Statistical NLP - Assignment 5
Problem Set Questions

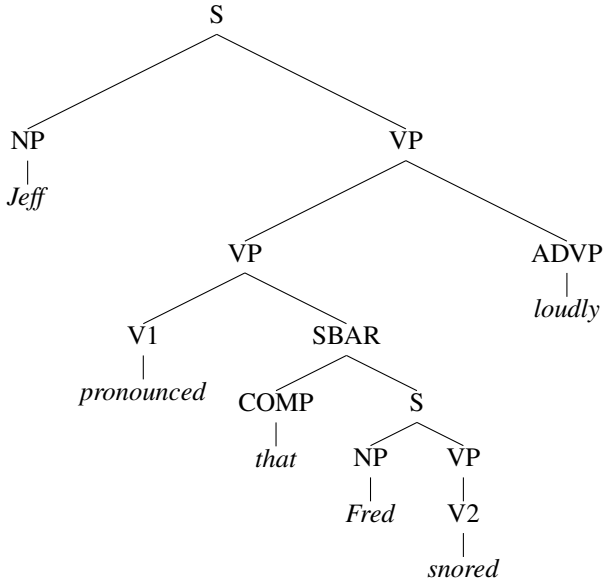
Daniel Rivera Ruiz
Department of Computer Science
New York University
New York, NY 10003
drr342@nyu.edu

1 Parsing

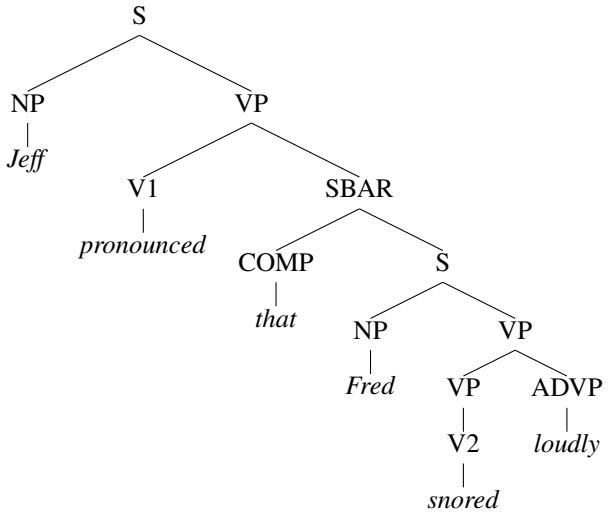
1. Probabilistic Context-Free Grammar:

Rule	Probability
S → NP VP	1.0
NP → John	0.1667
NP → Sally	0.3333
NP → Bill	0.1667
NP → Fred	0.1667
NP → Jeff	0.1667
VP → V1 SBAR	0.3333
VP → VP ADVP	0.3333
VP → V2	0.3333
V1 → said	0.3333
V1 → declared	0.3333
V1 → pronounced	0.3333
SBAR → COMP S	1.0
COMP → that	1.0
V2 → snored	0.3333
V2 → ran	0.3333
V2 → swam	0.3333
ADVP → loudly	0.3333
ADVP → quickly	0.3333
ADVP → elegantly	0.3333

2. Parse trees for *"Jeff pronounced that Fred snored loudly"* and their probabilities:



$$
\begin{aligned}
P =& (\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Jeff})(\text{VP} \rightarrow \text{VP ADVP})(\text{VP} \rightarrow \text{V1 SBAR}) \\
& (\text{ADVP} \rightarrow \textit{loudly})(\text{V1} \rightarrow \textit{pronounced})(\text{SBAR} \rightarrow \text{COMP S})(\text{COMP} \rightarrow \textit{that}) \\
& (\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Fred})(\text{VP} \rightarrow \text{V2})(\text{V2} \rightarrow \textit{snored}) \\
=& (1.0)(\tfrac{1}{6})(\tfrac{1}{3})(\tfrac{1}{3})(\tfrac{1}{3})(\tfrac{1}{3})(1.0)(1.0)(1.0)(\tfrac{1}{6})(\tfrac{1}{3})(\tfrac{1}{3}) \\
\boldsymbol{P} =& \mathbf{3.8104 \times 10^{-5}}
\end{aligned}
$$

