Natural Language Processing Homework3

Hong Pengfei 1002949

July 2020

1 Language Model

1 Take log on both side:

$$log(p(D)) = \sum_{s \in D} \sum_{w_i \in s} log(p(w_i|w_{i-1}))$$

The sum of all the $log(p(w_i|w_{i-1})$ in the document for a given bigram w_i, w_j equals to the logprob of the bigrams $log(p(x_j|x_i))$ times its count appearing in the document.

$$\sum_{s \in D} \sum_{w_i \in s} log(p(w_i|w_{i-1})) = \sum_{i}^{|V|} \sum_{j}^{|V|} count(w_i, w_j) * log(p(x_j|x_i))$$

To maximize $log(p(D))$, subject to $\forall i \in [1, ..., |V|], \sum_j^{|V|} p(w_j|w_i) = 1$ where $|V|$ is the vocabulary size in all Documents.
Equivalently, we can optimize the auxiliary optimization function using Lagrange multiplier ($\sum_j^{|V|} p(w_j|w_i) - 1 = 0$):

$$\mathcal{L} = \sum_{i}^{|V|} \sum_{j}^{|V|} log(count(w_i, w_j) * p(x_j|x_i)) + \sum_{i=1}^{|V|} \lambda_i (\sum_{j=1}^{|V|} p(w_j|w_i) - 1)$$

$\forall p(w_i|w_{i-1})$, we take derivative of $\mathcal{L}$ with respect to $p(w_j|w_i)$ and set it to 0:

$$\frac{\partial \mathcal{L}}{\partial p(w_i|w_{i-1})} = \frac{count(w_i, w_j)}{p(w_j|w_i)} + \lambda_i = 0$$

$$p(w_j|w_i) = -\frac{count(w_i, w_j)}{\lambda_i} \tag{1}$$

Plugging 1 into $\sum_j^{|V|} p(w_j|w_i) - 1 = 0$ we have:

$$\sum_{j}^{|v|} p(w_j|w_i) = \sum_{j}^{|v|} -\frac{count(w_i, w_j)}{\lambda_i} = \frac{\sum_j^{|v|} count(w_i, w_j)}{-\lambda_i} = 1$$

$$\lambda_i = -\sum_{j}^{|v|} count(w_i, w_j) \tag{2}$$

We plug 2 into 1, therefore

$$p(w_j = b|w_i = a) = \frac{count(w_i = a, w_j = b)}{\sum_j^{|V|} count(w_i = a, w_j = b)} = \frac{count(a, b)}{count(a)}$$

2 probability of all bigrams:

	John	loves	Mary	likes	NLP	END
START	0.5	0	0.5	0	0	0
John	0	1	0	0	0	0
loves	0	0	1	0	0	0
Mary	0	0	0	0.5	0	0.5
likes	0	0	0	0	1	0
NLP	0	0	0	0	0	1

The idea is to add a hyperparameter λ to interpolate between unigram probability and bigram probability:

$$p(u|v) = \lambda p(u) + (1 - \lambda)p(u|v)$$

, where $p(u)$ is the probability output by unigram model, and $p(u|v)$ is the probability output by bigram model. In this way, we can deal with unknown bigrams.
Furthermore, the hyperparamter λ is tuned on the validation set.

2 Dependency Parsing

Projective trees Please see the supplmentary material figure 1. 1. the projective trees are circled out in red. there are total 7 projective trees. if root can only have one children.
if root is allowed to have multiple children, then see figure 2 2 there are 12 projective trees.

Non-projective trees Please see the supplementary material figure 1 1 the non projective trees. there are total 2 non-projective trees (if including projective trees there are 9).
if root is allowed to have multiple children, then see figure 2 2 there are 4 nonprojective trees.

3 Context Free grammars

3.1 PCFG

Using CKY algorithm but change the max sign to sum sign can work on unique subproblems and efficiently and systematically get the solution.

