UNIVERSITY OF CALIFORNIA

Los Angeles

Arc-Eager Construction Provides Learning Advantage Beyond Stack Management

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Arts in Linguistics

by

Phillip A. Barnett

2021

ABSTRACT OF THE THESIS

Arc-Eager Construction Provides Learning Advantage Beyond Stack Management

by

Phillip A. Barnett

Master of Arts in Linguistics
University of California, Los Angeles, 2021
Professor Tim Hunter, Chair

Psycholinguistic research has posited arc-eager left corner parsing as a psychologically viable candidate for the human parsing mechanism (Resnik, 1992). Using probabilistic left-corner grammars (PLCGs), as introduced by Manning and Carpenter (1997), as a testbed, this thesis examines the probabilistic mechanisms involved in arc-eager tree construction.[1] By moving attachment decisions earlier in the decision tree, arc-eager PLCGs gain probabilistic advantage over their arc-standard counterparts due to recursive left-corner embeddings, tree productions of the form $A \to A\gamma$, which are abundant in datasets like the Penn Treebank, and which many would argue have psychological reality. This advantage is fully independent of the well-documented stack management advantage seen in arc-eager constructions of right-branching structures.

[1]The python module `ae-plcg`, which was created and used by the present investigation to model and evaluate arc-standard and arc-eager PLCGs, can be found at `https://github.com/phill-barnett/ae-plcg`, along with several fully trained and testable model parameter sets, detailed in Section 3.1 of this thesis.

The thesis of Phillip A. Barnett is approved.

Jesse A. Harris

Dylan Bumford

Tim Hunter, Committee Chair

University of California, Los Angeles

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Many theories of syntax posit underlying hierarchical structures for sentences, often referred to as *trees*. These trees recursively group constituents into larger phrases, ultimately assigning a single grouping and tag to the entire sentence. Trees can be constructed in different orders for different purposes. Context-free grammars (CFGs), for instance, construct a tree top-down, starting with the root node and expanding nonterminals until all terminal leaf nodes have been enumerated.

One of the questions posited by language processing research is that of the *human parsing mechanism*; what construction order algorithm do we use to turn linear utterances into phrase-structure trees? Many researchers have weighed in on the very specific way the brain turns sentences into hierarchical structures for composing meaning (e.g., Johnson-Laird, 1983; Abney and Johnson, 1991; Bengio et al., 2003). Early psychological perception research revealed that humans use two types of processing when decoding and encoding their experience of the world: bottom-up, where structure and meaning are built up from smaller components in the stimulus; and top-down, where structure and meaning are constructed based on expectation. This gave natural rise to the consideration of both bottom-up and top-down mechanisms within the human parsing mechanism. Human language processing research (e.g., Resnik, 1992) has shown that the human parsing mechanism aligns closely with a particular construction order known as *left-corner*, wherein bottom-up left-to-right processing of input is used to generate top-down predictions about the remaining structure. This construction order is discussed more explicitly in Section 1.4.1.

There are two main flavors of left-corner construction order: arc-standard and arc-eager. The difference between the two lies in the way they consolidate the predicted phrase types sought by the parser and the observed phrase types the parser has already detected in the input. Arc-standard construction fully constructs a tree before making these *attachment* decisions, whereas arc-eager must immediately make the decision as to whether a newly posited node is the goal node, or a daughter node of the same category. Manning and Carpenter (1997) propose a language model based on arc-standard left-corner construction order, which they term a probabilistic left-corner grammar (PLCG), which I will use as a testbed of comparison for these two types of left-corner construction algorithms.

This work recreates and extends the architecture of the original Manning and Carpenter (1997) paper to compare and explore the effects of arc-standard and arc-eager left-corner construction order on the learned parameters of the resulting language models. Chapter 1 presents an overview of probabilistic language models, situating PLCGs within the landscape. Chapter 2 presents a thorough description of the arc-standard and arc-eager language models created for the present investigation, and ensures mathematically that arc-eager PLCGs constitute a complete language model, congruent to the original arc-standard PLCG. Finally, Chapter 3 gives a preliminary evaluation of these models and analyzes their structure and results to determine the differences in the way they learn and generate probabilities.

1.1 Probabilistic Language Models

Probabilistic language models date back to the 1950s work of computer scientists like Ray Solomonoff (see Solomonoff, 1997). The terms *grammar* and *language model* are often used interchangeably. However, for the sake of clarity, I disambiguate them here as follows: a grammar is a function that takes a string as input and returns a Boolean value based on whether the string is a set member of the grammar, $G :: \Sigma^* \rightarrow \texttt{Bool}$, while a language model is a function from strings to probabilities, $\Gamma :: \Sigma^* \rightarrow [0, 1]$. A statistical language model is generally defined as a probability distribution over a string of words given a grammar. The probability of a sentence $W = (w_1, ..., w_n)$ is the product of probabilities of each of its words w given the preceding words, as in (1.1). In principle, the probability of the next word in an utterance is conditioned on all preceding words: $P\left(w_n \,|\, w_1, ..., w_{n-1}\right)$. However, without some abstraction away from this notion, utterances would be assigned a probability based on the full sentence frequency. In a corpus of 50,000 unique sentences, each sentence would be assigned a probability of $1/50000$. Even worse, sentences not contained in the corpus would recieve a probability of 0. Thus, we must assume that utterances are composed of smaller chunks that have some probability of occuring, as well as some probability of occuring in a given structural relation to the rest of the utterance. The human brain, being the great pattern-recognizer that it is, finds patterns in language and extrapolates a mental language model from them.

$$P(W) = \prod_{i=1} P(w_i \mid w_1, \ldots, w_{i-1}) \tag{1.1}$$

1.1.1 Bias in language models

Inductive learning is the process of discovering patterns within data based on surface observations. Mitchell (1980) points out that learning is only possible through bias about which generalizations are possible within the training data. From this perspective, Universal Grammar can be thought of as the notion that humans are genetically hard-wired with some sort of *inductive bias* pre-installed that gives shape to the patterns they are capable of finding. When constructing a probabilistic language model, all decisions about the way the model will learn probabilities introduce inductive bias.

As Bengio et al. (2003) point out, language models, like all models that assign probability to groups of discrete random variables (in this case words), suffer from what is known as the *curse of dimensionality*; modeling a string of n words given a grammar containing g words could yield up to $(g^n - 1)$ free parameters (p. 1137). As the grammar and length of utterance grows, the model parameters become computationally unwieldy.

To address this computational problem, researchers have proposed methods for generating probabilistic language models that introuduce a type of inductive bias called *independence assumptions*. Instead of conditioning each word/structure's probability on all preceding words/structures in the sentence, informational sacrifice is made as certain aspects of the preceding variables are highlighted and others are thrown out.

The present study focuses on language models of syntactic structure. More specifically, we are most interested in models that assign probabilities to strings based on the posited or observed phrase-structure trees that underlie them. Within such models, the probability of a string given a grammar is equivalent to the sum of the probabilities of all possible string-tree pairs for that sentence, as shown in (1.2). Since only trees that yield the string will receive non-zero probability, this is equivalent to (1.3).

However, assigning probabilities to strings is not essential to comparing construction orders. Thus, the present study focuses only on the probabilities that such models assign to the trees themselves.

$$P\left(s \mid G\right) = \sum_{t} P\left(s, t \mid G\right) \tag{1.2}$$

$$= \sum_{\{t : yield(t) = s\}} P\left(t \mid G\right) \tag{1.3}$$

1.2 *n*-gram models

All statistical models use variables. The most basic approaches to modelling involve modelling only observed variables, while more complex models make use of latent, or hidden, variables, variables that never appear on the surface, but are posited by the model as a means of explaining the observable structure. Language models are no exception; there are language models that only pay attention to the surface structure of language, and there are language models that make hypotheses about and model the hidden structure of language. The most naive approach to modelling language is n-gram modelling. These models rely purely on the observed statistical frequency of words in a training corpus given the preceding $n-1$ words.

Step	Complete Configuration	Context	Action	
0	[ ]	( )	GEN	the
1	[the]	(the)	GEN	catfish
2	[the catfish]	(catfish)	GEN	howls
3	[the catfish howls]	(howls)	GEN	at
4	[the catfish howls at]	(at)	GEN	midnight

Table 1.1: The complete configuration and conditioning context of a simple bigram model's derivation of the sentence *the catfish howls at midnight*.

