Instructions

- This homework has two parts: Q1 and Q2 are written questions and Q3 is a programming assignment with some written questions. Each part needs to be submitted as follows:
  - Submit the answers to the written questions as a `pdf` file on Canvas for the assignment corresponding to Homework 4 Written. This should contain the written answers to Q1 and Q2. Name the pdf file as `LastName_FirstName.pdf`. We recommend students type answers with LaTeX or word processors for this part. A scanned handwritten copy would also be accepted, try to be clear as much as possible. No credit may be given to unreadable handwriting.
  - The programming assignment requires you to work on boilerplate code and text data, which are provided in a zip file on Canvas. Submit the code to the programming assignment in a zip that contains (1) `POS_tagging.ipynb`, which will contain both the code **and the written answers to Q3**, and (2) a PDF version of the notebook `POS_tagging.pdf`. You can convert the notebook to a PDF by clicking "File" → "Download as" → "PDF."
    This submission is to be made on Canvas for the assignment corresponding to Homework 4 Programming. Name the zip file as `LastName_FirstName.zip`.
  - You can also submit output files to Gradescope "HW4 POS Tagging" to check your models' performance. The uploaded file `test_labels.txt` should contain the predicted tags from your POS tagger. You can see your score for Q3 programming questions as soon as you submit it.
- For the written questions, write out all steps required to find the solutions so that partial credit may be awarded.
- We generally encourage collaboration with other students. You may discuss the questions and potential directions for solving them with another student. However, you need to write your own solutions and code separately, and not as a group activity. Please list the students with whom you collaborated.

1. Suppose we are tagging a sentence "They run programs" with two types of tags: N (noun) and V (verb). The initial probabilities, transition probabilities and emission probabilities computed by an HMM model are shown below.

	N	V
π	0.8	0.2

Table 1: Initial probabilities

	N	V
N	0.4	0.6
V	0.8	0.2

Table 2: Transition probabilities

	they	run	programs
N	0.6	0.2	0.2
V	0	0.6	0.4

Table 3: Emission probabilities

   (a) Given the HMM model above, compute the probability of the given sentence "They run programs" by using the Forward AND Backward Algorithm. Please show all the steps of calculations. [6 pts]

   (b) Tag the sentence "They run programs" by using the Viterbi Algorithm. Please show all the steps of calculations. [4 pts]

Solution. a) The table for the forward algorithm is as follows (read it from left to right):

	they	run	programs
N	$0.8(0.6) = \mathbf{0.48}$	$0.48(0.4) + 0(0.8)$ $= 0.192(0.2)$ $= \mathbf{0.0384}$	$0.384(0.4) + 0.1728(0.8)$ $= 0.1536(0.2)$ $= \mathbf{0.03072}$
V	$0.2(0) = \mathbf{0}$	$0.48(0.6) + 0(0.2)$ $= 0.288(0.6)$ $= \mathbf{0.1728}$	$0.0384(0.6) + 0.1728(0.2)$ $= 0.0576(0.4)$ $= \mathbf{0.02304}$

Table 4: Forward algorithm

If we sum up the values in the last, right-most column, we get the probability of the given sentence from the forward algorithm, which is $0.03072 + 0.02304 = \mathbf{0.05376}$.

The table for the backward algorithm can be seen on the next page because it doesn't fit on this page (read it from right to left).

	start_state	they	run	programs
N	$0.8(0.6)(0.112)$ $= \mathbf{0.05376}$	$0.4(0.2)(0.32)$ $+ 0.6(0.6)(0.24)$ $= \mathbf{0.112}$	$0.4(0.2)(1)$ $+ 0.6(0.4)(0.1)$ $= \mathbf{0.32}$	1
V	$0.2(0)(0.08)$ $= \mathbf{0}$	$0.8(0.2)(0.32)$ $+ 0.2(0.6)(0.24)$ $= \mathbf{0.08}$	$0.8(0.2)(1)$ $+ 0.2(0.4)(1)$ $= \mathbf{0.24}$	1

Table 5: Backward algorithm

If we sum up the values in the first, left-most column, we get the probability of the given sentence from the backward algorithm, which is $0.5376 + 0 = \mathbf{0.05376}$. Clearly, the forward and backward algorithm produce the same probability, which is verification they are right.

b) The table for the Viterbi algorithm can be seen below:

	they	run	programs
N	$0.8(0.6) = \mathbf{0.48}$	$N = 0.48(0.4) = 0.192$ $V = 0(0.8) = 0$ $\implies \max(N, V) = 0.192$ $= 0.192(0.2)$ $= \mathbf{0.0384}$ Backpointer: **N**	$N = 0.038(0.4) = 0.015$ $V = 0.173(0.8) = 0.138$ $\implies \max(N, V) = 0.138$ $= 0.138(0.2)$ $= \mathbf{0.0276}$ Backpointer: **V**
V	$0.2(0) = \mathbf{0}$	$N = 0.48(0.6) = 0.288$ $V = 0(0.2) = 0$ $\implies \max(N, V) = 0.288$ $= 0.288(0.6)$ $= \mathbf{0.1728}$ Backpointer: **N**	$N = 0.0384(0.6) = 0.023$ $V = 0.1728(0.2) = 0.035$ $\implies \max(N, V) = 0.035$ $= 0.035(0.4)$ $= \mathbf{0.0138}$ Backpointer: **V**