First step: find all the nonterminal labels for span = 1 . that is for word: Mary, loves, John

- For Mary - N (0.1) or V (0.1)
- For loves - N (0.1) or V (0.4)
- For John - N (0.3) or V (0.1)

Second step: find all the nonterminals labels for span = 2 that is for words: Mary loves & loves John.
For "Mary loves" it can be those probability

- N $\rightarrow$ N V = 0.1 * 0.1 * 0.4 = 4e-3
- N $\rightarrow$ N N = 0.3 * 0.1 * 0.1 = 3e-3
- V $\rightarrow$ V V = 0.1 * 0.1 * 0.4 = 4e-3
- V $\rightarrow$ V N = 0.1 * 0.1 * 0.1 = 1e-3

Therefore taking the sum of possibility for each label, we have:

- N $\rightarrow$ N V = 0.1 * 0.1 * 0.4 = 7e-3
- V $\rightarrow$ V V = 0.1 * 0.1 * 0.4 = 5e-3

Similarly for "loves John"

- N $\rightarrow$ N V = 0.1 * 0.1 * 0.1 = 1e-3
- N $\rightarrow$ N N = 0.3 * 0.1 * 0.3 = 9e-3
- V $\rightarrow$ V V = 0.1 * 0.4 * 0.1 = 4e-3
- V $\rightarrow$ V N = 0.1 * 0.4 * 0.3 = 12e-3

Therefore taking the maximum possibility for each label, we have:

- N $\rightarrow$ N N = 0.3 * 0.1 * 0.3 = 1e-2
- V $\rightarrow$ V N = 0.1 * 0.4 * 0.3 = 16e-3

Finally for the whole span "Mary loves John"

- S → N N = 0.2 * 0.1 *1e-2 + 0.2 * 7e-3 * 0.3) = 2e-4 + 42e-5 = 62e-5
- S → N V = 0.8 * 0.1 * 16e-3 + 0.8 * 7e-3 * 0.1) = 1.84e-3

Therefore, the total probability for the sentence is 62e-5 + 1.84e-3 = 0.00246

3.2 WCFG

Using CKY algorithm instead of times the probability, we add the scores together, it can work on unique subproblems and efficiently and systematically get the solution.

First step: find all the nonterminal labels for span = 1 . that is for word: Mary, loves, John

- For John - N (1) or V (-1.5)
- For loves - N (-1) or V (1.5)
- For Mary - N (0.5) or V (-0.5)

Second step: find all the nonterminals labels for span = 2 that is for words: Mary loves & loves John.
For "John loves" it can be those probability

- N → N V = -1 + 1 + 1.5 = 1.5
- N → N N = 1 + 1 - 1 = 1
- V → V V = -1 + -1.5 + 1.5 = -1
- V → V N = -2 + -1.5 + -1 = -4.5

Therefore taking the maximum score for each label, we have:

- N → N V = -1 + 1 + 1.5 = 1.5
- V → V V = -1 + -1.5 + 1.5 = -1

Similarly for "loves Mary"

- N → N V = -1 + -1 + -0.5 = -2.5
- N → N N = 1 + -1 + 0.5 = 0.5
- V → V V = -1 + 1.5 + -0.5 = 0
- V → V N = -2 + 1.5 + 0.5 = 0

Therefore taking the maximum score for each label, we have:

- N → N N = 1 + -1 + 0.5 = 0.5
- V → V N = -2 + 1.5 + 0.5 = 0

Finally for the whole span "John loves Mary"

- S → N N = max(-1 + 1 + 0.5, -1 + 1.5 + 0.5) = max(0.5, 1) = 1
- S → N V = max(2 + 1 + 0, 2 + 1.5 + -0.5) = max(3, 3) = 3

Therefore, the structure can be any
(S ((John N) (V (V loves) (N Mary))))
(S (N (John N) (V loves)) (V Mary))))
(S ((John N) (V (V loves) (V Mary))))
with score 3.

3.3 4th most probable tree

every time stores 4 most probable tags for a given span and tag, and then pick the most probable parent based on all rules applied to those 4 tags.

First step: find all the nonterminal labels for span = 1 . that is for word: John, loves, Mary

- For John - N (1) or V (-1.5)
- For loves - N (-1) or V (1.5)
- For Mary - N (0.5) or V (-0.5)

Second step: find all the nonterminals labels for span = 2 that is for words: John loves & loves Mary.
For "John loves" it can be those probability

- N → N V = -1 + 1 + 1.5 = 1.5
- N → N N = 1 + 1 - 1 = 1
- V → V V = -1 + -1.5 + 1.5 = -1
- V → V N = -2 + -1.5 + -1 = -4.5

Therefore taking the maximum score for span "loves Mary", each tag can have two scores and smaller than maximum of 4 scores, so we will keep them all.

