COS 484: Natural Language Processing **(Due: 03/03/2025)**

Assignment #2

Instructors: Danqi Chen, Tri Dao, Vikram Ramaswamy

Course Policy: Read all the instructions below carefully before you start working on the assignment and before you make a submission. The course assignment policy is available at https://princeton-nlp.github.io/cos484/. When you're ready to submit, please follow the instructions found here: http://bit.ly/COS_NLP_Submission

- This assignment contains 2 parts, a theoretical and a programming part. The former consists of 3 problems, and the latter has 2, for a total of **5 problems**.
- We *highly* recommended that you typeset your submissions in LaTeX. Use the template provided on the website for your answers. If you have never used LaTeX, you can refer to the short guide here: http://bit.ly/WorkingWithLaTeX. Include your name and NetIDs with your submission. If you wish to submit hand-written answers, you can scan and upload the pdf.
- Assignments must be uploaded to Gradescope **before class (01:59pm Eastern)** on the due date mentioned above.
- As per the late-day policy outlined on the course website, you have 4 late days to use at your discretion throughout the semester. Once you run out of late days, late submissions will incur a penalty of 10% for each day, up to a maximum of 3 days beyond which submissions will not be accepted.
- All programming problems in this class will be completed in Google Colab using Python. If you would like to get familiar with this environment, you may complete the problems in this introductory Colab notebook (This will not be graded). If you've never worked with Google Colab before, take a look through this introduction guide: http://bit.ly/WorkingWithColab. **The answers to the written questions proposed in the programming part should be answered in the Colab notebook.**

Theoretical Part

Submission Policy: Submit a single PDF for the answers to all questions in this part.

Problem 1: Hidden Markov Models for Tagging (13 points)

We are interested in labeling each word in the following corpus of text with a set of *sentiment tags*:

```
the bread smells delicious
the coffee was bitter and awful
```

The set of tags available to us is $\{+, -, O\}$, representing positive sentiment, negative sentiment and neutral sentiment, respectively. We assume Markov assumption and output independence, i.e. each tag only depends on the previous tag and each word only depends on its corresponding tag. This results in a hidden Markov model (HMM) over the tags and words. We are also given the following parameters of the HMM:

	and	**awful**	**bitter**	**bread**	**coffee**	**delicious**	**smells**	**the**	**was**
+	0.0	0.0	0.1	0.05	0.05	0.7	0.05	0.0	0.05
-	0.0	0.7	0.15	0.0	0.05	0.0	0.05	0.0	0.05
O	0.2	0.05	0.05	0.1	0.1	0.05	0.05	0.2	0.2

Table 1: Emission probabilities, ϕ. Rows denote hidden tags y_j and columns are observations x_j.

	+	-	O
$\emptyset$	0.3	0.3	0.4
+	0.4	0.2	0.4
-	0.1	0.5	0.4
O	0.2	0.2	0.6

Table 2: Transition probabilities, θ. Rows denote tags y_j and columns are over y_{j+1}. Transition from null state represents the initial probability of a state, i.e. $\theta_{\emptyset \to +} = \pi(+)$.

(a) (4 points) Write down the formula to compute the ***joint probability*** of the tag sequence $y = \langle O, O, O, + \rangle$ and the sentence $x = \langle$`the bread smells delicious`$\rangle$ given the above parameters.

(b) (9 points) Given a word sequence $x = \langle$`coffee smells bitter`$\rangle$, what is the *least likely* tag sequence for this text using Viterbi decoding? Show your steps.

Problem 2: Understanding word2vec (10 points)

Given a sequence of words $w_1, \ldots, w_T$ and context size c, the training objective of skip-gram that we learned in the class is:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log P(w_{t+j} \mid w_t),$$

where $P(w_o \mid w_t)$ is defined as:

$$P(w_o \mid w_t) = \frac{\exp\left(\mathbf{u}_{w_t}^{\intercal}\mathbf{v}_{w_o}\right)}{\sum_{k\in V}\exp\left(\mathbf{u}_{w_t}^{\intercal}\mathbf{v}_k\right)},$$

where $\mathbf{u}_k$ represents the "target" vector and $\mathbf{v}_k$ represents the "context" vector, for every $k \in V$.