While the actual probability of each word in the sentence is best expressed as $P\left(w_i \mid w_1, \ldots, w_{i-1}\right)$, due to the core independence assumption of an *n*-gram model, the probability of a word in an *n*-gram model can be expressed as $P\left(w_i \,\middle|\, w_{i-(n-1)}, \ldots, w_{i-1}\right)$. Independence assumptions like these are essential to making language models computable, and indeed, we can assume from the infinite, novel capacity of human language, that the human mind must also be making some sort of independence assumptions about the structure of language. As in (1.1), sentence probabilities in n-gram models are the product of each word's probability. Regardless of the independence assumption made, probabilities for each conditioning context must sum to 1 in order to ensure a complete language model. In an n-gram model, this means that for any given context, whether it is composed of the preceding word, two words, or 200 words, summing across the probabilities of all possible words for that context will result in a sum of 1, as in (1.4).

$$\sum_{w} P\left(w \,\middle|\, w_{i-(n-1)}, \ldots, w_{i-1}\right) = 1 \tag{1.4}$$

By only relying on the surface structure of the data they describe, these models intuitively fall short of accurately modelling language when we consider longer-distance relational structures in language data, such as reduplication or reflexive pronoun relationships. An unbounded n would be needed to deal with the recursive nature of right- and left-branching structures, which humans tend to have little trouble understanding or encoding. See (Manning and Schütze, 1999, Ch. 6), for a more in-depth discussion of n-gram models.

1.3 Probabilistic context-free grammars (PCFGs)

To capture the recursive nature of language, linguists and computer scientists employ context-free grammars (CFGs), which are simple phrase-structure grammars that describe a language with a set of production rules. These production rules take the form $A \rightarrow \gamma$ where A is a nonterminal category and γ is a tuple of terminal and/or nonterminal variables. CFGs take phrase-structure trees, like the one in 1.1, and return a Boolean acceptability rating.

Probabilistic language models known as probabilistic context-free grammars (PCFGs) can be easily constructed from CFGs by simply assigning probabilities to each of the production rules in the grammar. Deriving sentence probabilities is slightly more abstract with PCFGs as words are now formulated as the result of the generative actions that produced them, rather than preceding words, as in n-gram models. Sentence probabilities in this case are straightforwardly calculated by taking the product of the probabilities of all generative actions needed to construct the tree ($C_1, \ldots, C_n$). The probability

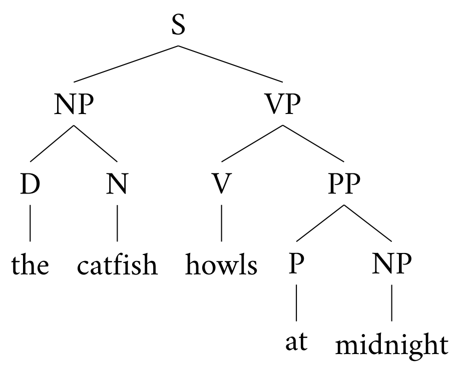

Figure 1.1: The CFG tree structure representing the sentence *The catfish howls at midnight.*

of each of these actions should theoretically be conditioned on all previous actions, $P\left(C_i \mid C_1, ..., C_n\right)$, but instead, PCFGs use a drastic independence assumption: the probability of an action depends only on the node being expanded. PCFGs only feature one generative action, which we can call EXPAND, that uses CFG production rules to construct trees top-down, $P\left(\text{EXPAND} A \rightarrow \gamma \mid A\right)$. In this way, like n-gram models, PCFGs use only a limited construction history to determine probabilities. A more thorough breakdown of PCFGs can be found in (Manning and Schütze, 1999, Ch. 11).

These probabilities are most straightforwardly calculated by dividing the number of times a certain production rule $A \rightarrow \gamma$ occurs within the training data by the total number of times A occurs in the corpus. This probability can be represented as $P\left(A \rightarrow \gamma \mid A\right)$. Note the drastic independence assumption here: the probability that A is rewritten as γ is only conditioned on the existence of A, hence the "context-free" moniker; CFGs do not not look higher in the tree than the current node. Also like *n*-gram models, all locally competing expansions also sum to 1, as in (1.5).

$$\sum_{\gamma} P\left(A \rightarrow \gamma \mid A\right) \tag{1.5}$$

1.4 Probabilistic Left-Corner Grammars (PLCGs)

1.4.1 Construction order

In the case of a PCFG, probabilities are derived from production rules used in a constituency tree. These probabilities correspond roughly to the actions of a simple top-down or bottom-up CFG parsing algorithm. While a PCFG does not look at the stack contents, like an actual parser would, it constructs the tree in the same order. In top-down CFG parsing, the parser looks first at the root category, which can be any nonterminal category in the grammar, and then uses the rules of the grammar to rewrite this nonterminal and its resulting nonterminals, ultimately generating the leaves of the parse tree. Bottom-up parsing conversely starts at the leaf nodes, rewriting nodes left-to-right as complete constituents are identified, ultimately identifying a root category for the string.

As one might expect, the literature is full of parsing algorithms for formal grammars that follow different construction orders. Among the most psychologically plausible of these algorithms is a class of tree construction algorithms known as *left-corner* (LC) (Resnik, 1992). Manning and Carpenter (1997) propose an alternative method for making a probabilistic language model using a CFG grammar that uses left-corner construction order to override the CFG's context-free assumption when conditioning probabilities. They term this class of language models *probabilistic left-corner grammars* (PLCGs) as they take advantage of left-corner construction order's mixed top-down/bottom-up approach to tree construction that builds phrases based on observed categories (like bottom-up) but also makes predictions, or *projections* as we'll call them, about which categories may come next based on a *goal category*. This goal category effectively gives context to the CFG-style productions within the tree.

Left-Corner Construction Algorithm

- Construct the left corner[1] lc of the tree bottom-up.
- Having observed lc, generate predicted nonterminals for the remaining daughter nodes using the grammar and construct them top-down.
- Consolidate predicted and observed nonterminals.

Figure 1.2: Left-corner (LC) construction of a parse tree follows this general procedure of using the tree's left corner to generate predictions for the rest of the string, and ultimately consolidating predicted and observed nonterminals, which verifies that an accurate parse has been found.

The procedure in Figure 1.2 gives the general procedure for left-corner tree construction. While the term *left-corner* (LC) represents a large class of tree-construction algorithms (see Demers, 1977; Nederhof, 1993), the particular left-corner algorithm used by Manning and Carpenter (1997) is de-

