

# Natural Language Processing

---

## Homework 3

---

Ivan Christian  
1003056

### 1. Language Model

#### Part 1

Prove that

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

Given that

$$w_1, w_2, w_3, w_4, \dots, w_n$$

are sequences of words in the sentence, we want to have the probability to predict the next word given the current word. As such we are able to get this criteria that we want, which is

$$p(w_{n-1}|w_n)$$

However, every current and next words are dependent on previous words that results in the following:

$$p(w_{n-1}|w_1, w_2, w_3, \dots, w_n)$$

The bigram model, however, approximates the probability of a word given bigram all the previous words,

$$P(w_n|w_1^{n-1})$$

Using the history approximation we can get the criteria

$$p(w_n|w_{n-1})$$

as the bigram model would assume such that it is only dependent on the previous word instead of the whole history (Markov assumptions).

The intuitive way to maximise a probability is to employ the **Maximum Likelihood Estimation** which includes counting the appearance of words in the sentence. This would get us

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

as the probability calculation and hence proving that the

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

## Part 2

Training Corpus:

⟨START⟩ John loves Mary ⟨END⟩

⟨START⟩ Mary likes NLP ⟨END⟩

All the count of each word in the training corpus

```
count(<START>) = 2
count(<END>) = 2
count(John) = 1
count(Mary) = 2
count(loves) = 1
count(likes) = 1
count(NLP) = 1
```

All the probability for the bigram

```
count(<START>,John)/count(<START>) = 1/2
count(John, loves)/count(John) = 1/1
count(loves, Mary)/count(loves) = 1/1
count(Mary, <END>)/count(Mary) = 1/2
count(<START>, Mary)/count(<START>) = 1/2
count(Mary, likes)/count(Mary) = 1/2
count(likes, NLP)/count(likes) = 1/1
count(NLP,<END>)/count(NLP) = 1/1
```

Part 3

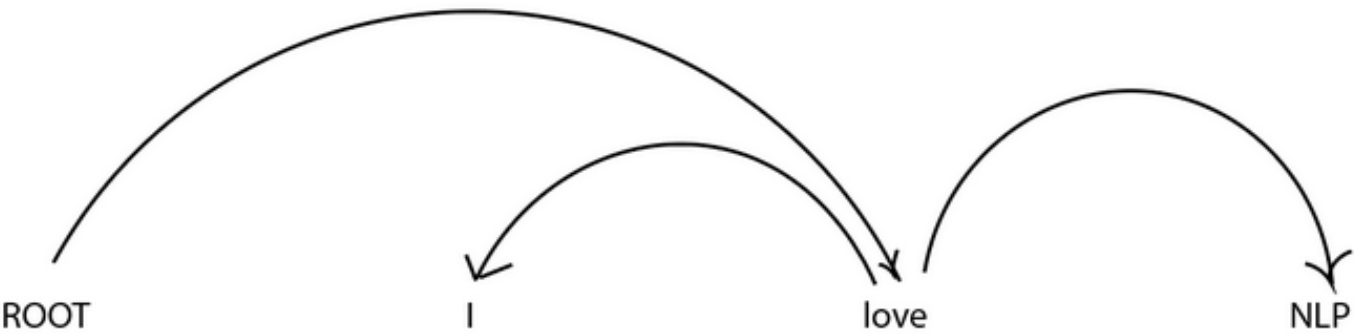
To accomodate to the missing [John, <END>] case we would need to check for unknown cases and label them as <unk> . This is done so that the unknown cases are not automatically set to zero and this can be done by smoothing the probability. Smoothing the probability would mean adding 1 to both the denominator and the numerator to take into account unknown cases such as [John, <END>] . This can be done in trrough the following

