ELEC-E5510 – Speech Recognition
Assignment 3

Nguyen Xuan Binh, 887799

15/11/2023

Part 1: N-gram language models

Question 1.1

Fetch the 1-gram counts again and compute the maximum likelihood estimates for the following 1-gram probabilities by hand (you can use a calculator):

- **P(in)**
- **P(a)**
- **P(</s>) (end of sentence symbol)**

When I run this command, I obtain the 1-gram counts of the train.txt file
$ ngram-count -order 1 -text train.txt

The reported counts for each unigrams are reported as

<s>	the	box	is	in	bag	</s>	it	a	on
3	4	3	3	3	2	3	2	1	1

SRILM also reports the count of <s>. We need to ignore it and don't include it in the counts since <s> is always assumed in the beginning of the sentence and is never generated later. Therefore, the total number of counts is

$$\sum_j C\left(w_i\right) = 4 + 3 + 3 + 3 + 2 + 3 + 1 + 1 = 22$$

The maximum likelihood estimation (MLE) of the 1-gram can be computed as

$$P\left(w_i\right) = \frac{C\left(w_i\right)}{\sum_j C\left(w_j\right)}$$

The maximum likelihood estimates for the following 1-gram are:

$$P(in) = \frac{C(in)}{\sum_j C(w_j)} = \frac{3}{22} = 0.136$$

$$P(a) = \frac{C(a)}{\sum_j C(w_j)} = \frac{1}{22} = 0.045$$

$$P(< /s >) = \frac{C(</s>)}{\sum_j C(w_j)} = \frac{3}{22} = 0.136$$

Question 1.2

Use ngram-count to get necessary counts and compute the following 2-gram and 3-gram estimates (maximum likelihood) below by hand. Note that the notation P(bag | in the) means the probability that the word "bag" appears after "in the" (for example in the sentence "in the bag"). The variable X in the equations below may represent any word in the vocabulary (there are 9 words excluding <s>). This means you need to report even n-grams that do not appear in our corpus.

P(X | is) for all X
P(X | in the) for all X

We will now calculate the maximum likelihood estimate for 2-grams using:

$$P(w_i \mid w_j) = \frac{C(w_j, w_i)}{C(w_j)}$$

and similarly for 3-grams using:

$$P(w_i \mid w_j, w_k) = \frac{C(w_j, w_k, w_i)}{C(w_j, w_k)}$$

From the train.txt file of Question 1.1

```
<s> the box is in the bag </s>
<s> it is in a bag on the box </s>
<s> is the box in it </s>
```

We can observe that the only words following ”is” are the words ”in” and ”the”. As a result, all other words will have MLE = 0 with respect to “is” in bigram model

$$P(box|is) = P(is|is) = P(bag|is) = P(it|is) = P(a|is) = P(on|is) = P(< /s > |is) = 0$$

We can also confirm the count by using the command

$ ngram-count -order 2 -text train.txt | grep "^is"

```
nguyenb5@kosh ~/SR_ex3 % ngram-count -order 2 -text train.txt | grep "^is"
is      3
is in   2
is the  1
```

The MLE of the two rest cases are calculated as follows:

$$P(in|is) = \frac{C(is\,,in)}{C(is)} = \frac{2}{3} = 0.67$$

$$P(the|is) = \frac{C(is\,,the)}{C(is)} = \frac{1}{3} = 0.33$$

For the trigram model, we can similarly use this command for the count

All the 3 grams that start with ”in the”

According to Figure 2, the only word following ”in the” is ”bag”. Thus:
$ ngram-count -order 3 -text train.txt | grep "^in the"

```
nguyenb5@kosh ~/SR_ex3 % ngram-count -order 3 -text train.txt | grep "^in the"
in the  1
in the bag      1
```

As a result, all other words except bag has MLE = 0 in the trigram model given “in the”

$$P(the|in\,the) = P(box|in\,the) = P(is|in\,the) = P(in|in\,the) = P(it|in\,the) = P(a|in\,the) = P(on|in\,the) = P(< /s > |in\,the) = 0$$

The MLE of the word “bag” in the trigram model is

$$P(bag|in\,the) = \frac{C(in\,the\,,bag)}{C(in\,the)} = \frac{1}{1} = 1$$

Question 1.3

Get the 2-gram counts again.

a) Using interpolated absolute discounting (D=0.5) and compute P(in | is) and P (</s> | is) by hand.

