Homework 1

Problem 1

A) Write regular expressions.

1. Language 1:

/(\b[A-Za-z]+\b)\s\1/

(From the given examples, I use \s here to represent a break between two words. If we should consider other punctuations, we should change \s into other expression.)

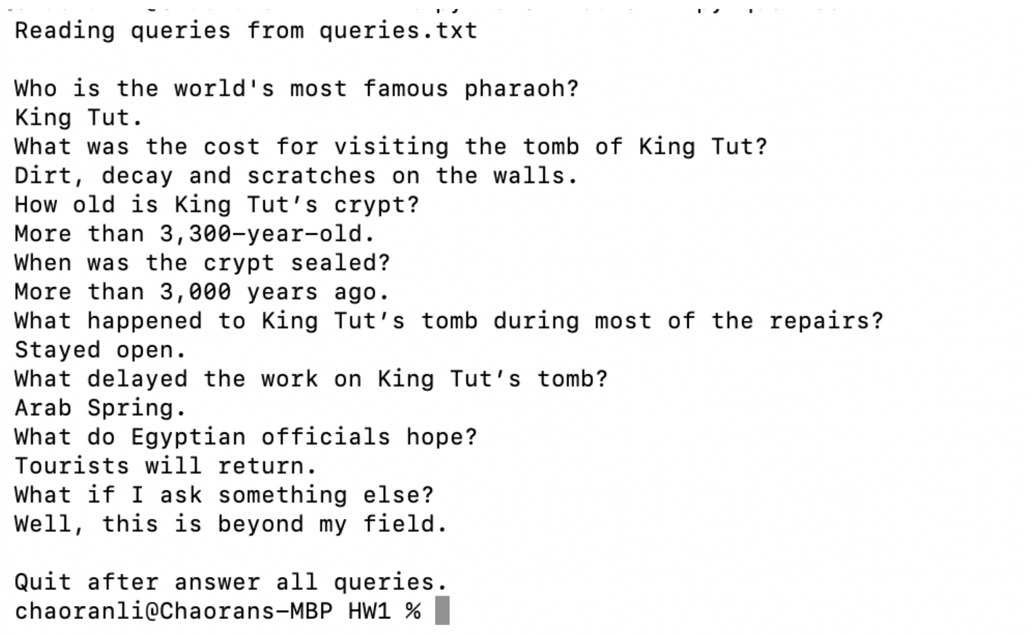
2. Language 2:

/(?=.*hedge)(?=.*fund)/

(From the given example of 'funds', I do not specify \b for hedge and fund.)

B) Produce a QA system.

Below is the running result for reading queries from a file.



```
Reading queries from queries.txt

Who is the world's most famous pharaoh?
King Tut.
What was the cost for visiting the tomb of King Tut?
Dirt, decay and scratches on the walls.
How old is King Tut's crypt?
More than 3,300-year-old.
When was the crypt sealed?
More than 3,000 years ago.
What happened to King Tut's tomb during most of the repairs?
Stayed open.
What delayed the work on King Tut's tomb?
Arab Spring.
What do Egyptian officials hope?
Tourists will return.
What if I ask something else?
Well, this is beyond my field.

Quit after answer all queries.
chaoranli@Chaorans-MBP HW1 %
```

It can also respond to console input.

Problem 2

S1: Sales of the company to return to normalcy.

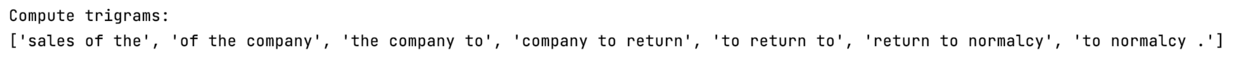
S2: The new products and services contributed to increase revenue.

A) Compute trigram probabilities.