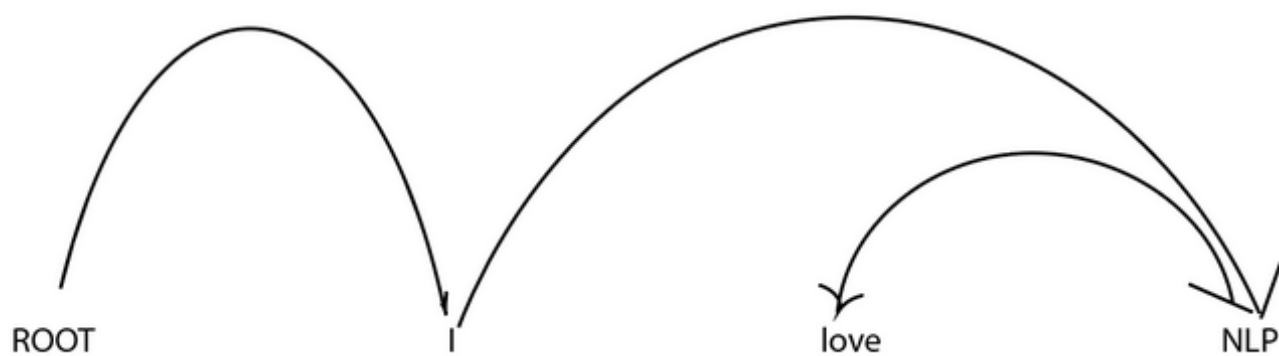
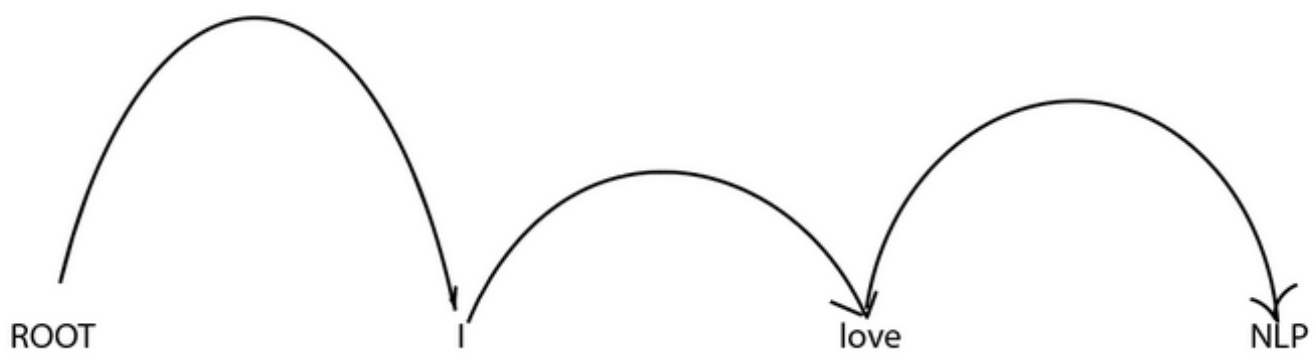
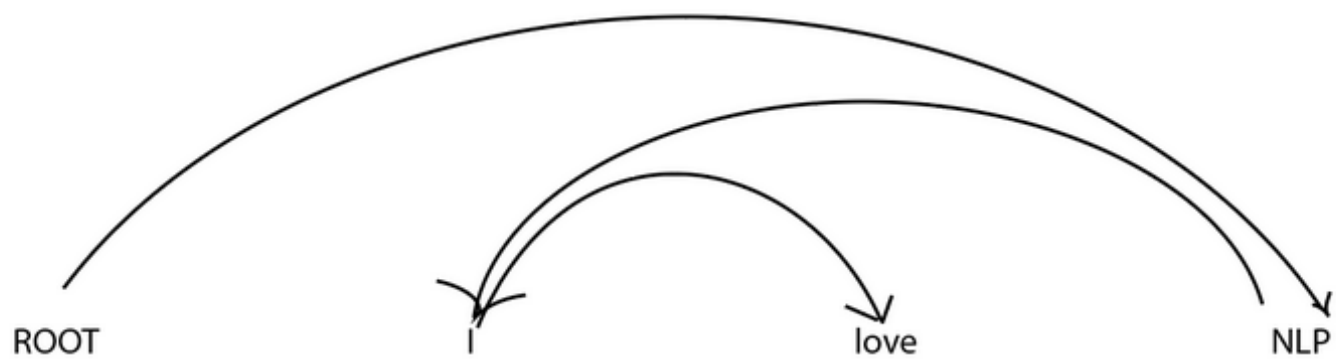
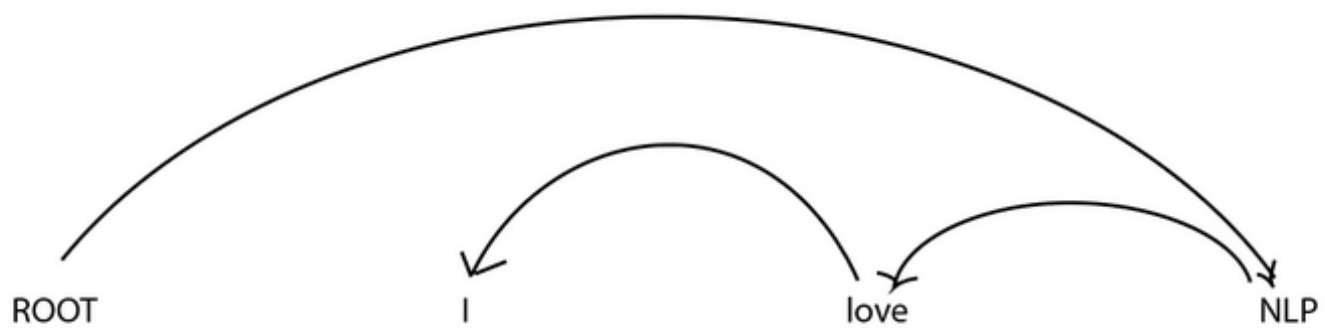
$$p(D) = \frac{count(a, b) + 1}{count(a) + 1}$$

