6.864, Fall 2007: Problem Set 1

Total points: 120 points

Due date: 5pm, 26th September 2007

Submit to Igor Malioutov either by email to `igorm@csail.mit.edu`, or by hand to Stata G369

Late policy: 5 points off for every day late, 0 points if handed in after 1pm on October 1st 2007

Question 1 (20 points)

A probabilistic context-free grammar $G = (N, \Sigma, R, S, P)$ in Chomsky Normal Form is defined as follows:

- N is a set of non-terminal symbols (e.g., `NP, VP, S` etc.)
- Σ is a set of terminal symbols (e.g., *cat, dog, the*, etc.)
- R is a set of rules which take one of two forms:
  - $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
  - $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
- $S \in N$ is a distinguished start symbol
- P is a function that maps every rule in R to a probability, which satisfies the following conditions:
  - $\forall r \in R,\ P(r) \geq 0$
  - $\forall X \in N,\ \sum_{X \rightarrow \alpha \in R} P(X \rightarrow \alpha) = 1$

Now assume we have a probabilistic CFG G', which has a set of rules R which take one of the two following forms:

- $X \rightarrow Y_1 Y_2 \ldots Y_n$ for $X \in N$, $n \geq 2$, and $\forall i,\ Y_i \in N$
- $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$

Note that this is a more permissive definition than Chomsky normal form, as some rules in the grammar may have more than 2 non-terminals on the right-hand side. An example of a grammar that satisfies this more permissive definition is as follows:

S	→	NP	VP		0.7
S	→	NP	NP	VP	0.3
VP	→	Vt	NP		0.8
VP	→	Vt	NP	PP	0.2
NP	→	DT	NN	NN	0.3
NP	→	NP	PP		0.7
PP	→	P	NP		1.0

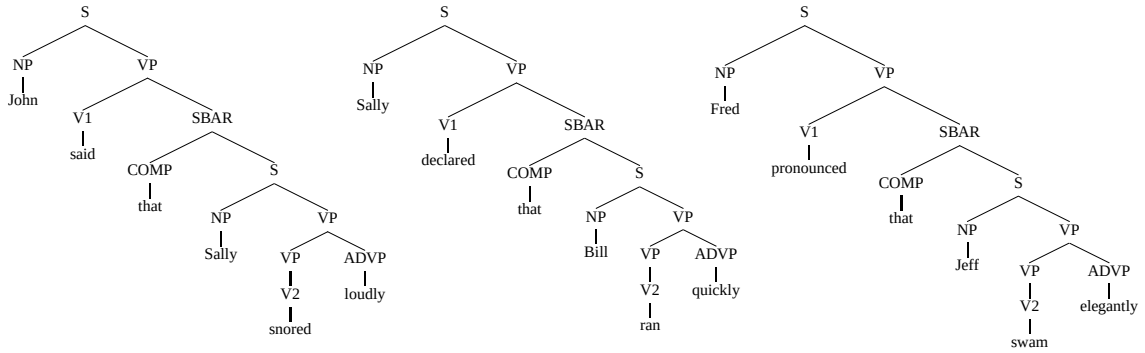
Vt	→	saw	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
IN	→	with	0.5
IN	→	in	0.5

Question 1(a): Describe how to transform a PCFG G', in this more permissive form, into an "equivalent" PCFG G in Chomsky normal form. By equivalent, we mean that there is a one-to-one function f between derivations in G' and derivations in G, such that for any derivation T' under G' which has probability p, $f(T')$ also has probability p. (Note: one major motivation for this transformation is that we can then apply the dynamic programming parsing algorithm, described in lecture, to the transformed grammar.) Hint: think about adding new rules with new non-terminals to the grammar.

Question 1(b): Show the resulting grammar G after applying your transformation to the example PCFG shown above.

