HOMEWORK 7: GRAPHICAL MODELS

10-301/10-601 Introduction to Machine Learning (Summer 2022)
https://www.cs.cmu.edu/~hchai2/courses/10601/
OUT: Wednesday, July 20
DUE: Wednesday, July 27 at 1:00 PM
TAs: Sana, Brendon, Ayush, Boyang (Jack), Chu

Summary In this assignment you will implement a new named entity recognition system using Hidden Markov Models. You will begin by going through some multiple choice and short answer warm-up problems to build your intuition for these models and then use that intuition to build your own HMM models.

START HERE: Instructions

- **Collaboration Policy**: Please read the collaboration policy here: https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~hchai2/courses/10601/#Syllabus
- **Submitting your work:** You will use Gradescope to submit answers to all questions and code. Please follow instructions at the end of this PDF to correctly submit all your code to Gradescope.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in LaTeX. Each derivation/proof should be completed in the boxes provided. You are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template. If you do not follow the template, your assignment may not be graded correctly by our AI assisted grader.
  - **Programming:** You will submit your code for programming questions on the homework to Gradescope (https://gradescope.com). After uploading your code, our grading scripts will autograde your assignment by running your program on a virtual machine (VM). When you are developing, check that the version number of the programming language environment (Python 3.9.6) and versions of permitted libraries (`numpy` 1.21.2) match those used on Gradescope. You have 10 free Gradescope programming submissions. After 10 submissions, you will begin to lose points from your total programming score. We recommend debugging your implementation locally first before submitting your code to Gradescope.
- **Materials:** The data and reference output that you will need in order to complete this assignment is posted along with the writeup and template on the course website.

Instructions for Specific Problem Types

For "Select One" questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ● Henry Chai
- ○ Marie Curie
- ○ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ● Henry Chai
- ○ Marie Curie
- ⊗ Noam Chomsky

For "Select all that apply" questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- □ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are scientists?

- ■ Stephen Hawking
- ■ Albert Einstein
- ■ Isaac Newton
- ⊠ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-601

10-6301

Written Questions (53 points)

1 LaTeX Bonus Point (1 points)

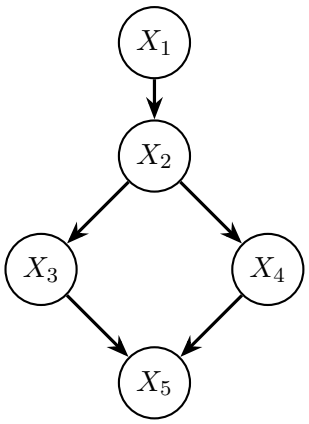
1. (1 point) **Select one:** Did you use LaTeX for the entire written portion of this homework?

   ◯ Yes

   ◯ No

2 Bayesian Networks (10 points)

Consider the joint distribution over the binary random variables X_1, X_2, X_3, X_4, X_5 represented by the Bayesian Network shown in the figure.



1. (1 point) Write the joint probability distribution $P(X_1, X_2, X_3, X_4, X_5)$, factorized as much as possible, using the conditional independence assumptions expressed by the above network.

   Your Answer

2. (1 point) How many parameters would we need to represent the joint distribution **without** the conditional independence assumptions expressed by the Bayesian Network?

   Your Answer

3. (1 point) How many parameters would we need to represent the joint distribution **with** the conditional independence assumptions expressed by the Bayesian Network?

   Your Answer

4. (1 point) Which variables are in the Markov boundary of X_4? Note that the Markov boundary is the smallest possible Markov blanket.

Your Answer

5. By sampling from the network, we are able to estimate the following conditional probability tables:

$X_1 = 0$	0.3
$X_1 = 1$	0.7

	$X_1 = 0$	$X_1 = 1$
$X_2 = 0$	0.8	0.25
$X_2 = 1$	0.2	0.75

	$X_2 = 0$	$X_2 = 1$
$X_3 = 0$	0.5	0.6
$X_3 = 1$	0.5	0.4

	$X_2 = 0$	$X_2 = 1$
$X_4 = 0$	0.3	0.2
$X_4 = 1$	0.7	0.8

	$X_3 = 0, X_4 = 0$	$X_3 = 0, X_4 = 1$	$X_3 = 1, X_4 = 0$	$X_4 = 1, X_3 = 1$
$X_5 = 0$	0.4	0.7	0.8	0.5
$X_5 = 1$	0.6	0.3	0.2	0.5

Table 1: Estimated Conditional Probability Tables

In these tables, the columns represent the condition; for example, the top left entry in the last table corresponds to $P(X_5 = 0 \mid X_3 = 0, X_4 = 0)$. Using the values in Table 1, compute the following probabilities. Round to **four decimal places** after the decimal point.