Similarly for "loves Mary"

- N → N V = -1 + -1 + -0.5 = -2.5
- N → N N = 1 + -1 + 0.5 = 0.5
- V → V V = -1 + 1.5 + -0.5 = 0
- V → V N = -2 + 1.5 + 0.5 = 0

Finally for the whole span "Mary loves John"

- S → N (John) N (N loves V Mary) = -1 + 1 + -2.5 = -2.5
- S → N (John) N (N loves N Mary) = -1 + 1 + 0.5 = 0.5
- S → N (N John V loves) N (Mary) = -1 + 1.5 + 0.5 = 1
- S → N (N John N loves) N (Mary) = -1 + 1 + 0.5 = 0.5
- S → N (John) V (V loves V Mary) = 2 + 1 + 0 = 3
- S → N (John) V (V loves N Mary) = 2 + 1 + 0 = 3
- S → N (N John V loves) V (Mary) = 2 + 1.5 + -0.5 = 3
- S → N (N John N loves) V (Mary) = 2 + 1 + -0.5 = 2.5

Therefore, the 4th structure is tree

$$(S(N(NJohn)(Nloves))(VMary))$$

with weight(score) of 2.5

4

Transition based Parsing.

1. Stack: ROOT — Buffer: The cat sat on the pat — action: sh
2. Stack: ROOT The — Buffer: cat sat on the pat — action: sh
3. Stack: ROOT The cat — Buffer: sat on the pat — action: la
4. Stack: ROOT cat — Buffer: sat on the pat — action: sh
5. Stack: ROOT cat sat — Buffer: on the pat — action: la
6. Stack: ROOT sat — Buffer: on the pat — action: sh
7. Stack: ROOT sat on — Buffer: the pat — action: sh
8. Stack: ROOT sat on the — Buffer: pat — action: sh
9. Stack: ROOT sat on the pat — Buffer: Empty — action: la
10. Stack: ROOT sat on pat — Buffer: Empty — action: la
11. Stack: ROOT sat pat — Buffer: Empty — action: ra
12. Stack: ROOT sat — Buffer: Empty — action: ra
13. Stack: ROOT — Buffer: Empty — action: FINISHED

Transition based parsing Complexity The Worst case complexity is $O(n)$. because it need to finish a process of adding all the word from the buffer to the stack (using sh), and pop all the word off the stack using (la or ra). so total number of actions to do will always be $2 * n$ which is linear with the number of words n.

Eager transition based parsing

1. Stack: ROOT — Buffer: The cat sat on the pat — action: sh
2. Stack: ROOT The — Buffer: cat sat on the pat — action: la
3. Stack: ROOT — Buffer: cat sat on the pat — action: sh
4. Stack: ROOT cat — Buffer: sat on the pat — action: la
5. Stack: ROOT — Buffer: sat on the pat — action: ra
6. Stack: ROOT sat — Buffer: on the pat — action: sh
7. Stack: ROOT sat on — Buffer: the pat — action: sh
8. Stack: ROOT sat on the — Buffer: pat — action: la
9. Stack: ROOT sat on — Buffer: pat — action: la
10. Stack: ROOT sat — Buffer: pat — action: ra
11. Stack: ROOT sat pat — Buffer: Empty — action: re
12. Stack: ROOT sat — Buffer: Empty — action: re
13. Stack: ROOT — Buffer: Empty — action: FINISHED

Eager transition based parsing complexity The worse complexity is still $O(n)$. because it need to finish the process of moving all the words from the buffer to stack using ra or shift. and remove all the words from the stack either by la or re. so the total number of actions will be executed $2n$ times with complexity of $O(n)$.