(a)(5 points) Derive the following gradient (probability w.r.t context vector):

$$-\frac{\partial \log P(w_o \mid w_t)}{\partial \mathbf{v}_{w_o}}$$

(b)(5 points) Assume we train this model on a large corpus (e.g. English Wikipedia). Describe at least two effects of choosing different context sizes c for training the word vectors $\mathbf{u}_w$, e.g. what would you expect if we used context size $c = 1$, 5, or 100?

Problem 3: Named entity recognition with neural networks (14 points)

Let us consider a simple neural network model that can be used for named entity recognition (NER). We can have the network predict a label for each token separately using features from a window around it.

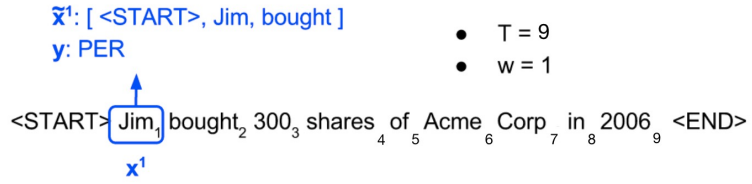

Figure 1: An example of an input sequence and the first window ($\tilde{\mathbf{x}}_1$) from this sequence.

Let $\mathbf{x} \stackrel{\text{def}}{=} \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ be an input sequence of length T and $\mathbf{y} \stackrel{\text{def}}{=} \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$ similarly be an output sequence. Here, $\mathbf{x}_t \in \mathbb{R}^{1\times|V|}$ is a one-hot (row) vector[1] representing the t-th word for a vocabulary V. $\mathbf{y}_t \in \mathbb{R}^{1\times C}$ is also a one-hot row vector representing the ground truth label for $\mathbf{x}_t$, where C is the number of labels.

In a window-based classifier, every input sequence is split into T new data points, each representing a window and its label. Each data point is constructed using a window of size $2w+1$ centered at $\mathbf{x}_t$: $\tilde{\mathbf{x}}_t \stackrel{\text{def}}{=} [\mathbf{x}_{t-w}, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_{t+w}]$. We pad special start tokens (`<START>`) to the beginning of the sentence and special end tokens (`<END>`) to the end of the sentence when constructing windows centered at starting and ending tokens. For example, a window of size 1 around "Jim" in the sentence above would be [`<START>`, `Jim`, `bought`], and a window of size 2 would be [`<START>`, `<START>`, `Jim`, `bought`, `300`].

We can use a simple feedforward neural net that uses a word embedding layer, a single hidden layer with a `Sigmoid` activation, a `Softmax` output layer and the cross-entropy loss to predict $\mathbf{y}_t$ from $\tilde{\mathbf{x}}_t$:

$$\begin{aligned}
\mathbf{e}_t &= [\mathbf{x}_{t-w}E, \ldots, \mathbf{x}_t E, \ldots, \mathbf{x}_{t+w}E] \\
\mathbf{h}_t &= \texttt{Sigmoid}(\mathbf{e}_t W^\top + \mathbf{b}_1) \\
\hat{\mathbf{y}}_t &= \texttt{Softmax}(\mathbf{h}_t U^\top + \mathbf{b}_2) \\
J &= \texttt{CE}(\mathbf{y}_t, \hat{\mathbf{y}}_t) = -\sum_i y_{t,i} \log(\hat{y}_{t,i}),
\end{aligned}$$

Here, $E \in \mathbb{R}^{|V|\times D}$ are word embeddings which are learned during training, where D is the size of the word embedding. Note that as $\mathbf{x}_t$ is a one-hot vector, $\mathbf{x}_t E$ denotes the row vector of the embedding matrix that corresponds to the embedding of $\mathbf{x}_t$. $\mathbf{h}_t \in \mathbb{R}^{1\times H}$, where H is the size of the hidden layer, and $\hat{\mathbf{y}}_t \in \mathbb{R}^{1\times C}$, where $C = 5$ is the number of classes being predicted: PER (person), ORG (organization), LOC (location), MISC (miscellaneous), and O (non-entity).