(a) (2 points) $P(X_1 = 0, X_2 = 1, X_3 = 0, X_4 = 1, X_5 = 0)$

Answer

Work

(b) (2 points) $P(X_3 = 1)$

Answer	Work

(c) (2 points) $P(X_1 = 1|X_3 = 1)$

Answer	Work

3 Hidden Markov Models (10 points)

1. (2 points) **Select all that apply:** Let Y_t be the state at time t. Which of the following are true under the (first-order) Markov assumption in an HMM?

   - ☐ The states are independent
   - ☐ The observations are independent
   - ☐ $Y_t \perp\!\!\!\perp Y_{t-1} \mid Y_{t-2}$
   - ☐ $Y_t \perp\!\!\!\perp Y_{t-2} \mid Y_{t-1}$
   - ☐ None of the above

2. (2 points) **Select all that apply:** Which of the following independence assumptions hold in an HMM?

   - ☐ The current observation x_t is conditionally independent of all other observations given the current state Y_t
   - ☐ The current observation x_t is conditionally independent of all other states given the current state Y_t
   - ☐ The current state Y_t is conditionally independent of all states given the previous state
   - ☐ The current observation x_t is conditionally independent of Y_{t-2} given the previous observation x_{t-1}
   - ☐ None of the above

3. In the remaining questions, you will see two quantities and decide which relations could hold between them.

   As a reminder, $\alpha_t(s_j) = P(Y_t = s_j, x_{1:t})$ and $\beta_t(s_j) = P(x_{t+1:T} \mid Y_t = s_j)$. Assume that there are T observations and $T \geq 5$, J possible hidden states so each $Y_t \in \{s_1, \ldots, s_J\}$, and that we do not use pseudocounts. Pseudocounts are explained in Section 9.4.

   (a) (2 points) **Select all that apply:** Which of the following are possible relations between $\sum_{i=1}^{J}(\alpha_5(s_i)\beta_5(s_i))$ and $P(x_{1:T})$?

   $\square$ $=$

   $\square$ $>$

   $\square$ $<$

   (b) (2 points) **Select all that apply:** Which of the following are possible relations between $P(Y_4 = s_1, Y_5 = s_2, x_{1:T})$ and $\alpha_4(s_1)\beta_5(s_2)$?

   $\square$ $=$

   $\square$ $>$

   $\square$ $<$

   (c) (2 points) **Select all that apply:** Which of the following are possible relations between $\alpha_5(s_i)$ and $\beta_5(s_i)$?

   $\square$ $=$

   $\square$ $>$

   $\square$ $<$

4 Forward-Backward Algorithm (15 points)

The following questions should be completed before you start the programming component of this assignment. To help you prepare to implement the forward-backward algorithm (see Section 9.5 for a detailed explanation), we have provided a small example for you to work through by hand. This toy data set consists of a training set of three sequences with three unique words and two states, and a validation set with a single sequence composed of the same words occurring in the training set.

Training set:

```
you     D
eat     C
fish    D

you     D
fish    D
eat     C

eat     C
fish    D
```

The training word sequences are:

$$\mathbf{x}^{(1)} = [\texttt{you eat fish}]^T$$
$$\mathbf{x}^{(2)} = [\texttt{you fish eat}]^T$$
$$\mathbf{x}^{(3)} = [\texttt{eat fish}]^T$$

And the corresponding tags are:

$$\mathbf{y}^{(1)} = [D\ C\ D]^T$$
$$\mathbf{y}^{(2)} = [D\ D\ C]^T$$
$$\mathbf{y}^{(3)} = [C\ D]^T$$

Validation set:

```
fish
eat
you
```

The validation word sequence is:

$$\mathbf{x}_{\text{validation}} = \left[\texttt{fish eat you}\right]^T$$

In this section and the following section (Viterbi Decoding), we define:

- Observations $x_t \in \{\texttt{you}, \texttt{eat}, \texttt{fish}\}$
- States $Y_t \in \{C, D\}$
- $\mathbf{B}$ is the transition matrix, where $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$. Here $\mathbf{B}$ is a 4×4 matrix, as we are also accounting for the START/END states.
- $\mathbf{A}$ is the emission matrix, where $A_{jk} = P(X_t = o_k \mid Y_t = s_j)$. Here $\mathbf{A}$ is a 4×3 matrix, as we are also accounting for the START/END states.