5 Supplementary material

6 Supplementary material

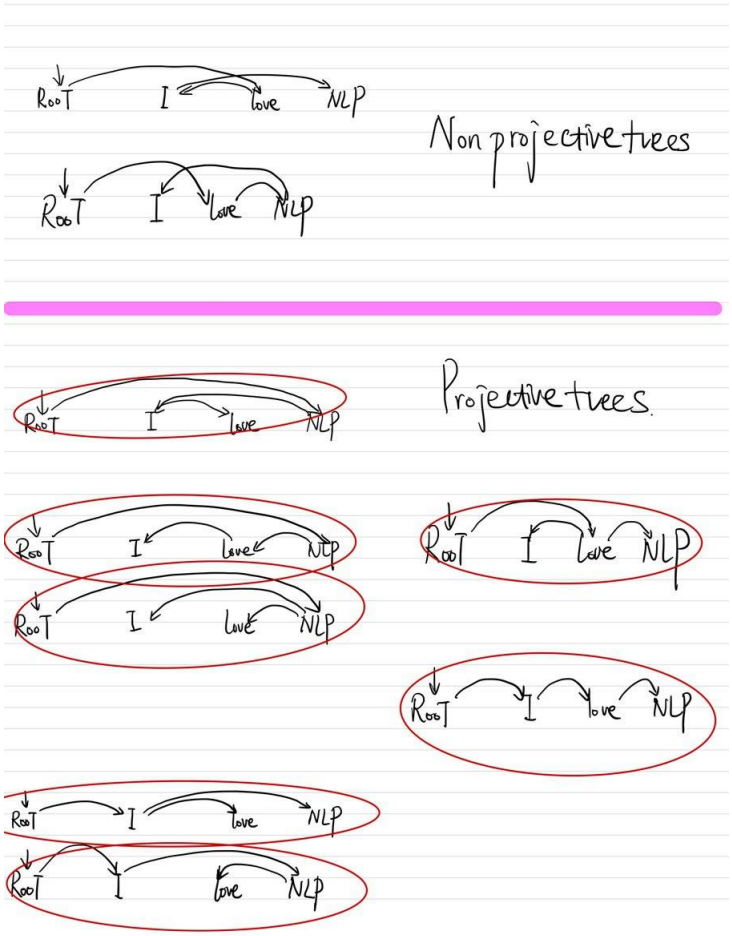

Figure 1: dep trees (root can only have one children) with projective trees circled out in red

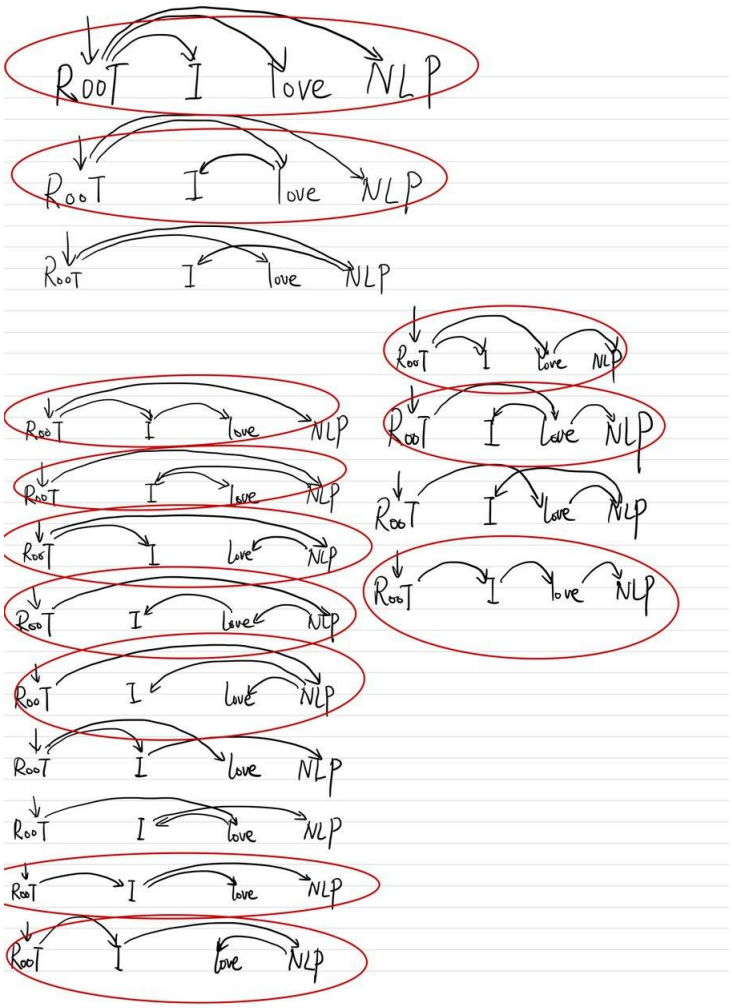

Figure 2: dep trees (root can have multiple children) with projective trees circled out in red