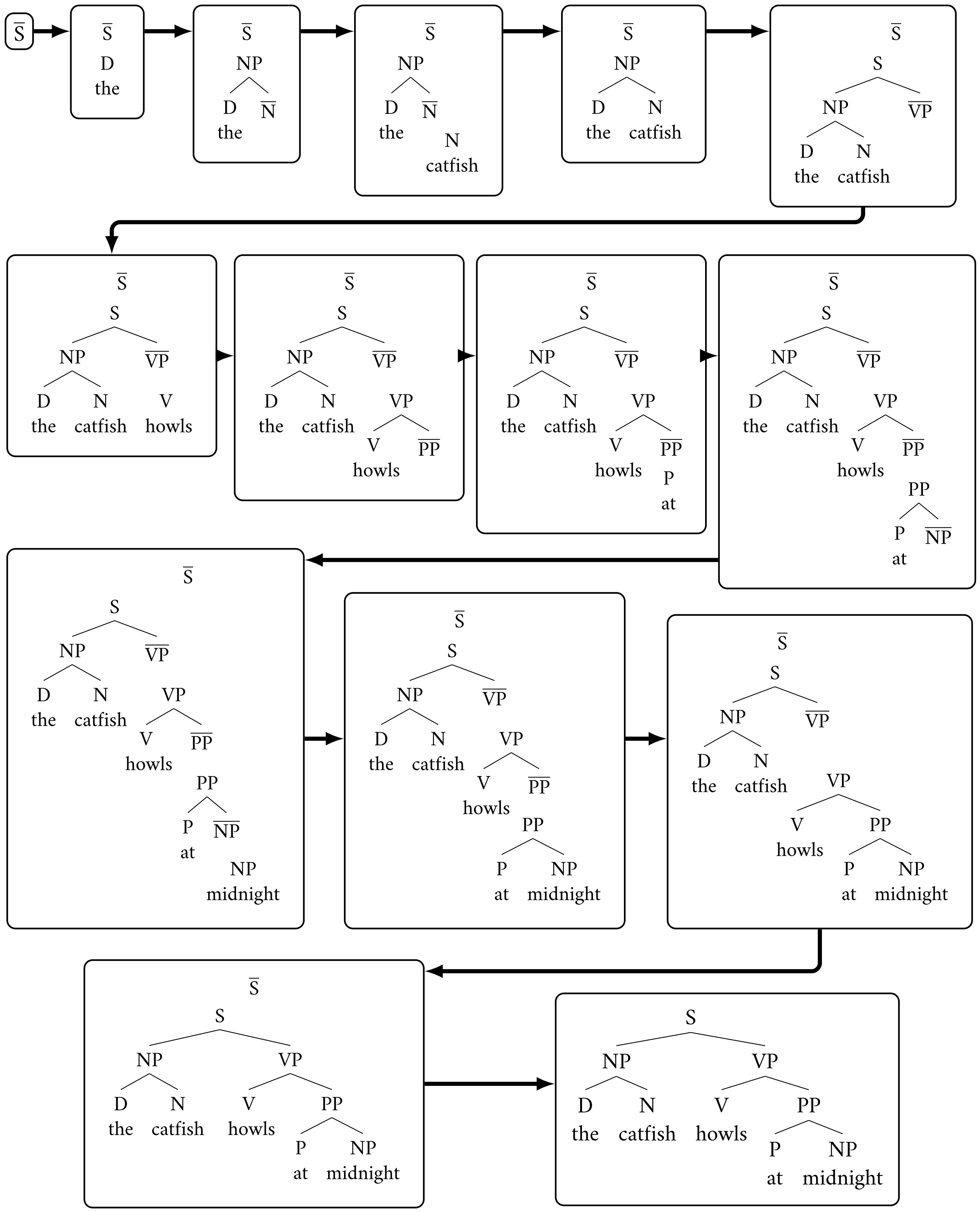

Figure 1.3: Arc-standard left-corner construction order. Each step is transformed to the next by generative LC parse operations. The complete list of actions for this construction is given in Table 1.2. PLCGs are trained on lists of these parse operations given the state of the stack.

signed to generate exactly one unique *left-corner derivation* per unique tree. An LC derivation is represented as a list of generative *actions* used to construct the tree. Table 1.3 highlights the difference in construction actions that PCFGs and PLCGs condition their probabilities on.

Step	Stack	Context	Action	Rule
0	$[\overline{\text{S}}]$	$(\overline{\text{S}}, \overline{\text{S}})$	SHIFT	D $\rightarrow$ the
1	$[\text{D}, \overline{\text{S}}]$	$(\text{D}, \overline{\text{S}})$	PROJECT	NP $\rightarrow$ D N
2	$[\overline{\text{N}}, \text{NP}, \overline{\text{S}}]$	$(\overline{\text{N}}, \overline{\text{N}})$	SHIFT	N $\rightarrow$ catfish
3	$[\text{N}, \overline{\text{N}}, \text{NP}, \overline{\text{S}}]$	$(\text{N}, \overline{\text{N}})$	ATTACH	
4	$[\text{NP}, \overline{\text{S}}]$	$(\text{NP}, \overline{\text{S}})$	PROJECT	S $\rightarrow$ NP VP
5	$[\overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\overline{\text{VP}}, \overline{\text{VP}})$	SHIFT	V $\rightarrow$ howls
6	$[\text{V}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\text{V}, \overline{\text{VP}})$	PROJECT	VP $\rightarrow$ V PP
7	$[\overline{\text{PP}}, \text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\overline{\text{PP}}, \overline{\text{PP}})$	SHIFT	P $\rightarrow$ at
8	$[\text{P}, \overline{\text{PP}}, \text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\text{P}, \overline{\text{PP}})$	PROJECT	PP $\rightarrow$ P NP
9	$[\overline{\text{NP}}, \text{PP}, \overline{\text{PP}}, \text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\overline{\text{NP}}, \overline{\text{NP}})$	SHIFT	NP $\rightarrow$ midnight
10	$[\text{NP}, \overline{\text{NP}}, \text{PP}, \overline{\text{PP}}, \text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\text{NP}, \overline{\text{NP}})$	ATTACH	
11	$[\text{PP}, \overline{\text{PP}}, \text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\text{PP}, \overline{\text{PP}})$	ATTACH	
12	$[\text{VP}, \overline{\text{VP}}, \text{S}, \overline{\text{S}}]$	$(\text{VP}, \overline{\text{VP}})$	ATTACH	
13	$[\text{S}, \overline{\text{S}}]$	$(\text{S}, \overline{\text{S}})$	ATTACH	

Table 1.2: The context and action pairs encountered in the arc-standard left-corner derivation of the tree for *the catfish howls at midnight*, which is given in Figure 1.1. The LC tree construction in Figure 1.3 follows these actions. Note that ATTACH does not take a rule, as it operates exclusively on the stack.

	n-gram	PCFG	PLCG
Algorithm	Left-to-right	Top-down	Left-corner
Actions	GENERATE	EXPAND	SHIFT PROJECT ATTACH
Probability	$P\left(\text{GEN } w_1 \,\middle|\, w_{i-(n-1)}, \ldots, w_{i-1}\right)$	$P\left(\text{EXPAND } A \rightarrow \gamma \mid A\right)$	$P\left(\text{ACTION} \mid lc, gc\right)$

Table 1.3: Comparison of *n*-gram , PCFG, and PLCG model actions and probabilistic conditioning. Note that by using the machinery of a parser, the PLCG is able to condition on a more dynamic set of actions.

1.4.2 The drastic independence assumption of PLCGs

As discussed in Section 1.3, PCFGs make the independence assumption that the only thing affecting the probability of a construction action is the category of the node being constructed, the parent of the current local subtree. While the independence assumption made by PLCGs is also drastic, it is vitally different. While in principle each action should be conditioned on all of the previous actions in the derivation, to approximate this while drastically reducing computational load, the original Manning and Carpenter (1997) PLCG conditions each action on the partial state of a left-corner parser at the current point in the derivation. If a parser were constructing the tree[2], it would need to keep a running list of predicted and observed nonterminals, called a *stack*, to keep track of the structure it generates from the input string. The original PLCG generates a stack for each step of the construction and conditions on the topmost symbol on the stack, which is the left-corner of the current tree, and the topmost predicted nonterminal (e.g., $\overline{\text{S}}$, $\overline{\text{NP}}$, etc.) on the stack, which is the current goal category. As stated above, the goal category is what gives a PLCG more context than a PCFG.

The PLCGs presented in this paper make the independence assumption that action probabilities are conditioned only on their left-corner and current goal. This means that the probabilities generated by the PLCG are dependent not only on the current category like CFGs, but also on the context of the current subtree in the sentence. In other words, subject-NPs, where the current goal is $\overline{\text{S}}$, and object-NPs, where the current goal is $\overline{\text{VP}}$, are generated using different sections of the probability space. This is important to highlight, as it was the primary motivation behind the development of the original Manning and Carpenter (1997) PLCG.

Action	Rule Type	Input string	Stack
SHIFT	C $\rightarrow w_1$	$w_1, w_2, \ldots, w_n \rightarrow w_2, \ldots, w_n$	$[\gamma] \rightarrow [\text{C}, \gamma]$
PROJECT	X $\rightarrow$Y Z	No Change	$[\text{Y}, \gamma] \rightarrow [\overline{\text{Z}}, \text{X}, \gamma]$
ATTACH	None	No Change	$[\text{X}, \overline{\text{X}}, \gamma] \rightarrow [\gamma]$

Table 1.4: A formal definition for each of the actions available to an arc-standard left-corner parser. These actions, along with the corresponding rule, make up the derivations for an AS-PLCG. Here, terminal rules are represented by C $\rightarrow w$, while nonterminal rules are represented by X $\rightarrow$Y Z.

1.4.3 Learning probabilities with PLCGs

Manning and Carpenter (1997) provides probabilities for left-corner derivations using their original

[2]As we will discuss in Chapter 2, the PLCG training process does not actually parse trees, but uses an oracle-generating algorithm that utilizes the already-enumerated structure of a tree to straightforwardly generate the one correct derivation without any hypothesizing/guesswork needed.