Note: Pseudocounts (see Section 9.4) should be used to estimate $\mathbf{B}$ and $\mathbf{A}$. We will also adhere to the conditions listed in Section 9.4 for entries in $\mathbf{B}$ and $\mathbf{A}$ involving the START/END states; these entries will be filled in for you and **do not require pseudocounts/should not be normalized**.

For all numerical answers, round to **four decimal places** after the decimal point. Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur.

1. (4 points) Fill in the following tables with the estimates of $\mathbf{B}$ (left) and $\mathbf{A}$ (right).

	START	C	D	END
START	0	??	??	??
C	0	??	??	??
D	0	??	??	??
END	0	0	0	0

	`you`	`eat`	`fish`
START	1	1	1
C	??	??	??
D	??	??	??
END	1	1	1

2. (2 points) Compute $\alpha_1(C)$ and $\alpha_1(D)$, the α values associated with states C and D for the first word in the validation sequence.

$\alpha_1(C)$

Work

$\alpha_1(D)$

Work

3. (2 points) Compute $\alpha_2(C)$, the α value associated with state C for the second word in the validation sequence.

$\alpha_2(C)$	Work

4. (2 points) Compute $\beta_2(D)$, the β value associated with state D for the second word in the validation sequence.

$\beta_2(D)$	Work

5. (3 points) Predict the state for the third word in the validation sequence.

State	Work

6. (2 points) Compute the log-likelihood for the entire validation sequence, "`fish eat you`."

Likelihood	Work

5 Viterbi Decoding (6 points)

In the Viterbi algorithm, we seek to find the most probable hidden state sequence $\hat{Y_1}, \ldots, \hat{Y_T}$ given the observations $x_1, \ldots, x_T$.

We define:

- $\omega_t(s_k)$ to be the probability of most probable sequence of t states ending at state s_k, given the first t observations:

$$\omega_t(s_k) = \max_{y_1,\ldots,y_{t-1}} P(x_{1:t}, y_{1:t-1}, Y_t = s_k) \tag{1}$$

- $b_t(s_k)$ to be the backpointer that stores the path through hidden states that gives us the highest probability:

$$b_t(s_k) = \operatorname*{argmax}_{y_1,\ldots,y_{t-1}} P(x_{1:t}, y_{1:t-1}, Y_t = s_k) \tag{2}$$

We outline the Viterbi Algorithm below:

1. Initialize $\omega_0(\text{START}) = 1$
2. For $1 \leq t \leq T+1$, we calculate

$$\omega_t(s_j) = \max_{k \in \{1,\ldots,J\}} A_{jx_t} B_{kj} \omega_{t-1}(s_k)$$
$$b_t(s_j) = \operatorname*{argmax}_{k \in \{1,\ldots,J\}} A_{jx_t} B_{kj} \omega_{t-1}(s_k)$$

We can obtain the most probable sequence by backtracing through the backpointers as follows:

1. $\hat{Y_T} = b_{T+1}(END)$
2. For $t = T-1, \ldots, 1$:
   $\hat{Y_t} = b_{t+1}(\hat{Y}_{t+1})$
3. Return $\hat{Y_1}, \ldots, \hat{Y_T}$

For the following questions, consider the Hidden Markov Model specified below and assume we observe the sequence $\mathbf{x} = [\texttt{you eat}]^T$.

	C	D	END
START	0.5	0.5	0
C	0.5	0.3	0.2
D	0.3	0.6	0.1

	you	eat	fish
C	0.5	0.4	0.1
D	0.3	0.5	0.2

Note that the values in these matrices aren't necessarily the same as those you estimated in the previous section. Showing your work in these questions is optional, but it is recommended to help us understand where any misconceptions may occur. Only your answer in the left box will be graded.

1. (2 points) Compute $\omega_1(C)$ and $\omega_1(D)$. Round to **four decimal places** after the decimal point.

$\omega_1(C)$	Work

$\omega_1(D)$	Work

2. (2 points) Compute $\omega_2(C)$ and $\omega_2(D)$. Round to **four decimal places** after the decimal point.

