Assignment 2

Gengcong Yan - 1009903
ELEC-E5510 - Speech Recognition

November 23, 2021

1 Question 1

According to formula in introduction, we compute the maximum likelihood estimates for the following 1-gram probabilities:

$$\begin{aligned} P(in) &= 3/22 \\ P(a) &= 1/22 \\ P(< /s >) &= 3/22 \end{aligned} \tag{1}$$

2 Question 2

According to the similar computation above, we get the following 2-gram and 3-gram maximum likelihood estimates below by hand. V means the vocabularies in the model.

2-gram:

$$\begin{aligned} P(in|is) &= 2/3 \\ P(the|is) &= 1/3 \\ P(x|is) &= 0, x \in \{V \setminus (in, the)\} \end{aligned} \tag{2}$$

3-gram:

$$\begin{aligned} P(bag|inthe) &= 1 \\ P(x|inthe) &= 0, x \in \{V \setminus (bag)\} \end{aligned} \tag{3}$$

3 Question 3

3.1 A

After using interpolated absolute discounting (D=0.5) , we compute $P(in|is)$ and $P(< /s > |is)$ by hand. The results are the same with the results computed in the program later.

$$\begin{aligned} P(in|is) &= \frac{1.5}{3} + \frac{3}{22} * \frac{2}{3} = 12/22 \\ P(< /s > |is) &= \frac{0}{3} + \frac{3}{22} * \frac{2}{3} = 1/22 \end{aligned} \tag{4}$$

3.2 B

After we applied the smoothing methods, even if word sequences do not occurred in the training data, the maximum likelihood estimation won't give zero probabilities for them. This allows us to avoid extreme values such as positive and negative infinity when calculating. To keep a language model from assigning zero probability to these unseen words, we'll have to shave off a bit of probability mass from some more frequent events and give it to the words never seen.

4 Question 4

4.1 A

The log-probabilities of above sentences mean the probabilities of the whole sentence shows up in log form. In the original probability calculation the multiplication method was used, but since we have used the logarithmic form, here we are talking about the probability of all words added together to get the total probability of the whole sentences.

4.2 B

The log probabilities of 3 sentences in *test.txt* are **-3.35. -2.83 and -2.51**, respectively. So The most probable sentences according to the model is $< s > it - is < /s >$.

4.3 C

The shorter and simpler the sentence, the more likely it is. The log probabilities of $< s > it < /s >$ is -0.67 better than sentences above.

5 Question 5

5.1 A

The log probabilities of 3 models for the real Finnish model are **-264042. -237930 and -235817**, respectively. 3-gram model gives the best probability for the test data.

5.2 B

The proportion of out-of-vocabulary (OOV) words in the test data is $27097/91849 = \mathbf{0.295}$.

6 Question 6

6.1 A

In the morgh n-gram models, The log probabilities of 3 models for the real Finnish model are **-662489. -503229 and -465729**, respectively. Still, morgh 3-gram model gives the best probability for the test data.

6.2 B

Now in the morgh n-gram models, the proportion of out-of-vocabulary (OOV) words in the test data is $0/186121 = \mathbf{0}$.

7 Task2

7.1 Task 2.1

In FFNN model, the computation result shows below, the whole sentence is **a girl likes eating by itself**.

$$\begin{gathered} Seed = agirl \\ likes(0.9947) \\ eating(0.9824) \\ by(0.4952) \\ itself(0.5320) \\ < /s > (0.9826) \end{gathered} \tag{5}$$

In RNN model, the computation result shows below, the whole sentence is **a girl likes eating by herself**.

$$\begin{gathered} Seed = agirl \\ likes(0.7446) \\ eating(0.9364) \\ by(0.7516) \\ herself(0.4646) \\ < /s > (0.9991) \end{gathered} \tag{6}$$

The code of the compuation show below,

```
# initialize a model
# word embeddings are 2d
# rnn word history embeddings are 3d

rnn_model =  RNN(vocab_to_ind, 2, 3)

# 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"

words=['<s>','a','girl']

with torch.no_grad():
#     prediction = rnn_model(['<s>','a','girl'])
    prediction = rnn_model(words)
    pred=torch.argmax(prediction[-1])
    print(ind_to_word[int(pred)],prediction[-1][pred])

while ind_to_word[int(pred)]!="</s>":
    words.append(ind_to_word[int(pred)])
    with torch.no_grad():
        prediction = rnn_model(words)
        pred=torch.argmax(prediction[-1])
        print(ind_to_word[int(pred)],prediction[-1][pred])
```

7.2 Task 2.2

The differences between FFNN and RNN are the ways of handling historical information. By selecting words sequences of a specific length as input to the model, FFNN establishes internal relationships between words sequences within that length, but considers them to be independent of each other whenever they are beyond that length. This makes the computation easier and faster, but ignores the relationships between distant words. The RNN keeps the historical information from previous representations up to date and retains it, and uses it in the calculations in combination with the information from current words in neural networks.

The problem of n-gram language model is sparsity. Increasing n makes sparsity problems worse, typically we can not have n too bigger. Increasing n or increasing corpus increases model size. A fixed-window neural language model improves over n-gram language model. There is no sparsity problem and you do not need to store all observed n-grams. And RNN model can process any length input. Computation for step t can (in theory) use information from many steps back. Model size doesn't increase for longer input and same wights applied on every time step.