PLCG.[3] As with all phrase-structure language models, to find the probability of a string s, given a left-corner language model G, we must sum over the probability of each string-tree pair given G as shown above in (1.2), which is equivalent to (1.3). Within the Manning and Carpenter model, the probability of an individual tree is the sum probability of all possible left-corner derivations of that tree. As there is only one possible left-corner derivation of any given tree, by the chain rule, the probability of a tree can be found by taking the product of the probabilities of the parser actions $(C_1, ..., C_m)$ contained in the derivation given the action history, as in (1.6).

$$P(t) = P(d) = \prod_{i=1}^{m} P\left(C_i \,|\, C_1, ..., C_{i\text{-}1}\right) \tag{1.6}$$

Without diving into neural networks, it is not practical to condition each action on the entire parser action history. To address this, Manning and Carpenter (1997) made a drastic, yet effective, assumption that the only part of the history an action is conditioned on is the topmost goal category on the parser's stack and the topmost category on the stack overall. As we will see in Section 1.4.4, in cases wherein the topmost goal category and the topmost category are not the same entity, this is equivalent to conditioning on the current goal category, which we will call gc, and the most recently constructed nonterminal category, which is the left-corner of the in-progress tree or subtree, which we will call lc.

1.4.4 Arc-standard probabilities

$$P\left(C_i = \textsc{shift}\ T \rightarrow w \,|\, C_1, ..., C_{i-1}\right) = \begin{cases} P_S(T \rightarrow w | gc) & \text{if ToS is predicted } \overline{\text{NT}} \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

$$P\left(C_i = \textsc{attach} \,|\, C_1, ..., C_{i-1}\right) = \begin{cases} P_A(lc, gc) & \text{if ToS is observed NT} \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

$$P\left(C_i = \textsc{project}\ A \rightarrow \gamma \,|\, C_1, ..., C_{i-1}\right) = \begin{cases} (1 - P_A(lc, gc))P_P(A \rightarrow \gamma | lc, gc) & \text{if ToS is observed NT} \\ 0 & \text{otherwise} \end{cases} \tag{1.9}$$

[3]Equations (1.6)-(1.12) are adapted from Manning and Carpenter (1997); ToS = Top of Stack.

The probability of shifting using a certain terminal rule $T \rightarrow w$ is zero when the top of the stack (ToS) is not a predicted nonterminal $\overline{\mathrm{NT}}$. Otherwise it is conditioned on the goal category and effectively represents lexical frequencies dependant on the current goal context. When to SHIFT is fully deterministic under this formulation. Note that when the top of the stack is a predicted nonterminal, $lc = gc$. So in practice SHIFT parameters ($P_S(T \rightarrow w|gc)$) can be trained the same way as ATTACH ($P_A(lc, gc)$) and PROJECT ($P_P(N \rightarrow c|lc, gc)$). These arc-standard parsing operations obey the following:

$$\begin{aligned} \sum_{T \rightarrow w} P_S(T \rightarrow w|gc) &= 1 && (1.10) \\ \text{if } lc \neq gc, P_A(lc, gc) &= 0 && (1.11) \\ \sum_{\{A \rightarrow \gamma : \gamma = lc, \ldots\}} P_P(A \rightarrow \gamma|lc, gc) &= 1 && (1.12) \end{aligned}$$

As we will see in Section 2.3, a PLCG is trained using a corpus of annotated parse trees, such as the Penn Treebank. Each tree is converted to its one possible left-corner derivation, effectively a list of parser actions. These parser actions are tallied according to the state of the stack immediately before the action. Due to this training process, only actions that make progress toward the eventual goal receive non-zero probability. This prevents dead end derivations and guarantees that the probability of all possible rule projections sum to 1, as in (1.12). Since all other probabilities are complements to each other within this system, it defines a language model.

CHAPTER 2

Extending the PLCG

2.1 Arc-eager left-corner construction order

Note that in the fourth and sixth steps of the left-corner derivation illustrated in Figure 1.1, more symbols end up on the stack due to the presence of multiple active predicted and observed nonterminals; nonterminals are only taken off the stack once their parent tree is fully constructed. If this parser encounters a deep right-branching structure, like a bottom-up parser, it would have to keep every unresolved nonterminal in the right-branching structure on the stack until the subtree was complete, which means the parser must proceed in linear, rather than constant, time depending on the depth of the right-branching structure.

Human sentence processing research reveals that humans have no problem comprehending and encoding left- and right-branching structures, like those in (α) and (β), but experience processing overload with center-embedded structures like that in (γ). As Resnik (1992) points out, arc-standard left-corner parsing does not accurately mirror the the processing difference observed with these different types of nested structures. A better candidate is *arc-eager* left-corner construction.

(α) The coyote's cousin's nephew despises rattlesnakes. *Left-branching*

(β) These are the locusts that plagued the merchant that sold the cow that sang. *Right-branching*

(γ) The hawk that the buzzard that the lizard fought met was grey. *Center-embedding*

The final step of the left-corner parse (see Figure 1.2), consolidation of predicted and observed nonterminals, can be performed at the very end of the construction of the tree or subtree, which is known as *arc-standard* left-corner parsing, or at the earliest possible step in the parse, which is know as

Step	Stack	Context	Action	Rule		
0	[$\overline{\text{S}}$]	($\overline{\text{S}}$, $\overline{\text{S}}$)	SHIFT$_u$	D	$\rightarrow$	the
1	[D, $\overline{\text{S}}$]	(D, $\overline{\text{S}}$)	PROJECT$_u$	NP	$\rightarrow$	D N
2	[$\overline{\text{N}}$, NP, $\overline{\text{S}}$]	($\overline{\text{N}}$, $\overline{\text{N}}$)	SHIFT$_a$	N	$\rightarrow$	catfish
3	[NP, $\overline{\text{S}}$]	(NP, $\overline{\text{S}}$)	PROJECT$_a$	S	$\rightarrow$	NP VP
4	[$\overline{\text{VP}}$]	($\overline{\text{VP}}$, $\overline{\text{VP}}$)	SHIFT$_u$	V	$\rightarrow$	howls
5	[V, $\overline{\text{VP}}$]	(V, $\overline{\text{VP}}$)	PROJECT$_a$	VP	$\rightarrow$	V PP
6	[$\overline{\text{PP}}$]	($\overline{\text{PP}}$, $\overline{\text{PP}}$)	SHIFT$_u$	P	$\rightarrow$	at
7	[P, $\overline{\text{PP}}$]	(P, $\overline{\text{PP}}$)	PROJECT$_a$	PP	$\rightarrow$	P NP
8	[$\overline{\text{NP}}$]	($\overline{\text{NP}}$, $\overline{\text{NP}}$)	SHIFT$_a$	NP	$\rightarrow$	midnight

Table 2.1: The context and action pairs encountered in the arc-eager left-corner derivation of the tree for *the catfish howls at midnight*, which is given in Figure 1.1. The arc-eager LC tree construction in Figure 2.1 follows these actions.

arc-eager left-corner parsing. An arc-standard left-corner parser typically has access to three actions: SHIFT , which places the nonterminal category of the next token in the input string on top of the stack, which becomes lc for the following parser action; PROJECT , which generates predicted nonterminals onto the stack based on rules of type $A \rightarrow (\gamma_{lc}, ..., \gamma_m)$; and ATTACH , which deletes an observed nonterminal and its predicted counterpart if they are at the top of the stack. An arc-eager left-corner parser, on the other hand, does not have access to ATTACH , but instead uses SHIFT$_a$ (shift-attach), which is equivalent to SHIFT immediately followed by ATTACH within the arc-standard framework, and PROJECT$_a$ (project-attach), which is equivalent to PROJECT immediately followed by a special version of ATTACH that can consolidate observed-predicted pairs regardless of where they are in the stack as long as they are adjacent. In other words, an arc-eager left-corner parser must make attachment decisions as soon as possible, as it does not have the ability to ATTACH once the tree has been fully constructed. I will use SHIFT$_u$ (shift-unattached) and PROJECT$_u$ (project-unattached) for the arc-eager equivalents of SHIFT and PROJECT .

As an illustration, when parsing a simple sentence like *Bill walked*, an arc-standard left-corner parser would shift Bill onto the stack as an NP and then project S $\rightarrow$ NP, VP, resulting in the following stack configuration: [$\overline{\text{VP}}$, S, $\overline{\text{S}}$]. By contrast, after shifting, an arc-eager left-corner parser would PROJECT$_a$ using S $\rightarrow$ NP, VP, deciding immediately that the observed S generated by the projection is indeed the S it is looking for, leaving just a predicted VP on the stack: [$\overline{\text{VP}}$].

