SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.040 Natural Language Processing, Summer 2020
Homework 3
Due: 17 July 2020, 5pm

1. Language Model (10 points)

Suppose we use a bigram language model to model the a training corpus $\mathcal{D}$ which is composed of many sentences $\boldsymbol{s}$, the probability of $\mathcal{D}$ can be computed as follows:

$$p(\mathcal{D}) = \prod_{\boldsymbol{s}\in\mathcal{D}} \prod_{w_i\in\boldsymbol{s}} p(w_i = b|w_{i-1} = a), \tag{1}$$

where w_i is the i-th word in the sentence $\boldsymbol{s}$.

Prove that the bigram probability $p(w_i = b|w_{i-1} = a)$ which maximizes $p(\mathcal{D})$ is equal to:

$$p(w_i = b|w_{i-1} = a) = \frac{\text{count}(a, b)}{\text{count}(a)} \tag{2}$$

Note that $\text{count}(a, b), \text{count}(a)$ denote the number of occurrences of the bigram (a, b), unigram a in the corpus $\mathcal{D}$. Please clearly show all the steps of your proof.

(5 points)

Answer:

$$\begin{aligned}
\max_{p(b|a)} \log p(\mathcal{D}) &= \max_{p(b|a)} \sum_{\boldsymbol{s}\in\mathcal{D}} \sum_{w_i\in\boldsymbol{s}} \log p(w_i = b|w_{i-1} = a) \\
&= \max_{p(b|a)} \sum_{a\in\mathcal{V}} \sum_{b\in\mathcal{V}} \log p(b|a)^{\text{count(a,b)}} \\
\text{s.t.} \quad & \sum_{b\in\mathcal{V}} p(b|a) = 1, \forall a \in \mathcal{V}
\end{aligned}$$

Use Lagrange multiplier:

$$\max_{p(b|a)} \log p(\mathcal{D}) = \max_{p(b|a)} \sum_{a\in\mathcal{V}} \sum_{b\in\mathcal{V}} \log p(b|a)^{\text{count(a,b)}} + \sum_{a\in\mathcal{V}} \lambda_a (\sum_{b\in\mathcal{V}} p(b|a) - 1)$$

We differentiate $\log p(\mathcal{D})$ with respect to $p(b|a)$, obtaining

$$\frac{\text{count(a,b)}}{p(b|a)} + \lambda_a$$

Let the above equation equal to 0, we will have:

$$p(b|a) = -\frac{\text{count(a,b)}}{\lambda_a}$$

According to the constraints $\sum_{b\in\mathcal{V}} p(b|a) = 1$,

$$\sum_{b\in\mathcal{V}} -\frac{\text{count}(a, b)}{\lambda_a} = 1$$

Thus,

$$-\lambda_a = \sum_{b \in \mathcal{V}} \text{count(a,b)} = \text{count(a)}$$

Plug this term into $p(b|a)$, we will have,

$$p(b|a) = \frac{\text{count}(a, b)}{\text{count}(a)}$$

Suppose we have a training corpus consisting of 2 sentences:

- ⟨START⟩ John loves Mary ⟨END⟩
- ⟨START⟩ Mary likes NLP ⟨END⟩

we model this corpus using a bigram language model. Please compute the probability of all the bigrams in the corpus according to equation (2).

(2 points)

Answer:

$$\begin{aligned} p(John|\langle\text{START}\rangle) &= 0.5 \\ p(Mary|\langle\text{START}\rangle) &= 0.5 \\ p(loves|John) &= 1 \\ p(Mary|loves) &= 1 \\ p(\langle\text{END}\rangle|Mary) &= 0.5 \\ p(likes|Mary) &= 0.5 \\ p(NLP|likes) &= 1 \\ p(\langle\text{END}\rangle|NLP) &= 1 \end{aligned}$$

Now we have a test sentence:

⟨START⟩ Mary likes John ⟨END⟩

As you can see, the bigram *likes John*, *John* ⟨*END*⟩ has never appeared in the training corpus. Thus, we are not able to compute the probability of this sentence. One solution is to interpolate bigram and unigram models. Can you think of a way to interpolate the bigram and unigram models so that the new model can compute the probability of sentences with unseen bigrams? Write down the formulation of your interpolated model and explain your idea.

