UNIVERSITY OF CALIFORNIA, SAN DIEGO

Automatic Grammar Correction: Using PCFGs and Whole Sentence Context

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

in

Computer Science

by

Vineet Kumar

Committee in charge:

Professor Roger Levy, Chair
Professor Charles Elkan
Professor Lawrence Saul

2012

The thesis of Vineet Kumar is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2012

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would first like to thank my adviser Roger Levy for his invaluable mentorship and giving me the opportunity to work on this project. I would also like to thank Albert Park for answering all my questions as I extended his work on automatic grammar correction. I also owe a great debt of gratitude to my committee members Charles Elkan and Lawrence Saul for taking time to review my work and give me feedback.

VITA

2007	Bachelor of Technology in Information Technology, Malaviya National Institute of Technology, Jaipur, India
2012	Master of Science in Computer Science, University of California, San Diego

ABSTRACT OF THE THESIS

Automatic Grammar Correction: Using PCFGs and Whole Sentence Context

by

Vineet Kumar

Master of Science in Computer Science

University of California, San Diego, 2012

Professor Roger Levy, Chair

We explore the problem of automatic grammar correction and extend the work of [Park and Levy, 2011]. We use a noisy channel model that uses whole sentence context to generate a grammatically correct sentence with the highest probability. Our major contribution is to explore the idea of using a better language model than n-gram to represent the rules of the English language. We use Probabilistic Context Free Grammar (PCFG) and explain how we can combine it with noise models that are represented with Weighted Finite State Transducers (wFST) to build our noisy channel model. We also extend V-expectation semirings [Eisner, 2002] to CKY parsing, a popular parsing algorithm for parsing a sentence of a language.

Chapter 1

Introduction

How do humans make mistakes that lead to a grammatically incorrect sentence? We can perhaps categorize some of these mistakes as selecting an incorrect form of a word, inserting and deleting words and making spelling errors. Table 1.1 lists some examples of common grammatical mistakes:

Table 1.1: Examples of some common grammatical mistakes

Mistake Category	Correct Sentence	Incorrect Sentence
Wordform choice	John *walks*	John *walk*
Spelling	*People* talk	*Poeple* talk
Insertion and Deletion	I *will* go.	I go

Thus, if we consider grammatical errors as noise that is added to an error-free sentence, we can build a noisy channel model that represents the mistakes people make. In this thesis, we explore the problem of doing automatic grammar correction using a noisy channel model and whole sentence context. We state our problem formally in the section below.

1.1 Problem Statement

We explore the problem of predicting the most likely grammatically correct sentence given an incorrect sentence. This task is also called as automatic grammar

correction. In this thesis, we explore the automatic grammar correction problem using a noisy channel model and extend the work of [Park and Levy, 2011].

A noisy channel model is a generative model that has a component that generates error free sentences. We call this component the **language model**. It further has a noise component that modifies the original sentence by adding errors or noise stochastically. We call this component the **noise model**. Thus, a noisy channel model can be considered as a generative model of language that emits all sentences of a language and some of those sentences have errors[1]. This model generates sentences that we observe and may wish to correct (if they are grammatically incorrect).

[Park and Levy, 2011] introduce us to using noisy channel models for automatic grammar correction. Their noisy channel model also has two components - a language model and a noise model. They use a n-gram model as the language model, and use various noise models (spelling correction, wordform, prepositional error, article choice error, among other models) which they represent using Weighted Finite State Transducers (wFST). We will introduce wFST later in Section 2.1.2. At a high level a wFST can be seen as a string transformer, that given an input and state of the system, generates a new output. Figure 1.1 depicts their noisy channel model. Notice that both their language and noise model can be represented using a wFST and the composition of the language and model is again a wFST.

[Park and Levy, 2011] also state that one can view the task of grammar correction as a machine translation task. A standard machine translation task involves translating a sentence from a Language F (say French) to Language E (say English). If we consider the grammatically incorrect language to be Language F, and the grammatically correct language to be Language E, then automatic grammar correction can be seen as a task in translating from the grammatically incorrect to grammatically correct language. However, for learning the parameters of the model, statistical machine translation make use of parallel corpus (available say by two same books translated in Language E and F). Unfortunately, no such

[1] A language model generates all error-free sentences; in combination with the noise model our noisy channel model generates error-free and erroneous sentences

parallel corpus exists for grammar correction. Thus a supervised learning approach cannot be used. Thus as suggested by [Park and Levy, 2011], we use unsupervised learning for our task. The parallel to machine translation, however is useful for the task of evaluation of our models, which we will discuss later in Section 4.3.

In our thesis, we use a noisy channel model too. We however use a Probabilistic Context Free Grammar (PCFG) as our language model. We use the same noise models as [Park and Levy, 2011], that is our noise models are represented by a wFST. We explain PCFG in Section 2.1.1. A PCFG, at a high level is a generative model of language that emits all possible sentences of a language with some probability. Another way to look at PCFG is that it generates sentences that are probabilistically distributed over all the sentences of the language. We believe a PCFG would do a better job in capturing the structure of a sentence than a n-gram model, and thus when combined with a noise model will help us make better predictions. Table 1.2 summarizes how our noisy channel model is different.

In this thesis, our major contribution is to explain how a PCFG language model can be combined with a wFST noise model to build a noisy channel model. We use this noisy channel model to predict the most likely grammatically correct sentence for a given observed sentence.

1.2 Organization of the Thesis

In rest of the thesis, we explain the background needed to understand our noisy channel model. We further, demonstrate with the help of a toy example, how training and decoding work for our model. In later chapters of our thesis, we explain how we implement our model. We also explain changes we made to some open source software tools we use, and the data sets for our task. We conclude by discussing the results we get by using a wordform choice noise model with our language model and discussing future work.

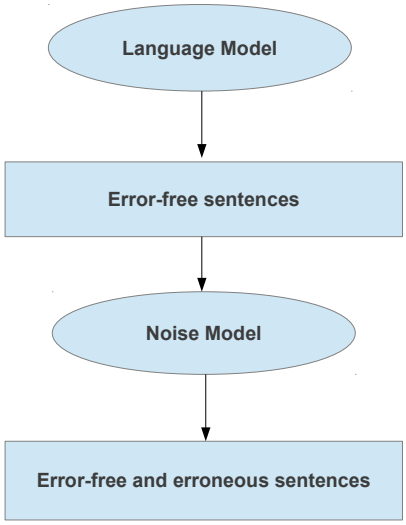

Figure 1.1: [Park and Levy, 2011] Noisy channel model: the language model is a n-gram model, represented using wFST and the noise model is also represented using wFST, the composition gives back a wFST

Table 1.2: How our noisy channel model differs from [Park and Levy, 2011]

Noisy Channel Model	Language Model	Noise Model
[Park and Levy, 2011]	n-gram Model	wFST
Our Noisy Channel Model	**PCFG**	wFST

Chapter 2

Background

2.1 Noisy Channel Model

As explained in Section 1.1, our noisy channel model is constructed by the composition of a language and a noise model. The language model generates error-free sentences distributed probabilistically over all the sentences of a language (in our case English language). The noise model introduces errors stochastically to an error-free sentence. Thus, when a language model is composed with a noise model, we get a noisy channel model that generates error-free as well as erroneous sentences. This noisy channel is a generative model of language that generates the sentences that we may wish to correct. Figure 2.1 shows an example sentence that can be generated by our noisy channel model.

We use Probabilistic Context Free Grammar (**PCFG**) as our **language model** and Weighted Finite State Transducer (**wFST**) as our **noise model**. We will now define PCFG and wFST, so as to facilitate further discussion of our model.

2.1.1 Probabilistic Context Free Grammar (PCFG)

A PCFG is a 5 tuple (N, T, R, S, P) where:

- N is a finite set of **non-terminals**
- T is a finite set of terminals, or **words** of our language.