The idea of smoothing is to give some probability mass to n-grams that have not occurred in the training data. Thus, probability mass has to be taken from n-grams that occur in the training data. There are several smoothing methods, but here we consider absolute discounting with interpolation. The equation for the 2-gram probability then becomes

$$P\left(w_i \mid w_j\right) = \frac{max\left(0, C\left(w_j, w_i\right) - D\right)}{C\left(w_j\right)} + P\left(w_i\right)b\left(w_j\right)$$

where

$$b\left(w_j\right) = \frac{\#\, of\ different\ words\ appearing\ after\ w_j}{C\left(w_j\right)} D$$

- **Computation of P(in | is)**

In this case, we have:

- D = 0.5
- # of different words appearing after “is” are 2 (“in” and “the” from Question 1.1)
- C(in) = 3 from question 1.1
- C(is, in) = 2 from question 1.1

The back-off weight is therefore

$$b(is) = \frac{2}{3} \times 0.5 = \frac{1}{3}$$

Now we can calculate P(in | is)

$$P(in \mid is) = \frac{max(0, C(is, in) - 0.5)}{C(in)} + P(in)b(is)$$

$$= \frac{max(0, 2 - 0.5)}{3} + \frac{3}{22} \times \frac{1}{3} = \frac{1.5}{3} + \frac{1}{22} = \frac{6}{11} = 0.545$$

- **Computation of P(</s> | is)**

In this case, we have:

- $b(is) = 0.3333$ (result above)
- C(is, </s>) = 0 from question 1.1

Now we can calculate P(</s> | is)

$$P(< /s > \mid is) = \frac{max(0, C(</s>, is) - 0.5)}{C(is)} + P(< /s >)b(is)$$

$$= \frac{max(0, 0 - 0.5)}{3} + \frac{3}{22} \times \frac{1}{3} = 0 + \frac{1}{22} = \frac{1}{22} = 0.04545$$

Now I run the testing commands to check the probabilities for each n-gram separately

**$ ngram-count -order 2 -interpolate -cdiscount1 0 -cdiscount2 0.5 **
-text train.txt -lm 2gram.lm

$ ngram -lm 2gram.lm -ppl test.txt -debug 2

```
p( in | is ...)         = [2gram] 0.545455 [ -0.263241 ]
```

```
p( </s> | is ...)       = [1gram] 0.0454546 [ -1.34242 ]
```

We can see that SRILM has produced the same results as the hand calculations.

b) Compare the results you got in Question 2. What has changed? Why?

Before smoothing, P(in|is) = 0.67 and P(< /s > |is) = 0
After smoothing, P(in|is) = 0.545 and P(< /s > |is) = 0.045.

The probability P(in | is) = 0.545 is reduced from 0.67. This decrease occurs because absolute discounting reduces the probability of observed n-grams to give some of that probability mass to unseen n-grams. On the other hand, the probability P(< /s > | is) = 0.045 is now greater than 0, which indicates that after smoothing, there is now a non-zero chance of the sentence ending after the word "is." This probability is assigned based on the discounting from other more frequent n-grams.

The interpolation part of the smoothing uses a weighted average of the higher-order model (0.5 for bigram) and the lower-order model (unigram model). The discount D is subtracted from the observed counts to reserve some probability mass for unseen n-grams, and the back-off weight $b(w_j)$ determines how much of this reserved mass to distribute to a particular unseen n-gram based on the lower-order model probabilities.

The main reason for these changes is to address the zero-probability problem for unseen events. In real-world language, just because a particular word sequence hasn't been observed in a training corpus doesn't mean it's impossible. Smoothing techniques, such as interpolated absolute discounting, redistribute some probability mass to these unseen events, ensuring that the model can handle rarer bigrams/trigrams.

Question 1.4

a) What are the log-probabilities of the above sentences?

