Programming Assignment 6: Hidden Markov Models

10-301/10-601 Introduction to Machine Learning (Summer 2023)
https://www.cs.cmu.edu/~hchai2/courses/10601/
OUT: Thursday, July 20th
DUE: Thursday, July 27 at 11:59 PM
TAs: Alex, Andrew, Sofia, Tara, Markov, Neural the Narwhal

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:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., "Jane explained to me what is asked in Question 2.1"). Second, write your solution *independently*: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only. See the Academic Integrity Section on the course site for more information: https://www.cs.cmu.edu/~hchai2/courses/10601/
- **Late Submission Policy:** See the late submission policy here: https://www.cs.cmu.edu/~hchai2/courses/10601/
- **Submitting your work:**
  - **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). **You are only permitted to use the Python Standard Library modules and `numpy`**. When you are developing, check that the version number of the programming language environment (e.g. Python 3.9.12) and versions of permitted libraries (`numpy` 1.23.0) match those used on Gradescope. You have a **total of 10 Gradescope programming submissions.** Use them wisely. In order to not waste code submissions, we recommend debugging your implementation on your local machine (or the linux servers) and making sure your code is running correctly first before any Gradescope coding submission.
  - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. You must typeset your submission using LaTeX. If your submission is misaligned with the template, there will be a **2% penalty** (e.g., if the homework is out of 100 points, 2 points will be deducted from your final score). Each derivation/proof should be completed in the boxes provided. Do not move or resize any of the answer boxes. If you do

not follow the template, your assignment may not be graded correctly by our AI assisted grader.

For multiple choice or select all that apply questions, shade in the box or circle in the template document corresponding to the correct answer(s) for each of the questions. For LaTeX users, replace `\choice` with `\CorrectChoice` to obtain a shaded box/circle, and don't change anything else.

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-~~6~~301

Written Questions (32 points)

1 Forward-Backward Algorithm (26 points)

The following questions should be completed before you start the programming component of this assignment. To help you prepare to implement the HMM forward-backward algorithm (see Section 4.5 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 tags, 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 question, we define:

- Each observed state $x_t \in \{1, 2, 3\}$, where 1 corresponds to `you`, 2 corresponds to `eat`, and 3 corresponds to `fish`
- Each hidden state $Y_t \in \{C, D\}$. Let $s_1 = C$ and $s_2 = D$.
- $\mathbf{B}$ is the transition matrix, where $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$. Here $\mathbf{B}$ is a 2×2 matrix.
- $\mathbf{A}$ is the emission matrix, where $A_{jk} = P(X_t = k \mid Y_t = s_j)$. Here $\mathbf{A}$ is a 2×3 matrix. As an example, A_{23} denotes $P(X_t = 3 \mid Y_t = s_2)$, or the probability that X_t corresponds to `fish` given the hidden state $Y_t = D$.
- $\boldsymbol{\pi}$ describes Y_1's initialization probabilities: $\pi_j = P(Y_1 = s_j)$.

  For pseudo-code of the Forward-Backward Algorithm, refer to 4.5.

Note: Pseudocounts used in section 4.4 should also be used here.

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. Only your answer in the left box will be graded.

1. (5 points) Compute $\alpha_2(C)$, the α value associated with the tag "C" for the second word in the validation sequence.

$\alpha_2(C)$	Work

2. (3 points) Compute $\beta_2(D)$, the β value associated with the tag "D" for the second word in the validation sequence.

$\beta_2(D)$	Work

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

Tag	Work

4. (3 points) Compute the log-likelihood for the entire validation sequence, "`fish eat you`", using e as the log base.

Log-Likelihood	Work

5. (6 points) In the forward algorithm, recall that the entries in $\boldsymbol{\alpha}$ can be computed using the following dynamic programming algorithm:

```
$\alpha_1(s_j) = \pi_j \cdot A_{s_j,x_1}$ for all $s_j$
For $t = 1, 2, 3, \ldots, T$
    $\alpha_t(s_j) = A_{s_j,x_t} \sum_k B_{s_k,s_j} \alpha_{t-1}(s_k)$ for all $s_j$
```

To implement the forward algorithm in log-space, we can derive $\log\big(\alpha_t(s_j)\big)$ in terms of $\log(\boldsymbol{\alpha}_{t-1})$, $\log \mathbf{B}$ and $\log \mathbf{A}$:

$$\begin{aligned}
&\log\big(\alpha_t(s_j)\big)\\
&= \log\big(A_{j,x_t} \textstyle\sum_k B_{kj} \alpha_{t-1}(s_k)\big)\\
&= \log(A_{j,x_t}) + \log\big(\textstyle\sum_k B_{kj} \alpha_{t-1}(s_k)\big)\\
&= \log(A_{j,x_t}) + \log\Big(\textstyle\sum_{k=1}^J e^{\log\left(\alpha_{t-1}(k) B_{kj}\right)}\Big)\\
&= \log(A_{j,x_t}) + \log\Big(\textstyle\sum_{k=1}^J e^{\log\left(\alpha_{t-1}(k)\right) + \log\left(B_{kj}\right)}\Big)
\end{aligned}$$