(3 points)

Answer:

$$\hat{p}(w_i|w_{i-1}) = \lambda_1 p(w_i|w_{i-1}) + \lambda_2 p(w_i), \quad \lambda_1 + \lambda_2 = 1$$

where $p(w_i|w_{i-1}), p(w_i)$ are bigram/unigram language models, respectively.

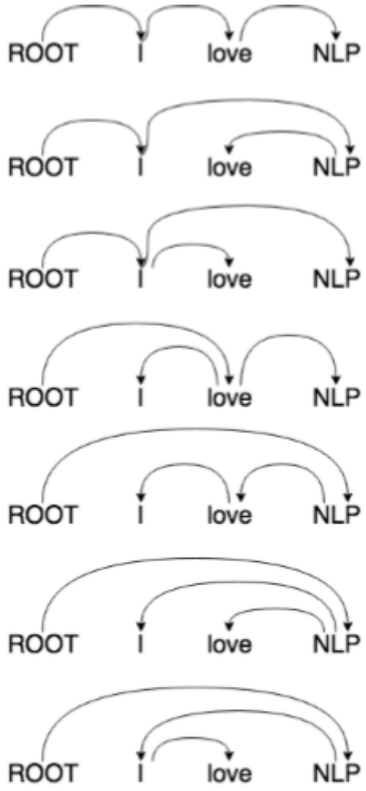
2. Dependency Parsing (10 Points)

Consider the task of unlabeled dependency parsing (i.e., there is no label on the arcs) and consider the sentence "I love NLP". Draw all the possible *projective* dependency trees:

ROOT I love NLP

Note: as we discussed during class, we assume the special ROOT symbol appears to the left of the first word. (5 points)

Answer:

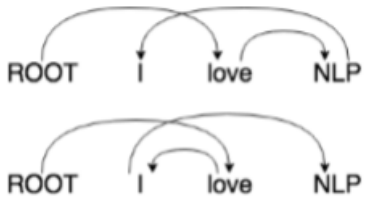


Draw all the possible unlabeled *non-projective* dependency trees for the same sentence.

ROOT I love NLP

(5 points)

Answer:



3. Context Free Grammars (12 Points)

The probabilistic CFG (PCFG) defines a distribution of parse trees, and therefore is able to assign each parse tree a probability, where the leaf nodes of each tree form a word sequence or a sentence. Consider the following PCFG (the probabilities for the rules appear in the

parentheses) where **S** is the designated root non-terminal (i.e., all parse trees should have this as the root):

S	→	**N N**	(0.2)	**S**	→	**N V**	(0.8)
N	→	John	(0.3)	**V**	→	John	(0.1)
N	→	loves	(0.1)	**V**	→	loves	(0.4)
N	→	love	(0.1)	**V**	→	love	(0.2)
N	→	Mary	(0.1)	**V**	→	Mary	(0.1)
N	→	**N V**	(0.1)	**V**	→	**V V**	(0.1)
N	→	**N N**	(0.3)	**V**	→	**V N**	(0.1)

Now, find a way to *efficiently* calculate the probability associated with the sentence "Mary loves John":

$$p(\text{Mary loves John})$$

Clearly show the steps that lead to your answer.

(6 points)

Hint: there might be several trees whose leave nodes form the desired word sequence "John saw Mary" *(this sequence is also called the "yield" of a tree), each of which comes with a probability. The above term is essentially the sum of all such probabilities. However, instead of enumerating all trees, you may need an efficient procedure to find the sum.*

Answer: Similar to the CKY algorithm discussed in class, given PCFG $G = (N, \Sigma, S, R)$, let us define $\pi[i, j, X]$ as the sum of the probabilities of all the possible (partial) parse trees that has a root $X \in N$ and covers the word span $w_i, ..., wj - 1$:

$$\pi[i, j, X] = \sum_{X \to YZ, i+1 \le s \le j-1} p(X \to YZ) \times \pi[i, s, Y] \times \pi[s, j, Z], i \le j - 1$$

$$\pi[i, i+1, X] = \begin{cases} p(X \to Y), & \text{if } X \to Y \in R \\ 0, & \text{otherwise} \end{cases}$$

