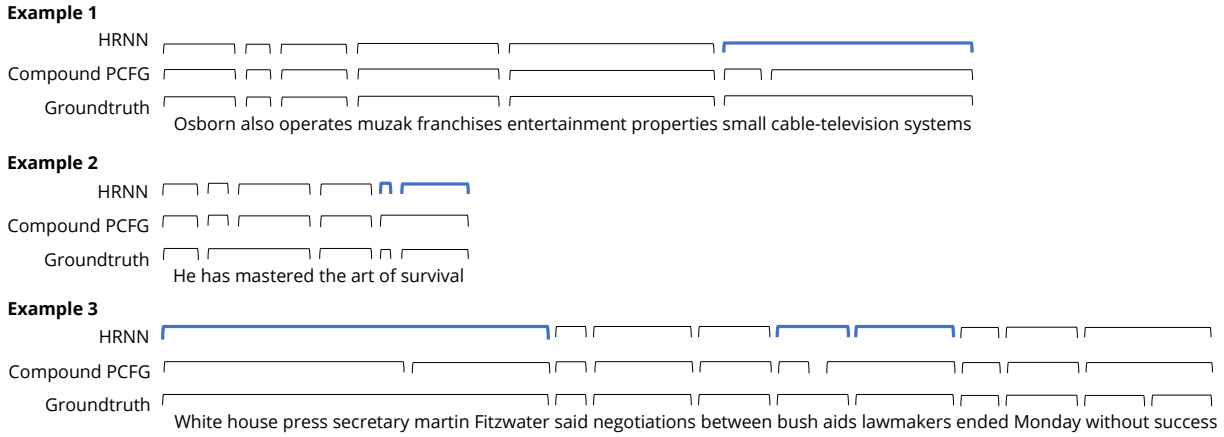


Figure 4.1: Examples of chunking structures produced by HRNN and Compound PCFG. The difference is highlighted in bold. We also show ground-truth chunks for reference.

Osborn, *also* and *operates* as individual chunks (Example 1). In general, HRNN not only effectively learns the chunking patterns from Compound PCFG but also can smooth out its noise and achieve higher performance for unsupervised chunking.

Chapter 5

Conclusion and Future Work

In this section, we summarize our work and highlight the main contributions. We also discuss the future direction and the extension of some of the ideas mentioned in this work.

5.1 Conclusion

In this thesis, we addressed a new task of syntactic structure discovery, namely, unsupervised chunking. The goal of this task is to identify chunks/phrases without any supervision from the annotated linguistic structures. Furthermore, finding spans like noun and verb phrases is very crucial in the process of understanding text. Hence, unsupervised chunking has real world applications and would benefit several downstream tasks like information extraction, named-entity recognition etc.

We introduced a knowledge transfer approach for the task of unsupervised chunking. Specifically, we proposed a hierarchical (HRNN) as a student model and employed recent advances in unsupervised parsing namely, Compound PCFG as a teacher model. Our student HRNN model learns from the chunk labels induced by the teacher Compound PCFG. Our knowledge transfer approach can be summarized in two steps.

First, we presented a maximal left-branching heuristic to induce chunk labels from Compound PCFG. Our simple and effective heuristic does not require any supervision of annotated grammars while converting parse trees into chunking labels. Second, we designed a HRNN that learns from these heuristically induced chunk labels. Our HRNN has a lower word-level RNN and an upper chunk-level RNN. We presented a gating mechanism that switches between two RNNs in a soft manner and also serves as a chunking label. Such a design makes sure that

predictions at every step are aware of the previously predicted chunks. This offers an added advantage over RNNs for the task of chunking by providing autoregressiveness.

With our thorough experiments, we demonstrated the effectiveness of our student HRNN model which achieved an improvement of more than five percentage points based on either the LM-based chunker or Compound PCFG on the CoNLL-2000 dataset. This large margin implies that our HRNN can indeed smooth out the noise of heuristics and capture the chunking patterns. We also evaluate our model on a different language (German) and different domain (Online reviews). This evaluation highlights the robustness of our HRNN model. We showed that our maximal left-branching chunking heuristic outperforms all other heuristics when employed on top of Compound PCFG. This strengthens our conjecture that left-branching structures indicate closely related words. We also presented rigorous analysis on the student model's architecture. Overall, our knowledge transfer approach largely bridges the gap between supervised and unsupervised chunking.