Table 6: Viterbi algorithm

The above table not only includes probabilities for each state, along with all the necessary calculations you want to see, but also the backpointers. I could have created a separate backpointers table, and that's what you would do in practice, but honestly, that's not really important in a question like this. From the above, we want to use that table to tag the sequence. We start from the end of the sentence, which is at the right-most column ("programs"). From here, we can see the most likely tag for "programs" is N, because the "N" row has a probability 0.0276 and the "V" row has the probability 0.0138. We see its backpointer is V, because the V tag contributed the most to the calculation of its probability. So, we go to the next word in the sentence, "run". Since the backpointer for "program" was V, we look at "run", tag it as V, since "run" being a verb contributed the most to the probability of "programs" being a noun, and then we look at the backpointer of "run", which is "N". As such, we finally reach the final word, "they", and tag it as "N" since that was the backpointer. Therefore, from the above description, we have arrived at the conclusion that "programs" is tagged with N, "run" is tagged with V, and "they" is tagged with N. This sentence tagging produces the sequence N, V, N, or Noun, Verb, Noun. This result makes sense from an English-speaker perspective because before I did the Viterbi algorithm, I predicted it would be tagged as NVN anyway. ■

2. Suppose we are parsing a sentence "submit the homework through Canvas" with a Probability Context Free Grammar (PCFG) in Chomsky Normal Form. We give the PCFG as follows.

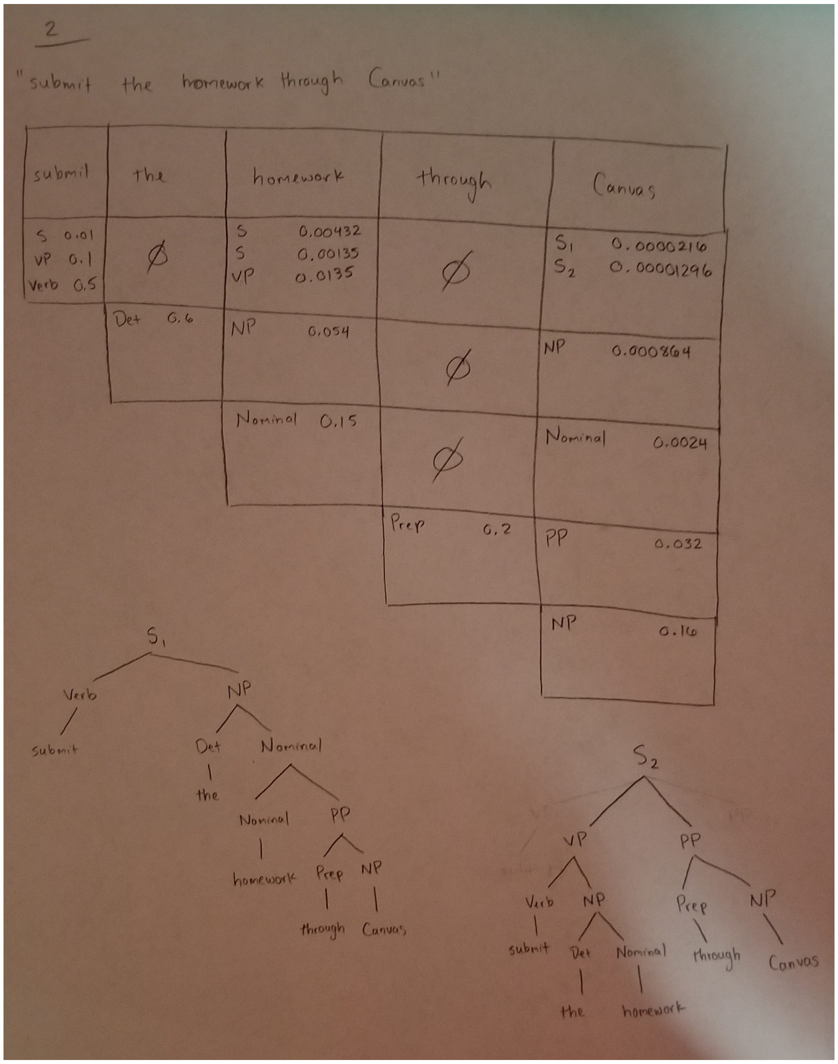
Rule	Probability
S $\longrightarrow$ NP VP	0.8
S $\longrightarrow$ submit	0.01
S $\longrightarrow$ Verb NP	0.05
S $\longrightarrow$ VP PP	0.03
NP $\longrightarrow$ Canvas	0.16
NP $\longrightarrow$ Gradescope	0.04
NP $\longrightarrow$ Det Nominal	0.6
Nominal $\longrightarrow$ Nominal Noun	0.2
Nominal $\longrightarrow$ Nominal PP	0.5
Nominal $\longrightarrow$ homework	0.15
VP $\longrightarrow$ submit	0.1
VP $\longrightarrow$ contain	0.04
VP $\longrightarrow$ Verb NP	0.5
VP $\longrightarrow$ VP PP	0.3
PP $\longrightarrow$ Prep NP	1.0
Det $\longrightarrow$ the	0.6
Prep $\longrightarrow$ through	0.2
Verb $\longrightarrow$ submit	0.5