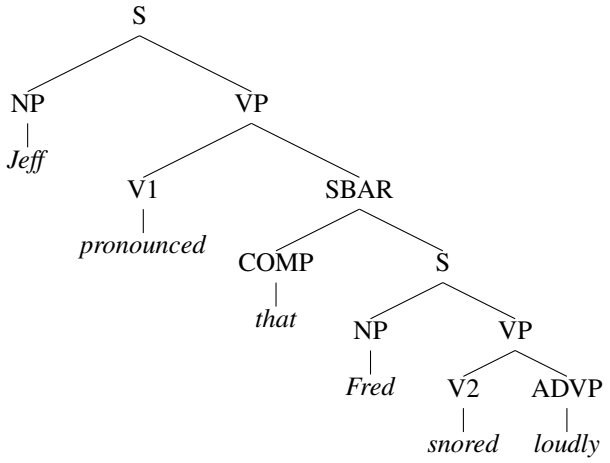


$$
\begin{aligned}
P =& (\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Jeff})(\text{VP} \rightarrow \text{V1 SBAR})(\text{V1} \rightarrow \textit{pronounced}) \\
& (\text{SBAR} \rightarrow \text{COMP S})(\text{COMP} \rightarrow \textit{that})(\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Fred}) \\
& (\text{VP} \rightarrow \text{VP ADVP})(\text{VP} \rightarrow \text{V2})(\text{V2} \rightarrow \textit{snored})(\text{ADVP} \rightarrow \textit{loudly}) \\
=& (1.0)(\tfrac{1}{6})(\tfrac{1}{3})(\tfrac{1}{3})(1.0)(1.0)(1.0)(\tfrac{1}{6})(\tfrac{1}{3})(\tfrac{1}{3})(\tfrac{1}{3})(\tfrac{1}{3}) \\
\boldsymbol{P} =& \mathbf{3.8104 \times 10^{-5}}
\end{aligned}
$$

3. Modifications to the PCFG.
The problem with the current PCFG is the rule VP → VP ADVP, which allows the ADVP to be attached to *any* VP. If said VP takes the form V2, a *low* attachment (consistent with the original corpus) will result. However, if it takes the form V1 SBAR, the undesired *high* attachment will be produced. To solve this problem, it suffices to eliminate the recursive rule and force the right-hand side VP to take the form of a V2:

Rule	Probability
S → NP VP	1.0
NP → John	0.1667
NP → Sally	0.3333
NP → Bill	0.1667
NP → Fred	0.1667
NP → Jeff	0.1667
VP → V1 SBAR	**0.5**
VP → V2 ADVP	**0.5**
V1 → said	0.3333
V1 → declared	0.3333
V1 → pronounced	0.3333
SBAR → COMP S	1.0
COMP → that	1.0
V2 → snored	0.3333
V2 → ran	0.3333
V2 → swam	0.3333
ADVP → loudly	0.3333
ADVP → quickly	0.3333
ADVP → elegantly	0.3333

Under this new grammar, the only possible parse tree for *Jeff pronounced that Fred snored loudly* is as follows:



With a probability equals to

$$\begin{aligned} P =& (\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Jeff})(\text{VP} \rightarrow \text{V1 SBAR})(\text{V1} \rightarrow \textit{pronounced}) \\ & (\text{SBAR} \rightarrow \text{COMP S})(\text{COMP} \rightarrow \textit{that})(\text{S} \rightarrow \text{NP VP})(\text{NP} \rightarrow \textit{Fred}) \\ & (\text{VP} \rightarrow \text{V2 ADVP})(\text{V2} \rightarrow \textit{snored})(\text{ADVP} \rightarrow \textit{loudly}) \\ =& (1.0)(\tfrac{1}{6})(\tfrac{1}{2})(\tfrac{1}{3})(1.0)(1.0)(1.0)(\tfrac{1}{6})(\tfrac{1}{2})(\tfrac{1}{3})(\tfrac{1}{3}) \\ \boldsymbol{P} =& \mathbf{2.5720 \times 10^{-4}} \end{aligned}$$