$\omega_2(C)$	Work

$\omega_2(D)$	Work

3. (2 points) **Select one:** Which of the following is the most likely sequence of hidden states given the observed sequence?

   ◯ $Y_1 = C, Y_2 = C$

   ◯ $Y_1 = D, Y_2 = D$

   ◯ $Y_1 = D, Y_2 = C$

   ◯ $Y_1 = C, Y_2 = D$

   ◯ Not enough information.

6 Forward-Backward Algorithm Vectorization (5 points)

The forward-backward algorithm is a dynamic programming approach to efficiently compute forward and backward probabilities for a given sequence. To help with your implementation of the algorithm in the programming section, in this section you will derive vectorized expressions for the forward and backward probabilities in log space. For an explanation of working in log space and the log-sum-exp trick, see Section 9.5.3 and HW7 Recitation.

Express the logarithms of the following alpha vectors $\boldsymbol{\alpha}_i$ and beta vectors $\boldsymbol{\beta}_i$ in terms of $\mathbf{x}$, $\log \mathbf{A}$, $\log \mathbf{B}$ (indexing them as appropriate), and $\log \boldsymbol{\alpha}_t$ and $\log \boldsymbol{\beta}_t$ for any $t \neq i$. Your solutions must rely on matrix operations (e.g. matrix multiplication, elementwise multiplication $\odot$, elementwise addition $\oplus$). You may use the `LogSumExp` function, and assume it is applied row-wise for a matrix (yielding a column vector as a result). You may also assume NumPy-style broadcasting and that all $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ are column vectors.

(a) (1 point) $\log(\boldsymbol{\alpha}_1)$

Your Answer

(b) (2 points) $\log(\boldsymbol{\alpha}_2)$

Your Answer

(c) (2 points) $\log(\boldsymbol{\beta}_2)$

Your Answer

7 Empirical Questions (6 points)

Return to these questions after implementing your `learnhmm.py` and `forwardbackward.py` functions. Please ensure that you have used the log-sum-exp trick in your programming as described in Section 9.5.3 before answering these empirical questions.

Using the full data set **en_data/train.txt** in the handout, use your implementation of `learnhmm.py` to learn HMM parameters using the first 10, 100, 1000, and 10000 sequences in the file. Use these learned parameters to perform prediction on both the English **train.txt** and the **validation.txt** files with your implementation of `forwardbackward.py`. Construct a plot with number of sequences used for training on the x-axis (log-scale with base e) and average log likelihood across all sequences from the English **train.txt** and the **validation.txt** on the y-axis (see Section 9.5 for details on computing the log data likelihood for a sequence).

Fill in the table with the resulting log likelihood values, rounded to two decimal places, and include your plot in the large box. To receive credit for your plot, you must submit a computer generated plot. **DO NOT** hand draw your plot.

1. (4 points) Fill in this table with your computed log-likelihood values.

# Sequences	Train Average Log-Likelihood	Validation Average Log-Likelihood
10	??	??
100	??	??
1000	??	??
10000	??	??

2. (2 points) Put your plot below:

Plot

8 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found here.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.
3. Did you find or come across code that implements any part of this assignment? If so, include full details.

Your Answer

9 Programming (90 points)

9.1 The Task

In the programming section you will implement a named entity recognition system using Hidden Markov Models (HMMs). Named entity recognition (NER) is the task of classifying named entities, typically proper nouns, into pre-defined categories, such as person, location, or organization. Consider the example sequence below, where each word is appended with a tab and then its tag:

“	O
Rhinestone	B-ORG
Cowboy	I-ORG
”	O
(	O
Larry	B-PER
Weiss	I-PER
)	O
-	O
3:15	O

`Rhinestone` and `Cowboy` are labeled as an organization (`ORG`), while `Larry` and `Weiss` is labeled as a person (`PER`). Words that aren't named entities are assigned the `O` tag. The `B-` prefix indicates that a word is the beginning of an entity, while the `I-` prefix indicates that the word is inside the entity.

NER is an incredibly important task for a machine to analyze and interpret a body of natural language text. For example, when designing a system that automatically summarizes news articles, it is important to recognize the key subjects in the articles. Another example is designing a trivia bot. If you can quickly extract the named entities from the trivia question, you may be able to more easily query your knowledge base (e.g. type a query into Google) to request information about the answer to the question.

On a technical level, the main task is to implement an algorithm to learn the HMM parameters given the training data and then implement the forward-backward algorithm to perform a smoothing query which we can then use to predict the hidden tags for a sequence of words.