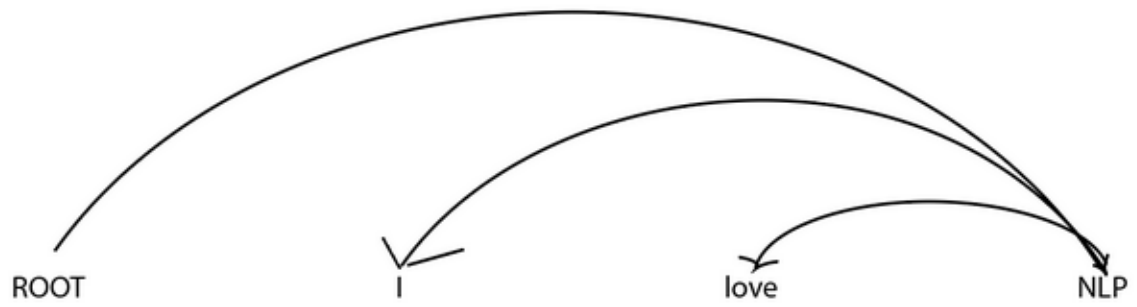
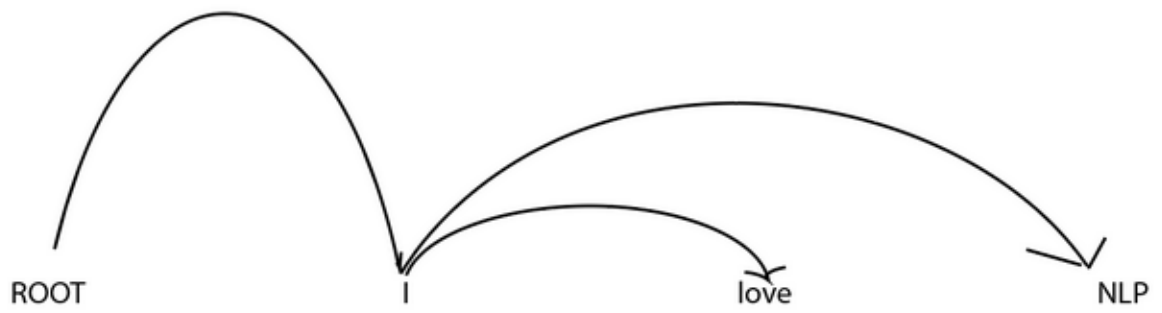
Through this, it would be highly unlikely to have a 0 probability for unknown cases.

2. Dependency Parsing

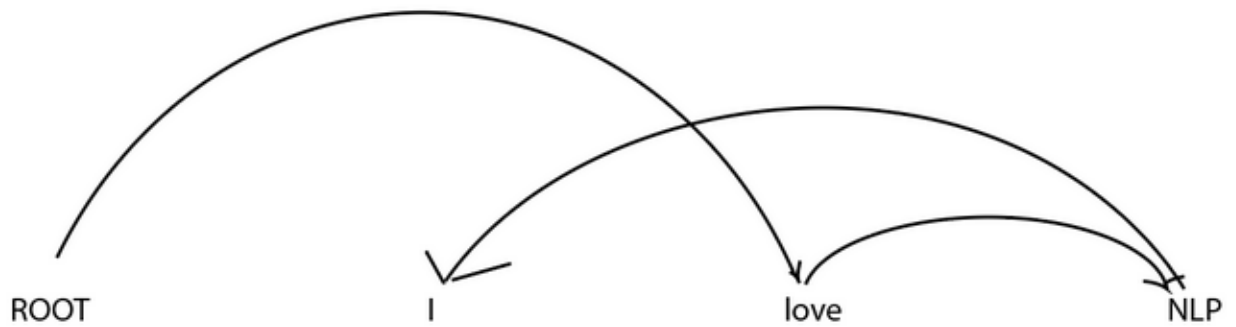
Part 1 (Projective)