2 Language Modeling

1. The Kneser-Ney n-gram model is given by the following equation:

$$P_{KN}\left(w_i|w_{i-n+1}^{i-1}\right) = \frac{max\left(N_{1+}(\bullet, w_{i-n+1}, \ldots, w_i) - D, 0\right)}{N_{1+}(\bullet, w_{i-n+1}, \ldots, \bullet)} + \lambda P_{KN}\left(w_i|w_{i-n+2}^{i-1}\right)$$

Where $N_{1+}(\bullet, w_{i-n+1}, \ldots, w_i) = |(w', w_{i-n+1}, \ldots, w_i) : C(w', w_{i-n+1}, \ldots, w_i) > 0|$ is the *continuation count* i.e., the number of unique words that precede the n-gram $w_{i-n+1}, \ldots, w_i$ in the training corpus. In addition to the model described in the previous equation, we assume the following conditions:

- The vocabulary of the training corpus is V and its size is $|V|$.
- The n-grams observed in the corpus are organized in a dictionary: the keys of the dictionary are the words in the vocabulary $w_1, w_2, \ldots, w_{|V|}$, and the values are sets $S_1, S_2, \ldots, S_{|V|}$ of the $(n-1)$-grams that were observed preceding each key.

To calculate the inference time complexity we notice that:

- $N_{1+}(\bullet, w_{i-n+1}, \ldots, w_i)$ can be calculated in constant time by indexing into the dictionary and retrieving $|S_i|$.
- $N_{1+}(\bullet, w_{i-n+1}, \ldots, \bullet)$ requires to traverse all the keys in the dictionary and therefore the calculation is linear in $|V|$.

The time complexity of the quotient in the model equation is thus $O(|V|)$. Finally, we observe that the n-gram model will consist of n such quotients (one for each recursive call of P_{KN}), and therefore the overall complexity will be $O(|V| \cdot n)$.

To calculate the memory complexity of the model, we first estimate the size of its dictionaries:

- All the dictionaries have a fixed number of keys (equals to $|V|$) regardless of the degree of the model.
- At the n-gram level, the set associated with each key can have at most $|V|^{n-1}$ elements. This is the extreme case where all the possible permutations of the words in the vocabulary (allowing repetitions) are observed during training.

With this, the memory complexity of the dictionaries at the n-gram level is clearly $O(|V| \cdot |V|^{n-1}) = O(|V|^n)$. When performing the recursion steps, the accumulated memory complexity will be $O(|V|^n + |V|^{n-1} + \ldots)$, which reduces to $O(|V|^n)$.
Finally, we have to consider the memory required to store the model parameters. Usually, there is a finite set of discount parameters D, which are stored in a constant amount of memory. The number of λ parameters depends on the degree of the model, thus requiring $O(n)$ memory to store them.
Therefore, the overall memory complexity of the model will be $O(|V|^n + n) = O(|V|^n)$.

Inference time complexity of the Kneser-Ney n-gram model $= O(|V|n)$
Memory complexity of the Kneser-Ney n-gram model $= O(|V|^n)$

2. To calculate the inference time complexity of the neural language model proposed by Mnih and Hilton (2007) described in the assignment, we make the following observations:

- The calculation of p_i involves a summation over vector-matrix multiplications. The multiplications are between a vector e_w of dimension $(1 \times d)$ and a matrix C_j of dimension $(d \times d)$: these can be performed in $O(d^2)$ time. There are $(n-1)$ such multiplications inside the summation, and therefore the overall complexity is $O((n-1)d^2) = O(nd^2)$.
- If the value of p_i is given, calculating v_i can be accomplished in $O(d)$ time: the multiplication of the vectors p_i and e_w is $O(d)$ and adding the scalar bias b_w is $O(1)$. Accounting for the time that takes to compute p_i it yields $O(nd^2 + d) = O(nd^2)$.
- Finally, to compute $P\left(w_i|w_{i-n+1}^{i-1}\right)$ we need to sum $e^{v_i(v)}$ over all v in the vocabulary V. Given that the exponentiation can be done in constant time, the time complexity of the final step is $O(|V|) \cdot O(v_i)$.