Question 2 (20 points)

Nathan L. Pedant decides to build a treebank. He finally produces a corpus which contains the following three parse trees:



Clarissa Lexica then purchases the treebank, and decides to build a PCFG, and a parser, using Nathan's data.

Question 2(a): Show the PCFG that Clarissa would derive from this treebank.

Question 2(b): Show two parse trees for the string "Jeff pronounced that Fred snored loudly", and calculate their probabilities under the PCFG.

Question 2(c): Clarissa is shocked and dismayed, (see 2(b)), that "Jeff pronounced that Fred snored loudly" has two possible parses, and that one of them—that Jeff is doing the pronouncing loudly—has relatively high probability, in spite of it having the ADVP *loudly* modifiying the "higher" verb, *pronounced*. This type of high attachment is never seen in the corpus, so the PCFG is clearly missing something. Clarissa decides to fix the treebank, by altering some non-terminal labels in the corpus. Show one such transformation that results in a PCFG that gives zero probability to parse trees with "high" attachments. (Your solution should systematically refine some non-terminals in the treebank, in a way that slightly increases the number of non-terminals in the grammar, but allows the grammar to capture the distinction between high and low attachment to VPs.)

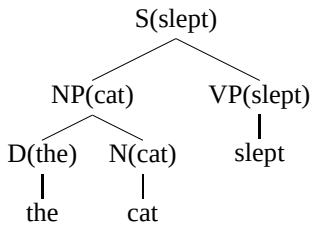
Question 3 (20 points)

We will refer to a "lexicalized PCFG" in Chomsky normal form, as a PCFG $G = (N, \Sigma, R, S, P)$ similar to that in question 1, where each of the rules in R takes one of the following three forms:

- $X(h) \rightarrow Y_1(h)\ Y_2(w)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$.
  e.g., `NP(man)` $\rightarrow$ `NP(man) PP(with)`.
- $X(h) \rightarrow Y_1(w)\ Y_2(h)$ for $X \in N$, and $Y_1, Y_2 \in N$, and $h, w \in \Sigma$.
  e.g., `S(snores)` $\rightarrow$ `NP(man) VP(snores)`.
- $X(h) \rightarrow h$ for $X \in N$, and $h \in \Sigma$
  e.g., `NP(man)` $\rightarrow$ `man`.

Here the symbols in the grammar rules are of the form $X(h)$, where X is a symbol such as `NP`, `VP`, etc., and h is a lexical item such as *man*, *snores*, etc.

In addition, for any symbol of the form $X(h)$, there is a probability $P_S(X(h))$ which is the probability of $X(h)$ being chosen as the root of a parse tree. As one example, the tree



would have probability

$P_S(\texttt{S(slept)})\times$
$P(\texttt{S(slept)} \rightarrow \texttt{NP(cat) VP(slept)}|\texttt{S(slept)})\times$
$P(\texttt{NP(cat)} \rightarrow \texttt{D(the) N(cat)}|\texttt{NP(cat)})\times$
$P(\texttt{D(the)} \rightarrow \texttt{the}|\texttt{D(the)})\times$
$P(\texttt{N(cat)} \rightarrow \texttt{cat}|\texttt{N(cat)})\times$
$P(\texttt{VP(slept)} \rightarrow \texttt{slept}|\texttt{VP(slept)})$

Question 3(a): Describe a dynamic programming algorithm, similar to the one in lecture, which finds the highest scoring parse tree under a grammar of this form. Your algorithm should make use of a dynamic programming table $\pi[i, j, k, X]$ where

$$\pi[i, j, k, X] = \text{highest probability for any parse tree whose root is the symbol } X(w_k), \text{ and which spans words } i \ldots j \text{ inclusive}$$

For example, if the sentence being parsed is $w_1, w_2, \ldots w_6$ = *the cat sat on the mat*, then $\pi[4, 6, 4, \texttt{PP}]$ would store the maximum probability for any parse tree whose root is $PP(\text{on})$, and which spans the string *on the mat*.

