Assignment 6
CSE 517: Natural Language Processing - University of Washington
Winter 2022

Solution 1.
In this problem, we are given with two text file one with a masked sentences and other with the bigram language model vocabulary. Our goal is to decode the sequences by substituting the masked character with a character from the bigram models vocabulary. To solve this task, I used the Viterbi algorithm, which is based on dynamic programming; this technique is used for efficiently finding the optimum state sequence.

I began by loading the two masked text files and the bigram vocabulary text file. After preproces sing the masked text file by reading and parsing the sentences, I create the bigram dictionary (usi ng the bigram vocabulary text file) by separating the word 1 given word 2 and its probability. I al so added an index to another dictionary that I created of unique words (keys) that appeared in the bigram vocabulary dictionary. The dictionary snippet for character 's>' is shown below as an exa mple.

```
<s>': {'<start>': 1.1345535078125355e-06,
               'T': 0.0031052729508829094,
               'h': 0.042082858711782564,
               '<s>': 1.1345535078125355e-06,
               :
               :
               '.': 0.04468665901221233,
               '<eos>': 1.1345535078125355e-06,
               'S': 0.0059870388607267495,
               :
               :
               '{': 2.269107015625071e-06,
               '}': 1.1345535078125355e-06},
```

Following this, I wrote a function to calculate the likelihood of words given word 1 and word 2 from my bigram dictionary. For this step, a code snippet is provided below.

```
def probability_of_bigram_words( word_1, word_2):
    if word_1 in biagram_dict.keys() and word_2 in biagram_dict[word_1].keys():
        return biagram_dict[word_1][word_2]
```

The next stage was to find the word 2 with the highest probability given a word 1 ($P(word\_2\ |word\_1) = ln(probability)$), and if the probability is infinite, return zero.

To implement the Viterbi algorithm, I referred to the Wikipedia resource which provides an example to implement the code.

In my Viterbi algorithm, I began by defining a function which takes arguments observation(sentences) (Y= ($y_1, y_2 \ldots y_k$}), space, ($y_n \in O = \{o_1, o_2 \ldots o_k\}$), state space ($x_n \in S = \{s_1, s_2 \ldots s_k\}$), bigram model.

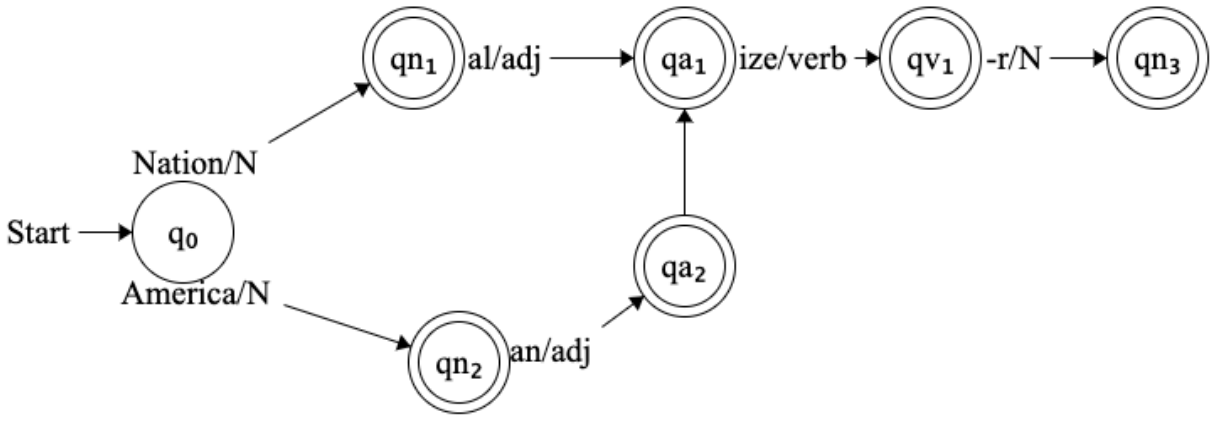
I defined trellis of possible outcomes by holding the probability of each state given each sentence and similarly to saving the back pointers that remembers each state. I initialize the START probability values and then run a loop to determine the probability of each hidden state.