9.2 The Dataset

WikiANN is a “silver standard” dataset that was generated without human labelling. The English Abstract Meaning Representation (AMR) corpus and DBpedia features were used to train an automatic classifier to label Wikipedia articles. These labels were then propagated throughout other Wikipedia articles using the Wikipedia's cross-language links and redirect links. Afterwards, another tagger that self-trains on the existing tagged entities was used to label all other mentions of the same entities, even those with different morphologies (prefixes and suffixes that modify a word in other languages). Finally, the amassed training examples were filtered by “commonness” and “topical relatedness” to pick more relevant training data.

The WikiANN dataset provides labelled entity data for Wikipedia articles in 282 languages. We will be primarily using the English subset, which contains 14,000 training examples and 3,300 test examples, and the French subset, which contains around 7,500 training examples and 300 test examples.

9.3 File Formats

The contents and formatting of each of the files in the handout folder is explained below.

1. **train.txt** This file contains labeled text data that you will use in training your model in the Learning problem (Section 9.4). Specifically, the text contains one word per line that has already been preprocessed, cleaned and tokenized. Every sequence has the following format:

   `<Word0>\t<Tag0>\n<Word1>\t<Tag1>\n ...  <WordN>\t<TagN>\n`

   where every `<WordK>\t<TagK>` unit token is separated by a newline. Between each sequence is an empty line. If we have two three-word sequences in our data set, the data will look like so:

   ```
   <Word0>\t<Tag0>\n
   <Word1>\t<Tag1>\n
   <Word2>\t<Tag2>\n
   \n
   <Word0>\t<Tag0>\n
   <Word1>\t<Tag1>\n
   <Word2>\t<Tag2>
   ```

   Note: Word 2 of the second sequence does not end with a newline because it is the end of the data set.

   Since the `<START>` and `<END>` tags do not have associated words, they do not appear in **train.txt**. The `parse_args` function in the starter code automatically includes these special tags for you.

2. **validation.txt**: This file contains labeled validation data that you will use to evaluate your model. This file has the same format as **train.txt**.

3. **index_to_word.txt, index_to_tag.txt**: These files contain a list of all words or tags that appear in the data set. The format is simple:

index_to_word.txt	**index_to_tag.txt**
`<Word0>\n`	`<Tag0>\n`
`<Word1>\n`	`<Tag1>\n`
`<Word2>\n`	`<Tag2>\n`
⋮	⋮

   In your functions, you will convert the string representation of words or tags to indices corresponding to the location of the word or tag in these files. For example, if *Austria* is on line 729 of **index_to_word.txt**, then all appearances of *Austria* in the data sets should be converted to the index 729.

4. **predicted.txt**: This file contains labeled data that you will use to debug your implementation. The labels in this file are generated by running a reference implementation using the features from **train.txt**. This file has the same format as **train.txt** (and therefore should not include the `<START>` and `<END>` tags).

5. **metrics.txt**: This file contains the metrics you will compute for the validation data. The first line should contain the average log likelihood, and the second line should contain the prediction accuracy. There should be a single space after the colon preceding the metric value; see the reference output file for more details.

6. **hmmtrans.txt, hmmemit.txt**: These files contain pre-trained model parameters of an HMM that you

can use to test your implementation of the Learning and Evaluation and Decoding problems (Sections 9.4, 9.5). Both files are formatted as space-separated matrices, where each entry corresponds to an entry in the transition or emission matrix. In the case of transition probabilities, the entries correspond to the probability of transitioning from one tag to another. Similarly, in the case of emission probabilities, the entries correspond to the probability of emitting a particular word, given a current tag. Elements in the same row are separated by a space. Each row corresponds to a line of text, using `\n` to create new lines.

hmmtrans.txt:

```
<ProbS1S1> <ProbS1S2> ...  <ProbS1SN>\n
<ProbS2S1> <ProbS2S2> ...  <ProbS2SN>\n...
```

Above, the entry `<ProbS2S1>` corresponds to the probability $P(Y_t = s_1|Y_{t-1} = s_2)$, where Y_i are tags.

hmmemit.txt:

```
<ProbS1Word1> <ProbS1Word2> ...  <ProbS1WordN>\n
<ProbS2Word1> <ProbS2Word2> ...  <ProbS2WordN>\n...
```

Above, the entry `<ProbS2Word1>` corresponds to the probability $P(X_t = x_1|Y_t = s_2)$, where X_t is a word and Y_t is a tag.