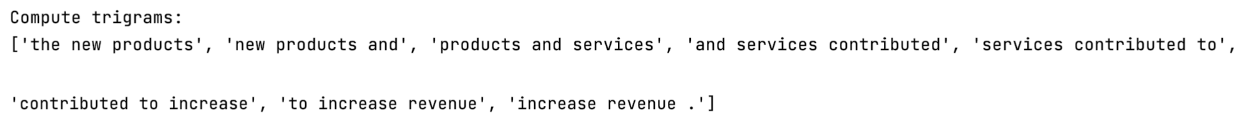
1. Compute the tragrams:

S1: ['sales of the', 'of the company', 'the company to', 'company to return', 'to return to', 'return to normalcy', 'to normalcy .']

S2: ['the new products', 'new products and', 'products and services', 'and services contributed', 'services contributed to', 'contributed to increase', 'to increase revenue', 'increase revenue .']



```
Compute trigrams:
['sales of the', 'of the company', 'the company to', 'company to return', 'to return to', 'return to normalcy', 'to normalcy .']
```



```
Compute trigrams:
['the new products', 'new products and', 'products and services', 'and services contributed', 'services contributed to',

'contributed to increase', 'to increase revenue', 'increase revenue .']
```

2. Compute trigram counts and probabilities without smoothing.

See related tables in /tables.

3. Compute trigram counts, probabilities and re-constituted counts with Laplace (add-one) smoothing.

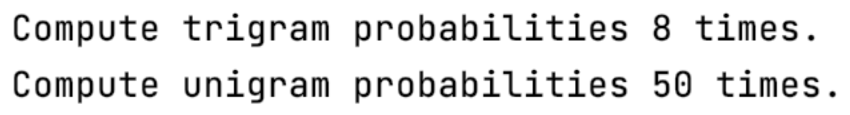
See related tables in /tables.

4. Compute trigram probabilities with Katz back-off smoothing.

See related tables in /tables.

For S1:

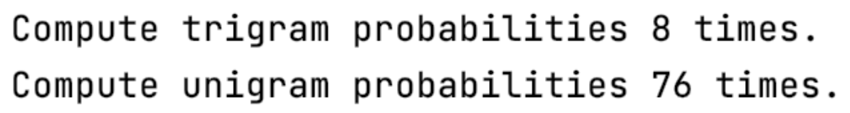
Running results:



```
Compute trigram probabilities 8 times.
Compute unigram probabilities 50 times.
```

For S2:

Running results:



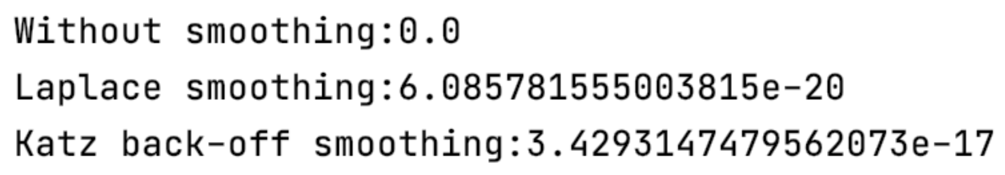
```
Compute trigram probabilities 8 times.
Compute unigram probabilities 76 times.
```

From my observation, all zero count in trigram level can all find non-zero count in bigram level. Hence, for the sentence we have, zero unigram probabilities are computed.

5. Compute three kind of total trigram probabilities.

For S1:

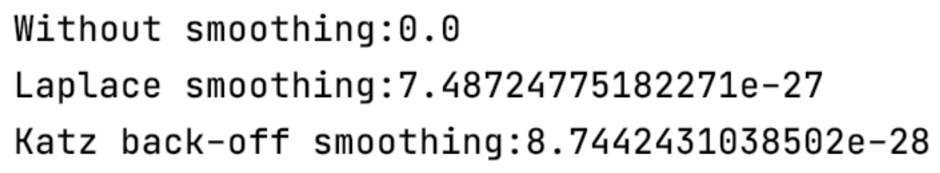
Running results:



```
Without smoothing:0.0
Laplace smoothing:6.085781555003815e-20
Katz back-off smoothing:3.4293147479562073e-17
```

For S2:

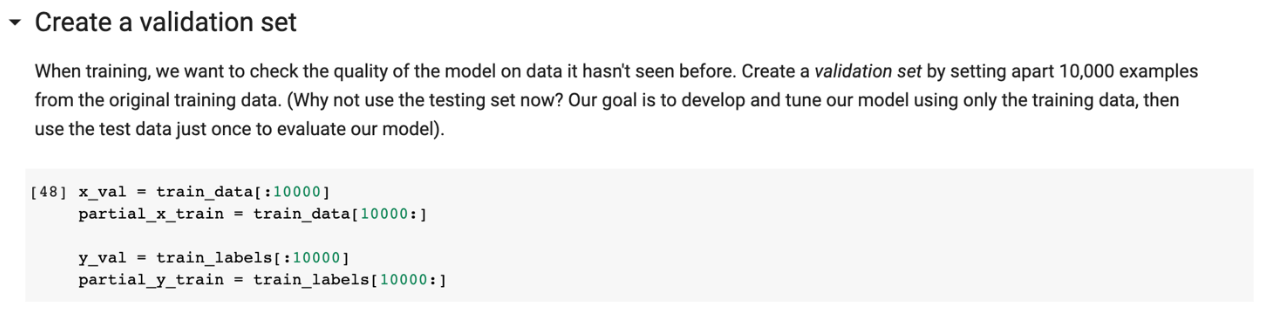
Running results:



```
Without smoothing:0.0
Laplace smoothing:7.48724775182271e-27
Katz back-off smoothing:8.7442431038502e-28
```

B) Learn a neural language model on Google cloud.

Actually, a validation set with size of 10000 has been given in the source code.



▾ Create a validation set

When training, we want to check the quality of the model on data it hasn't seen before. Create a *validation set* by setting apart 10,000 examples from the original training data. (Why not use the testing set now? Our goal is to develop and tune our model using only the training data, then use the test data just once to evaluate our model).

```
[48] x_val = train_data[:10000]
     partial_x_train = train_data[10000:]

     y_val = train_labels[:10000]
     partial_y_train = train_labels[10000:]
```

Hence, I use loss, perplexity and accuracy graphs to evaluate different models.

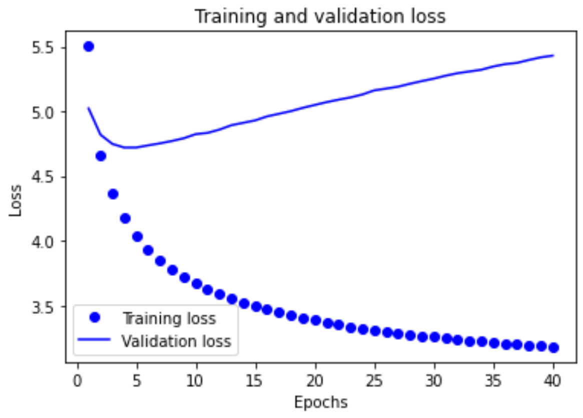
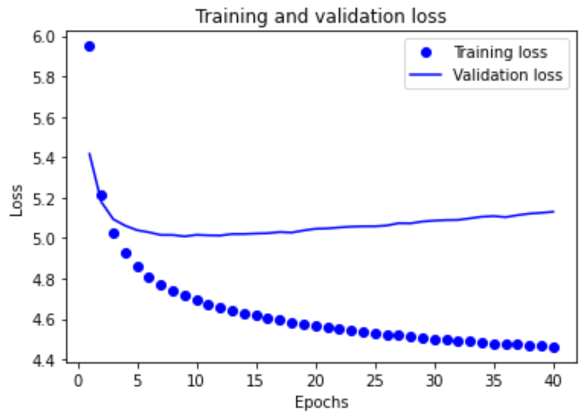
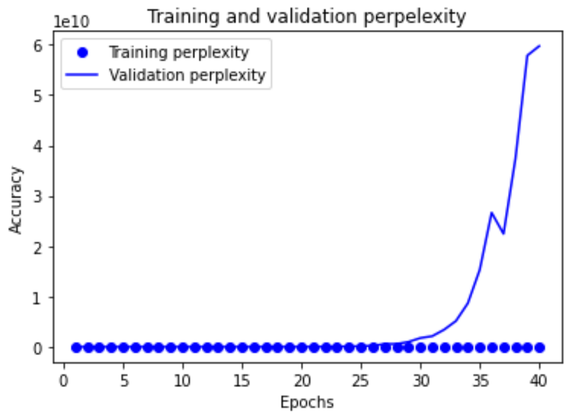
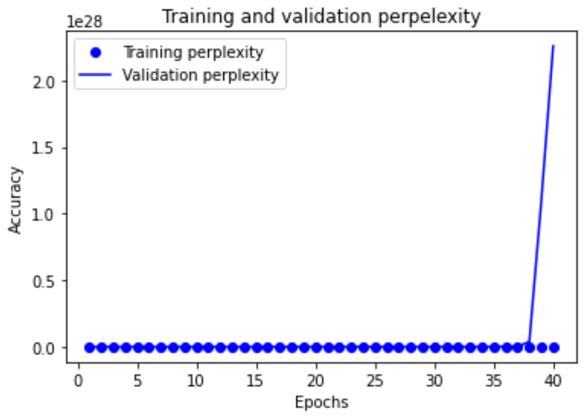
I use the same first two layers for all models:

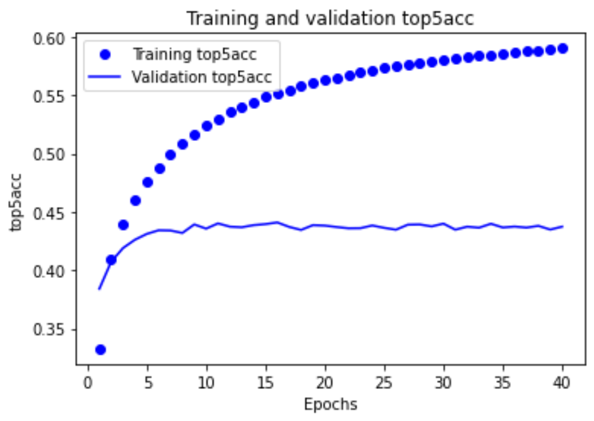
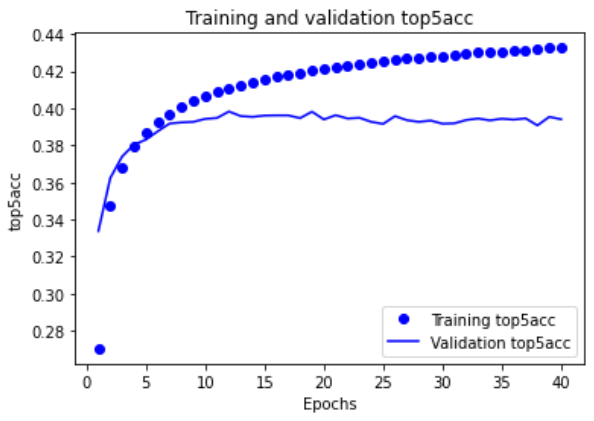
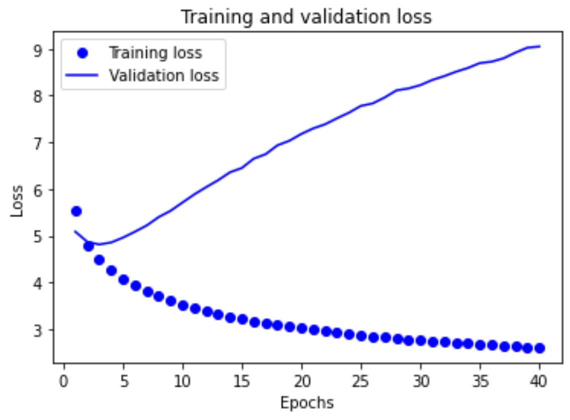
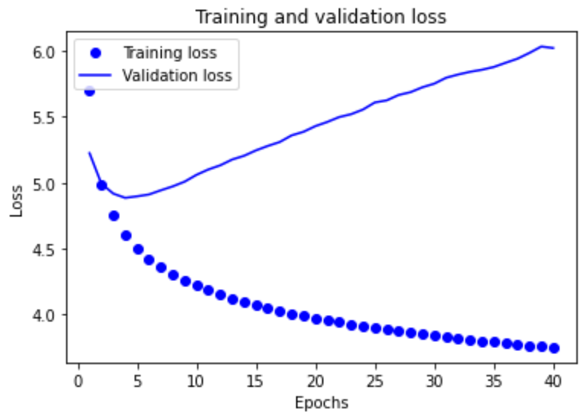
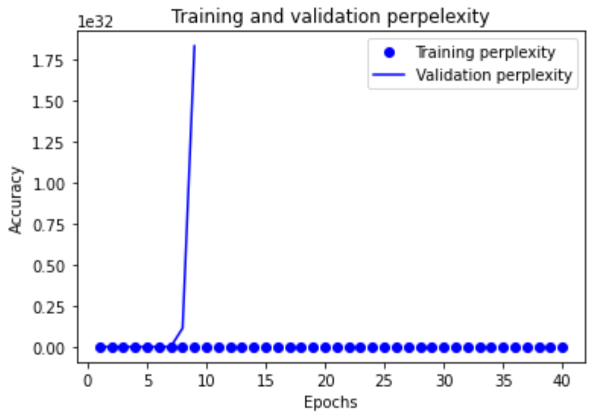
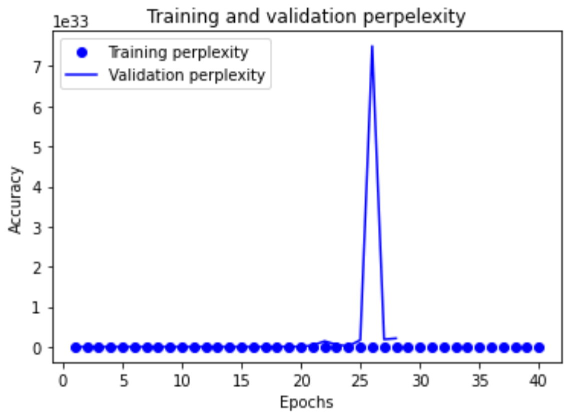
```
model = keras.Sequential()
model.add(keras.layers.Embedding(vocab_size, 50,
weights=[embedding_matrix]))
model.add(keras.layers.Reshape([50*N]))
```