With this, the final inference time complexity of the neural language model is $O(|V|nd^2)$.

To calculate the memory complexity of the model, we only need to account for the memory required by its parameters:

- There are $(n-1)$ matrices C_j of dimension $(d \times d)$, the memory required to allocate them is $O((n-1)d^2) = O(nd^2)$.
- There are $|V|$ vectors e_w of dimension $(1 \times d)$, the memory required to allocate them is $O(|V|d)$.
- There are $|V|$ scalar bias factors b_w, the memory required to allocate them is $O(|V|)$.

It is worth noticing that p_i and v_i do not require memory allocation because they can be calculated on the fly while executing the algorithm.
Finally, the resulting memory complexity of the model is $O(nd^2 + |V|d + |V|) = O(nd^2 + |V|d)$.

Inference time complexity of the neural language model $= \boldsymbol{O(|V|nd^2)}$
Memory complexity of the neural language model $= \boldsymbol{O(nd^2 + |V|d)}$

3. To reduce the inference time complexity of the neural language model of the previous section, we will consider the parametrization proposed in the assignment:

$$P\left(w_i|w_{i+n-1}^{i-1}\right) = \sum_{c \in C} P\left(w_i|c, w_{i+n-1}^{i-1}\right) P\left(c|w_{i+n-1}^{i-1}\right) \mathbb{I}\left[w_i \in W_c\right]$$

Where C is a set of word clusters $c_1, c_2, \ldots, c_N$ and W_c is the set of words in the cluster c. First we will analyze individually the time complexity of each of the three parts conforming the model:

- By definition, the indicator function $\mathbb{I}\left[w_i \in W_c\right]$ returns 1 if $w_i \in W_c$ and 0 otherwise. If we construct a dictionary over the vocabulary such that (*key, value*) = (w, c), searching for w_i to retrieve its cluster can be done in constant time $O(1)$. It is important to perform this step first because once we define in which cluster w_i is located, we only calculate the probabilities within that cluster and the summation over c reduces to one element.
- The first probability, $P\left(w_i|c, w_{i+n-1}^{i-1}\right)$, is equivalent to the probability for the original model, with the difference that we are calculating it in the context of cluster c, and therefore the vocabulary is reduced to W_c. Thus, the time complexity is $O(|W_c|nd^2)$.
- To calculate the second probability, $P\left(c|w_{i+n-1}^{i-1}\right)$, we need to find a way to express a cluster c in terms of the original neural model. Given that a cluster is nothing but a set of words, intuitively we can express it as the average of the vectors e_w of the words w that are in it:

$$e_c = \frac{1}{|W_c|} \sum_{w \in W_c} e_w$$

  Calculating this vector can be done in $O(|W_c|d)$ time. With this definition, the second probability is also equivalent to the one in the original model, only that in this case the context is the number of clusters N. Accounting for both the vector calculation and the probability itself, the complexity of this part is $O(Nnd^2 + |W_c|d)$.

Finally, adding up the complexities of the three parts and with some algebraic manipulation we arrive to the inference time complexity for the overall model:

$$\boldsymbol{O((|W_c| + N)nd^2)}$$

It is worth noticing that in the extreme case where $N = 1$, it must follow that $|W_c| = |V|$ and the resulting complexity is the same as in the original model. Instead of doing this, we will try to choose $|W_c|$ and N as to minimize the inference time complexity of the modified model.
We cannot optimize over W_c alone because there is no way of knowing *a priori* in which cluster we will find the word of interest w_i, and so we optimize for the average case by making clusters of equal size. Under this condition, the problem at hand reduces to minimizing $|W_c| + N$ given that $|W_c| \cdot N = |V|$.
The solution to this optimization problem is straightforward: $|W_c| = N = \sqrt{|V|}$, and the resulting inference time complexity for the optimized neural language model is

$$\boldsymbol{O(\sqrt{|V|}nd^2)}$$