9.4 Learning

Your first task is to write a program `learnhmm.py` to learn the Hidden Markov Model parameters needed to apply the forward-backward algorithm (See Section 9.5). You will need to estimate the transition probabilities $\mathbf{B}$ and the emission probabilities $\mathbf{A}$. For this assignment, we model each of these probabilities using a multinomial distribution with parameters $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$, and $A_{jk} = P(X_t = x_k \mid Y_t = s_j)$. These can be estimated using maximum likelihood, which results in the following parameter estimates:

1. $P(Y_t = s_k \mid Y_{t-1} = s_j) = B_{jk} = \frac{1+N_{Y_t=s_k, Y_{t-1}=s_j}}{\sum_{p=1}^{J}(1+N_{Y_t=s_p, Y_{t-1}=s_j})}$, where $N_{Y_t=s_k, Y_{t-1}=s_j}$ is the number of times tag s_j is followed by tag s_k in the training data set.
2. $P(X_t = x_k \mid Y_t = s_j) = A_{jk} = \frac{1+N_{X_t=x_k, Y_t=s_j}}{\sum_{p=1}^{M}(1+N_{X_t=x_p, Y_t=s_j})}$, where $N_{X_t=x_k, Y_t=s_j}$ is the number of times that the tag s_j emits the word x_k in the training data set.

Note we add 1 to each count to make a pseudocount. This is slightly different from pure maximum likelihood estimation, but it is useful in improving performance when evaluating unseen transitions or emissions on the validation set.

We include *both* the special tags `<START>` and `<END>` in the $\mathbf{A}$ and $\mathbf{B}$ matrices, which makes indexing easier. As a result, after learning $\mathbf{A}$ and $\mathbf{B}$, you must ensure that the following conditions hold:

1. $P(Y_t = \texttt{<START>}|Y_{t-1} = s_j) = B_{j,\,\texttt{<START>}} = 0$ for all j, since nothing can transition to `<START>`.
2. $P(Y_t = s_k|Y_{t-1} = \texttt{<END>}) = B_{\texttt{<END>},\,k} = 0$ for all k, since `<END>` cannot transition to any other tag. This implies that the row in $\mathbf{B}$ corresponding to the `<END>` tag is *not* a probability distribution.
3. $P(X_t = x_k|Y_t = \texttt{<START>}) = A_{\texttt{<START>},k} = P(X_t = x_k|Y_t = \texttt{<END>}) = A_{\texttt{<END>},k} = 1$ for all k. While setting these values to 0 may be more natural since the special `<START>` and `<END>` tags never emit anything, setting these values to 1 allows us to explicitly consider fewer edge cases in the forward-backward algorithm and write cleaner (and equally correct) code.

Your implementation should read in the training data set (**train.txt**) and then estimate $\mathbf{B}$ and $\mathbf{A}$ using the above maximum likelihood solutions with pseudocounts.

Your outputs should be in the same format as **hmmtrans.txt** and **hmmemit.txt** (including the same number of decimal places to ensure there are no rounding errors during prediction). The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python learnhmm.py [args...]
```

Where `[args...]` is a placeholder for five command-line arguments: `<train_input>` `<index_to_word>` `<index_to_tag>` `<hmmemit>` `<hmmtrans>`. These arguments are described below:

1. `<train_input>`: path to the training input `.txt` file (see Section 9.2)
2. `<index_to_word>`: path to the `.txt` file that specifies the dictionary mapping from words to indices. The tags are ordered by index, with the first word having index of 0, the second word having index of 1, etc.
3. `<index_to_tag>`: path to the `.txt` file that specifies the dictionary mapping from tags to indices. The tags are ordered by index, with the first tag having index of 0, the second tag having index of 1, etc.
4. `<hmmemit>`: path to output `.txt` file to which the emission probabilities ($\mathbf{A}$) will be written. The file output to this path should be in the same format as the handout `hmmemit.txt` (see Section 9.2)
5. `<hmmtrans>`: path to output `.txt` file to which the transition probabilities ($\mathbf{B}$) will be written. The file output to this path should be in the same format as the handout `hmmtrans.txt` (see Section 9.2).

As an example, the following command would run your program on the toy dataset provided in the handout.