**$ ngram-count -order 2 -interpolate -cdiscount1 0 -cdiscount2 0.5 **
-text train.txt -lm 2gram.lm

```
nguyenb5@kosh ~/SR_ex3 % ngram -lm 2gram.lm -ppl test.txt -debug 2
reading 10 1-grams
```

The reported log-probabilities on the test file are:

```
<s> it is in the bag </s>
        p( it | <s> )   = [2gram] 0.212121 [ -0.673416 ]
        p( is | it ...)         = [2gram] 0.318182 [ -0.497325 ]
        p( in | is ...)         = [2gram] 0.545455 [ -0.263241 ]
        p( the | in ...)        = [2gram] 0.257576 [ -0.589095 ]
        p( bag | the ...)       = [2gram] 0.147727 [ -0.830539 ]
        p( </s> | bag ...)      = [2gram] 0.318182 [ -0.497325 ]
1 sentences, 5 words, 0 OOVs
0 zeroprobs, logprob= -3.35094 ppl= 3.61818 ppl1= 4.67938
```

log P(<s> it is in the bag </s>) = **-3.35**

```
<s> it is in the box </s>
        p( it | <s> )   = [2gram] 0.212121 [ -0.673416 ]
        p( is | it ...)         = [2gram] 0.318182 [ -0.497325 ]
        p( in | is ...)         = [2gram] 0.545455 [ -0.263241 ]
        p( the | in ...)        = [2gram] 0.257576 [ -0.589095 ]
        p( box | the ...)       = [2gram] 0.659091 [ -0.181055 ]
        p( </s> | box ...)      = [2gram] 0.234848 [ -0.629212 ]
1 sentences, 5 words, 0 OOVs
0 zeroprobs, logprob= -2.83334 ppl= 2.96636 ppl1= 3.68696
```

log P(<s> it is in the box </s>) = **-2.83**

```
<s> it is </s>
        p( it | <s> )   = [2gram] 0.212121 [ -0.673416 ]
        p( is | it ...)         = [2gram] 0.318182 [ -0.497325 ]
        p( </s> | is ...)       = [1gram] 0.0454546 [ -1.34242 ]
1 sentences, 2 words, 0 OOVs
0 zeroprobs, logprob= -2.51316 ppl= 6.8821 ppl1= 18.0543
```

log P(<s> it is </s>) = **-2.51**

b) Which sentence is the most probable one according to the model?

The sentence is the most probable if it has the highest log probability
Therefore, according to the model, the most probable sentence is

<s> it is </s>

c) Give an example of a sentence (non-empty, no out-of-vocabulary words) whose probability is even higher than any of the above.

I create a custom shell script that generates 10 sentences, then they are recorded into a temporary file in order to calculate their probabilities. Because ngram does not have the feature of limiting the word, I truncate all random sentences to retain at most 3 words.

```
#!/bin/bash

touch temp_sentences.txt
touch temp_test.txt
touch log_results.txt

# Remove all contents from the two text files
echo -n "" > temp_sentences.txt
echo -n "" > temp_test.txt
echo -n "" > log_results.txt

# Generate 10 sentences and save them to a temporary file
ngram -lm 2gram.lm -gen 10 > temp_sentences.txt

# Process each sentence
while IFS= read -r line; do
    # Truncate to the first 5 words plus the end-of-sentence token
      truncated_sentence=$(echo $line | awk '{print "<s> " $1, $2, $3 "
</s>"}')
    echo "Truncated Sentence: $truncated_sentence"

      # Write the truncated sentence to a temporary file for probability
calculation
    echo $truncated_sentence >> temp_test.txt

done < temp_sentences.txt

# Calculate the probability of the truncated sentence
ngram -lm 2gram.lm -ppl temp_test.txt -debug 2 > log_results.txt
```

Some of the random sentences that has higher log probability than"it is" are "the box" and "it"

<s> the box </s>
p(the | <s>) = [2gram] 0.257576 [-0.589095]
p(box | the ...) = [2gram] 0.659091 [-0.181055]
p(</s> | box ...) = [2gram] 0.234848 [-0.629212]
1 sentences, 2 words, 0 OOVs
0 zeroprobs, logprob= -1.39936 ppl= 2.92721 ppl1= 5.00819

<s> it </s>
p(it | <s>) = [2gram] 0.212121 [-0.673416]
p(</s> | it ...) = [2gram] 0.318182 [-0.497325]
1 sentences, 1 words, 0 OOVs
0 zeroprobs, logprob= -1.17074 ppl= 3.8492 ppl1= 14.8163

Question 1.5

The file /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz contains 10000 sentences that are not included in the training data. This test data can be used for evaluating the models that you just created. Use the ngram tool to compute the log-probability of the test data for each model (omit the -debug flag to avoid excess output).