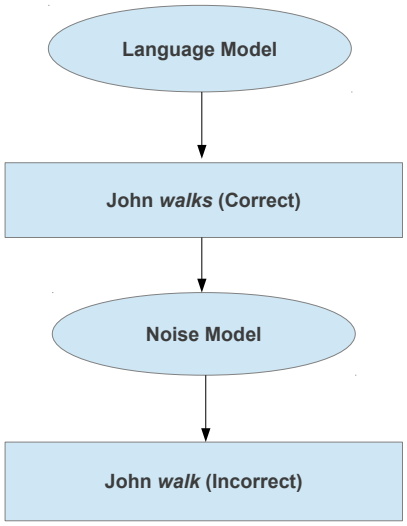

Figure 2.1: An example sentence generated by our noisy channel model. Notice how the word *walks* was replaced by *walk*.

- R is a set of **expansion rules** of the form $X \rightarrow Y_1 Y_2 ... Y_n$, where $X \in N$, $Y_i \in (T \cup N)$ and $i = 1$ to n.
- S is a unique **start symbol**, such that $S \in N$.
- P is a **probability function** that maps a rule R to $[0, 1]$.
- For any $X_i \in N$, we have the following constraint:

$$\sum_{X \rightarrow Y_1 ... Y_n \in R : X = X_i} P(X \rightarrow Y_1 ... Y_n) = 1 \tag{2.1}$$

Equation 2.1 states that the probability of all rules with same non-terminal on the LHS must sum to 1.

Thus, starting with the Root symbol S and using the expansion rules R, till we end in terminals, we can generate a group of words or a **sentence**. The sentence generated by using a set of rules is also referred as **yield** and the rules can be conveniently represented by a **derivation tree**. Table 2.1 shows a toy PCFG. Note that this PCFG generates sentences of the form $(a+b)(a+b)(a+b)^*$. Figure 2.2 shows some sample derivation trees for our toy PCFG along with the yield.

Table 2.1: A toy PCFG with start symbol as S, $N = \{S, A\}$ and $T = \{a, b\}$. Probability for each rule is specified in the brackets

Rule-Number	Rule	P(Rule)
R_1	$S \rightarrow AA$	(1)
R_2	$A \rightarrow AA$	(0.25)
R_3	$A \rightarrow a$	(0.25)
R_4	$A \rightarrow b$	(0.5)

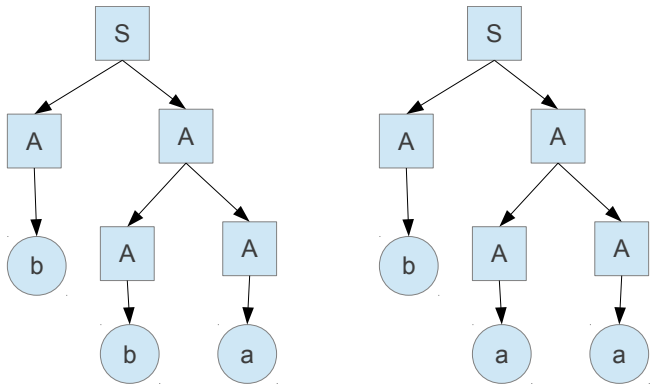

Figure 2.2: Derivation trees for sentences generated by our toy PCFG. Left tree generates the sentence "b b a" and right tree generates the sentence "b a a"

Probability of generating a sentence by a PCFG is simply the **product of the probabilities of the rules** used. Let us look at the probabilities of the trees generated by our PCFG. We will use the probabilities from Table 2.1 for our calculations:

$$\begin{aligned} P(\text{“}bba\text{”}) &= P(S \rightarrow AA) \cdot P(A \rightarrow b) \cdot P(A \rightarrow AA) \cdot P(A \rightarrow b) \cdot P(A \rightarrow a) \\ &= 1 \cdot 0.5 \cdot 0.25 \cdot 0.5 \cdot 0.25 \\ &= 0.015625 \end{aligned}$$

$$\begin{aligned} P(\text{“}baa\text{”}) &= P(S \rightarrow AA) \cdot P(A \rightarrow b) \cdot P(A \rightarrow AA) \cdot P(A \rightarrow a) \cdot P(A \rightarrow a) \\ &= 1 \cdot 0.5 \cdot 0.25 \cdot 0.25 \cdot 0.25 \\ &= 0.078125 \end{aligned}$$

In this section, we defined a PCFG, explained how it can generate sentences, and how to calculate the probability of a yield. We hope this gives the reader a general

idea of how a PCFG can be seen as generative model of a language. We will now define a wFST.

2.1.2 Weighted Finite State Transducer (wFST)

A wFST[1] is a 6 tuple (Q, T, T, Q_s, Q_f, P) where:

- Q represents a finite set of **states**.
- T represents a finite set of **words** .
- T represents a **state transition** $(Q_0^a) \rightarrow (Q_1^b)$, where $Q_0, Q_1 \in Q$ and $a, b \in T$. Thus, at state Q_0 and after receiving "a" the model changes its state to Q_1 and emits the word "b". In effect, it transformed the input word "a" to "b".
- Q_s is a set of **initial states** .
- Q_f is a set of **final states** .
- P is a **probability function**, such that $P(T_i) \in [0, 1]$ where $T_i \in T$
- Additionally, for every input word a_i we have the following constraint:

$$\sum_{Q_i^a \rightarrow Q_j^b \in\ T\ \ a=a_i} P(Q_i^a \rightarrow Q_j^b) = 1 \tag{2.2}$$

Equation 2.2 allows us to represent conditional probability distribution. It can be inferred as 'What is the probability of generating an output word **given an input word**?'. We need this as our noise model, which takes the input from the language model, and generates an output ("b" generated when input word was "a")

Figure 2.3 shows a Toy Noise Model represented by a wFST. This wFST indicates that the word "a" is never changed, but the word "b" can be changed to "a" with a probability of p_1. Thus if this transducer receives a word "b" it can

[1] our wFST representation does not care about the final weight of the state. We can assume the final state probability to be 1

change it to "a" with a probability of p_1 and it remains as "b" with a probability of $(1 - p_1)$. Changing the word from "b" to "a" when it was not intended by our language model, can be thought of as an **error**.

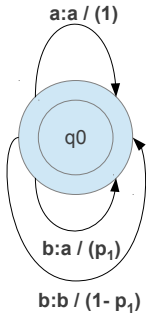

Figure 2.3: A toy wFST with $Q \in \{q0\}$, $T = \{a, b\}$, each state transition is represented with an arrow, and probability is represented in brackets

Table 2.2: State transitions for toy Noise Model in Figure 2.3

State Transition	P(Transition)
$q_0^a \rightarrow q_0^a$	1
$q_0^b \rightarrow q_0^a$	p_1
$q_0^b \rightarrow q_0^b$	$(1 - p_1)$

We have now defined PCFG and wFST, explaining how a PCFG acts as a generative model of a language, and how a wFST can introduce errors. We will now talk about the free parameters of our language and noise models.

2.2 Free Parameters

We did not mention until now how we learn the rules and probabilities for our language model, or how we learn the parameters and state transitions for our noise model. We will, for the rest of the chapter assume that the language model is **fixed**. We will discuss in Section 3.1 how we learn the language model. We will also assume that the **state transitions** for our noise model are **fixed**. Thus the **only unknown** we have is the **probabilities of state transition** for our noise

model. For example, for our toy noise model, we want to learn p_1, which is the only free parameter.

We have now explained the free parameters of our noise model and will now explain how we train our model and learn noise model parameters.

2.3 Training

Our noisy channel model generates error-free as well as erroneous sentences. A supervised learning is an approach where we have labeled data. For our task, such labeled data would be a corpus of incorrect sentence and the corresponding correct sentence. However, as we will explain below, we do not have access to such a corpus and thus cannot use supervised learning. Thus, we use an unsupervised learning approach, which can be used in cases where there is no labeled data, and all we have is observed data (in our case a data set of incorrect grammar sentences). Our aim in unsupervised learning is to learn the free parameters of our model such that the likelihood of the observed data is maximized.