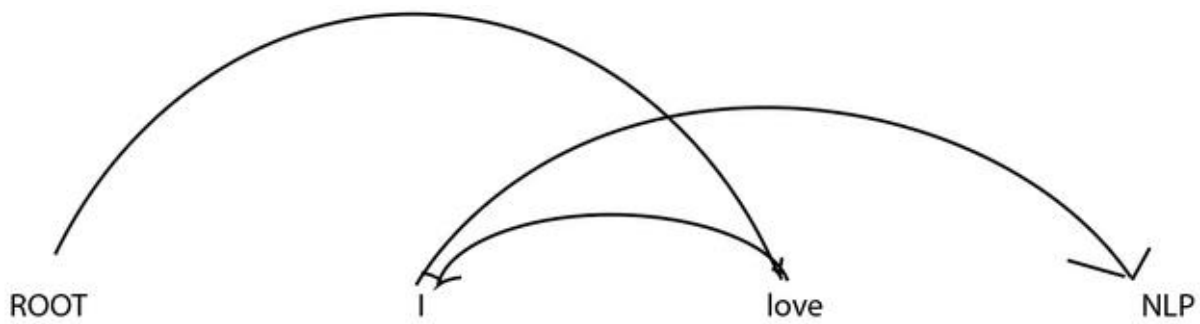






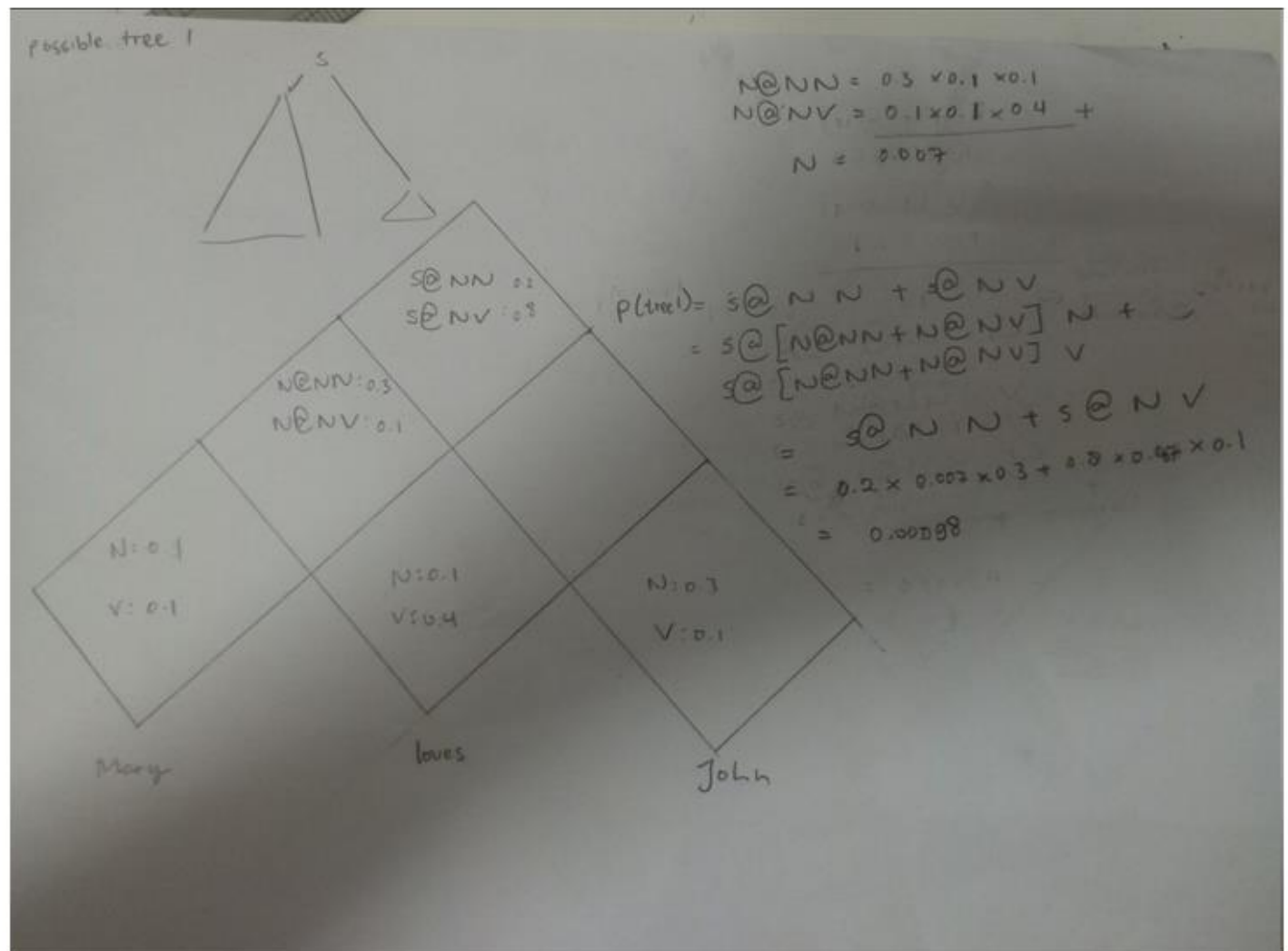
## Part 2 (Non Projective)





### 3. Context Free Grammar

#### Part 1



Possible tree 2



$$p(\text{subtree 2}) = S @ NN + S @ NV$$

$$\begin{aligned} N @ NN &= 0.3 \times 0.1 \times 0.3 \\ N @ NV &= 0.1 \times 0.1 \times 0.1 \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{sum} = 0.01$$

$$\begin{aligned} V @ VV &= 0.1 \times 0.4 \times 0.1 \\ V @ VN &= 0.1 \times 0.4 \times 0.3 \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} \text{sum} = 0.016$$

$$\begin{aligned} p(\text{tree 2}) &= 0.2 \times 0.1 \times 0.01 + 0.8 \times 0.1 \times 0.016 \\ &= 0.00148 \end{aligned}$$



$$p(\text{'Mary loves John'}) = 0.00098 + 0.00148$$

$$p(\text{' Mary loves John '}) = 0.00246$$

## Part 2

Instead of multiplying in the usual CKY algorithm to find the max tree probability, you instead sum it. Doing so would make it similar to sum the log probabilities of probability in a vanilla CKY algorithm. This would achieve the same result for maximising the probability to get the best possible weight for the algorithm. This would maximise the score using the weights corresponding to the nodes.