```
$ python learnhmm.py toy_data/train.txt toy_data/index_to_word.txt \
toy_data/index_to_tag.txt hmmemit.txt hmmtrans.txt
```

After running the command above, the **hmmemit.txt** and **hmmtrans.txt** output files should match the reference files provided in the `toy_output` directory.

9.5 Evaluation and Decoding

9.5.1 Forward Backward Algorithm and Minimal Bayes Risk Decoding

Your next task is to implement the forward-backward algorithm. Suppose we have a set of sequence consisting of T words, $x_1, \ldots, x_T$. Each word is associated with a tag $Y_t \in \{1, \ldots, J\}$ (including the <START> and <END> tags). In the forward-backward algorithm we seek to compute $P(Y_t \mid x_{1:T})$ for each t up to a multiplicative constant. This is done by first breaking $P(Y_t \mid x_{1:T})$ into a "forward" component and a "backward" component as follows:

$$\begin{aligned} P(Y_t = s_j \mid x_{1:T}) &\propto P(Y_t = s_j, x_{t+1:T} \mid x_{1:t}) \\ &\propto P(Y_t = s_j \mid x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j, x_{1:t}) \\ &\propto P(Y_t = s_j \mid x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j) \\ &\propto P(Y_t = s_j, x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j) \end{aligned}$$

Then, we can efficiently compute $P(Y_t = s_j, x_{1:t})$ and $P(x_{t+1:T} \mid Y_t = s_j)$ using dynamic programming.

Forward Algorithm

Define $\alpha_t(s_j) = P(Y_t = s_j, x_{1:t})$. This can be rearranged into the following expression:

$$\alpha_t(s_j) = P(x_t | Y_t = s_j) \sum_{k=1}^{J} P(Y_t = s_j | Y_{t-1} = s_k) \alpha_{t-1}(s_k) \tag{3}$$

Using this definition, the α's can be computed using the following dynamic programming procedure:

$\alpha_0(\texttt{<START>}) = 1$
$\alpha_0(s_m) = 0$ `for all` $s_m \neq$ <START>
for `t = 1,...,T:`
    for `j = 1,...,J:`
        $\alpha_t(s_j) = A_{j,x_t} \sum_k (B_{k,j} \alpha_{t-1}(s_k))$

Backward Algorithm

Define $\beta_t(s_j) = P(x_{t+1:T} \mid Y_t = s_j)$. This can be rearranged into the following expression:

$$\beta_t(s_j) = \sum_{k=1}^{J} \beta_{t+1}(s_k) P(x_{t+1} | Y_{t+1} = s_k) P(Y_{t+1} = s_k | Y_t = s_j) \tag{4}$$

Just like the α's, the β's can also be computed using the following dynamic programming procedure:

$\beta_{T+1}(\texttt{<END>}) = 1$
$\beta_{T+1}(s_m) = 0$ `for all` $s_m \neq$ <END>
for `t = T,...,1:`
    for `j = 1,...,J:`
        $\beta_t(s_j) = \sum_k (\beta_{t+1}(s_k) A_{k,x_{t+1}} B_{j,k})$

Note that, for both the forward and backward algorithms, $A_{k,x_{T+1}} = 1$ where $k =$ <END>, as defined in section 9.4.

Forward-Backward Algorithm As stated above, the goal of the Forward-Backward algorithm is to compute $P(Y_t = s_j \mid x_{1:T})$ up to a constant factor. This can be done using the following relation:

$$P(Y_t = s_j \mid x_{1:T}) \propto P(Y_t = s_j, x_{1:t}) P(x_{t+1:T} \mid Y_t = s_j)$$

After running your forward and backward passes through the sequence, the conditional probabilities (up to a constant factor) are:

$$P(Y_t \mid x_{1:t}) \propto \alpha_t \odot \beta_t$$

where $\odot$ is the element-wise product.

Minimum Bayes Risk Prediction We will assign tags using the minimum Bayes risk predictor, defined for this problem as follows:

$$\hat{Y}_t = \underset{j \in \{1,\ldots,J\}}{\operatorname{argmax}} P(Y_t = s_j \mid x_{1:T})$$

To resolve ties, select the tag that appears earlier in the **index_to_tag.txt** input file.

Computing the Log Likelihood of a Sequence When we compute the log likelihood of a sequence, we are interested in the computing the quantity $\log(P(x_{1:T}))$. We can rewrite this in terms of values we have already computed in the forward-backward algorithm as follows:

$$\begin{aligned}
\log P(x_{1:T}) &= \log \left(\sum_j P(x_{1:T}, Y_t = s_j) \right) \\
&= \log \left(\sum_j P(Y_t = s_j, x_{1:t}) P(x_{t+1:T} | Y_t = s_j) \right) \\
&= \log \left(\sum_j \alpha_t(s_j) \beta_t(s_j) \right)
\end{aligned}$$

In the formula above, note that the log-likelihood is the same for any choice of t; it can therefore be helpful to choose a convenient value of t that reduces computation.

9.5.2 Implementation Details

You should now write a program `forwardbackward.py` that implements the forward-backward algorithm. The program will read in validation data and the parameter files produced by `learnhmm.py`. The autograder runs and evaluates the output from the files generated, using the following command:

```
$ python forwardbackward.py [args...]
```

Where `[args...]` is a placeholder for seven command-line arguments:`<validation_input>` `<index_to_word>` `<index_to_tag>` `<hmmemit>` `<hmmtrans>` `<predicted_file>` `<metric_file>`. These arguments are described in detail below:

1. `<validation_input>`: path to the validation input `.txt` file that will be evaluated by your forward backward algorithm (see Section 9.2)
2. `<index_to_word>`: path to the `.txt` file that specifies the dictionary mapping from words to indices. The tags are ordered by index, with the first word having index of 0, the second word having index of 1, etc. This is the same file as was described for `learnhmm.py`.
3. `<index_to_tag>`: path to the `.txt` file that specifies the dictionary mapping from tags to indices. The tags are ordered by index, with the first tag having index of 0, the second tag having index of 1, etc. This is the same file as was described for `learnhmm.py`.
4. `<hmmemit>`: path to input `.txt` file which contains the emission probabilities ($\mathbf{A}$).
5. `<hmmtrans>`: path to input `.txt` file which contains transition probabilities ($\mathbf{B}$).
6. `<predicted_file>`: path to the output `.txt` file to which the predicted tags will be written. The file should be in the same format as the `<validation_input>` file.
7. `<metric_file>`: path to the output `.txt` file to which the metrics will be written.

As an example, the following command would run your program on the toy dataset provided in the handout.

```
$ python3 forwardbackward.py toy_data/validation.txt \
toy_data/index_to_word.txt toy_data/index_to_tag.txt \
toy_output/hmmemit.txt toy_output/hmmtrans.txt \
predicted.txt metrics.txt
```

After running the command above, the `<predicted_file>` output should be:

```
fish    D
eat     C
you     D
```

And the <metric_file> output should be:

```
Average Log-Likelihood: -5.100082181210599
Accuracy: 0.3333333333333333
```

where average log-likelihood and accuracy are evaluated over the validation set.

Take care that your output has the exact same format as shown above. There should be a single space after the colon preceding the metric value (e.g. a space after `Average Log-Likelihood:`). Each line should be terminated by a Unix line ending `\n`.

9.5.3 Log-Space Arithmetic for Avoiding Underflow

Handling underflow properly is a critical step in implementing an HMM. The most generalized way of handling numerical underflow due to products of small positive numbers (like probabilities) is to calculate everything in log-space, i.e., represent every quantity by their logarithm.

For this homework, using log-space starts with transforming Eq.(3) and Eq.(4) into logarithmic form - you may find the recitation handout helpful. Please use base e (natural log) for logarithm calculation.

After transforming the equations into log form, you may discover calculations of the following type:

$$\log \sum_i \exp\left(v_i\right)$$

This may be programmed as is, but $\exp\left(v_i\right)$ may cause underflow when v_i is large and negative. One way to avoid this is to use the log-sum-exp trick. We provide the pseudocode for this trick in Algorithm 1:

Algorithm 1 Log-Sum-Exp Trick

procedure LOGSUMEXP($(v_1, v_2, \cdots, v_n)$)
    $m = \max(v_i)$ for $i = \{1, 2, \cdots, n\}$
    return $m + \log(\sum_i \exp(v_i - m))$

Note: The autograder test cases account for numerical underflow using the Log-Sum-Exp Trick. If you do not implement forwardbackward.py with the trick, you will only receive partial credit.

9.6 Starter Code

To help you start this assignment, we have provided starter code in the handout.

9.7 Gradescope Submission

You should submit your `learnhmm.py` and `forwardbackward.py` to Gradescope. Please do not use other file names. This will cause problems for the autograder to correctly detect and run your code.