Step 1:

$$\pi[0, 1, N] = 0.1, \pi[0, 1, V] = 0.1$$
$$\pi[1, 2, N] = 0.1, \pi[1, 2, V] = 0.4$$
$$\pi[2, 3, N] = 0.3, \pi[2, 3, V] = 0.1$$

Step 2:

$$\begin{aligned}\pi[0, 2, N] &= p(N \to NN) \times \pi[0, 1, N] \times \pi[1, 2, N] + p(N \to NV) \times \pi[0, 1, N] \times \pi[1, 2, V] \\ &= 0.3 \times 0.1 \times 0.1 + 0.1 \times 0.1 \times 0.4 = 0.007\end{aligned}$$

Similarly,

$$\begin{aligned}\pi[0, 2, V] &= p(V \to VV) \times \pi[0, 1, V] \times \pi[1, 2, V] + p(V \to VN) \times \pi[0, 1, V] \times \pi[1, 2, N] \\ &= 0.1 \times 0.1 \times 0.4 + 0.1 \times 0.1 \times 0.1 = 0.005\end{aligned}$$

$$\begin{aligned}\pi[1, 3, V] &= p(V \to VV) \times \pi[1, 2, V] \times \pi[2, 3, V] + p(V \to VN) \times \pi[1, 2, V] \times \pi[2, 3, N] \\ &= 0.1 \times 0.4 \times 0.1 + 0.1 \times 0.4 \times 0.3 = 0.016\end{aligned}$$

$$\begin{aligned}\pi[1, 3, N] &= p(N \to NV) \times \pi[1, 2, N] \times \pi[2, 3, V] + p(N \to NN) \times \pi[1, 2, N] \times \pi[2, 3, N] \\ &= 0.1 \times 0.1 \times 0.1 + 0.3 \times 0.1 \times 0.3 = 0.01\end{aligned}$$

Step 3:

$$\begin{aligned}\pi[0,3,S] &= p(S \to NN) \times (\pi[0,1,N] \times \pi[1,3,N] + \pi[0,2,N] \times \pi[2,3,N]) \\ &\quad + p(S \to NV) \times (\pi[0,1,N] \times \pi[1,3,V] + \pi[0,2,N] \times \pi[2,3,V]) \\ &= 0.2 \times (0.1 \times 0.01 + 0.007 \times 0.3) + 0.8 \times (0.1 \times 0.016 + 0.007 \times 0.1) = 0.00246\end{aligned}$$

We have discussed during class how to make use of the CKY algorithm to find the most probable tree structure for an input sentence based on a given *probabilistic* context free grammar (PCFG).

In fact, it is also possible to make use of the CKY algorithm to find the most probable parse tree for a given sentence based on a *weighted* CFG (WCFG). The difference between a WCFG and a PCFG is the former assigns to each grammar rule a *weight* rather than a *probability*, and a tree is scored as the *sum* of the weights of each WCFG rule involved in the tree (rather than a *product* of the probabilities of all rules involved in a tree for the case of PCFG). In other words, the score of a parse tree T is now defined as:

$$score(T) = \sum_{r \in T} weight(r)$$

where $r \in T$ is a rule in the WCFG that appears in the tree T, whose weight is $weight(r)$.

Now, consider the following WCFG (the weights for the rules appear in the parentheses):

S	→	**N N**	(−1.0)	**S**	→	**N V**	(+2.0)
N	→	John	(+1.0)	**V**	→	John	(−1.5)
N	→	loves	(−1.0)	**V**	→	loves	(+1.5)
N	→	love	(−3.0)	**V**	→	love	(+2.5)
N	→	Mary	(+0.5)	**V**	→	Mary	(−0.5)
N	→	**N V**	(−1.0)	**V**	→	**V V**	(−1.0)
N	→	**N N**	(+1.0)	**V**	→	**V N**	(−2.0)

Briefly explain how to modify the CKY algorithm discussed in class so that it works for WCFG.