2.3.1 Lack of Parallel Training Corpus

[Park and Levy, 2011] specify that an automatic grammar correction task can also be viewed as machine translation from one language to another. Let us assume that the error-free (grammatically correct) sentences belong to a Language E, and the erroneous sentences belong to a Language F. Thus automatic grammar correction, can be viewed as translation from language F (grammatically incorrect) to language E (grammatically correct). However, unlike machine translation we do not have access to a large parallel corpus of text in the languages E and F. Thus, we cannot use supervised learning and we take a **unsupervised learning** approach.

2.3.2 What does training mean for our noise model ?

We stated earlier, that we wish to learn the value of p_1, that being the only free parameter of our model (Figure 2.3). p_1 is the probability of state transition $b \rightarrow a$. Also, Equation 2.2 specifies that for all transitions with same input word, the probability of transitions sum to 1. Thus, in our toy noise model:

$$P(b \rightarrow a) + P(b \rightarrow b) = 1$$

Now, let us say we have just one observed sentence S^1_{obs} = "*bba*". Now the first two words of this sentence "b" can only be generated if we traverse $b \rightarrow b$ in the noise model. However, "a" can either be generated by traversing $a \rightarrow a$ or $b \rightarrow a$. Thus, we have two paths, which we denote as derivations for generating "b b a" (Table 2.3). We denote the number of times a state transition by C. C_i denotes the counts for a single derivation:

Table 2.3: Noise Model Transitions to generate S^1_{obs} = "*bba*"

Derivation	$Word_1$ = "b"	$Word_2$ = "b"	$Word_3$ = "a"	$C_i(b \rightarrow b)$	$C_i(b \rightarrow a)$
D_1	$b \rightarrow b$	$b \rightarrow b$	$b \rightarrow a$	2	1
D_2	$b \rightarrow b$	$b \rightarrow b$	$a \rightarrow a$	2	0

Let us now assume that the D_1 occurs with a probability of $P(D_1)$ and D_2 occurs with $P(D_2)$. To calculate p_1, we need to take the ratio of expected counts of $b \rightarrow a$ and the total counts with b as the input symbol, given an observed sentence:

$$p_1 = \frac{E[C(b \rightarrow a)|S^1_{obs}]}{E[C(b \rightarrow a)|S^1_{obs}] + E[C(b \rightarrow b)|S^1_{obs}]} \tag{2.3}$$

We have seen two derivations for our observed sentence. Thus the expected counts can be computed as:

$$E[C(b \rightarrow a)|S^1_{obs}] = \frac{\sum_{i=1}^{2} C_i(b \rightarrow a) \cdot P(D_i|S^1_{obs})}{\sum_{i=1}^{2}(C_i(b \rightarrow a) + C_i(b \rightarrow b)) \cdot P(D_i|S^1_{obs})} \tag{2.4}$$

if D denotes the set of all derivations for a sentence S^i_{obs}, we can generalize Equation 2.4 as follows:

$$E[C(b \rightarrow a)|S^i_{obs}] = \frac{\sum_{D_i \in D} C_{D_i}(b \rightarrow a) \cdot P(D_i|S^i_{obs})}{\sum_{D_i \in D} C_{D_i}(b \rightarrow a) + C_{D_i}(b \rightarrow b) \cdot P(D_i|S^i_{obs})} \tag{2.5}$$

If we have n observed sentences denoted by S^i_{obs} for i = 1 to n , we need to sum all the expected counts given each sentence, to get the total counts. Thus, Equation 2.3 becomes:

$$p_1 = \frac{\sum_{i=1}^{n} E[C(b \rightarrow a)|S^i_{obs}]}{\sum_{i=1}^{n} E[C(b \rightarrow a)|S^i_{obs}] + E[C(b \rightarrow b)|S^i_{obs}]} \tag{2.6}$$

Equation 2.5 needs us to compute $P(D_i|S^i_{obs})$ which can be specified as:

$$P(D_i|S^i_{obs}) = \frac{P(D_i, S^i_{obs})}{P(S^i_{obs})} \tag{2.7}$$

The total probability of a sentence $P(S^i_{obs})$ is the sum of the joint probability of all derivations and the sentence:

$$P(S^i_{obs}) = \sum_{D_j \in D} P(D_j, S^i_{obs}) \tag{2.8}$$

In the next section, we explain how to compute the joint probability of a derivation and observed sentence.

2.3.3 Calculating the total probability ($P(S^i_{obs})$) of an observed sentence

If we compose our noisy channel model with an observed sentence, it restricts the output of the model to the observed sentence. This is because all other sentences will lead to a non-final state in the wFST, and thus would be discarded by the model.

We can construct a wFST trivially that generates an observed sentence. Let us revisit S^1_{obs} = "*bba*". This sentence has three words. To construct a wFST, we will add a state for each word, and a final state, that indicates end of the sentence. Thus, we get a total of four states, and for each state transition we use the same word as the input and output. We further use unit probabilities for each transition.

We can now convert this resultant wFST (noise model composed with an observed sentence) to a word lattice. This means we can discard the output from each transition, and parse this word lattice with our PCFG language model. Let

us summarize the steps we have discussed so far. (Figure 2.4) shows the first three steps:

- For each observed sentence, create an observed wFST with unit probabilities and input and output word same as the word in the sentence. Add a final state indicating the end of the sentence.
- Compose the noise model with this observed sentence, restricting the noise model to only generate the observed sentence.
- Convert this composed wFST to a word lattice by discarding the output words
- While Parsing the input word lattice with the PCFG multiply the lattice word probability with the rule probability. This gives us the probability for that observed sentence.

This completes discussion for all the parameters needed for Equation 2.4. We now discuss the task of getting counts for each state transition for each sentence.

Note that the procedure described above is general and computes the total probability of an observed sentence. If we wish to compute the joint probability, with respect to one derivation, $P(S_{obs}i, D_i)$, by creating word lattice for D_i and parsing it with our PCFG. We will now discuss how we can keep track of counts for the state transition, more specifically, we are interested in calculating $C_{D_i}(b \rightarrow a)$ and $C_{D_i}(b \rightarrow b)$

2.3.4 Keeping track of counts C_{D_i}

Calculating C_{D_i} for an observed sentence can be a huge book-keeping task (as we will need to do it for every derivation for a sentence), and we use the framework of V-expectation semirings [Eisner, 2002]. V-expectation semirings, can be used to keep track of expected counts of arc transitions. Since in our toy example, we want to count the number of expected state transitions for $b \rightarrow b$ and $b \rightarrow a$, we can create a vector of length = 2, where the first component of vector indicates expected number of traversals for $b \rightarrow a$ and the second component

indicates the expected number of traversals for $b \rightarrow b$. We will denote this tuple of probability and Vector as **Weight** Wt for a state transition:

$$Wt = (P, V)$$

where the first component P is a probability and the second V is a vector denoting the expected state transition counts. We initialize Weight by making the vector component value 1 for the state transition of interest and 0 for all other values, and then multiplying this with the probability of the state transition. We illustrate the process for our noise model. Table 2.5 and Figure 2.5 summarize the same.

- Our arcs of interest are $b \rightarrow a$ and $b \rightarrow b$. We assign $b \rightarrow a$ as index 0 in our vector and $b \rightarrow b$ as index 1.
- $b \rightarrow a$ which has $P(b \rightarrow a)$ as p_1
  Now has $Wt(b \rightarrow a) = (p1, [p1\ 0])$
- $b \rightarrow b$ which has $P(b \rightarrow b)$ as $(1 - p_1)$
  Now has $Wt(b \rightarrow b) = ((1 - p_1), [0\ 1 - p_1])$
- $a \rightarrow a$ which we do not wish to keep track of this, thus it has an empty vector component. $Wt(a \rightarrow a) = (1, [\ ])$

We have now introduced the framework necessary for book-keeping to keep track of the state transition counts. We now need to discuss how PCFG parsing works with V-expectation semirings. For this we will introduce a hypergraph that is a compact way of representing derivation trees for PCFG parsing.