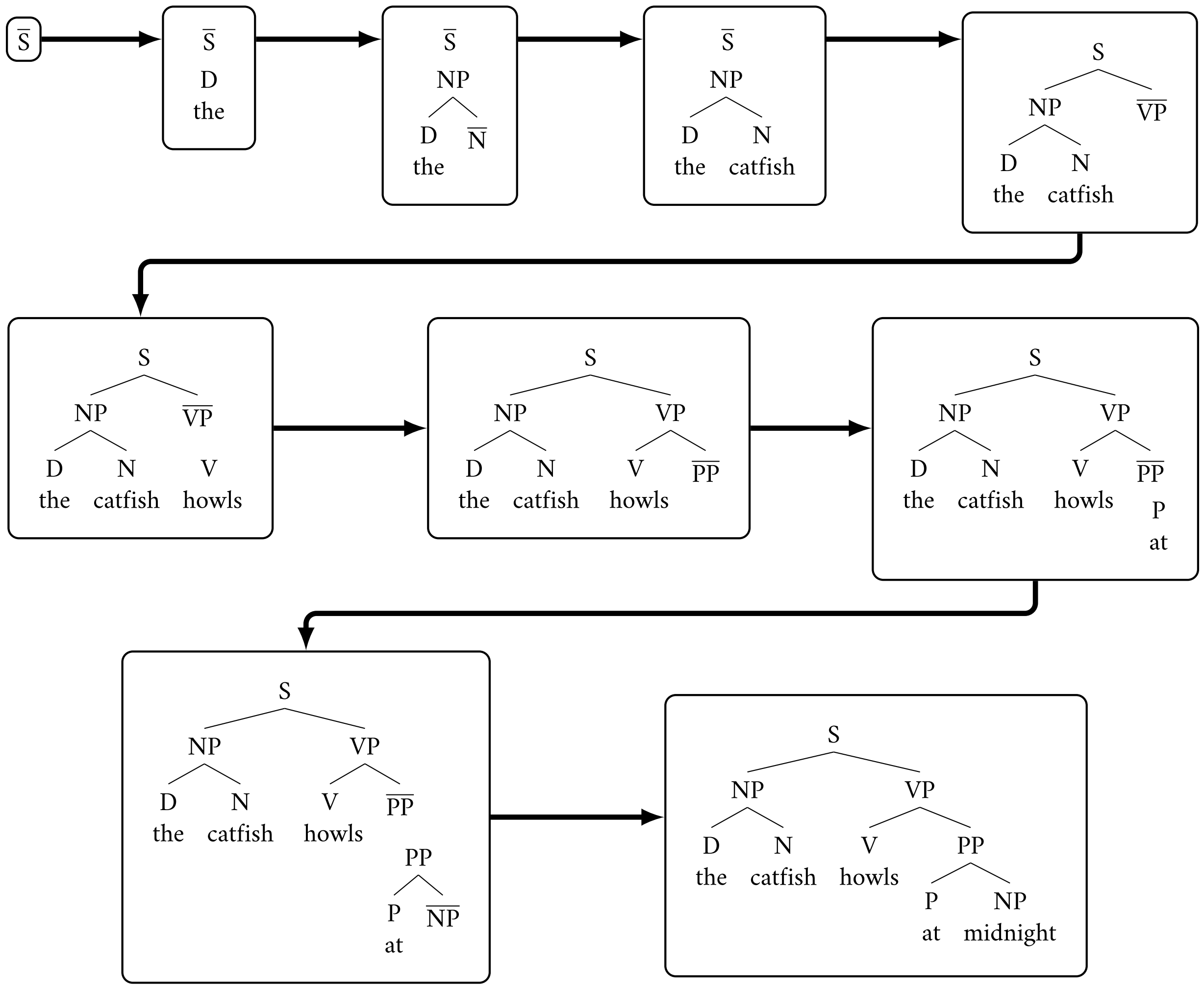

Figure 2.1: Arc-eager left-corner construction order. Arrow labels correspond to generative LC parse operations.

	AS-PLCG	AE-PLCG
Algorithm	arc-standard	arc-eager
Actions	SHIFT	SHIFT_u
	PROJECT	PROJECT_u
	ATTACH	SHIFT_a
		PROJECT_a

Table 2.2: Comparison of the arc-standard and arc-eager conceptions of a PLCG. Formal definitions for arc-standard actions are given in Table 1.4; arc-eager actions are given in Table 2.3.

Action	Rule Type	Input string	Stack
SHIFT_u	C $\to w_1$	$w_1, w_2, ..., w_n \to w_2, ..., w_n$	$[\gamma] \to [\text{C}, \gamma]$
PROJECT_u	X $\to$Y Z	No Change	$[\text{Y}, \gamma] \to [\overline{\text{Z}}, \text{X}, \gamma]$
SHIFT_a	C $\to w_1$	$w_1, w_2, ..., w_n \to w_2, ..., w_n$	$[\overline{\text{C}}, \gamma] \to [\gamma]$
PROJECT_a	X $\to$Y Z	No Change	$[\text{Y}, \overline{\text{X}}, \gamma] \to [\overline{\text{Z}}, \gamma]$

Table 2.3: A formal definition for each of the actions available to an arc-eager left-corner parser. These actions, along with the corresponding rule, make up the derivations for an AE-PLCG. Here, terminal rules are represented by C $\to w$, while nonterminal rules are represented by X $\to$Y Z.

2.2 Deriving probabilities from an arc-eager PLCG

In Section 1.4.4, we saw that when selecting its next action, an arc-standard left-corner parser must first decide deterministically between SHIFT and (ATTACH $\vee$ PROJECT), and then probabilistically between ATTACH and PROJECT . An arc-eager parser, on the other hand, first chooses deterministically between (SHIFT_u $\vee$ SHIFT_a) and (PROJECT_u $\vee$ PROJECT_a), and then probabilistically between its remaining options. The SHIFT_u / SHIFT_a probability P_{SM} of an action is only activated when the top of the stack is a predicted nonterminal. Otherwise, the PROJECT_u / PROJECT_a probability P_{PC} is activated. Thus, probabilities for SHIFT_u and SHIFT_a are complimentary (2.1), as are probabilities for PROJECT_u and PROJECT_a (2.2).

$$\forall gc \in G \sum_{T \to w} (P_{SM}(\text{SHIFT}_u\ T \to w | gc) + P_{SM}(\text{SHIFT}_a\ T \to w | gc)) = 1 \quad (2.1)$$

$$\forall (lc, gc) \in G \sum_{A \to \gamma} (P_{PC}(\text{PROJECT}_u\ A \to \gamma | lc, gc) + P_{PC}(\text{PROJECT}_a\ A \to \gamma | lc, gc)) = 1 \quad (2.2)$$

Notice that that division between $\textsc{shift}_u$ / $\textsc{shift}_a$ and $\textsc{project}_u$ / $\textsc{project}_a$ also divides which types of CFG rules are used in each action. Terminal generation is fully handled by $\textsc{shift}_u$ and $\textsc{shift}_a$ while $\textsc{project}_u$ and $\textsc{project}_a$ only use nonterminal rules. This creates an apparent symmetry in the system. Unlike the arc-standard configuration, which features a rule-defying ATTACH , all of the actions in an arc-eager parser can be abstracted to $\textsc{action}\ A \rightarrow \gamma$. It also means that each action is dependant on its rule-mate. To derive the rule-specific probability of an action, you must compute the probability mass occupied by the general action given the current gc, as well as the probability of the rule-specific action within its pairwise probability space (P_{SM} or P_{PC}).

$$P\left(C_i = \textsc{shift}_u\ T \rightarrow w \,|\, C_1, ..., C_{i\text{-}1}\right) = \begin{cases} P_{SM}(\textsc{shift}_u\ T \rightarrow w|gc) & \text{if ToS is } \overline{\text{NT}} \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$P\left(C_i = \textsc{shift}_a\ T \rightarrow w \,|\, C_1, ..., C_{i\text{-}1}\right) = \begin{cases} P_{SM}(\textsc{shift}_a\ T \rightarrow w|gc) & \text{if ToS is } \overline{\text{NT}} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

$$P\left(C_i = \textsc{project}_u\ A \rightarrow \gamma \,|\, C_1, ..., C_{i\text{-}1}\right) = \begin{cases} P_{PC}(\textsc{project}_u\ A \rightarrow \gamma|lc, gc) & \text{if ToS is NT} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

$$P\left(C_i = \textsc{project}_a\ A \rightarrow \gamma \,|\, C_1, ..., C_{i\text{-}1}\right) = \begin{cases} P_{PC}(\textsc{project}_a\ A \rightarrow \gamma|lc, gc) & \text{if ToS is NT} \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

Since within each action pair, the probability of the grammar's rules given any (lc, gc) pair still sums to one due to the training process, as in (2.1) and (2.2), and all other probabilities are complements of each other, the *arc-eager* version of a PLCG also defines a language model.