Find the most probable parse tree (as well as its weight) based on your algorithm for the following sentence (if there is more than one most probable parse trees, find them all):

John loves Mary

Clearly show the steps that lead to your answer. *(6 points)*

Answer: Step 1:

$$\begin{aligned}\pi[0,1,N] = 1.0, \pi[0,1,V] = -1.5 \\ \pi[1,2,N] = -1.0, \pi[1,2,V] = 1.5 \\ \pi[2,3,N] = 0.5, \pi[2,3,V] = -0.5\end{aligned}$$

Step 2:

$$\begin{aligned}\pi[0,2,N] &= \max\{weight(N \to NN) + \pi[0,1,N] + \pi[1,2,N], weight(N \to NV) + \pi[0,1,N] + \pi[1,2,V]\} \\ &= \max\{1+1-1, -1+1+1.5\} = 1.5 \quad \text{(best split rule: N→ NV)}\end{aligned}$$

Similarly,

$$\pi[0,2,V] = \max\{weight(V \to VV) + \pi[0,1,V] + \pi[1,2,V], weight(V \to VN) + \pi[0,1,V] + \pi[1,2,N]\}$$
$$= \max\{-1 - 1.5 + 1.5, -2 - 1.5 - 1\} = -1 \quad \text{(best split rule: V}\to \text{ VV)}$$

$$\pi[1,3,V] = \max\{weight(V \to VV) + \pi[1,2,V] + \pi[2,3,V], weight(V \to VN) + \pi[1,2,V] + \pi[2,3,N]\}$$
$$= \max\{-1 + 1.5 - 0.5, -2 + 1.5 + 0.5\} = 0 \quad \text{(best split rule: V}\to \text{ VV or V}\to \text{ VN)}$$

$$\pi[1,3,N] = \max\{weight(N \to NV) + \pi[1,2,N] + \pi[2,3,V], weight(N \to NN) + \pi[1,2,N] + \pi[2,3,N]\}$$
$$= \max\{-1 - 1 - 0.5, 1 - 1 + 0.5\} = 0.5 \quad \text{(best split rule: N}\to \text{ NN)}$$

Step 3:

$$\pi[0,3,S] = \max\{weight(S \to NN) + \max(\pi[0,1,N] + \pi[1,3,N], \pi[0,2,N] + \pi[2,3,N]),$$
$$weight(S \to NV) + \max(\pi[0,1,N] + \pi[1,3,V], \pi[0,2,N] + \pi[2,3,V])\}$$
$$= \max\{-1 + \max(1 + 0.5, 1 + 0.5), 2 + \max(1 + 0, 1.5 - 0.5)\} = 3 \text{(best split rule: S}\to \text{ NV)}$$

Now we rebuild the parse tree.

If the split point is 1, the most probable parse tree will be

$$(\text{S} \to \text{NV}, (\text{N} \to \text{John}), (\text{V} \to \text{VV}, (\text{V} \to \text{loves}, \text{V} \to \text{Mary})))$$

$$(\text{S} \to \text{NV}, (\text{N} \to \text{John}), (\text{V} \to \text{VN}, (\text{V} \to \text{loves}, \text{N} \to \text{Mary})))$$

If the split point is 2, the most probable parse tree will be

$$(\text{S} \to \text{NV}, (\text{N} \to \text{NV}, (\text{N} \to \text{John}, \text{V} \to \text{loves})), \text{V} \to \text{Mary})$$

There are totally 3 most probable trees and the score for each tree is 3.