2.3.5 Hypergraph and PCFG parsing

We use CKY parsing [Jurafsky and Martin, 2009], to parse a word lattice. CKY parsing keeps track of the parsing via a parsing chart, in which the root terminal node is noted down along with the word start index and end index. For example, with respect to our PCFG (Table 2.1), consider the rule $R_3 : A \rightarrow a$. Let us say the sentence to be parsed was "b a b". A CKY Parsing chart will have an item denoted as $\langle A, R_3, 1, 2 \rangle$. which means the $word_1 =$ "a" can be parsed by

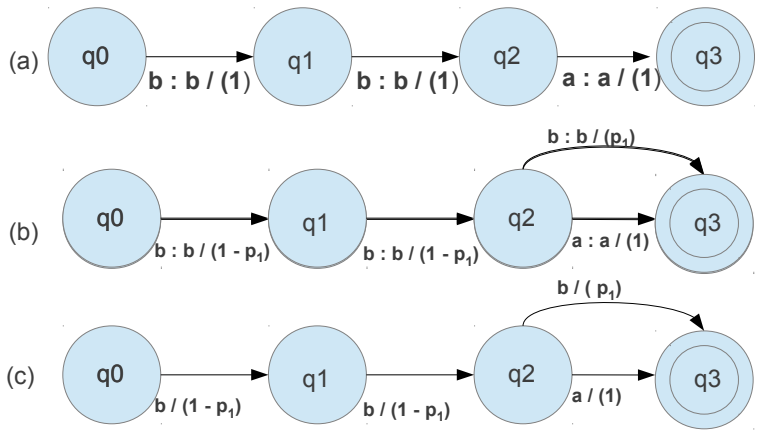

Figure 2.4: The figure shows the operations needed before parsing an observed sentence "b b a" with PCFG a) Shows the output wFST b) Shows the wFST when noise model is composed with observed wFST c) Shows the word lattice to be parsed

Table 2.4: The V-expectation semiring of [Eisner, 2002]. V is a vector space representing the features of arcs.

$$
\begin{array}{rcl}
\mathbb{K} & \in & \mathbb{R}_{\geq 0} \times V \\
(p_1, v_1) \oplus (p_2, v_2) & \stackrel{\text{def}}{=} & (p_1 + p_2, v_1 + v_2) \\
(p_1, v_1) \otimes (p_2, v_2) & \stackrel{\text{def}}{=} & (p_1 p_2, p_1 v_2 + v_1 p_2) \\
\text{if } p^* \text{ defined, } (p, v)^* & \stackrel{\text{def}}{=} & (p^*, p^* v p^*) \\
\bar{0} & \stackrel{\text{def}}{=} & (0, \mathbf{0}) \\
\bar{1} & \stackrel{\text{def}}{=} & (1, \mathbf{0})
\end{array}
$$

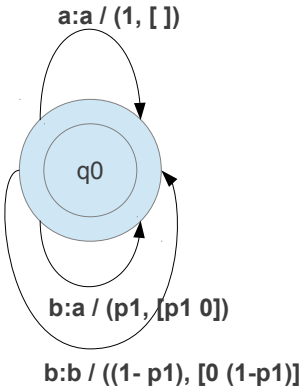

Figure 2.5: Noise model with V-expectation semiring

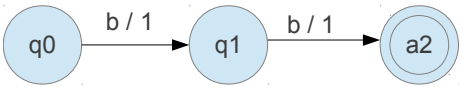

Figure 2.6: Word lattice for "b b"

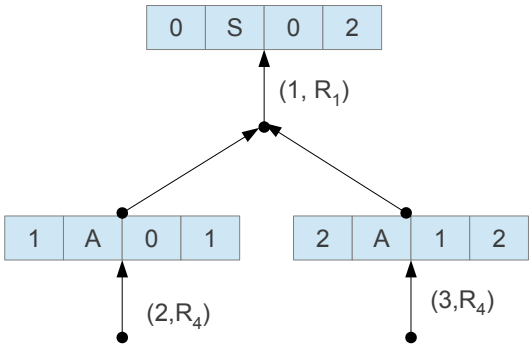

Figure 2.7: Sample hyper-graph for "b b"

using R_3, and the word has index 1 in the sentence. However, keeping a state for every parse is not very efficient, and hyper-graph allows us to represent the parsing state in a compact way.

Formally, a hypergraph [Li and Eisner, 2009] is a pair $\langle V, E \rangle$, where V is a set of nodes and E is a set of hyperedges, with each hyperedge connecting a set of antecedent nodes to a single consequent node. each node corresponds to an item in the parsing chart. A hyperedge, represents a PCFG rule that was used to construct that item in the parsing chart. We will denote the set of antecedent nodes on a hyperedge e by $T(e)$. We will denote set of incoming edges on a node v as $I(v)$.

Each node consists of a tuple: (Node-Number, RootSymbol, $word_{start}$, $word_{end}$). And each hyperedge is depicted by (Hyperdge-number, PCFG-Rule-number) Figure 2.7 shows a Hypergraph generated for the sentence "b b" (Figure 2.6)

When parsing with a word lattice that has a probability assigned to the word, we need to account it by multiplying the lattice word probability with the PCFG rule used. Its interesting to note that we need to do this only for rules that

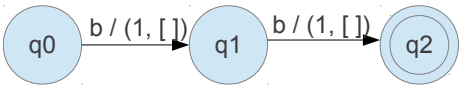

Figure 2.8: Word lattice for "b b" with semiring

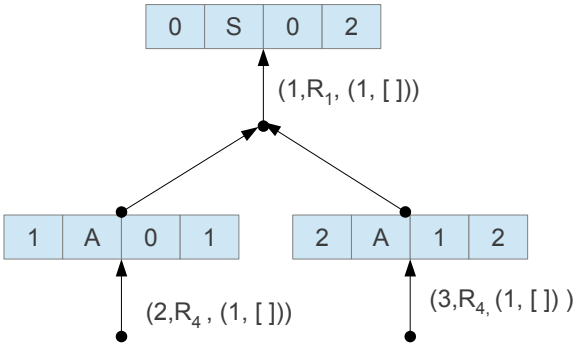

Figure 2.9: Sample hyper-graph for "b b" with semiring.

expand to a terminal symbol. For example, for our toy PCFG, we need to account for word lattice probability for $R_3 : A \rightarrow b$ and $R_4 : A \rightarrow a$ The probability of generating a hyper-edge e is given by Equation 2.9 where R_e represents the Rule used to construct e and L_e represents Lattice word used, if lattice word does not appear in rule expansion, then $P(L_e) = 1$

$$P(e) = P(R_e) * P(L_e) \tag{2.9}$$

And thus we can compute the probability of this hypergraph as follows:

$$P(\text{“}bb\text{''}) = P(e_1) * P(e_2) * P(e_3) \tag{2.10}$$

Our word lattice however, does not have a probability component. Infact, to keep track of state transition counts, we had added a V-expectation Semiring which we denote by Wt. Let us now explain how we use Weights:

- Add Wt to a hyperedge e. Thus hyperedge is represented as (Hyperdge-number, PCFG-Rule-number, Weight-lattice)
- Compute Weight of a hyperedge by using an Equation similar to Equation

2.9. Notice, we have used $\otimes$ as defined in Table 2.4:

$$Wt(e) = P(R_e) \otimes Wt(L_e) \tag{2.11}$$

- And finally an equation to compute Weight of hypergraph using Equation 2.10

$$Wt(\text{“}bb\text{”}) = Wt(e_1) \otimes Wt(e_2) \otimes Wt(e_3) \tag{2.12}$$

Figure 2.9 shows the semiring representation of the sample hypergraph for word lattice for “b b”(Figure 2.8)

2.3.6 Putting all the pieces together for training:

We mention earlier that we use unsupervised learning. We begin by intializing our parameters for noise model with random values, and then using Expectation-Maximization[Dempster et al., 1977] to update the value of our parameters, until we see convergence. In this section, we summarize the training algorithm:

- Initialize the unknown parameters of the noise model with random values.
- For each observed sentence, create a wFST as described in Section 2.3.3. Add Weights with probability of 1, and empty vector.
- Count the number of arcs we want to keep track of for our training. This will give us a length of the vector to be used. Compute the weight for each noise model state transition by the process described in 2.3.4.
- Compose Noise Model with the output wFST. To compute the weight of the resultant wFST use $\otimes$ and $\oplus$ operations.
- Convert the wFST to a word lattice.
- Parse word lattice using CKY Algorithm and the V-expectation modifications for hypergraphs as described in Section 2.3.5.