Tree 1 possibility

only N will exit

(2)

$S@NN: -1.0$   
 $S@NV: 2.0$

$N@NN: 1.0$   
 $N@NV: -1.0$

$N: +1.0$   
 $V: -1.5$

$N: -1.0$   
 $V: +1.5$

$N: +0.5$   
 $V: -0.5$

John

Ives

Mary

max ①  $\rightarrow N@NN = 1 + 1 + (-1) = 1$   
 $\rightarrow N@NV = -1 + 1.5 + 1 = 1.5$

max ②  $= \{N@NV\}$   
 $S@NN = -1.0 + 1.5 + 0.5 = 1.0$   
 $S@NV = 2.0 + 1.5 - 0.5 = 3.0$

max ③  $= \max(\text{Tree 1 possibility}) = 3.0 \Rightarrow S@NV$

(2)

only 2  $\frac{1}{2}$  11

9

$$N = +1.0$$
$$V = -1.5$$

John

N: -1.0  
V: +1.5

loves

$N = +0.5$   
 $V = -0.5$

Maru

$$\begin{aligned} \max \{ & \rightarrow N @ N N = 1 + 1 + (-1) \\ & \quad = 1 \\ & \rightarrow N @ N V = -1 + 1 + 1 \\ & \quad = 1.5 \end{aligned}$$

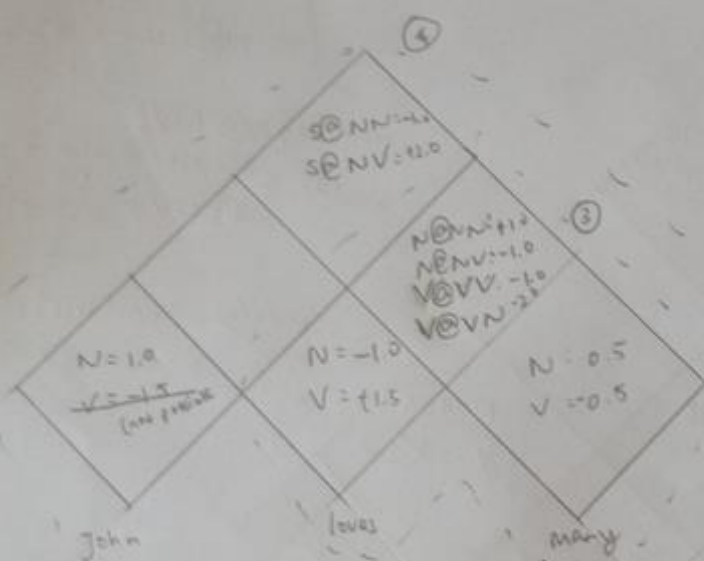
$$\max \textcircled{1} = \{N @ NV\}$$

$$\max(2) = \begin{cases} S @ NN = -1.0 + 1.5 + 0.5 \\ S @ NV = 2.0 + 1.5 - 0.5 \\ = 3.0 \end{cases}$$

$$\begin{aligned} \max(2) &= \max(\text{Tree possibility}) \\ &= 3.0 \Rightarrow 5 @ (N @ N @ N) \end{aligned}$$



Tree 2 possibility



$$\text{max } (5) = \begin{cases} N@NN = 1.0 + (-1.0) + (0.5) \\ \quad = 0.5 \\ N@NV = -1.0 - 1.0 - 0.5 \\ \quad = -2.5 \\ V@VV = 1.5 - 1.0 - 0.5 \\ \quad = 0 \\ V@VN = -2.0 + 1.5 + 0.5 \\ \quad = 0 \end{cases}$$

$$\text{max } (3) = N@NN$$

$$\text{max } (4) > S@NN = 0.5$$

Hence the most probable tree would be  $S@N@NV)V$

Based on the weights given and the calculation we know that the maximum possible weights that can be gotten is from tree 1 which makes the score of 3.0.