Now, find the 4th most probable parse tree (as well as its weight) for the following sentence:

John loves Mary

This requires a modification to the CKY algorithm. Clearly describe the algorithm, and clearly show the steps that lead to your answer. *(6 points)*

Answer:

Step 1:

$$\pi[0,1,N] = 1.0, \pi[0,1,V] = -1.5$$
$$\pi[1,2,N] = -1.0, \pi[1,2,V] = 1.5$$
$$\pi[2,3,N] = 0.5, \pi[2,3,V] = -0.5$$

Step 2:

$$\pi[0,2,N] = \text{Top-4}\{weight(N \to NN) + \pi[0,1,N] + \pi[1,2,N],$$
$$weight(N \to NV) + \pi[0,1,N] + \pi[1,2,V]\}$$
$$= \text{Top-4}\{1 + 1 - 1, -1 + 1 + 1.5\} = \{1, 1.5\}$$

$$\begin{aligned}\pi[0,2,V] &= \text{Top-4}\{weight(V \to VV) + \pi[0,1,V] + \pi[1,2,V],\\ &\qquad weight(V \to VN) + \pi[0,1,V] + \pi[1,2,N]\}\\ &= \text{Top-4}\{-1-1.5+1.5, -2-1.5-1\} = \{-1, -4.5\}\end{aligned}$$

$$\begin{aligned}\pi[1,3,V] &= \text{Top-4}\{weight(V \to VV) + \pi[1,2,V] + \pi[2,3,V],\\ &\qquad weight(V \to VN) + \pi[1,2,V] + \pi[2,3,N]\}\\ &= \text{Top-4}\{-1+1.5-0.5, -2+1.5+0.5\} = \{0, 0\}\end{aligned}$$

$$\begin{aligned}\pi[1,3,N] &= \text{Top-4}\{weight(N \to NV) + \pi[1,2,N] + \pi[2,3,V],\\ &\qquad weight(N \to NN) + \pi[1,2,N] + \pi[2,3,N]\}\\ &= \text{Top-4}\{-1-1-0.5, 1-1+0.5\} = \{-2.5, 0.5\}\end{aligned}$$

Step 3:

$$\begin{aligned}\pi[0,3,S] &= \text{Top-4}\{weight(S \to NN) + \text{Top-4}\{\pi[0,1,N] + \pi[1,3,N], \pi[0,2,N] + \pi[2,3,N]\},\\ &\qquad weight(S \to NV) + \text{Top-4}\{\pi[0,1,N] + \pi[1,3,V], \pi[0,2,N] + \pi[2,3,V]\}\}\\ &= \text{Top-4}\{-1 + \text{Top-4}\{1+0.5, 1-2.5, 1.5+0.5, 1+0.5\},\\ &\qquad 2 + \text{Top-4}\{1+0, 1+0, 1-0.5, 1.5-0.5\}\} = \{3, 3, 2.5, 3\}\end{aligned}$$

The 4th most probable parse tree is,

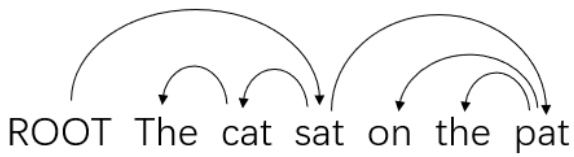
$$(\text{S} \to \text{NV}, (\text{N} \to \text{NN}, (\text{N} \to \text{John}, \text{N} \to \text{loves})), \text{V} \to \text{Mary})$$

4. Transition-based Parsing (18 Points)

Recall the "arc-standard" transition-based algorithm for parsing discussed in class involves the following actions:

- **sh** (shift): shift the next word in the buffer (i.e., the head of the buffer) to the stack
- **la** (left-arc): add an arc from the topmost word on the stack, s_1, to the second-topmost word, s_2, and pop s_2
- **ra** (right-arc): add an arc from the second-topmost word on the stack, s_2, to the topmost word, s_1, and pop s_1

Now, consider the following dependency tree:



List down the sequence of actions involved for parsing the sentence into the dependency tree. Assume the initial configuration is: the stack has one element at the top: ROOT, and the buffer contains the sequence of words from the input sentence, with the first word being the head of the buffer (i.e., the word "The" appears at the beginning of the buffer). *(6 points)*

Note: it is not required for you to draw the configuration (i.e., status of stack, buffer, partial tree) at each step, but you only need to correctly list down the sequence of actions. Some brief explanations are fine, but make them concise.