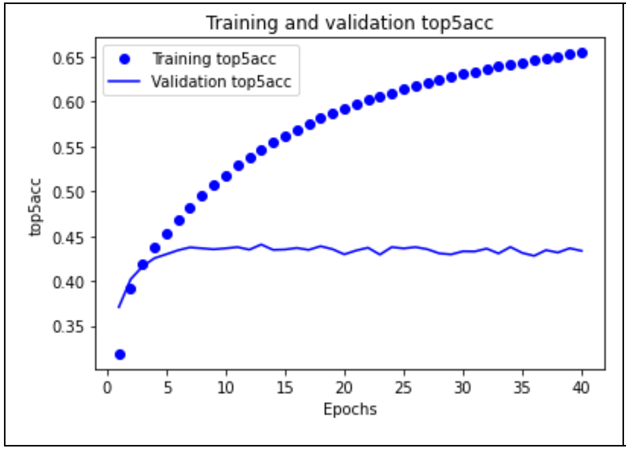
After this, each model have different layer settings:

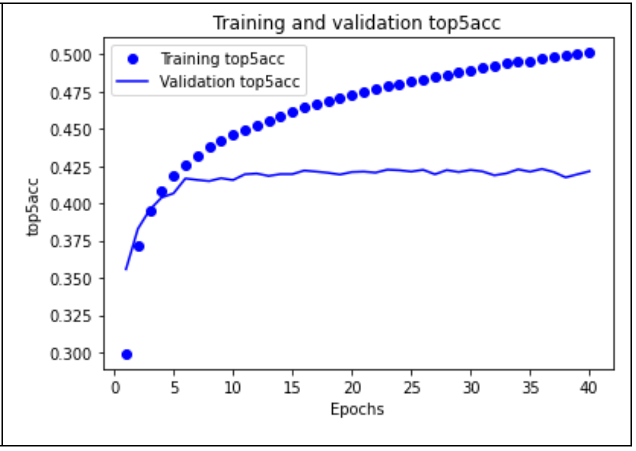
```
1 Intermediate Layer
model.add(keras.layers.Dense(vocab_size, activation=tf.nn.softmax))

2 Intermediate Layers
```

```
#model.add(keras.layers.Dense(16, activation=tf.nn.relu))
model.add(keras.layers.Dense(vocab_size, activation=tf.nn.softmax))

3 Intermediate Layers (a)
model.add(keras.layers.Dense(16, activation=tf.nn.relu))
model.add(keras.layers.Dense(1024, activation=tf.nn.relu))
model.add(keras.layers.Dense(vocab_size, activation=tf.nn.softmax))

3 Intermediate Layers (b)
model.add(keras.layers.Dense(16, activation=tf.nn.relu))
model.add(keras.layers.Dense(128, activation=tf.nn.relu))
model.add(keras.layers.Dense(vocab_size, activation=tf.nn.softmax))
```

1 intermediate layer	2 intermediate layers
	
	

	
3 intermediate layers (a)	3 intermediate layers (b)
	
	





To be honest, the given 2 intermediate layers model has pretty good performance in loss and perplexity. I tried many parameters and some of them can have higher accuracy like 3 intermediate layers model (a).