- This will give us the total weight of the hypergraph for the observed sentence. Let us denote it as $Wt_{hg} = (P_{hg}, V_{hg})$. P_{hg} denotes the total probability of the hypergraph, which also is $P(S^i_{obs})$. V_{hg} is a vector of expected counts. This would be the **Expectation Step**.

- Find the new parameter values using Equation 2.6 **Maximization Step**

- Update the noise model parameters and re-iterate the process, till we see convergence of parameters

2.3.7 Training demonstration for a toy corpus

We will now demonstrate training for our toy PCFG and noise model with the help of a toy training corpus (Table 2.6):

- We begin by converting $Sobs_1$ to a wFST with same output and input symbols, state transition probability of 1, and a state for each word in the sentence.

- Next, we compose noise model with observed wFST and use semiring $\otimes$ instead of $*$ and $\oplus$ instead of $+$.

- Next, we need to convert this wFST to a word lattice that can be parsed with our language model (PCFG). This is done by removing the output symbol. Figure 2.10 shows the computation to generate the word lattice.

- Next we use CKY algorithm and a hypergraph (Section 2.3.5. This gives us the hypergraph depicted in Figure 2.11 and Table 2.7

- We can then compute the Weight of the hypergraph using inside-outside algorithm as described in [Li and Eisner, 2009]:

$$Wt(S^1_{obs}) = (\frac{(1-p1)^2(2p1+1)}{32}, [\frac{(1-p1)^2 p1}{16} \ \frac{(1-p1)^2(2p1+1)}{16}])$$

- The above weight can be broken into two components: $P_(S^i_{obs})$ and expected counts vector. We had assigned $b \to a$ as index 0 and $b \to b$ as index 1. Let

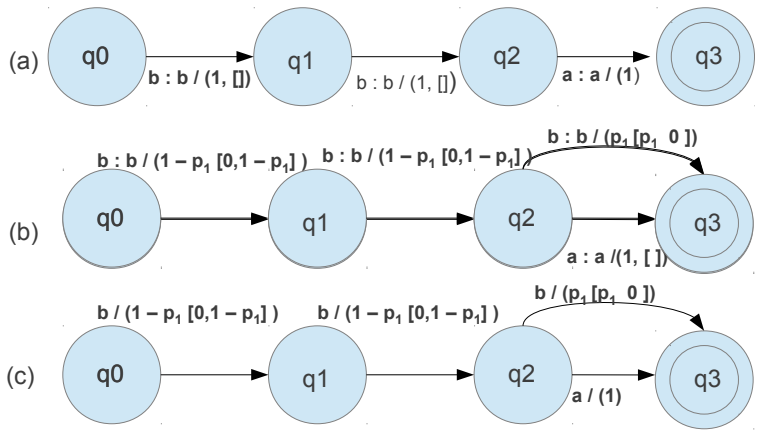

Figure 2.10: Figure depicting word lattice computation for "b b a" (a) depicts output wFST, (b) depicts resultant wFST when composed with a noise model and (c) depicts the word lattice

us list them down below for clarity:

$$P(S^i_{obs}) = \frac{(1-p_1)^2(2p_1+1)}{32}$$
$$E[b \to a] = \frac{(1-p_1)^2 p_1}{16}$$
$$E[b \to b] = \frac{(1-p_1)^2(2p_1+1)}{16}$$

- We need to divide the expected counts with $P_{S^1_{obs}}$ to use in Equation 2.6. This gives us:

$$E[b \to a | S^1_{obs}] = \frac{2p_1}{2p_1+1}$$
$$E[b \to b | S^1_{obs}] = 2$$

- Similarly, we can compute Wt for S^2_{obs}. Figure 2.12 and Table 2.9

$$Wt(S^2_{obs}) = (\frac{(1-p_1)(2p_1+1)^2}{64}, [\frac{(1+2p_1)(1-p_1)p_1}{16} \ \frac{(1-p_1)(2p_1+1)^2}{64})]$$

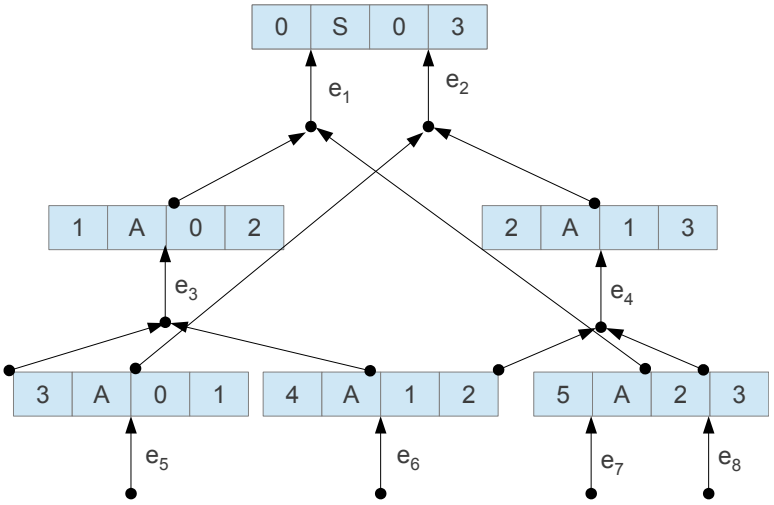

Figure 2.11: Hyper-graph for composition of Noise model and "b b a"

And thus we get:

$$E[b \to a|S^2_{obs}] = \frac{4p_1}{2p_1+1}$$
$$E[b \to b|S^2_{obs}] = 1$$

- Using Equation 2.6 we can get an update rule for p_1 as follows:

$$\begin{aligned} p_1 \leftarrow & \frac{E[b \to a|S^1_{obs}] + E[b \to a|S^2_{obs}]}{E[b \to a|S^1_{obs}] + E[b \to a|S^2_{obs}] + E[b \to a|S^1_{obs}] + E[b \to b|S^2_{obs}]} \\ &= \frac{\frac{2p_1}{2p_1+1} + \frac{4p_1}{2p_1+1}}{\frac{2p_1}{2p_1+1} + \frac{4p_1}{2p_1+1} + 2 + 1} \\ &= \frac{\frac{6p_1}{2p_1+1}}{\frac{6p_1}{2p_1+1} + 3} \\ &= \frac{2p_1}{4p_1+1} \end{aligned}$$

- Running E-M with the above update rule(Table 2.8) gives us a value of 0.25 after 16 iterations