Answer: Action sequence is **sh, sh, la, sh, la, sh, sh, sh, la, la, ra, ra**

Action	Transition	Stack	Buffer	Remark
/	/	[ROOT]	[The cat sat on the pat]	
sh	shift	[ROOT The]	[cat sat on the pat]	
sh	shift	[ROOT The cat]	[sat on the pat]	
la	left-arc	[ROOT cat]	[sat on the pat]	pop The
sh	shift	[ROOT cat sat]	[on the pat]	
la	left-arc	[ROOT sat]	[on the pat]	pop cat
sh	shift	[ROOT sat on]	[the pat]	
sh	shift	[ROOT sat on the]	[pat]	
sh	shift	[ROOT sat on the pat]	[]	
la	left-arc	[ROOT sat on pat]	[]	pop the
la	left-arc	[ROOT sat pat]	[]	pop on
ra	right-arc	[ROOT sat]	[]	pop pat
ra	right-arc	[ROOT]	[]	pop sat

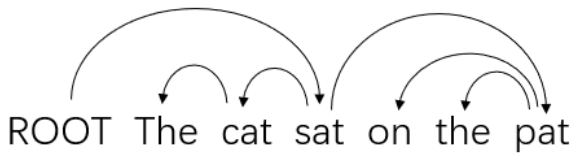
Give a worst-case time complexity analysis for the above "arc-standard" transition system for unlabelled dependency parsing (assume the input sentence has n words). Clearly explain your answer. *(2 points)*

Answer: $\mathcal{O}(n)$. For each word, there is a shift action to push it onto the stack and a left-arc or right-arc action to pop it. If the input sequence has n words, the total number of actions is 2n.

Now, let us consider a new "arc-eager" system with the following new actions (note that the definitions to **la** and **ra** are now different):

- **sh** (shift): shift the next word in the buffer (i.e., the head of the buffer) to the stack
- **re** (reduce): pop the stack
- **la** (left-arc): add an arc from the head of the buffer b_1, to the topmost word on the stack s_1, and pop s_1
- **ra** (right-arc): add an arc from the topmost word on the stack, s_1, to the head of the buffer b_1, and shift b_1 to the stack

Again, consider the same dependency tree:



List down the sequence of actions involved for parsing the sentence into the dependency tree. Assume the initial configuration is: the stack has one element at the top: ROOT, and the buffer contains the sequence of words from the input sentence, with the first word being the head of the buffer (i.e., the word "The" appears at the beginning of the buffer). *(8 points)*

Note: it is not required for you to draw the configuration (i.e., status of stack, buffer, partial tree) at each step, but you only need to correctly list down the sequence of actions. Some brief explanations are fine, but make them concise.

Answer: Action sequence: **sh, la, sh, la, ra, sh, sh, la, la ,ra, re, re**

Action	Transition	Stack	Buffer	Remark
/	/	[ROOT]	[The cat sat on the pat]	
sh	shift	[ROOT The]	[cat sat on the pat]	
la	left-arc	[ROOT]	[cat sat on the pat]	pop The
sh	shift	[ROOT cat]	[sat on the pat]	
la	left-arc	[ROOT]	[sat on the pat]	pop cat
ra	right-arc	[ROOT sat]	[on the pat]	shift sat
sh	shift	[ROOT sat on]	[the pat]	
sh	shift	[ROOT sat on the]	[pat]	
la	left-arc	[ROOT sat on]	[pat]	pop the
la	left-arc	[ROOT sat]	[pat]	pop on
ra	right-arc	[ROOT sat pat]	[]	shift pat
re	reduce	[ROOT sat]	[]	pop pat
re	right-arc	[ROOT]	[]	pop sat

Give a worst-case time complexity analysis for the above "arc-eager" transition system for unlabelled dependency parsing (assume the input sentence has n words). Clearly explain your answer. *(2 points)*

Answer: $\mathcal{O}(n)$. For each word, there is a shift action or right-arc to push it onto the stack and a left-arc or reduce action to pop it. If the input sequence has n words, the total number of actions is 2n.