In the backward algorithm, we also have a similar algorithm:

```
$\beta_T(s_j) = 1$ for all $s_j$
For $t = T-1, T-2, T-3, \ldots, 1$
    $\beta_t(s_j) = \sum_k \beta_{t+1}(s_k) A_{s_k,x_{t+1}} B_{s_j,s_k}$
```

We can also derive the backward algorithm in log-space by finding $\log\big(\beta_t(j)\big)$ in terms of $\log(\boldsymbol{\beta}_{t+1})$, $\log \mathbf{B}$ and $\log \mathbf{A}$:

$$\begin{aligned}
&\log\big(\beta_t(s_j)\big)\\
&= \log\big(\textstyle\sum_k (\beta_{t+1}(s_k) A_{kx_{t+1}} B_{jk})\big)\\
&= \log\Big(\textstyle\sum_k e^{\log\left(\beta_{t+1}(s_k) A_{kx_{t+1}} B_{jk}\right)}\Big)\\
&= \log\Big(\textstyle\sum_{k=1}^J e^{\log(A_{kx_{t+1}}) + \log\left(\beta_{t+1}(k)\right) + \log(B_{jk})}\Big)
\end{aligned}$$

Note how the above equations include terms that look like $\log \sum_i \exp(v_i)$. In your programming section, you shouldn't program this as-is, because the exp term will underflow when v_i is very negative. You'll instead implement this using a trick known as the log-sum-exp trick. For this written problem, you may assume access to a `logsumexp` function that takes in a matrix H and returns a vector v where $v_i = \log \sum_j \exp(H_{ij})$. In other words, `logsumexp` performs the log-sum-exp trick on each row of H.

Now it's time to vectorize the log-space equations above! For each of the following quantities, write one line of code to compute them from the given quantities. You are given that $\log \boldsymbol{\alpha}_t$ is a column vector which is the element-wise log of $\boldsymbol{\alpha}_t$, $\log \boldsymbol{\beta}_t$ is a column vector which is the element-wise log of

$\boldsymbol{\beta}_t$, $\log A$ is the element-wise log of the emission matrix, $\log B$ is the element-wise log of the transition matrix, $\log \boldsymbol{\pi}$ is the element-wise log of the initial probabilities, $\mathbf{x}$ is the input sequence (a list of words represented as matrix indices), and the `logsumexp` function as described above. You may only write your answer in terms of these quantities. You may assume that NumPy has been imported as `np` and that broadcasting is allowed (e.g. if M is an $a \times b$ matrix and v is a vector of length b, then $M + v$ will add v element-wise to every row of M).

(a) (2 points) $\log \boldsymbol{\alpha}_1$

Your Answer

```
% YOUR ANSWER
```

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

Your Answer

```
% YOUR ANSWER
```

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

Your Answer

```
% YOUR ANSWER
```

2 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 4.5.3 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 and average log likelihood across all sequences from the English **train.txt** and the **validation.txt** on the y-axis (see Section 4.5 for details on computing the log data likelihood for a sequence). Please use **log-scale** with base 10 for the x-axis.

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.

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

2. (2 points) Put your plot below:

Plot

3 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

4 Programming (98 points)

4.1 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.

4.2 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.

4.3 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 4.4). 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.

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. This index will also correspond to locations in the parameter matrices. For example, the word *Austria* corresponds to the parameters in column 729 of the matrix stored in **hmmemit.txt**. This will be useful for your forward-backward algorithm implementation (see Section 4.5).

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**.

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 detail.

6. **hmmtrans.txt, hmmemit.txt, hmminit.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 4.4, 4.5). The formats of the first two files are the same; each line in these files

consists of a conditional probability distribution. In the case of transition probabilities, this distribution corresponds to the probability of transitioning into another state, given a current state. Similarly, in the case of emission probabilities, this distribution corresponds to the probability of emitting a particular symbol, given a current state. 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...
```

hmmemit.txt:

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

The format of **hmminit.txt** is similarly defined except that it only contains a single probability distribution over starting states. Therefore, each row only has a single element.

hmminit.txt:

```
<ProbS1>\n
<ProbS2>\n...
```

4.4 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 4.5). There are three sets of parameters that you will need to estimate: the initialization probabilities $\boldsymbol{\pi}$, 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 $\pi_j = P(Y_1 = s_j)$, $B_{jk} = P(Y_t = s_k \mid Y_{t-1} = s_j)$, and $A_{jk} = P(X_t = k \mid Y_t = s_j)$. These can be estimated using maximum likelihood, which results in the following parameter estimates:

1. $P(Y_1 = s_j) = \pi_j = \frac{N_{Y_1=s_j}+1}{\sum_{p=1}^{J}(N_{Y_1=s_p}+1)}$, where $N_{Y_1=s_j}$ equals the number of times state s_j is associated with the first word of a sentence in the training data set.

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

3. $P(X_t = k \mid Y_t = s_j) = A_{jk} = \frac{N_{X_t=k,Y_t=s_j}+1}{\sum_{p=1}^{M}(N_{X_t=p,Y_t=s_j}+1)}$, where $N_{X_t=k,Y_t=s_j}$ is the number of times that the state s_j is associated with the word 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 cases during evaluation of your validation set.

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

Your outputs should be in the same format as **hmminit.txt**, **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:

```
$ python3 learnhmm.py [args...]
```

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

1. `<train_input>`: path to the training input `.txt` file (see Section 4.2)
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. `<hmminit>`: path to output `.txt` file to which the estimated initialization probabilities ($\boldsymbol{\pi}$) will be written. The file output to this path should be in the same format as the handout `hmminit.txt` (see Section 4.2).
5. `<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 4.2)
6. `<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 4.2).

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

```
$ python3 learnhmm.py toy_data/train.txt toy_data/index_to_word.txt \
toy_data/index_to_tag.txt toy_data/hmminit.txt toy_data/hmmemit.txt \
toy_data/hmmtrans.txt
```

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

4.5 Evaluation and Decoding

4.5.1 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, $x_1, \dots, x_T$. Each word is associated with a label $Y_t \in \{1, \dots, J\}$. In the forward-backward algorithm we seek to approximate $P(Y_t \mid x_{1:T})$ up to a multiplication constant. This is done by first breaking $P(Y_t \mid x_{1:T})$ into a "forward" component and a "backward" component as follows:

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

where $P(Y_t = s_j \mid x_1, \dots, x_t)$ and $P(x_{t+1}, \dots, x_T \mid Y_t = s_j)$ are computed by bottom-up dynamic programming approach.

Forward Algorithm

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

$$\alpha_t(s_j) = A_{jx_t} \sum_k B_{kj} \alpha_{t-1}(k) \quad (1)$$

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

```
for t = 1,...,T:
    for j = 1,...,J:
        if t == 1:
            $\alpha_1(s_j) = \pi_j * A_{j,x_1}$
        else:
            $\alpha_t(s_j) = A_{j,x_t} * \sum_k (\alpha_{t-1}(s_k) * B_{k,j})$
```

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:

$$\beta_t(s_j) = \sum_{k=1}^{J} A_{kx_{t+1}} \beta_{t+1}(s_k) B_{jk} \quad (2)$$

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

```
for t = T,...,1:
    for j = 1,...,J:
        if t == T:
            $\beta_T(s_j) = 1$
        else:
            $\beta_t(s_j) = \sum_k (A_{k,x_{t+1}} \beta_{t+1}(s_k) B_{j,k})$
```

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

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

After running your forward and backward passes through the sequence, you are now ready to estimate the conditional probabilities as:

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

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:

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

To resolve ties, select the tag that appears earlier in the `<index_to_tag>` 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:

$$\begin{aligned}\log P(x_{1:T}) &= \log\big(\sum_j P(x_{1:T}, Y_t = s_j)\big)\\ &= \log\big(\sum_j \alpha_T(s_j)\big)\end{aligned}$$

4.5.2 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:

```
$ python3 forwardbackward.py [args...]
```

Where `[args...]` is a placeholder for eight command-line arguments:`<validation_input>` `<index_to_word>` `<index_to_tag>` `<hmminit>` `<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 4.2)
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. `<hmminit>`: path to input `.txt` file which contains the estimated initialization probabilities ($\boldsymbol{\pi}$).
5. `<hmmemit>`: path to input `.txt` file which contains the emission probabilities ($\mathbf{A}$).
6. `<hmmtrans>`: path to input `.txt` file which contains transition probabilities ($\mathbf{B}$).
7. `<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.
8. `<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_data/hmminit.txt toy_data/hmmemit.txt \
toy_data/hmmtrans.txt toy_data/predicted.txt \
toy_data/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: -3.0438629330222424
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`.

4.5.3 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.(1) and Eq.(2) 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:

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

This may be programmed as is, but $\exp(v_i)$ 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

```
1: procedure LogSumExpTrick($(v_1, v_2, \cdots, v_n)$)
2:     $m = \max(v_i)$ for $i = \{1, 2, \cdots, n\}$
3:     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 might only receive partial credit.

4.6 Gradescope Submission

You should submit your `learnhmm.py` and `forwardbackward.py` to Gradescope. **Any other files will be deleted.** Please do not use other file names. This will cause problems for the autograder to correctly detect and run your code. Please go through the appendix at the end for information on starter-code.

Some additional tips: Make sure to read the autograder output carefully. The autograder for Gradescope prints out some additional information about the tests that it ran. For this programming assignment we've specially designed some buggy implementations that you might implement and will try our best to detect those and give you some more useful feedback in Gradescope's autograder. Make wise use of autograder's output for debugging your code.

Note: For this assignment, you have 10 submissions to Gradescope before the deadline, but only your last submission will be graded.