The ATTACH / PROJECT probability space split is removed as each of the arc-eager actions is constructive, rather than solely consoliditory like ATTACH . The deterministic choice between a SHIFT -like action and a PROJECT -like action nonetheless ensures that all probabilities will sum to 1.

2.3 Construction and evaluation of models

I constructed two models for training PLCGs[1]: an arc-standard version AS-PLCG that recreates the essential components of the original PLCG (Manning and Carpenter, 1997) and an arc-eager version AE-PLCG that is computationally identical to AS-PLCG with the exception of the function used to generate training oracles, which are effectively lists of actions, derivations of the trees in the corpus. I will use AS-PLCG and AE-PLCG to refer to the training algorithms I used to train arc-standard and arc-eager PLCGs. Trained models, complete with grammars and probabilities, will recieve an extension like *-F*, which I will use to refer to models trained on the *full* Penn Treebank (AS-PLCG-F and AE-PLCG-F).

Both training algorithms followed the same basic procedure, which is shown in Figure 2.2. This training algorithm is an operationalization of the probabilistic derivations in Section 2.2.

PLCG Training Procedure

- A corpus of annotated constituency trees are passed into the model as training input, one tree at a time.
- For each tree, a list of ACTIONS is generated constituting the one correct arc-standard or arc-eager LC derivation.
- Each action is tallied according to the current left-corner lc and goal category gc.
- The counts are normalized across the model by dividing each action's count by the total number of actions observed in the same (lc, gc) context.

Figure 2.2: The generalized training algorithm for both AS-PLCG and AE-PLCG.

All of the models I constructed use Sections 02-24 of the Penn Treebank (PTB) as training data. As trees were fed into the model, nodes underwent some light processing to improve generalizability. Specifically, all leaf nodes (words) were made lowercase to avoid encoding capitalization differences

[1]The python module `ae-plcg`, which was created and used by the present investigation to model and evaluate arc-standard and arc-eager PLCGs, can be found at `https://github.com/phill-barnett/ae-plcg`, along with several fully trained and testable model parameter sets, detailed in Section 3.1 of this thesis.

into the model parameters, nonterminal nodes that have numbered indices (such as NP-SUBJ-9)[2] were stripped of their indices to collocate their probabilities, and each tree's empty root note was given a universal label ROOT. This light processing was applied to all of the models presented here. The AS and AE models that go no further are labeled AS-PLCG-F and AE-PLCG-F. all of the AS-PLCG models learn using the same algorithm, and the same can be said for all AE-PLCG models. The rest of the models differ from AS-PLCG-F and AE-PLCG-F only in the way the training data is pre-processed before training.

	Parameters	Log-Likelihood	ΔLL	AIC	ΔAIC
AS-PLCG-F	116596	-7001030		14235252	
AE-PLCG-F	118696	-6964005	37025	14165401	69851

Table 2.4: Preliminary evaluation results for AS- and AE-PLCG-F, our baseline models, comparing model parameters, log-likelood calculation for full training corpus, and an Akaike Information Criterion (AIC) score calculated from these two numbers. The arc-eager model assigns a significantly higher LL at the cost of 2100 additional parameters. The AIC calculation suggests this tradeoff is worth it; the AE model is better at fitting the observed data.

To compare the AE and AS models, I found the maximum-likelihood values for each of the model parameters using the relative-frequency estimation procedure described in Figure 2.2 and then calculated the maximum likelihood of the models by estimating the log probability of the full PTB corpus of trees, which gives us the Log-Likelihood estimates in Table 2.4. As you can see in Table 2.4, the AE-PLCG assigns a significantly higher probability to the full corpus, but does so at the cost of 2100 additional parameters. These additional parameters are the result of the AE model using SHIFT$_a$ / PROJECT$_a$ in addition to SHIFT$_u$ / PROJECT$_u$ in contexts where the AS model uses only SHIFT / PROJECT. Any time we add more parameters to a model, it is likely to fit the data better, but at the risk of over-fitting. The Akaike Information Criterion (AIC) offers a crude estimation of the Kullback-Liebler discrepancy between a model and the underlying model that theoretically generates the data (Akaike, 1973, 1974). Each model's AIC score is calculated using the formula $AIC = 2k - 2l$, where k is the number of parameters and l is the maximum log-likelihood. The high AIC scores for AS- and AE-PLCG suggest a large divergence from this theoretical underlying model, which is bound to happen when using a

[2] Internal nodes are numbered for various reasons in the Penn Treebank.

data set with so many data points[3]. Nonetheless, the difference in AIC values for the two models is vastly different, which leads to an easy conclusion. When comparing model AIC scores, the lowest AIC score AIC_{min} is assigned to the best fitting model. A difference in AIC scores of less than 2 suggests support for both models, while a difference of more than 10 suggests essentially no support for the higher scoring model. Since Δ AIC is far greater than 10, we can conclude that AE-PLCG-F has a significant advantage over AS-PLCG-F. The next chapter includes an investigation into the source of this AE advantage.

[3]In this case, each action in each derivation in the corpus is a data point, so there are millions of data points per model.

CHAPTER 3

Discussion & Conclusions

The only difference between arc-standard and arc-eager left-corner constructions is that arc-eager collapses its consoliditory ATTACH operations into its constructive operations SHIFT_a and PROJECT_a. This effectively forces an arc-eager construction to make these ATTACH decisions earlier in the derivation. So before diving any deeper, we can conclude that the AE advantage seen in Table 2.4 must somehow arise from this difference. Resnik (1992) points out that arc-eager left-corner parsing is a more likely candidate for the human parsing mechanism due to its handling of stack depth. Specifically, arc-standard LC parsing is vulnerable to unbounded stack size when processing right-branching structures. Because arc-standard parsers wait until a tree is complete before attaching its root to its goal, as they traverse a right-branching structure, the goal category gets buried deeper and deeper on the stack, covered up by each new subtree's goal category and daughter nodes. Arc-eager parsers do not have this problem, as they consolidate predicted and observed nonterminals on-the-fly. Compare Tables 1.2 and 2.1 for an illustration of this difference. Manning and Carpenter (1997), along with their arc-standard PLCG, also propose a PLCG that uses an arc-eager-like system to resolve this AS deficiency for the sake of conditioning on stack size, but do not train or evaluate such a system. The motivation behind not training or evaluating this system may have been motivated in part due to a parsing problem that this difference creates for AE construction: as an AE parser reaches these ATTACH decision points earlier, the parsing decision tree features earlier branching points, which can explode the size of the decision space used for parsing. When conducting a beam search, where only the n most probable parses are kept in memory, this can lead to an over-abundance of possible parses, making low-probability options less likely to be retained, which is particularly problematic for rarer constructions.

So the question posed by the results of the baseline comparison above is not whether AE is a more likely candidate for the human parsing mechanism–it definitely is–but whether the advantage demonstrated here is the result of resolving the stack depth problem from AS, or if it spawns from some other source from within the arc-eager architecture. If the bounded stack size in the AE model were responsible for its advantage in the present evaluation, we would expect right-branching structures to create a larger advantage for the AE model, since right-branching structures are where AE holds this stack depth advantage. But when you think about it, the PLCG models presented here do not really care about stack size, as they are only conditioned on the topmost symbols of the stack. And in fact, looking at the conditioning contexts for a steps 5-9 in Table 1.2 and steps 4-8 in Table 2.1, we see that the two models, despite having wildly different stack depths, have access to the same stack information throughout the right-branching structure. So, due to the independence assumptions of PLCGs, collapsing the stack to only its topmost symbol or symbols, this handling of stack depth cannot contribute to the probabilistic advantage of the AE-PLCG.

This leads one to believe that the advantage, despite being something of a computational disadvantage for parsing, must arise from the earlier attachment decisions made by AE. But not all of these early decisions could be significantly advantageous. For instance, terminal productions, which are always unary, result in an immediate ATTACH action for non-left-corners once the node is shifted and result in no ATTACH action whatsoever for left corners. Because these attachment decisions are automatic and immediate in AS, both models will effectively behave the same way when constructing these nodes.