First, we are going to make a language model that allows the most common 60000 words from the corpus. We can use ngram-count to compute the 1-gram counts from the corpus by running

$ ngram-count -order 1 -text /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz -write countsfile_1gram.txt

```
812514  hopeaketun  4
812515  huipputapaamista    25
812516  sarjavoittajat  2
812517  aiheille    1
812518  kolmenlaisia    5
812519  kurpitsapää 1
```

Now, we sort the words by the counts and select only the 60000 most common words (the corpus actually contains over 800000 different word forms from above figure) by running

$ sort -n -r -k 2 countsfile_1gram.txt | head -n 60000 | cut -f 1 > 60000.words

```
59996   ajelehtimaan
59997   ajautuminen
59998   ajatuksissa
59999   ajatuksella
60000   ajattelutapa
```

Now, we proceed to create the 1-gram, 2-gram, 3-gram model using Kneser-Ney smoothing, including only the most common 60000 words

**$ ngram-count -order 1 -vocab 60000.words **
**-interpolate -text /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz **
-lm 1gram.lm.gz

**$ ngram-count -order 2 -vocab 60000.words -kndiscount1 -kndiscount2 **
**-interpolate -text /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz **
-lm 2gram.lm.gz

**$ ngram-count -order 3 -vocab 60000.words -kndiscount1 -kndiscount2 -kndiscount3 **
**-interpolate -text /work/courses/T/S/89/5150/general/data/stt/stt.train.txt.utf8.gz **
-lm 3gram.lm.gz

After training, three ngram.lm.gz files are generated in the same directory

```
1gram.lm.gz
2gram.lm
2gram.lm.gz
3gram.lm.gz
```

Finally, we run these commands for testing the three trained models above

$ ngram -lm 1gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz
$ ngram -lm 2gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz
$ ngram -lm 3gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz

```
nguyenb5@kosh ~/SR_ex3 % ngram -lm 1gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz: 9987 sentences, 91849 words, 27097 OOVs
0 zeroprobs, logprob= -264042 ppl= 3410.81 ppl1= 11960.4
nguyenb5@kosh ~/SR_ex3 % ngram -lm 2gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz: 9987 sentences, 91849 words, 27097 OOVs
0 zeroprobs, logprob= -237930 ppl= 1525.72 ppl1= 4725.81
nguyenb5@kosh ~/SR_ex3 % ngram -lm 3gram.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.txt.utf8.gz: 9987 sentences, 91849 words, 27097 OOVs
0 zeroprobs, logprob= -235817 ppl= 1429.6 ppl1= 4383.86
```

Using the ngram tool to compute the log-probability of the test data for each model, I collected the results in the table below

model	logprob	words	OOVs
1-gram	-264042	91849	27097
2-gram	-237930	91849	27097
3-gram	-235817	91849	27097

a) Which of the models gave the best probability for the test data?

From the logprob column, the probability is extremely small, which is essentially 0 chance for the testing data because the testing corpus is very large and the chance of generating it is analogous to the "monkey typing Hamlet". However, if comparing the magnitudes, the 3-gram model results in the best probability for the test data.

b) What is the proportion of out-of-vocabulary (OOV) words in the test data (the ngram tool prints the relevant information for this)?

We can see in the reported table above. In all 3 models, there are 27097 OOV words and 91849 words. Therefore, the proportion of OOV words in the test data is 27097/91849 = **29.5%**

Question 1.6

Compute the log-probabilities for the morphed test data as above using the morph 1-gram, 2-gram and 3-gram models. Note that you must use the morphed version of the test data:
/work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz. Otherwise the units of the model and the data do not match and you get lots of OOV words.

We can compute the log-probabilities for the morphed test data as above using the morph 1-gram, 2-gram and 3-gram models with these commands

**$ ngram-count -order 1 -interpolate -text **
/work/courses/T/S/89/5150/general/data/stt/stt.train.mrf.utf8.gz -lm 1gram.mrf.lm.gz

**$ ngram-count -order 2 -kndiscount1 -kndiscount2 -interpolate -text **
/work/courses/T/S/89/5150/general/data/stt/stt.train.mrf.utf8.gz -lm 2gram.mrf.lm.gz

**$ ngram-count -order 3 -kndiscount1 -kndiscount2 -kndiscount3 -interpolate -text **
/work/courses/T/S/89/5150/general/data/stt/stt.train.mrf.utf8.gz -lm 3gram.mrf.lm.gz