Note: your algorithm should also allow recovery of the parse tree which achieves the maximum probability.

Question 3(b): What is the running time of your algorithm?

Question 4 (20 points)

Recall the definition of a PCFG in Chomsky normal form from question 1. Now assume we have a probabilistic CFG G', which has a set of rules R which take one of the three following forms:

1. $X \rightarrow Y_1 Y_2$ for $X \in N$, and $Y_1, Y_2 \in N$
2. $X \rightarrow Y$ for $X \in N$, and $Y \in \Sigma$
3. $X \rightarrow Y$ for $X \in N$, and $Y \in N$

Note that this is very similar to a Chomsky normal form grammar, but that we are now allowed rules of form (3), such as `S -> VP`, where there is a single symbol on the right-hand-side of the rule, and this symbol is a non-terminal. We will refer to these new rules as **unary productions**. (Note that productions of the form in (2), such as `N -> dog`, will *not* be referred to as unary productions, as their right-hand-side is a terminal symbol.) We will refer to rules captured by cases (1) and (2) as **non-unary productions**.

As one example, the following grammar contains unary productions:

S	→	NP	VP	0.7
S	→	SBAR	VP	0.3
VP	→	Vi		0.4
VP	→	Vt	NP	0.4
VP	→	V3	SBAR	0.2
NP	→	NN		0.3
NP	→	DT	NN	0.7
SBAR	→	COMP	S	0.6
SBAR	→	S		0.4

Vi	→	sleeps	1.0
Vt	→	saw	1.0
V3	→	said	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
COMP	→	that	1.0

In this question, we'll attempt to convert a PCFG with unary productions into an "equivalent" PCFG which is in Chomsky normal form (note that we'll have to be careful with what we mean by "equivalent", we'll come to this shortly).

As a first step, we will use the classic transformation for (non-probabilistic) context-free grammars, that results in a new grammar that accepts the same set of strings as the original grammar, but which has all unary productions removed. Applying this transformation to the grammar above results in the CFG shown in figure 1 (note that the probabilities are missing – we'll fill them in soon).

This grammar transformation works in the following way. We form a new grammar G' from an existing grammar G by first taking all non-unary rules from G. Then, if there is any sequence of n productions in G

$$B_0 \rightarrow B_1 \rightarrow B_2 \ldots B_{n-1} \rightarrow \alpha$$

such that $B_i \rightarrow B_{i+1}$ for $i = 0 \ldots n-2$ are unary productions in the grammar, and $B_{n-1} \rightarrow \alpha$ is a non-unary production, then we add the rule

$$B_0 \rightarrow \alpha$$

to G'. For example, in the above example we have the sequence

$$\text{SBAR} \rightarrow \text{S} \rightarrow \text{NP VP}$$

S	→	NP	VP
S	→	SBAR	VP
VP	→	sleeps	
VP	→	Vt	NP
VP	→	V3	SBAR
NP	→	man	
NP	→	woman	
NP	→	telescope	
NP	→	DT	NN
SBAR	→	COMP	S
SBAR	→	NP	VP
SBAR	→	SBAR	VP

Vi	→	sleeps
Vt	→	saw
V3	→	said
NN	→	man
NN	→	woman
NN	→	telescope
DT	→	the
COMP	→	that

Figure 1: A transformed grammar, G'

so we add the rule SBAR → NP VP to G'. As another example, we have the sequence

$$\text{VP} \rightarrow \text{Vi} \rightarrow \text{sleeps}$$

so we add the rule VP → sleeps to G'.