5.2 Future work

As a part of future work, we would like to investigate the effect of self-supervised fine-tuning objectives on the task of unsupervised chunking. We hypothesized that sentence reconstruction might help the task of chunking. In this work, we explored different strategies for autoencoding as a downstream task, such as training with cross-entropy, reinforcement learning, and Gumbel-softmax. Our current results showed no further improvement in terms of F1 score when a self-supervised fine-tuning objective was employed. Further investigation and/or more stable training procedures such as [70, 71] may potentially help both the downstream task and the chunking.

Finally, we would like to explore the idea of jointly optimizing both chunking and parsing in an unsupervised setting. Specifically, this idea suggests performing multi-task learning through a shared representation of a neural model such as HRNN. Chunking can be looked at as a shallow version of parsing, and hence linguistic structures of both agree with each other to some extent. Although chunking does not fully describe a parse tree, more accurate chunking structures may aid in refining the parse tree via joint optimization. We leave exploring this hypothesis as our future work.

References

[1] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, William L Hamilton, and Jimmy Lin. Exploring the limits of simple learners in knowledge distillation for document classification with docbert. *ACL 2020*, page 72, 2020.

[2] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th international conference on computational linguistics*, pages 1638–1649, 2018.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] James K Baker. Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65(S1):S132–S132, 1979.

[5] Dana H Ballard. Modular learning in neural networks. In *AAAI*, volume 647, pages 279–284, 1987.

[6] Maria Barrett, Ana Valeria González-Garduño, Lea Frermann, and Anders Søgaard. Unsupervised induction of linguistic categories with records of reading, speaking, and writing. 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 2028–2038, 2018.

[7] Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. English web treebank. *Linguistic Data Consortium, Philadelphia, PA*, 2012.

[8] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, 2006.

[9] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: Analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[10] Rens Bod. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 865–872, 2006.

[11] Rens Bod. Unsupervised parsing with u-dop. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 85–92, 2006.

[12] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, 2016.

[13] Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5094–5101, 2018.

[14] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT press, 2014.

[15] Alexander Clark. Unsupervised induction of stochastic context-free grammars using distributional clustering. In *Proceedings of the ACL 2001 Workshop on Computational Natural Language Learning (ConLL)*, 2001.

[16] J. Cocke and J. T. Schwartz. Programming languages and their compilers: Preliminary notes. 1970.

[17] Van de Cruys and Tim. Two multivariate generalizations of pointwise mutual information. In *Proceedings of the Workshop on Distributional Semantics and Compositionality*, pages 16–20, 2011.

[18] Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, 2019.

[19] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[20] Stefano Federici, Simonetta Montemagni, and Vito Pirrelli. Shallow parsing and text chunking: A view on underspecification in syntax. *Cognitive Science research paper-university of Sussex CSRP*, pages 35–44, 1996.

[21] Nazanin Firoozeh, Adeline Nazarenko, Fabrice Alizon, and Béatrice Daille. Keyword extraction: Issues and methods. *Natural Language Engineering*, 26(3):259–291, 2020.

[22] Yoav Goldberg, Meni Adler, and Michael Elhadad. Em can find pretty good hmm postaggers (when given a good start). In *Proceedings of Association for Computational Linguistics-08: Human Language Technologies*, pages 746–754, 2008.

[23] Aria Haghighi and Dan Klein. Prototype-driven grammar induction. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 881–888, 2006.

[24] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[25] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[27] Nanjiang Jiang and Marie-Catherine de Marneffe. Evaluating bert for natural language inference: A case study on the commitmentbank. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 6086–6091, 2019.

[28] Christer Johansson. A context sensitive maximum likelihood approach to chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000.

[29] Mark Johnson. Pcfg models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[30] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. 1965.

[31] Jacob Kenton, Devlin Ming-Wei Chang, and Lee Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[32] Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sang-goo Lee. Are pre-trained language models aware of phrases? Simple but strong baselines for grammar induction. In *International Conference on Learning Representations*, 2019.

[33] Yoon Kim, Chris Dyer, and Alexander M Rush. Compound probabilistic context-free grammars for grammar induction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2369–2385, 2019.

[34] Dan Klein. *The unsupervised learning of natural language structure*. Stanford University, 2005.

[35] Dan Klein and Christopher D Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 128–135, 2002.

[36] Dan Klein and Christopher D Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd annual meeting of the association for computational linguistics (ACL-04)*, pages 478–485, 2004.

[37] Rob Koeling. Chunking with maximum entropy models. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000.

[38] Taku Kudo and Yuji Matsumoto. Chunking with support vector machines. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[39] Taku Kudoh and Yuji Matsumoto. Use of support vector learning for chunk identification. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 142–144, 2000.

[40] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[41] Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

[42] Gary T Leavens, Krishna Kishore Dhara, and Krishna Kishore Dhara. Concepts of behavioral subtyping and a sketch of their extension to component-based systems. 2000.