Finally, we run these commands for testing the three trained models above

$ ngram -lm 1gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz
$ ngram -lm 2gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz
$ ngram -lm 3gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz

```
nguyenb5@kosh ~/SR_ex3 % ngram -lm 1gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz: 9987 sentences, 186121 words, 0 OOVs
0 zeroprobs, logprob= -662489 ppl= 2388.84 ppl1= 3626.23
nguyenb5@kosh ~/SR_ex3 % ngram -lm 2gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz: 9987 sentences, 186121 words, 0 OOVs
0 zeroprobs, logprob= -503229 ppl= 368.196 ppl1= 505.558
nguyenb5@kosh ~/SR_ex3 % ngram -lm 3gram.mrf.lm.gz -ppl /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz
file /work/courses/T/S/89/5150/general/data/stt/stt.eval.mrf.utf8.gz: 9987 sentences, 186121 words, 0 OOVs
0 zeroprobs, logprob= -465729 ppl= 237.06 ppl1= 317.899
```

Using the ngram tool to compute the log-probability of the test data for each morphed model, I collected the results in the table below

model	logprob	words	OOVs
morphed 1-gram	-662489	186121	0
morphed 2-gram	-503229	186121	0
morphed 3-gram	-465729	186121	0

a) Which of the models is the best one?

Compared to Question 1.5, these 3 morphed models have even much exponentially lower probability. Out of the 3 morphed models, the 3-gram is the best one as it has the highest log-probability of -465729.

b) What is now the number of OOV morphs (the tool talks about words since it knows nothing about morphs)?

For the 3 morphed models, the OOV morphs are all 0.

Part 2: Neural Language Models

Task 2.1

Starting with a seed history, predict the next word. Use this predicted word as a part of the history for the next prediction. Keep predicting until you hit the end of the sentence token. Report the word sequences you'll get, along with the softmax layer output that each word got. You should return something like:
seed = "oops i"
did (0.9)
it(0.8)
again(0.9)
The seed history for FFNN is "a girl".
The seed history for RNN is "<s> a girl".

- For FFNL language model:

- The seed history for FFNN: "a girl"

```
# get the probability distribution for every word in vocabulary to follow "a girl"
with torch.no_grad():
    # Question 2.1
    # prediction = ffnn_model(['a', 'girl']) # likes is next probable word
    # prediction = ffnn_model(['girl', 'likes']) # eating is next probable word
    # prediction = ffnn_model(['likes', 'eating']) # by is next probable word
    # prediction = ffnn_model(['eating', 'by']) # itself is next probable word
    prediction = ffnn_model(['by', 'itself']) # </s> is next probable word
```

```
<s> 0.000000
a 0.000000
girl 0.000000
likes 0.994718
eating 0.000000
by 0.000000
herself 0.001884
</s> 0.000000
cat 0.000000
meat 0.000000
the 0.000000
fish 0.000000
itself 0.003398
```

```
<s> 0.000000
a 0.000000
girl 0.005493
likes 0.000000
eating 0.982447
by 0.000000
herself 0.000000
</s> 0.000471
cat 0.011590
meat 0.000000
the 0.000000
fish 0.000000
itself 0.000000
```

```
<s> 0.000208
a 0.000349
girl 0.005726
likes 0.000003
eating 0.000006
by 0.495222
herself 0.007952
</s> 0.002025
cat 0.010021
meat 0.226339
the 0.000387
fish 0.251268
itself 0.000494
```

```
<s> 0.000599
a 0.000178
girl 0.000287
likes 0.016776
eating 0.000288
by 0.004122
herself 0.431346
</s> 0.009951
cat 0.000229
meat 0.001846
the 0.000568
fish 0.001783
itself 0.532028
```

```
<s> 0.000190
a 0.000221
girl 0.001832
likes 0.000020
eating 0.004483
by 0.000895
herself 0.002450
</s> 0.982630
cat 0.003015
meat 0.000510
the 0.000208
fish 0.000537
itself 0.003009
```

1. likes: 0.994718
2. eating: 0.982447
3. by: 0.495222
4. itself : 0.532028
5. </s>: 0.982630

- For RNN language model:

- The seed history RNN: “<s> a girl”