We now come to the definition of equivalence. We will say that the PCFG G' is "equivalent" to a PCFG G if:

- For any string w, if $T(w)$ is the highest probability parse tree for w under the grammar G, and $T'(w)$ is the highest prob. parse under G', then these two parse trees have the same probability under their respective grammars.
- There is a function f such that $T(w) = f(T'(w))$. i.e., there is a function such that the highest probability parse tree in the original grammar can be recovered from the highest probability parse tree under G'.
- In some cases, we will allow the PCFG G' to be **deficient**. This means that we will relax the requirement on probabilities on rules to satisfy the condition $\forall X \in N,\ \sum_{X \rightarrow \alpha \in R} P(X \rightarrow \alpha) < 1$ rather than $\forall X \in N,\ \sum_{X \rightarrow \alpha \in R} P(X \rightarrow \alpha) = 1$, as in the definition in question 1.

Question 4(a): (10 points) Add probabilities to the grammar in figure 1, so that the new PCFG G' is equivalent to the old PCFG G. Describe the function f that maps a parse tree in G' to a parse tree in G.

Question 4(b): (10 points) Describe a strategy for creating an equivalent PCFG G' in Chomsky normal form for *any* PCFG G in the form described at the start of this question (i.e., a PCFG that may have unary productions in addition to Chomsky normal form rules). You may assume that for any unary rule, its probability is strictly less than 1. Describe also the function f used to recover the highest probability tree under G from the highest probability tree under G'. *Note that your resulting PCFG G' may be deficient in some cases.* Illustrate your transformation on the two PCFGs shown in figure 2 (these grammars will help you, in terms of illustrating some "tricky" cases that you'll run into with unary productions). **Hint: remember throughout this question that the goal of G' is to allow recovery of the maximum probability parse under G.**

Grammar 1:

S	→	NP	VP	0.7
S	→	SBAR	VP	0.3
VP	→	Vi		0.4
VP	→	Vt	NP	0.4
VP	→	V3	SBAR	0.2
NP	→	NN		0.3
NP	→	DT	NN	0.7
SBAR	→	COMP	S	0.4
SBAR	→	S		0.5
SBAR	→	NP	VP	0.1

Vi	→	sleeps	1.0
Vt	→	saw	1.0
V3	→	said	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
COMP	→	that	1.0

Grammar 2:

S	→	NP	VP	0.7
S	→	SBAR	VP	0.2
S	→	SBAR		0.1
VP	→	Vi		0.4
VP	→	Vt	NP	0.4
VP	→	V3	SBAR	0.2
NP	→	NN		0.3
NP	→	DT	NN	0.7
SBAR	→	COMP	S	0.4
SBAR	→	S		0.5
SBAR	→	X		0.1
X	→	S		0.1
X	→	NP	NP	0.9

Vi	→	sleeps	1.0
Vt	→	saw	1.0
V3	→	said	1.0
NN	→	man	0.7
NN	→	woman	0.2
NN	→	telescope	0.1
DT	→	the	1.0
COMP	→	that	1.0

Figure 2: Two grammars with unary productions

Question 5 (15 points)

Say we have a vocabulary $\mathcal{V}$, i.e., a set of possible words. We'd like to estimate a unigram distribution $P(w)$ over $w \in \mathcal{V}$. We observe n sample points, $w_1, w_2, \ldots w_n$ (note that this sample may not include all members of $\mathcal{V}$, particularly if n is small compared to $|\mathcal{V}|$.) For any word seen r times in the training sample, the Good-Turing estimate of its count is

$$GT(r) = (r+1) * \frac{N_{r+1}}{N_r},$$

where N_r is the number of members of $\mathcal{V}$ which are seen r times in the corpus. For any w which is observed in the training corpus, we make the estimate $P(w) = GT(C(w))/n$, where $C(w)$ is the number of times w is seen in the sample.

1. Can you see any problem with this estimation method for words with large values for $C(w)$?
2. Prove that under this definition $\sum_{w \in \mathcal{V}'} P(w) \leq 1$, where $\mathcal{V}'$ is the subset of $\mathcal{V}$ seen in the training corpus. If the "missing" probability mass $1 - \sum_{w \in \mathcal{V}'} P(w)$ is divided evenly amongst the words not

seen in the corpus, show that $P(w)$ for any word not in the corpus is $N_1/(n \times N_0)$ where N_0 is $|\mathcal{V}| - |\mathcal{V}'|$, and N_1 as before is the number of members of $\mathcal{V}$ seen exactly once in the corpus.