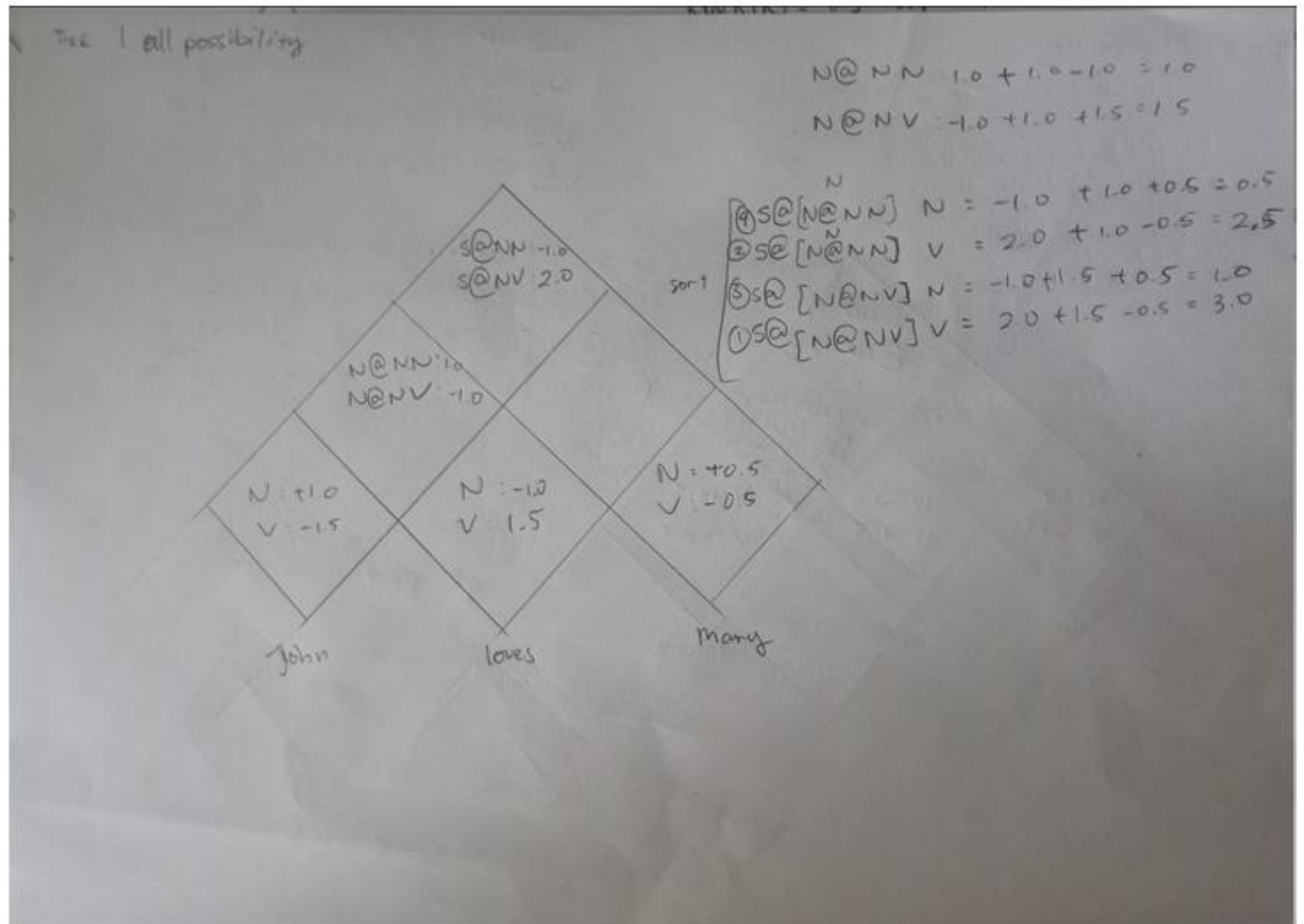
$$S@[N@NV]V$$

where John would have the tag N, loves would have the tag V, and Mary would have the tag V.

### Part 3

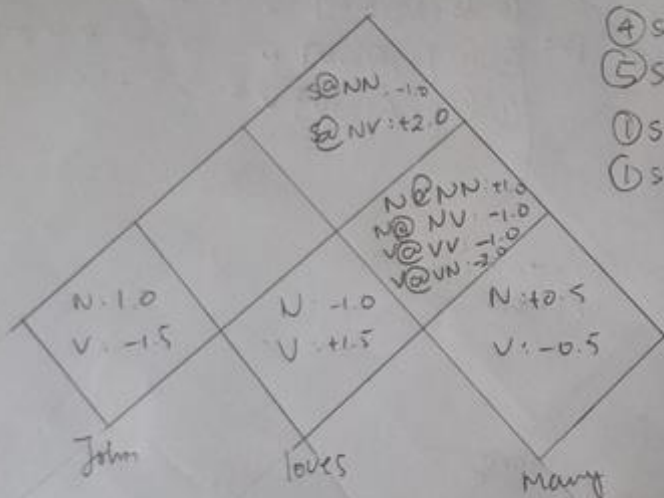
To find the 4th best score for John loves Mary, we need to modify the CKY algorithm such that it records all the possible values that the tree can make and then save it. filter it out such that there is only  $n$  best possible values left to consider and then take the  $n$ th best possible combination. This would mean that at each step record all the possible combinations of score, sort them out, then take only the  $n$  best scores that would be needed for the criteria.