In the loop, I took four different instances based on previous-word (w1) and current-word (w2) as mentioned below:

1) When both w1 and w2 are not masked, then probability will be max_prev_trellis_value + **probability of w2_given_w1**, and back pointer will be max_prev_trellis_label_idx
2) When w1 is not masked and w2 is masked, then probability will be max_prev_trellis_value + **probability of label_i_given_w1**, and back pointer will be max_prev_trellis_label_idx
3) When w1 is masked and w2 is not masked, then probability will be max of (**probability of w2_given_label_i-1 +** prev_trellis_value) among all labels, and back pointer will be argmax(**probability of w2_given_label_i-1 +** prev_trellis_value) among all labels
4) When both w1 and w2 are masked, then probability will be max of (**probability of label_i_given_label_i +** prev_trellis_value) among all labels, and back pointer will be argmax(**probability of label_i_given_label_i-1 +** prev_trellis_value) among all labels

Finally, I fill the probabilities for <eos> and back pointer as argmax of prev_trellis_value. Once I have all the back pointers filled, I simply traverse backward to construct the unmask sentence.

Solution 2.



Solution 3.
Part 1: Solution by hand attached

Part 2:

A) Space: O (nL)
- To fill in the data structure.
- The table is with O(n) columns and O(L) rows.

B) Runtime: O (nL^3)
- O (L^2): We need to determine max /argmax over label set L for each cell, which is of order 2.
- There are O(nL) cells.

References:

[1] https://en.wikipedia.org/wiki/Viterbi_algorithm
[2] Natural Language Processing by Jacob Eisenstein
[3] McKinney, Wes. "Pandas, python data analysis library." URL http://pandas. pydata. org (2015).
[4] https://www.audiolabs-erlangen.de/resources/MIR/FMP/C5/C5S3_Viterbi.html
[5] https://www.delftstack.com/howto/python/viterbi-algorithm-python/
[6] Ed discussion board for A3 Assignment and discord group.

Solution 3. Viterbi for second-Order Models

Second-order sequence model for triplets of labels, $s(x, i, y_i, y_{i+1}, y_{i+2})$

Given model,

$$\hat{y} = \underset{y \in \mathcal{L}^l}{\operatorname{argmax}} \sum_{i=0}^{l-1} s(x, i, y_i, y_{i+1}, y_{i+2})$$

Base case: $\heartsuit_2(y', y) = s(x, 0, \text{START}, y', y)$

For $i \in (3, \ldots l-1)$

$$\heartsuit_i(y', \hexagon) = \max_{y \in \mathcal{L}} s(x, l-1, y_{l-1}, y_l, \hexagon) + \heartsuit_{l-1}(y_{l-1}, y_l)$$

$$\heartsuit_{l-1}(y', y) = \max_{y_{l-2} \in \mathcal{L}} s(x, l-2, y_{l-2}, y_{l-1}, y_l) + \heartsuit_{l-2}(y_{l-2}, y_{l-1})$$

$$\heartsuit_{l-2}(y', y) = \max_{y_{l-3} \in \mathcal{L}} s(x, l-3, y_{l-3}, y_{l-2}, y_{l-1}) + \heartsuit_{l-3}(y_{l-3}, y_{l-2})$$

$$\vdots$$

$$\heartsuit_i(y', y) = \max_{y_{i-1} \in \mathcal{L}} s(x, i-1, y_{i-1}, y_i, y_{i+1}) + \heartsuit_{i-1}(y_{i-1}, y_i)$$

Hence,

$$\heartsuit_i(y', y) = \max_{y_{i-1} \in \alpha} s(x, i-1, y_{i-1}, y_i, y_{i+1}) + \heartsuit_{i-1}(y_{i-1}, y_i)$$

and backpointers will be

$$bp_i(y', y) = \operatorname{argmax}_{y_{i-1} \in \alpha} s(x, i-1, y_{i-1}, y_i, y_{i+1}) + \heartsuit_{i-1}(y_{i-1}, y_i).$$