Question 6 (20 points)

In lecture we saw a method for language modeling called *linear interpolation*, where the trigram estimate $\hat{P}(w_i \mid w_{i-2}, w_{i-1})$ is defined as

$$\hat{P}(w_i \mid w_{i-2}, w_{i-1}) = \lambda_1 \times P_{ML}(w_i \mid w_{i-2}, w_{i-1}) + \lambda_2 \times P_{ML}(w_i \mid w_{i-1}) + \lambda_3 \times P_{ML}(w_i)$$

Here $\lambda_1, \lambda_2, \lambda_3$ are weights for the trigram, bigram, and unigram estimates, and P_{ML} stands for the maximum-likelihood estimate.

One way to optimize the λ values (again, as seen in lecture), is to use a set of validation data, in the following way. Say the validation data consists of n sentences, $S_1, S_2, \ldots, S_n$. Define $\text{Count}_2(w_1, w_2, w_3)$ to be the number of times the trigram w_1, w_2, w_3 is seen in the validation sentences. Then the λ values are chosen to maximize the following function:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3 \in \mathcal{V}} \text{Count}_2(w_1, w_2, w_3) \log \hat{P}(w_3, |w_1, w_2)$$

Question: show that choosing λ values that maximize $L(\lambda_1, \lambda_2, \lambda_3)$ is equivalent to choosing λ values that *minimize the perplexity* of the language model on the validation data.

Question 7 (5 points)

In the lecture we saw an improved method for linear interpolation where the λ values are sensitive to the number of times the bigram (w_{i-2}, w_{i-1}) has been seen; the intuition behind this was that the more frequently this bigram has been seen, the more weight should be put on the trigram estimate.

Here we'll define a method that is similar in form to the method seen in lecture, but differs in some important ways. First, we define a function $\Phi(w_{i-2}, w_{i-1}, w_i)$ which maps trigrams into "bins", depending on their count. For example, we can define Φ as follows:

$$\begin{array}{lll}
\Phi(w_{i-2}, w_{i-1}, w_i) = 1 & \text{If} & Count(w_{i-2}, w_{i-1}, w_i) = 0 \\
\Phi(w_{i-2}, w_{i-1}, w_i) = 2 & \text{If} & 1 \leq Count(w_{i-2}, w_{i-1}, w_i) \leq 2 \\
\Phi(w_{i-2}, w_{i-1}, w_i) = 3 & \text{If} & 3 \leq Count(w_{i-2}, w_{i-1}, w_i) \leq 5 \\
\Phi(w_{i-2}, w_{i-1}, w_i) = 4 & \text{If} & 6 \leq Count(w_{i-2}, w_{i-1}, w_i)
\end{array}$$

The trigram estimate $\hat{P}(w_i \mid w_{i-2}, w_{i-1})$ is then defined as

$$\begin{aligned}
\hat{P}(w_i \mid w_{i-2}, w_{i-1}) \quad = \quad & \lambda_1^{\Phi(w_{i-2}, w_{i-1}, w_i)} \times P_{ML}(w_i \mid w_{i-2}, w_{i-1}) \\
& + \lambda_2^{\Phi(w_{i-2}, w_{i-1}, w_i)} \times P_{ML}(w_i \mid w_{i-1}) \\
& + \lambda_3^{\Phi(w_{i-2}, w_{i-1}, w_i)} \times P_{ML}(w_i)
\end{aligned}$$

Notice that we now have 12 smoothing parameters, i.e., λ_j^i for $i = 1 \ldots 4$ and $j = 1 \ldots 3$.

Question: Unfortunately this estimation method has a serious problem: what is it?