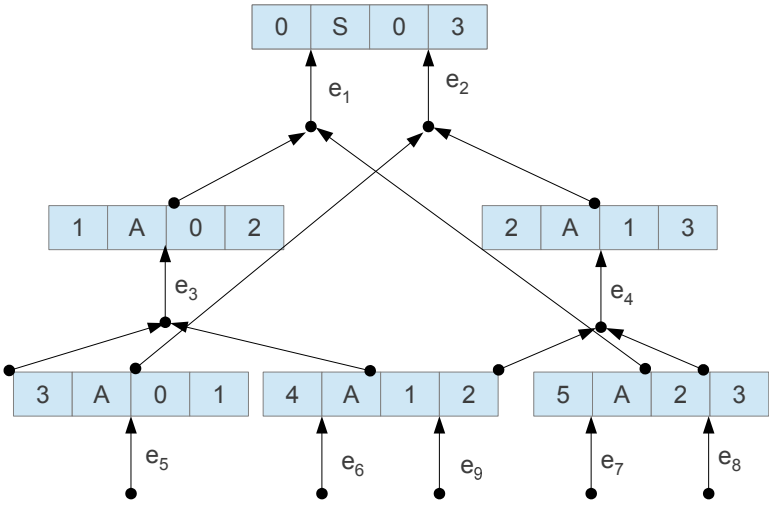

Figure 2.12: Hyper-graph for composition of Noise model and "b a a"

2.4 Decoding

The task of decoding is to generate the highest probability original sentence for an observed sentence. Let us say we observe a sentence S_{obs}, then we can have various S^i_{orig} for say $i = 1toN$. We are interested in finding the sentence which has highest $P(S^i_{orig}|S_{obs})$. Thus, once we learn the parameters of our noise model, we can compute $P(S^i_{orig}|S_{obs})$ as:

$$P(S^i_{orig}|S_{obs}) = \frac{P(S_{orig^i}, S_{obs})}{P(S_{obs})} \tag{2.13}$$

Note that we only care about the numerator $P(S_{orig}, S^i_{obs})$ as $P(S_{obs})$ is same for all S^i_{orig}. The computation is very similar to our training, we just dont need the Weight component for the word lattice and can work using the probability component. We re-iterate below for clarity:

- Convert the sentence to be decoded in a wFST with same input and output symbols and unit probabilities.
- Compose Noise Model (with learnt parameter values) with the above wFST
- Convert wFST to a word lattice

- Parse word lattice using the language PCFG. This will generate a hypergraph - whose **yield** generates all possible S^i_{orig}
- Return S^i_{orig} with the highest joint probability $P(S^i_{orig}, S_{obs})$

Table 2.5: Toy Noise Model for Figure 2.5

Arc	P	Wt
$q_0^a \rightarrow q_0^a$	1	(1, [ ])
$q_0^b \rightarrow q_0^a$	p1	(p_1, [p_1 0])
$q_0^b \rightarrow q_0^b$	(1 - p_1)	((1-p_1), [(0 (1-p_1)])

Table 2.6: A toy training corpus

$Sobs_1$	"b b a"
$Sobs_2$	"b a a"

Table 2.7: Table for Hyper-graph Figure 2.11

$Hyperedge$	PCGF Rule	$LatticeWeight$	Antecedent nodes
e_1	R_1	1 [ ]	1,5
e_2	R_1	1 [ ]	3,2
e_3	R_2	1 [ ]	3,4
e_4	R_2	1 [ ]	4,5
e_5	R_4	(1-p1) [0 (1-p1)]	$\emptyset$
e_6	R_4	(1-p1) [0 (1-p1)]	$\emptyset$
e_7	R_3	1 [ ]	$\emptyset$
e_8	R_4	p1 [p1 0]	$\emptyset$

Table 2.8: E-M table

Iteration	p_1
0	0.5
1	0.33333
2	0.28571
3	0.28571
4	0.26661
5	0.25806
6	0.25397
7	0.25197
8	0.25098
9	0.25049
10	0.25024
11	0.25012
12	0.25006
13	0.25003
14	0.25002
15	0.25001
16	0.25
17	0.25

Table 2.9: Table for Hyper-graph Figure 2.12

$Hyperedge$	PCFG Rule	$LatticeWeight$	Antecedent nodes
e_1	R_1	1 [ ]	1,5
e_2	R_1	1 [ ]	3,2
e_3	R_2	1 [ ]	3,4
e_4	R_2	1 [ ]	4,5
e_5	R_4	(1-p1) [0 (1-p1)]	$\emptyset$
e_6	R_4	(1-p1) [0 (1-p1)]	$\emptyset$
e_7	R_3	1 [ ]	$\emptyset$
e_8	R_4	p1 [p1 0]	$\emptyset$
e_9	R_4	p1 [p1 0]	$\emptyset$

Chapter 3

Design and Implementation

3.1 Language Model

We had stated earlier that we assume our language model to be fixed. In this section, we explain how we learn the rules and probabilities for our language. We use unlexicalized parsing [Klein and Manning, 2003] and use Sections 2-21 of WSJ sections of the Penn treebank [M. Marcus and Marcinkiewicz, 1993]. Penn treebank is a collection of 39,823 annotated trees. The annotations assign a structure to a sentence. We can extract rules for a language and thus build a Probabilistic Context Free Grammar (PCFG). For example, one tree from Penn tree bank looks as follows:

```
( (S
    (NP-SBJ (NNP Ms.) (NNP Haag) )
    (VP (VBZ plays)
      (NP (NNP Elianti) ))
    (. .) ))
```

The leaves of this tree generate a sentence of English. For example, above tree generates the sentence "Ms. Haag plans Elianti . "

The following describes the procedure to extract rules from an annotated Penn treebank tree:

- We first remove functional (NP-SBJ $\rightarrow$ NP) and empty tags from all trees.

We use tsurgeon [Levy and Andrew, 2006], an open source tool for tree manipulation. Thus, our original tree now becomes:

```
(
  (S
    (NP (NNP Ms.) (NNP Haag))
    (VP (VBZ plays)
      (NP (NNP Elianti)))
    (. .)))
```

- We now read the tree and extract the rules. This gives us rules as indicated in Table 3.1
- We further add a root symbol SROOT to each tree. Table 3.2 shows the updated set of rules.
- We now read the counts for each non-terminal (on LHS of the rule). We further normalize each rule by dividing it with the LHS count. Table 3.3 show the counts for each LHS rule and Table 3.4 shows the normalized counts. Thus we get a PCFG where each rule is specified by a probability.

We repeat the process for all 39,823 trees. We add the counts for all rules, and then normalize the rules by dividing it with count of LHS of the rule. We finally get a total of 924,021 rules of which 720,200 lead to terminal rule. An example of a terminal rule is: NNP $\rightarrow$ Elianti

3.2 CKY Parsing with semirings

We use CDEC [Dyer et al., 2010], an open source tool for parsing a word lattice with a PCFG. CDEC however does not understand V-expectation semirings. We need V-expectation semiring to accomplish training and learn parameters of our noise model. Similar to the changes described to work with semirings in hypergraph in Section 2.3.5, we extend CDEC so that a hypergraph stores the weight of the lattice and emits the total weight of the hypergraph for each training sentence. With these changes, we can train our noise model by passing a word lattice

Table 3.1: Rules generated for a sample tree

Rule	Counts(Rule)
S → NP VP .	1
VP → VBZ NP	1
NP → NNP	1
NP → NNP NNP	1
. → .	1
NNP → Elianti	1
NNP → Haag	1
NNP → Ms.	1
VBZ → plays	1

Table 3.2: Table 3.1 with updated rule for the root symbol

Rule	Counts(Rule)
SROOT → S	1
S → NP VP .	1
VP → VBZ NP	1
NP → NNP	1
NP → NNP NNP	1
. → .	1
NNP → Elianti	1
NNP → Haag	1
NNP → Ms.	1
VBZ → plays	1