I posit that the arc-eager advantage detected herein lies primarily in the actions taken by each model given a context of the shape (X, $\overline{\text{X}}$). An AS-PLCG encounters this type of conditioning context in two situations. One is where a left-corner X has been constructed bottom-up and the goal category is also X. This occurs any time a left-corner matches its parent node, wherein the AS-PLCG will choose to PROJECT. I will call these recursive left-corners for simplicity. The other situation that creates this context for AS-PLCG is the result of a non-left-corner being fully constructed (see Table 1.2 steps 3 and 10-13). At this point in the derivation, the AS-PLCG will ATTACH, eliminating both the observed and predicted X from the stack. In contrast, the AE-PLCG will only encounter this recursive left corner (X, $\overline{\text{X}}$) conditioning context in the first of these two scenarios because an AE derivation removes the predicted nonterminal from the stack using PROJECT$_a$. Thus, when an AS-PLCG encounters a recursive left-corner, it must PROJECT, but that PROJECT probability is competing with the ATTACH probability of that same context, which it has likely seen many more times, whereas the AE-PLCG must use PROJECT$_u$, which does not compete with another action type for probability mass[1]. So for any left-corner that appears frequently as a non-left-corner in the training data, the PROJECT probabilities for these recursive left-corner situations will be very low and, conversely, for any non-left-corner that appears frequently as a recursive left-corner, the ATTACH probability will be significantly lower. Without recursive left-corner productions, ATTACH probabilities would always equal 1 and would amount to a bookkeeping step without harming the probability of the construction.

	Parameters	Log-Likelihood	ΔLL	AIC	ΔAIC
AS-PLCG-F	116596	-7001030		14235252	
AE-PLCG-F	118696	-6964005	37025	14165401	69851
AS-PLCG-D	35048	-2961900		5993897	
AE-PLCG-D	37148	-2924875	37025	5924046	69851
AS-PLCG-UC	117530	-6979550		14194161	
AE-PLCG-UC	119618	-6942527	37024	14124289	69872
AS-PLCG-BR	185086	-6367714		13105600	
AE-PLCG-BR	186863	-6338194	29520	13050115	55486
AS-PLCG-BL	116930	-6999096		14232052	
AE-PLCG-BL	119030	-6962071	37025	14162201	69851
AS-PLCG-RX	37026	-991945		2057942	
AE-PLCG-RX	37026	-991945	0	2057942	0

Table 3.1: Preliminary evaluation results for AS- and AE-PLCG given varied training data comparing model parameters, maximum log-likelood, and an Akaike Information Criterion (AIC) score calculated from these two numbers. Descriptions of the training data variations are given at the beginning of Section 3.1.

3.1 Training Variants of AS- and AE-PLCG

To test the hypothesis that recursive left-corners are the main source of disadvantage for the AS-PLCG, I trained AS- and AE-PLCG in various ways by pre-processing the PTB before feeding it to the models. The evaluation results for all of these training variants are given in Table 3.1.[2] In PLCG-D, the models

[1] In this case, PROJECT$_u$ must only compete with other PROJECT$_u$ actions that use different production rules

[2] Trained model parameters for each of these variants are available at `https://github.com/phill-barnett/ae-plcg/tree/main/dfiles`. Model parameters are stored as JSON files, which can be easily loaded using the `lcg()` object from the `ae-plcg` python module.

were fed a delexicalized version of the PTB by replacing all leaf nodes with w. In PLCG-UC, the models were fed a version of PTB wherein unary productions were collapsed. PLCG-BR and PLCG-BL result from feeding the models a binarized version of the PTB, BR using right-factored binarization and BL using left-factored binarization. See Figure 3.1 for a more formal elaboration of the binarization process. Finally, in PLCG-RX, the models were fed a much smaller version of the PTB wherein trees featuring recursive left-corner productions were removed from the training data.

The delexicalized and unary-collapsed versions of the two models do not show a significant change in the difference between AS and AE. As predicted, due to the deterministic nature of the actions needed for these constructions, there is no difference in the way AS and AE handle these structures. This is true down to the tree level. Further close analysis of action-by-action probabilities, like the probabilities shown in 3.2, reveals that for the vast majority of AE actions, the corresponding action in the AS model has the exact same probability. Examining action-by-action probabilities reveals that in the event a SHIFT$_a$ and its corresponding SHIFT do not have the same probability, the difference is made up in a corresponding ATTACH . The only places the two models ever diverge are unary productions[3], which tend to be relatively small differences, and more significantly, at PROJECT points. Most of the time, the AE model is rewarded for making the earlier decision between PROJECT$_u$ and PROJECT$_a$ as it has been conditioned on the most likely scenario for each context. Specifically, the largest points of difference come from contexts of shape (X,$\overline{\text{X}}$), because these are the contexts where the AS model must decide between ATTACH and PROJECT . In the Penn Treebank, the vast majority of these contexts have a strong tendency to ATTACH , which leads the AS model to a rather low probability in instances where this is not the case. If the opposite were true, if most of these contexts led to a PROJECT , the AE model would still have the advantage as the AS model would be penalized in the instances where the opposite was true.

Notice that the left-binarized training corpus affects no change in the difference between AS- and AE-PLCG, while the right-binarized data results in a significant decrease in the AE advantage seen in the baseline models. When constructing the unbinarized tree in Figure 3.1, both models construct the

[3]It is quite curious that the UC models do not significantly impact the difference between AS and AE. Taken with the results from the RX variants, this is a strong indication that the ATTACH probabilities used in resolving unary productions are affected by recursive left-corner productions elsewhere, but do not contribute to a difference in the models themselves.

Context	Action	Rule	Prob.	Log Prob.	Cum. Log Prob.
Arc-Standard Actions					
($\overline{\text{ROOT}}$, $\overline{\text{ROOT}}$)	SHIFT	DT $\rightarrow w$	0.23	-1.47	-1.47
(DT, $\overline{\text{ROOT}}$)	PROJECT	NP-SBJ $\rightarrow$DT NN	0.24	-1.44	-2.91
($\overline{\text{NN}}$, $\overline{\text{NN}}$)	SHIFT	NN $\rightarrow w$	1	0	-2.91
(NN, $\overline{\text{NN}}$)	ATTACH		1	0	-2.91
(NP-SBJ, $\overline{\text{ROOT}}$)	PROJECT	S $\rightarrow$NP-SBJ VP .	0.76	-0.27	-3.18
($\overline{\text{VP}}$, $\overline{\text{VP}}$)	SHIFT	VBZ $\rightarrow w$	0.15	-1.92	-5.1
(VBZ, $\overline{\text{VP}}$)	PROJECT	VP $\rightarrow$VBZ PP	0	-6.13	-11.23
($\overline{\text{PP}}$, $\overline{\text{PP}}$)	SHIFT	IN $\rightarrow w$	0.88	-0.12	-11.35
(IN, $\overline{\text{PP}}$)	PROJECT	PP $\rightarrow$IN NP	0.89	-0.12	-11.47
($\overline{\text{NP}}$, $\overline{\text{NP}}$)	SHIFT	NN $\rightarrow w$	0.11	-2.22	-13.69
(NN, $\overline{\text{NP}}$)	PROJECT	NP $\rightarrow$NN	0.62	-0.48	-14.17
(NP, $\overline{\text{NP}}$)	ATTACH		0.73	-0.32	-14.49
(PP, $\overline{\text{PP}}$)	ATTACH		1	0	-14.49
(VP, $\overline{\text{VP}}$)	ATTACH		0.97	-0.03	-14.52
($\overline{.}$, $\overline{.}$)	SHIFT	. $\rightarrow w$	1	0	-14.52
(., $\overline{.}$)	ATTACH		1	0	-14.52
(S, $\overline{\text{ROOT}}$)	PROJECT	ROOT $\rightarrow$S	0.94	-0.06	-14.58
(ROOT, $\overline{\text{ROOT}}$)	ATTACH		1	0	-14.58
Arc-Eager Actions					
($\overline{\text{ROOT}}$, $\overline{\text{ROOT}}$)	SHIFT$_u$	DT $\rightarrow w$	0.23	-1.47	-1.47
(DT, $\overline{\text{ROOT}}$)	PROJECT$_u$	NP-SBJ $\rightarrow$DT NN	0.24	-1.44	-2.91
($\overline{\text{NN}}$, $\overline{\text{NN}}$)	SHIFT$_a$	NN $\rightarrow w$	1	0	-2.91
(NP-SBJ, $\overline{\text{ROOT}}$)	PROJECT$_u$	S $\rightarrow$NP-SBJ VP .	0.76	-0.27	-3.18
($\overline{\text{VP}}$, $\overline{\text{VP}}$)	SHIFT$_u$	VBZ $\rightarrow w$	0.15	-1.92	-5.1
(VBZ, $\overline{\text{VP}}$)	PROJECT$_a$	VP $\rightarrow$VBZ PP	0	-6.13	-11.23
($\overline{\text{PP}}$, $\overline{\text{PP}}$)	SHIFT$_u$	IN $\rightarrow w$	0.88	-0.12	-11.35
(IN, $\overline{\text{PP}}$)	PROJECT$_a$	PP $\rightarrow$IN NP	0.88	-0.12	-11.47
($\overline{\text{NP}}$, $\overline{\text{NP}}$)	SHIFT$_u$	NN $\rightarrow w$	0.11	-2.22	-13.7
(NN, $\overline{\text{NP}}$)	PROJECT$_a$	NP $\rightarrow$NN	0.41	-0.89	-14.59
($\overline{.}$, $\overline{.}$)	SHIFT$_a$	(., w)	1	0	-14.59
(S, $\overline{\text{ROOT}}$)	PROJECT$_a$	ROOT $\rightarrow$S	0.94	-0.06	-14.65

Table 3.2: Action-by-action probabilities as calculated by AS-PLCG-D and AE-PLCG-D for our running example, *the catfish howls at midnight*. Some slight changes were made to make the tree conform with PTB standards, including the re-labeling of nodes and addition of a ROOT and period phrase.