Table 7: Initial probabilities

(a) Parse the sentence "submit the homework through Canvas" by using the CKY Algorithm. You need to show the probabilities of all possible parses and choose the parse with the highest probability. Please show all the steps of calculations. [8 pts]

(b) Compute the probability of the given sentence "submit the homework through Canvas" by using the CKY Algorithm. Please show all the steps of calculations. [2 pts]

Solution. a)



From the above picture, one can see 2 parses were found, S_1 and S_2. The probability of each parse is as follows: $P(S_1) = 0.0000216$, $P(S_2) = 0.00001296$. The parse with the highest probability is S_1, and its parse tree is shown in the above picture, with the following chronological tags: Verb Det Nominal Prep NP.

b)

$P(S_1) = 0.0000216$
$P(S_2) = 0.00001296$

So, $P(S) = P(S_1) + P(S_2) =$ **0.00003456** ∎

3. For the programming task, you will implement several models for part-of-speech tagging. We will use the LSTM as the basic architecture for the model and try several modifications to improve performance. To do this, you will complete the Python code started in `POS_tagging.ipynb`, available on Canvas along with the train and test data (`train.txt` and `test.txt`) in `POS_tagging.zip`. For all the written questions to Q3 (accuracy reporting and error analysis), please provide your answers in the space provided **inside the notebook**.

   NOTE: for each model that you test, you will produce a file `test_labels.txt` that you can upload to Gradescope to evaluate your model's performance.

   (a) To begin, implement an LSTM tagger by completing the skeleton code provided in `BasicPOSTagger` in the notebook. The model will use *word embeddings* to represent the sequence of words in the sentence.

   For this problem, implement the forward pass and the training procedure for the tagger. After training for 30 epochs, the model should achieve at least 80% on the held-out data.

   In addition, compute the top-10 most frequent types of errors that the model made in labeling the data in `test.txt` (e.g. if the model labeled `NN` as `VB` 10 times) and report these errors in the written answer. Report the results in a table containing the top-10 mistakes, with the following columns: model tag type, ground truth tag type, error frequency, up to 5 example words that were tagged incorrectly. What kinds of errors did the model make and why do you think it made them? [5 pts]

   (b) Word-level information is useful for part-of-speech tagging, but what about character level information? For instance, the present tense of English verbs is marked with the suffix *-ing*, which can be captured by a sequential character model.

   For this problem, implement the character-level model by completing the skeleton code provided in `CharPOSTagger` in the notebook. Unlike part (a), this model will use *character* embeddings. After training for 30 epochs, the model should achieve at least 80% on the held-out data.

   In addition, compare the top-10 types of errors made by the character-level model with the errors made by the word-level model from part (a) and report these errors in the written answer. Report the error results in the same format as before. What kinds of errors does the character-level model make as compared to the original model, and why do you think it made them? [10 pts]

   (c) The previous models considered only a single direction, i.e. feeding the information from left to right (starting at word $i = 1$ and ending at word $i = N$). For the final model, you will implement a bidirectional LSTM that uses information from both left-to-right and right-to-left to represent dependencies between adjacent words in both directions. In order to decide if the word "call" is `VB` or `NN`, the model can use information from the following words to disambiguate (e.g. "call waiting" vs. "call him").

   For this problem, implement the bidirectional model by completing the skeleton code provided in `BiLSTMPOSTagger` in the notebook. Like the model in part (a), this will use word embeddings. After training for 30 epochs, the model should achieve at least 90% on the held-out data.

In addition, implement **one** of the three modifications listed in the notebook to boost your score: (a) different word embeddings to initialize the model (one different type), (b) different hyperparameters to initialize the model (five different combinations), (c) different model architecture on top of the BiLSTM (one different architecture). Explain in the written answer which modifications you have made, why you chose a certain modification over another (e.g. `Glove` over `word2vec`) and the resulting accuracy of the modified model.

Lastly, compare the top-10 errors made by this modified model with the errors made by the model from part (a). Report the error results in the same format as before. If you tested multiple different hyperparameters, compute the errors for the model with the highest accuracy on the validation data. What errors does the original model make as compared to the modified model, and why do you think it made them? [10 pts]