4th Best score for John loves Mary ( $n = 4$ ):



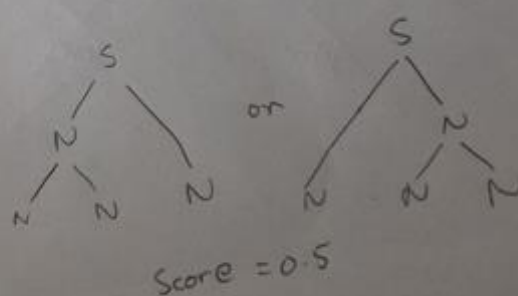
Tree 2 All possibility

$$\begin{aligned}
 N@NN &= 1.0 + (-1.0) + 0.5 = 0.5 \\
 N@NV &= -1.0 - 1.0 - 0.5 = -2.5 \\
 V@VV &= 1.5 - 1.0 - 0.5 = 0 \\
 V@VN &= -2.0 + 1.5 + 0.5 = 0
 \end{aligned}$$



④  $S@N[N@NN] \Rightarrow$  we have already calc for this  
 ⑤  $S@N[N@NV] = -1.0 + 1.0 - 2.5 = -2.5$   
 ①  $S@N[V@VV] = 2.0 + 1.0 + 0 = 3.0$   
 ①  $S@N[V@VN] = 2.0 + 1.0 + 0 = 3.0$