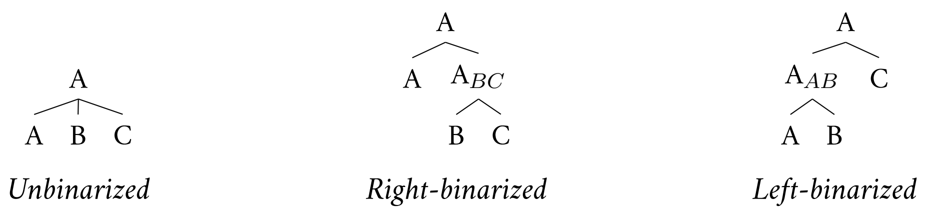

Figure 3.1: Demonstration of right- and left-factorization. Notice that in subtrees with recursive left corners, both binarization directions preserve this embedding. The key difference here is that right-binarization repositions B as a left-corner, while left-binarization creates a new left corner A_{AB}, which never results in a (A_{AB}, $\overline{\text{A}_{AB}}$) conditioning context.

left-embedded A bottom up, resulting in the context (A, $\overline{\text{A}}$). AS-PLCG then uses project A $\rightarrow$A B C to predict the remaining daughters. Because the nested A is a left-corner, and therefore constructed bottom-up, it is not later attached. B and C, however, come on to the stack as predicted nonterminals, and therefore must be attached once they are fully constructed. AS attach actions always occur in the context (X, $\overline{\text{X}}$), which happens to be the same conditioning context we see once a recursive left-corner is constructed. If X never occurs in a recursive left-corner position, the AS model will always attach in the context (X, $\overline{\text{X}}$), which means $P\left(\text{attach} \,\middle|\, X, \overline{\text{X}}\right) = 1$. However, whenever it does occur in a recursive left-corner position, X takes probability mass away from $P\left(\text{attach} \,\middle|\, X, \overline{\text{X}}\right)$ in favor of a project rule.

Left-binarization, as in PLCG-BL, results in no change in the difference between AS and AE, because all recursive left-corner and attach relationships are maintained. Using Figure 3.1 as illustration, while A_{AB} is created and serves as a left-corner, it will never occur in a *recursive* left-corner position (as it will always be daughter to A), nor will it ever occur outside of a left-corner position, meaning that the project probabilities associated with the node never have to compete with attach probabilities. Right-binarization, as in PLCG-BR, results in a decreased disadvantage for AS, as you can see in Table 3.1. The reason for this has nothing to do with A still being nested under A. Rather, notice that in left-binarization, the non-left-corner position of B and C is maintained. However, in the right-binarized tree, B is now a left-corner, which means it will no longer be attached, so the context (B, $\overline{\text{B}}$) recieves one less tally for attach . Thus, if B occurs elsewhere in the corpus as a recursive left corner,which is now more likely due to the binarization itself, the project probabilities in that instance will share less of their probability mass with attach, leading to higher probabilities for the derivation.

The PLCG-RX variants of the models were created as a final verification of this source of difference. All trees containing a recursive left-corner were left out of the training data. Thus, AS-PLCG will only encounter (X, $\overline{\text{X}}$) contexts when it needs to ATTACH , and AE-PLCG will never encounter this type of context at all. Thus for a given tree, AS SHIFT actions will have equivalent corresponding AE SHIFT$_u$ or SHIFT$_a$ actions, and PROJECT will have equivalent corresponding PROJECT$_u$ or PROJECT$_a$ actions and all ATTACH actions will have a probability 1, leading both models to the same probability, as well as the same number of parameters[4]. As Table 3.1 shows, the RX models confirm the hypothesis as there is no longer any difference whatsoever between AS and AE.

3.2 Conclusion

This paper presents a comparison of arc-standard and arc-eager left-corner constuction order using PLCGs as a testbed. We can conclusively see that the learning advantage AE-PLCG has over an AS-PLCG does not come from arc-eager construction order's finite stack size advantage, but rather from instances where a tree's parent node and left-corner are the same category. Future work should look toward connecting this to the psychological reality of the human parsing mechanism. There are many instances in the corpus where the AE model's probabilities strongly lean toward connecting or projecting, but the derivation calls for the opposite, resulting in a lower arc-eager probability. If humans experience processing difficulty in these cases, it would suggest that this advantage is based on linguistic reality, whereas a lack of processing difficulty in these cases could suggest the advantage observed here is simply an artifact of the model architecture.

[4]But Phill, you say, if AS-PLCG runs into a lot more contexts than AE-PLCG, of the form (X, $\overline{\text{X}}$) how does it not have more parameters? Well, because these contexts are no longer decision points. They are fully deterministic points of ATTACH . When the models are trained on the RX data, ATTACH is effectively reduced to a bookkeeping step that makes up the difference in the models at no probabilistic cost.

Bibliography

Abney, S. P. and Johnson, M. (1991). Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

Akaike, H. (1973). Maximum likelihood identification of gaussian autoregressive moving average models. *Biometrika*, 60(2):255–265.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Charniak, E. (1996). *Statistical language learning*. MIT press.

Demers, A. J. (1977). Generalized left corner parsing. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 170–182.

Dolatian, H. and Heinz, J. (2018). Modeling reduplication with 2-way finite-state transducers. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 66–77. Association for Computational Linguistics.

Goldsmith, J. (2002). Probabilistic models of grammar: Phonology as information minimization. *Phonological Studies*, 5:21–46.

Hale, J. (2016). Information-theoretical complexity metrics. *Language and Linguistics Compass*, 10(9):397–412.

Hale, J. (2017). Models of human sentence comprehension in computational psycholinguistics. In *Oxford Research Encyclopedia of Linguistics*.

Hunter, T. (2018). Formal methods in experimental syntax. *The Oxford Handbook of Experimental Syntax*.

Jelinek, F., Lafferty, J., and Mercer, R. (1992). Basic methods of probabilistic context free grammars. *Speech Recognition and Understanding, NATO ASI Series*, 75:345–360.

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

Johnson-Laird, P. N. (1983). *Mental models: Towards a cognitive science of language, inference, and consciousness*. Number 6. Harvard University Press.

Levy, R. (2013). Memory and surprisal in human sentence comprehension.

Manning, C. D. and Carpenter, B. (1997). Probabilistic parsing using left corner language models. In *Proceedings of the Fifth International Workshop on Parsing Technologies (IWPT-97)*, pages 147–158. MIT.

Manning, C. D. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Miller, G. A. and Chomsky, N. (1963). Finitary models of language users.

Mitchell, T. M. (1980). The need for biases in learning generalizations. *Department of Computer Science, Laboratory for Computer Science Research, Rutgers University*, pages 184–191.

Nederhof, M.-J. (1993). Generalized left-corner parsing. In *Proceedings of the sixth conference on European chapter of the Association for Computational Linguistics*, pages 305–314. Association for Computational Linguistics.

Resnik, P. (1992). Left-corner parsing and psychological plausibility. In *Proceedings of the 14th Conference on Computational Linguistics - Volume 1*, COLING '92, pages 191–197, USA.

Rørnes, K. M. (2019). Mental models in programming. Master's thesis.

Rosenkrantz, D. J. and II, P. M. L. (1970). Deterministic left corner parsing (extended abstract). In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*.

Shieber, S. M., Schabes, Y., and Pereira, F. C. (1995). Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36.

Solomonoff, R. J. (1997). The discovery of algorithmic probability. *Journal of Computer and System Sciences*, 55(1):73–88.