Problem 3

1. Compute the PPMI.

```
Part 1, compute the PPMI
Reading corpus from corpus_for_language_models.txt

Term-context matrix:
          said   of  board
chairman   110  312     15
company     21   55     26

Sum = 539

PPMIs:
PPMI('chairman', 'said') = 0.050588705239116516
PPMI('chairman', 'of') = 0.06841795927094263
PPMI('company', 'board') = 1.7446038343040187
PPMI('company', 'said') = 0.0
```

2. Compare the similarity.

```
Part 2, compute the similarity
Term-context matrix:
          said   of  board
chairman   110  312     15
company     21   55     26
sales        3    5      0
economy      1    4      0

Sum = 552

Similarities:
cos('chairman', 'company') = 0.9318204881744231
cos('company', 'sales') = 0.900683474458927
cos('company', 'economy') = 0.908212624797094
```

For the [chairman, company] has the biggest cosine value, they are the most similar among these pairs.

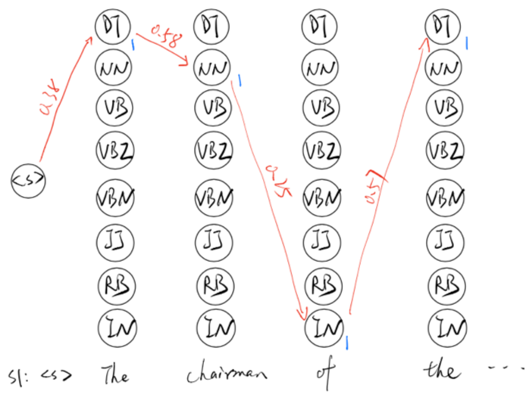
Problem 4

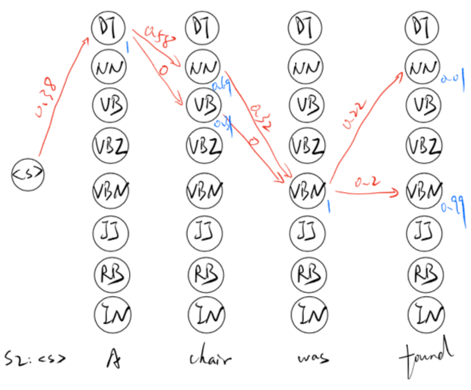
For sentences:

S1: The chairman of the board is completely bold.