Table 3.3: Counts of LHS for the tree

LHS	Counts(LHS)
SROOT	1
S	1
VP	1
NP	2
.	1
NNP	3
VBZ	1

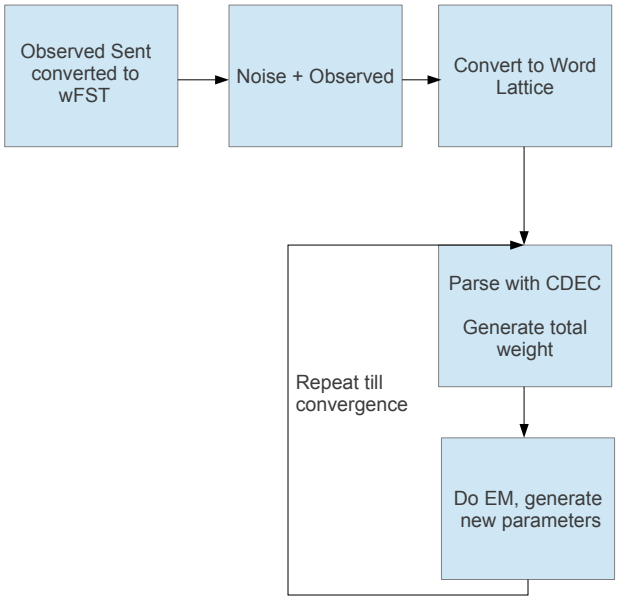

Figure 3.1: CDEC parsing for learning noise model parameters

Table 3.4: PCFG rules

Rule	Counts(Rule)
SROOT → S	1
S → NP VP .	1
VP → VBZ NP	1
NP → NNP	0.5
NP → NNP NNP	0.5
. → .	1
NNP → Elianti	0.33
NNP → Haag	0.33
NNP → Ms.	0.33
VBZ → plays	1

with semiring and doing Expectation-Maximization to learn updated parameters. Figure 3.1 shows how CDEC fits in the overall picture.

3.3 Corpus for learning noise model parameters

We use Korean ESL dataset provided by [Park and Levy, 2011] to train our noise models. This dataset is composed from 25,000 essays and comprised of 478,350 sentences written by Korean ESL students preparing for the Test of English as a Foreign Language (TOEFL) writing exam. These sentences were collected from open web postings by Koren ESL students asking for advice on their writing samples. For training and testing purposes the data set was split into a development set with 504 randomly selected sentences. The evaluation set consists of 1017 randomly selected sentences, and training set consists of the remaining sentences.

3.4 wFST operations

We need Weighted Finite State Transducer(wFST) to compose our noise models with an observed sentence, and then to parse the corresponding word lattice with the language model (PCFG). This is needed for both training and decoding purposes. We use OpenFST [Allauzen et al., 2007], an open source wFST tool. We use the parametric form of wFST , this allows us to do composition of noise model and observed sentence only once, and then update the word lattice with the updated parameters during EM training.

Chapter 4

Results and Analysis

In this chapter we discuss some issues with the grammar we learn for our language model. We then discuss our noise model, the parameters we learnt for it an end the chapter with an analysis of results.

4.1 Modifying Grammar

The grammar that we learn in Section 3.1 does not capture the difference between say a singular noun being combined with a plural verb and vice versa. Figure 4.1 shows the problem. The information about a NN being combined with VBZ is lost. We will call our grammar learnt with this problem as **Vanilla PCFG**. To address this issue, we move the information about NNS and NN up to its parent node. (Figure 4.2). We will call this grammar as **Fixed** PCFG. Fixing the PCFG is a work in progress, and we need to make it aware of more syntactic rules of grammar from our knowledge of english grammar to do a good job of grammar correction.

4.2 Noise Model

We use a wordform correction noise model. We try to model the mistakes people make while using the form of a verb or say using a singular form when a plural form was expected for a noun.

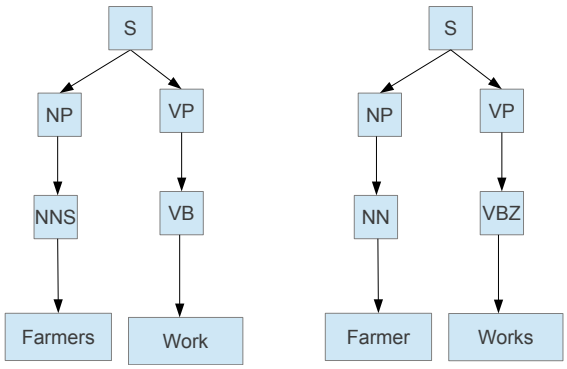

Figure 4.1: Some issues with the learnt grammar

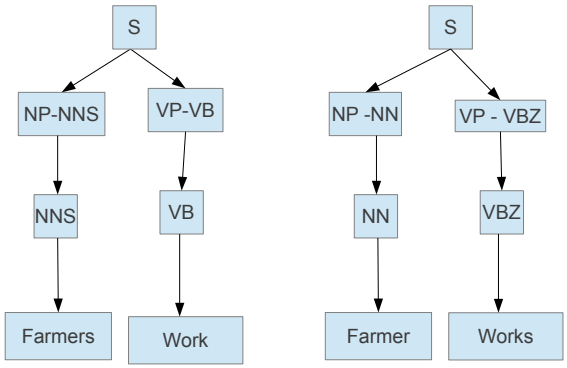

Figure 4.2: How the issues can be fixed by putting information in root node

4.2.1 Modeling the mistakes when using a noun

Nouns can be identified as being singular (E.g **cat**) or being plural (E.g **cats**). We try to model the mistakes people can make when say a plural form was intended to use. In other words, the noise model will try to estimate the probability of a singular changing to plural (cat → cats) or a plural changing to singular (cats → cat). Thus, our noise model will have two parameters: $p(singular \rightarrow plural)$, $p(plural \rightarrow singular)$

4.2.2 Modeling the mistakes when using a verb

Verbs can be classified in three main categories: regular verbs, irregular verbs and auxiliary verbs. A regular verb has four forms (Table 4.1), an irregular verb has five forms (Table 4.2) and an auxiliary verb has eight forms (Table 4.3). Notice that all the verb forms listed for an auxiliary verb, exist for a regular and irregular verb. Thus if we want to model an error as using the incorrect form of a verb (eg. walk → walking), we have 7 parameters for each form and total of 8 forms. This gives us 56 free parameters to learn.

Combining the noun and verb errors together gives us a noise model with 58 free parameters. We now describe how we construct our model.

4.2.3 wFST construction

For each word in an observed sentence, we identify it as a noun or a verb. We use CELEX [Baayen et al., 1995] to find all possible forms for a verb and a noun. We then identify the possible forms of the word. Let us say our observed word is cat. We ask CELEX what are its forms, CELEX tells us that the word cat is a noun, is singular, and has a form cats which is plural. We thus construct an arc cats → cat, and cat → cat and assign it probability $P_{PluralToSingular}$ and $P_{SingularToSingular}$ respectively. We repeat this process for each word in the observed sentence and act a unit probability transition for any word that is not a noun or a verb. We thus get a one state wFST which is both the start and the end state.

Table 4.1: Regular verb forms illustrated with verb walk

Form	Example
Base	walk
Past Simple	walked
Third Person Singular	walks
Present Participle	walking

Table 4.2: Irregular verb forms illustrated with verb fly

Form	Example
Base	fly
Past Simple	flew
Third Person Singular	flies
Present Participle	flying
Past Participle	flown

Table 4.3: Auxiliary verb forms illustrated with verb be

Form	Example
Base	am
Past Simple	was
Third Person Singular	is
Present Participle	being
Past Participle	been
Second Person	are
Infinitive	be
Second Person Past	were

4.2.4 Results

We use a training set of 1000 randomly chosen sentences from Korean ESL corpus. We see convergence of our model in 18 iterations. We run our model for both Vanilla PCFG and Fixed PCFG. Table 4.6 and Table 4.7 shows the parameters that we learn for our model when trained with Vanilla Grammar.Table 4.6 and Table 4.7 shows the parameters that we learn for our model when trained with Vanilla Grammar.