We define the weights and bias as for this RNN

```
# set parameters to ours
rnn_model.word_embed.weight = torch.nn.Parameter(torch.tensor(rnn_E.T, dtype=torch.float32))

rnn_model.rnn.weight_ih_l0 = torch.nn.Parameter(torch.tensor(rnn_W_in.T, dtype=torch.float32))
rnn_model.rnn.bias_ih_l0 = torch.nn.Parameter(torch.tensor(rnn_bias_in))

rnn_model.rnn.weight_hh_l0 = torch.nn.Parameter(torch.tensor(rnn_W_rec.T, dtype=torch.float32))
rnn_model.rnn.bias_hh_l0 = torch.nn.Parameter(torch.tensor(rnn_bias_rec))

rnn_model.out.weight = torch.nn.Parameter(torch.tensor(rnn_O.T, dtype=torch.float32))
rnn_model.out.bias = torch.nn.Parameter(torch.tensor(rnn_O_bias))
```

```
# get the probability distribution for every word in vocabulary to follow "<s> a girl"
with torch.no_grad():
    # prediction = rnn_model(['<s>','a','girl']) # likes is next probable word
    # prediction = rnn_model(['a','girl','likes']) # eating is next probable word
    # prediction = rnn_model(['girl','likes','eating']) # by is next probable word
    # prediction = rnn_model(['likes','eating','by']) # itself is next probable word
    prediction = rnn_model(['eating','by','itself']) # </s> is next probable word
```

```
<s> 0.000000
a 0.000014
girl 0.000229
likes 0.744596
eating 0.000527
by 0.000000
herself 0.249815
</s> 0.000111
cat 0.000372
meat 0.000000
the 0.000019
fish 0.000000
itself 0.004317
```

```
<s> 0.000000
a 0.000000
girl 0.000028
likes 0.000000
eating 0.897813
by 0.000000
herself 0.014170
</s> 0.000031
cat 0.000124
meat 0.000000
the 0.000000
fish 0.000000
itself 0.087834
```

```
<s> 0.000000
a 0.000000
girl 0.000000
likes 0.000000
eating 0.000000
by 0.751575
herself 0.000001
</s> 0.091189
cat 0.000000
meat 0.041398
the 0.000000
fish 0.115837
itself 0.000000
```

```
<s> 0.000000
a 0.000000
girl 0.001200
likes 0.000041
eating 0.208393
by 0.000000
herself 0.382621
</s> 0.005025
cat 0.004181
meat 0.000000
the 0.000000
fish 0.000000
itself 0.398538
```

```
<s> 0.000001
a 0.000002
girl 0.005115
likes 0.000005
eating 0.000438
by 0.004417
herself 0.171473
</s> 0.718418
cat 0.016642
meat 0.000156
the 0.000003
fish 0.000416
itself 0.082914
```

1. likes: 0.744596
2. eating: 0.897813
3. by: 0.751575
4. itself : 0.398538
5. </s>: 0.718418

Task 2.2

Explain how and why the predictions from FFNN LM and RNN LM differ. How do both models differ from n-gram models?

Result from task 2.1

FFNN model
1. likes: 0.994718
2. eating: 0.982447
3. by: 0.495222
4. itself : 0.532028
5. </s>: 0.982630

RNN model
1. likes: 0.744596
2. eating: 0.897813
3. by: 0.751575
4. itself : 0.398538
5. </s>: 0.718418

We can observe that the prediction of “likes”, “eating” and “itself” have higher probability by FFNN than RNN. However, for “by” and ending tag “</s>”, RNN has higher probability than FFNN. We can notice that there is a competition for “herself” and “itself” prediction between the models. In RNN, herself - itself has the probability of 0.382 and 0.398 respectively, meaning RNN believes they are both equally probable, but this is not the case for FFNN. This is logical because after “by”, both herself and itself are possible. Therefore, I think RNN may perform slightly better than the FFNN model. RNN is also not entirely confident with ending tag, but FFNN is almost surely of the sentence’s end. This means RNN is considering a longer sentence

How do both models differ from n-gram models?

The difference between the n-gram models and both neural networks (NN) models is that NNs still maintain the word’s meanings in context. For example, when we have two words with the same meaning such as impact/effect, one can be more common than the other in a certain fixed phrase such as “have a significant impact“ is more common than “effect”. However, in N-gram, it considers words on their counts instead of meaning relationships such as the NN models. Additionally, even though N-gram has a smoothing strategy, it is still not as effective as NN, as in NN, the probability of infrequent grams does not vanish totally as they are calculated from the word vectors via embedding.