[43] Yong-Hun Lee, Mi-Young Kim, and Jong-Hyeok Lee. Chunking using conditional random fields in korean texts. In *International Conference on Natural Language Processing*, pages 155–164, 2005.

[44] Willem JM Levelt. *An introduction to the theory of formal languages and automata*. John Benjamins Publishing, 2008.

[45] Bowen Li, Lili Mou, and Frank Keller. An imitation learning approach to unsupervised parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3485–3492, 2019.

[46] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. 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 1106–1115, 2015.

[47] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[48] C Maddison, A Mnih, and Y Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the international conference on learning Representations*. International Conference on Learning Representations, 2017.

[49] Jean Maillard, Stephen Clark, and Dani Yogatama. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *Natural Language Engineering*, 25(4):433–449, 2019.

[50] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[51] Antonio Molina and Ferran Pla. Shallow parsing using specialized hmms. *Journal of Machine Learning Research*, 2(4), 2002.

[52] Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. A survey on open information extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3866–3878, 2018.

[53] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[54] Mark A Paskin. Grammatical bigrams. 2002.

[55] John K Pate and Sharon Goldwater. Unsupervised syntactic chunking with acoustic cues: Computational models for prosodic bootstrapping. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, pages 20–29, 2011.

[56] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*, 2017.

[57] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 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, 2018.

[58] Lawrence Rabiner. First hand: The hidden markov model. *IEEE Global History Network.*, 2013.

[59] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[60] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[61] Lance A Ramshaw and Mitchell P Marcus. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*, pages 157–176. 1995.

[62] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[63] Roi Reichart and Ari Rappoport. Unsupervised induction of labeled parse trees by clustering with syntactic features. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 721–728, 2008.

[64] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2019.

[65] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[66] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[67] Erik Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*, 2000.

[68] Erik Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[69] Motoki Sano, Hiroyuki Shindo, Ikuya Yamada, and Yuji Matsumoto. Segment-level neural conditional random fields for named entity recognition. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 97–102, 2017.

[70] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[71] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[72] Yoav Seginer. Fast unsupervised incremental parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 384–391, 2007.

[73] Harshil Shah, Tim Xiao, and David Barber. Locally-contextual nonlinear crfs for sequence labeling. *arXiv preprint arXiv:2103.16210*, 2021.

[74] Yikang Shen, Zhouhan Lin, Chin-wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. In *International Conference on Learning Representations*, 2018.

[75] Yikang Shen, Zhouhan Lin, Athul Paul Jacob, Alessandro Sordoni, Aaron Courville, and Yoshua Bengio. 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, 2018.

[76] Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. In *International Conference on Learning Representations*, 2018.

[77] Yikang Shen, Yi Tay, Che Zheng, Dara Bahri, Donald Metzler, and Aaron Courville. StructFormer: Joint unsupervised induction of dependency and constituency structure from masked language modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (to appear)*, 2021.

[78] Haoyue Shi, Jiayuan Mao, Kevin Gimpel, and Karen Livescu. Visually grounded neural syntax acquisition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1842–1861, 2019.

[79] Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 151–161, 2011.

[80] Zach Solan, Eytan Ruppin, David Horn, and Shimon Edelman. Automatic acquisition and efficient representation of syntactic structures. *Advances in Neural Information Processing Systems*, pages 107–116, 2003.

[81] Jana Straková, Milan Straka, and Jan Hajic. Neural architectures for nested ner through linearization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5326–5331, 2019.

[82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[83] Andreas Veit, Tomas Matera, Lukas Neumann, Jiri Matas, and Serge Belongie. Coco-text: Dataset and benchmark for text detection and recognition in natural images. *arXiv preprint arXiv:1601.07140*, 2016.

[84] Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. Automated concatenation of embeddings for structured prediction. *arXiv preprint arXiv:2010.05006*, 2020.

[85] Adina Williams, Andrew Drozdov, and Samuel R. Bowman. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association for Computational Linguistics*, 6:253–267, 2018.

[86] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[87] J Gerard Wolff. Learning syntax and meanings through optimization and distributional analysis. pages 179–215, 1988.

[88] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. Luke: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, 2020.

[89] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. XLNET: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[90] Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *International Conference on Learning Representations*, 2017.

[91] D.H. Younger. Recognition and parsing of context-free languages in time n3. 1967.

[92] Tong Zhang, Fred Damerau, and David Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2(4):615–637, 2002.

[93] Yanpeng Zhao and Ivan Titov. Visually grounded compound pcfgs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4369–4379, 2020.

[94] GuoDong Zhou and Jian Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 473–480, 2002.