Table 4.4: Noun noise model parameters with Vanilla PCFG

	Singular	Plural
Singular	0.9028	0.0972
Plural	0.658	0.342

Table 4.5: Verb noise model parameters with Vanilla PCFG

	Base	Par	Th	Sec	Pa	PaPar	Inf.	Sec
Base	0.9792	0.0002	0.0204	0.0	0.0	0.0	0.0	0.0
Par	0.2610	0.6796	0.0	0.0	0.0593	0.0	0.0	0.0
Th	0.0970	0.0370	0.8658	0.0	0.0	0.0	0.0	0.0
Sec	0.0	0.0	0.5355	0.0	0.0	0.0	0.0980	0.3663
Pa	0.5456	0.0431	0.0259	0.0	0.3853	0.0	0.0	0.0
PaPar	0.0126	0.0002	0.5719	0.0	0.0	0.4152	0.0	0.0
Inf.	0.0	0.0	0.0007	0.0065	0.0	0.0001	0.1595	0.8331
Sec	0.0	0.0	0.0031	0.0924	0.0	0.0368	0.4471	0.4205

Table 4.6: Noun noise model parameters with Fixed PCFG

	Singular	Plural
Singular	0.9233	0.0767
Plural	0.6548	0.3451

4.3 Evaluation techniques

In this section, we explain our evaluation method. [Park and Levy, 2011] presented a novel approach to evaluate grammar correction model performance.

Table 4.7: Verb noise model parameters with Fixed PCFG

	Base	Par	Th	Sec	Pa	PaPar	Inf.	Sec
Base	0.9787	0.0045	0.0165	0.0	0.0001	0.0	0.0	0.0
Par	0.0235	0.8999	0.0	0.0	0.0765	0.0	0.0	0.0
Th	0.1096	0.0068	0.8809	0.0	0.0025	0.0	0.0	0.0
Sec	0.0	0.0	0.6112	0.0	0.0	0.0	0.0203	0.3684
Pa	0.5290	0.04079	0.0439	0.0	0.3861	0.0	0.0	0.0
PaPar	0.3262	0.0928	0.1013	0.0	0.0387	0.4409	0.0	0.0
Inf.	0.0	0.0	0.0	0.0129	0.0	0.049	0.8964	0.04156
Sec	0.0	0.0	0.0093	0.0783	0.0013	0.0	0.0	0.9108

If we denote the grammatically incorrect language as Language A, and the grammatically correct language as Language B. We can think of the task of grammar correction as translation from language A to language B. Thus, we can use standard machine translation evaluation techniques to evaluate our model. Machine translation evaluation relies on availability of a parallel corpus. Thus if we can obtain correct sentences for our sentences in development and testing set, we can evaluate our model using parallel corpus methods. We will now re-iterate the BLEU and METEOR metrics from [Park and Levy, 2011]

4.3.1 BLEU and METEOR

BLEU [Papineni et al., 2002] and METEOR [Lavie and Agrawal, 2007] are two state-of-the-art evaluation metrics currently being used in the field of machine translation. Both these metrics compare a machine translated output sentence to reference translations, and try to evaluate the similarity of the translation output to the reference. The reference translations are a set of high quality translations, often obtained from existing translations of the same document. Both BLEU and METEOR have a score between 0 and 1, where 1 is the best possible score and a higher score means better translation.

We use the reference sentences provided by [Park and Levy, 2011]. These reference translations were obtained using Amazon Mechanical Turk and later each correction was validated.

4.3.2 Results on development set

Out of 502 total sentences in the development set, 139 sentences failed to parse. This was mainly due to some word forms not being seen while training. We are yet to fix this issue and thus this failure was expected. Table 4.8 summarizes the results with BLEU and METEOR scores included.

Table 4.8: Results on development set with wordform noise model and using parameters learnt after using 1k training sentences

Total Sentences	502
Sentences that failed to parse	139
Average BLEU score for incorrect sentences:	0.705898
Average BLEU score for Vanilla Grammar:	0.6030409
Average BLEU score for Fixed Grammar:	0.630623
Average METEOR score for incorrect sentences:	0.812503
Average METEOR score for Vanilla grammar:	0.753504
Average METEOR score for Fixed Grammar:	0.768581

4.3.3 Analysis

We see that our model actually does a worse job and introduces more errors on the incorrect sentences. However, we see that our fixed grammar does a better job than Vanilla grammar. We think that due to strong independence assumptions in our grammar, some of the syntactic information is lost, and thus by fixing our grammar we can hope to do a better job at automatic grammar correction. This is part of our future work. Let us also look at some of the corrections our present model made that work well:

- Corrected: Such an effort will make our country help to overcome economic **difficulties** and create many jobs for the people.
- Incorrect: Such an effort will make our country help to overcome economic **difficulty** and create many jobs for the people

  Note that it correctly fixes the word **difficulty** to its right form.

  Now, let us also look at a sample correction where it performs bad:

- Correct: Teachers can only gain these **ability** through their careers , and some may even had to **went** through difficult time .
- Incorrect: Teachers can only gain these **abilities** through their careers , and some might even have to **go** through difficult time .

We see here that it is unable to associate these and abilities together. We can fix this by adding propagating lexical information up the tree about these. The problem of using go instead of went is however hard to fix.

Chapter 5

Conclusion and Future Work

In this thesis, we have presented a novel approach for automatic grammar correction. We have shown how we can use noisy channel model to correct grammatical errors. Our major contribution is to present how this model works when we use a Probabilistic Context Free Grammar (PCFG) as our language model along with wFST as noise model. We show how we can extend the framework of V-expectation Semirings to CKY parsing to learn the parameters of our noise model. In this thesis, we have worked with only one noise model - wordform correction.

Another important conclusion is that we can fix the grammar to remove the strong independence assumptions. We believe using syntactic knowledge of english grammar will help us fixing our grammar. As part of future work, We also plan to use more noise models as mentioned in [Park and Levy, 2011] - spelling errors, article choice errors, prepositional choice errors, word insertion models, word deletion models and combination of these models.

Bibliography

[Allauzen et al., 2007] Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., and Mohri, M. (2007). Openfst: a general and efficient weighted finite-state transducer library. In *Proceedings of the 12th international conference on Implementation and application of automata*, CIAA'07, pages 11–23, Berlin, Heidelberg. Springer-Verlag.

[Baayen et al., 1995] Baayen, H. R., Piepenbrock, R., and Gulikers, L. (1995). *The CELEX Lexical Database. Release 2 (CD-ROM).* Linguistic Data Consortium, University of Pennsylvania, Philadelphia, Pennsylvania.

[Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society B*, 39:1–38.

[Dyer et al., 2010] Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Ture, F., Blunsom, P., Setiawan, H., Eidelman, V., and Resnik, P. (2010). cdec: A decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of ACL*.

[Eisner, 2002] Eisner, J. (2002). Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Jurafsky and Martin, 2009] Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing, Second Edition.* Pearson.

[Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the Association for the Computational Linguistics.*

[Lavie and Agrawal, 2007] Lavie, A. and Agrawal, A. (2007). Meteor: an automatic metric for mt evaluation with high levels of correlation with human judgments. In *StatMT '07: Proceedings of the Second Workshop on Statistical Machine Translation*, page 228231. Association for Computational Linguistics.

[Levy and Andrew, 2006] Levy, R. and Andrew, G. (2006). Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *5th International Conference on Language Resources and Evaluation*.

[Li and Eisner, 2009] Li, Z. and Eisner, J. (2009). First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 40–51, Singapore.

[M. Marcus and Marcinkiewicz, 1993] M. Marcus, B. S. and Marcinkiewicz, M. (1993). Building a large annotated corpus of english: the penn treebank. In *Computational Linguistics*.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318.

[Park and Levy, 2011] Park, Y. A. and Levy, R. (2011). Automated whole sentence grammar correction using a noisy channel model. In *Proceedings of the 49th annual meeting on Association for Computational Linguistics*, ACL '11. Association for Computational Linguistics.