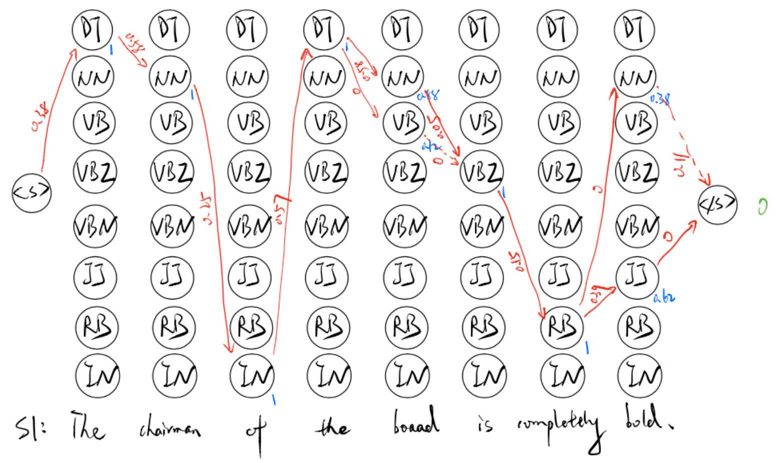
S2: A chair was found in the middle of the road.

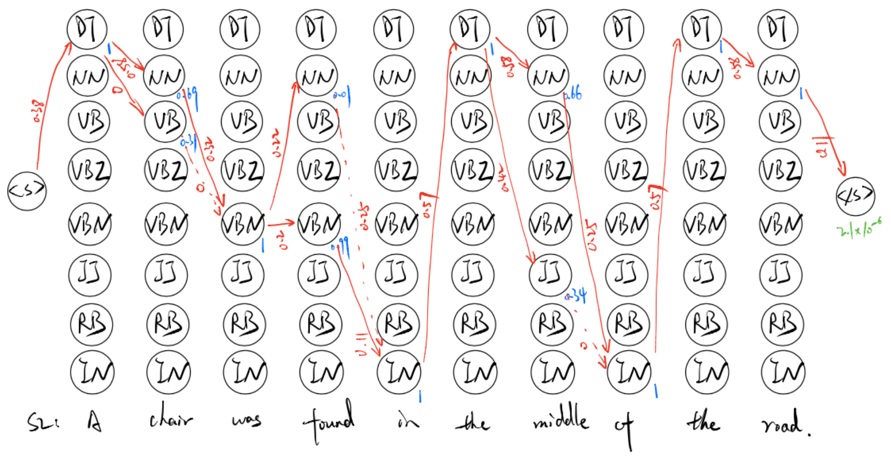
1. Present only the transition and observation likelihoods in the states reached after three steps.





2. Create the Viterbi table and populate it entirely.





3. Calculate the probability of assigning the tag sequence for each of the sentences.

For S1:

The transition probabilities from RB to NN and from JJ to </s> are all 0.

P(S1) = 0

This is strange. Because a sentence end with “bold” is quite normal. If the transition probabilities from JJ to </s> is not 0, we could tag S1 as:

S1: The/DT chairman/NN of/IN the/DT board/NN is/VBZ completely/RB bold/JJ.

For S2:

We can tag S2 as:

S2: A/DT chair/NN was/VBN found/VBN in/IN the/DT middle/NN of/IN the/DT road/NN.

P(S2) = 1 * 0.38 * 1 * 0.58 * 0.69 * 0.32 * 1 * 0.2 * 0.99 * 0.11 * 1 * 0.57 * 1 * 0.58 * 0.66 * 0.25 * 1 * 0.57 * 1 * 0.58 * 1 * 0.11 = 2.1E-6

4. Execute the Stanford POS-tagger.
Results:
S1: The_DT chairman_NN of_IN the_DT board_NN is_VBZ completely_RB bold_JJ ._.
S2: A_DT chair_NN was_VBD found_VBN in_IN the_DT middle_NN of_IN the_DT road_NN ._.
The result of S2 is more accurately, because it has a new tag named VBD.
Actually, the result of S1 is just the same as we get in part 3 if the transition probabilities from JJ to </s> is not 0. Like I said in part 3, it is strange and I think the data that Stanford POS-tagger use has some difference from ours.