4th Best combination

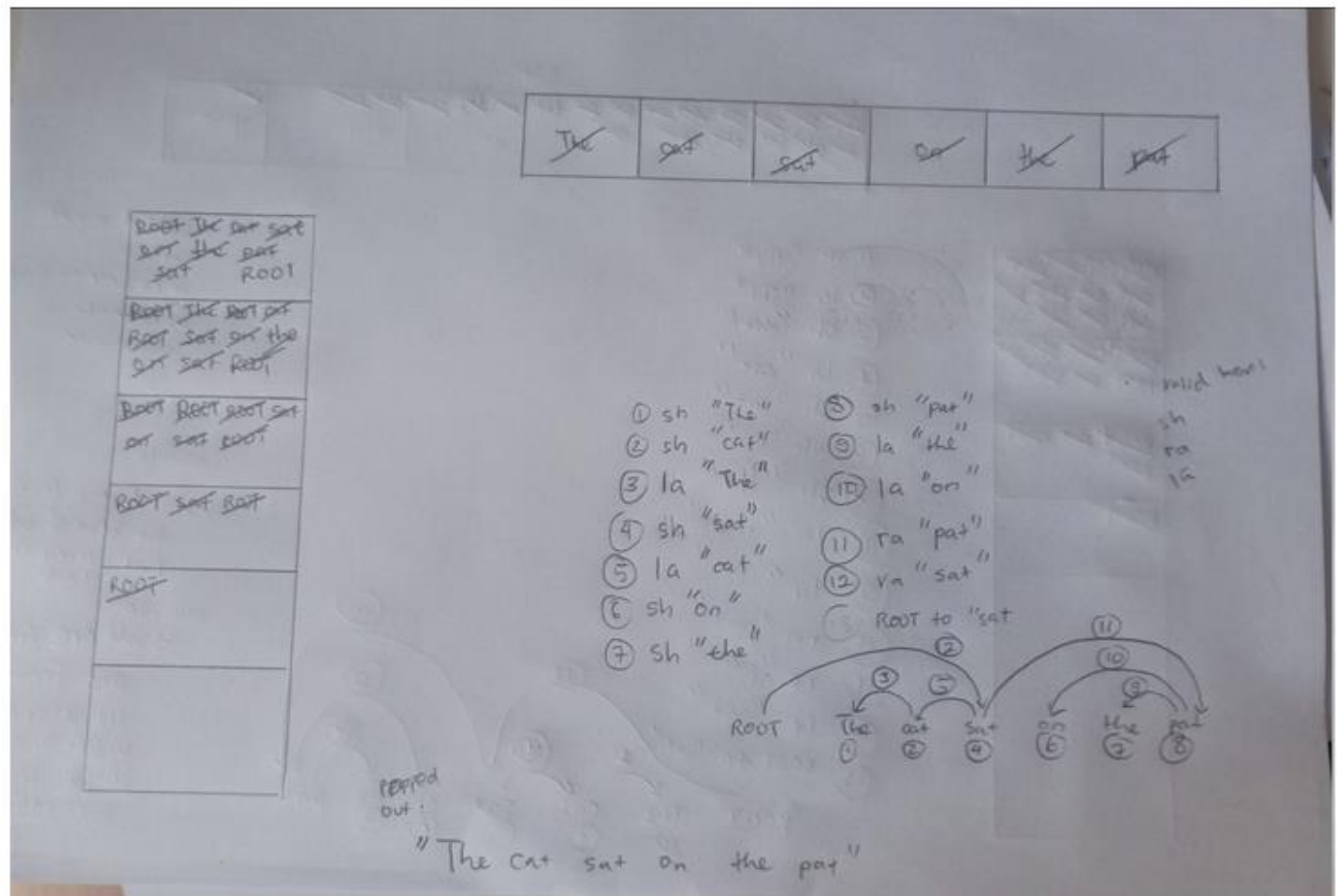


$S@[N@NN]NorS@N[N@NN]$

4th best score = 0.5

## 4. Transition-Based Parsing

### Part 1



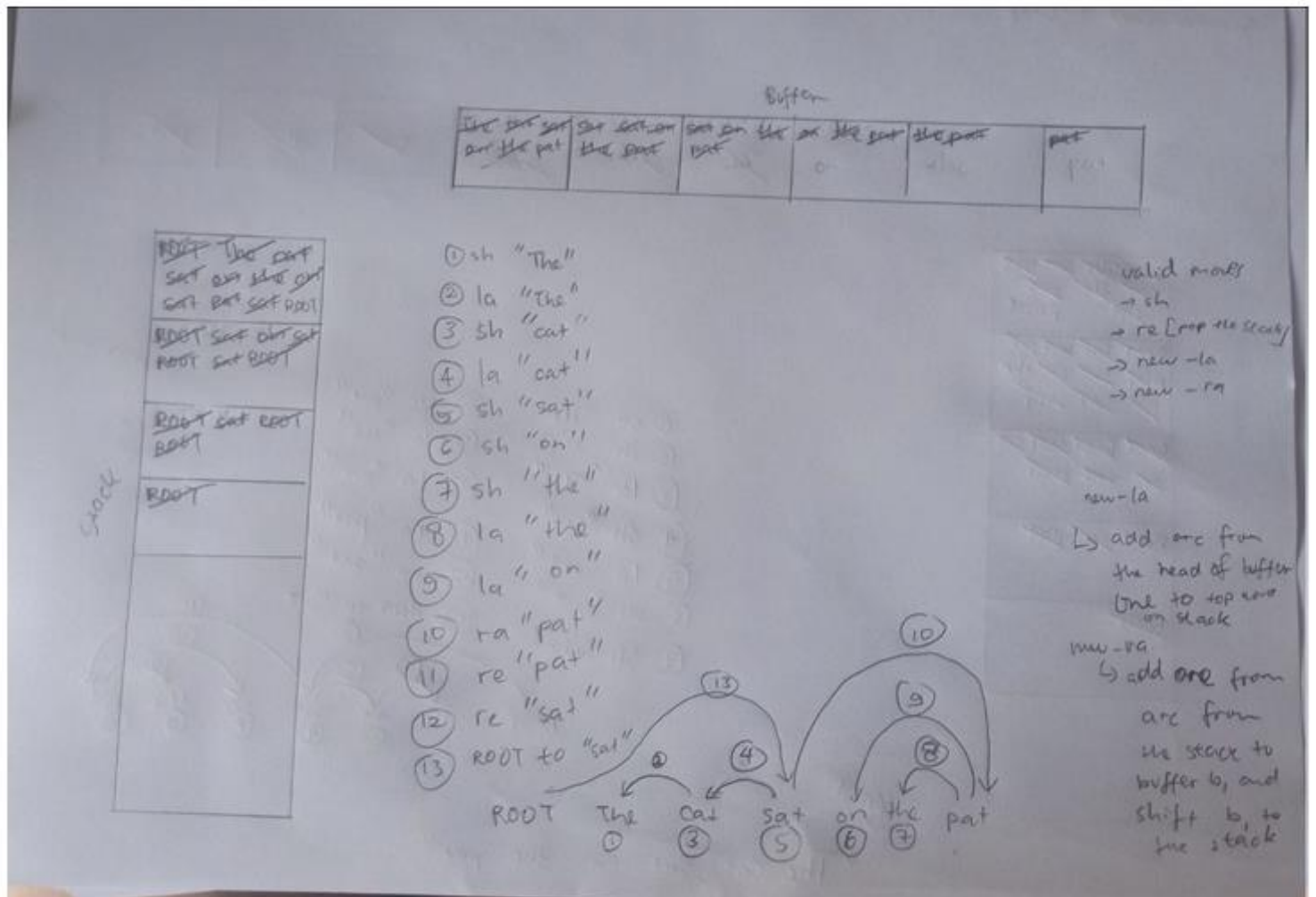
### Part 2

The worst time-complexity is

$$O(n)$$

This is because the transition based parsing at worst case only needs to go through the whole sentence once to move it from buffer through stack  $[O(n)]$  and removing the whole sentence from the stack, at worst case would take  $O(n)$  time complexity.

### Part 3



### Part 4

The worst time complexity is

$$O(n)$$

Similar to the standard approach it takes in  $O(n)$  complexity to move the whole sentence from buffer and popping requires  $O(n)$  to remove everything from the stack. Nothing seems to indicate that there will be a loop to keep checking for the whole sentence to parse the sentence and hence it is  $O(n)$  time complexity at worst.