Note that the notations we used here are slightly different from what we have seen in the class, in particular, we use $\mathbf{h}U^\top$ (here h is a row vector) instead of $U\mathbf{h}$ (here h is a column vector). This is for better compatability with the Pytorch convention that you will implement in the programming problems.

(a) (2 points) What are the parameters of this neural network that can be learned from the training data?

(b) (3 points) What are the dimensions of $\mathbf{e}_t$, W and U if we use a window of size w?

(c) (5 points) What is the computational complexity of predicting labels for a sentence of length T? Please write down all the different operations performed by the neural network during one forward pass and the computational complexity for each of them. Then sum them up to get the complexity of predicting labels for this sentence. You may provide your answer in big $\mathcal{O}$ notation and in terms of the variables defined in the problem (T, C, V, D, H, w).
To simplify things, assume that we can get the word embedding $\mathbf{x}_t E$ by table lookup instead of matrix multipli-

[1] In general, the term "one-hot" means that the vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the word.

cation (since $\mathbf{x}_t$ *is a one-hot vector), which can be considered as an operation with* $\mathcal{O}(1)$ *complexity.*

(d) (4 points) Describe at least 2 modeling limitations of this window-based FFN model for NER.

Programming Part

Submission Policy: The answers to the written questions for this part should be answered in the Colab notebook. **Do not include the answers to this part in the same PDF as the theory part.**

Problem 1: Hidden Markov Model for Named Entity Recognition (30 points)

In the following programming problems, you are going to implement models for the Named Entity Recognition (NER) task. NER is the task to associate the words in a sentence with their proper name tags. For example, "Marie Curie" may correspond to the tag `PER` (person) and "Princeton University" may correspond to the tag `ORG` (organization). In this programming assignment, will use a total of 5 tags: `PER` (person), `ORG` (organization), `LOC` (location), `MISC` (miscellaneous), and `O` (non-entity). For example, the correct tagging of the sentence "Steve Jobs founded Apple with Steve Wozniak ." is ⟨`PER`, `PER`, `O`, `ORG`, `O`, `PER`, `PER`⟩. Note that when consecutive words constitute a named entity, such as "Steve Jobs" in the previous example, they should both be tagged as `PER`.

In programming problem 1, you will implement the hidden Markov model (HMM) for this task.

(a) (8 points) First, implement a bigram hidden markov model (HMM) and estimate its parameters using the training data. Complete the TODOs in the `train()` function of the HMM model to update the initial state, transition and emission probabilities using the train corpus.

The default code performs greedy decoding using your HMM. Report the accuracy you observe on the training and validation sets. Also report the F-1 score you obtain for each of the 5 classes as well as the confusion matrix between the different classes *for the validation set*. Which pair of tags does the model have most difficulty separating according to the confusion matrix of the validation set?

(b) (8 points) Now, complete the implementation of Viterbi decoding for the HMM model in the function `viterbi_decode()`.

Report the train and validation accuracies. Also report F-1 scores per class and the confusion matrix *for the validation set*. What major differences do you observe compared to the matrix in (a)?

Link to notebook: Colab notebook.

Problem 2: Max-Entropy Markov Model for Named Entity Recognition (25 points)

The task is the same as the programming problem above. In programming problem 2, you will implement the MEMM model for this task.

(a) (6 points) Complete the `extract_feature` function in the `MEMM` class. We've implemented adding word features to the function, but we want you to add one-hot encoding features for the prior 4 tags. (HINT: You can use MEMM's `self.tag_encoder` to help generate one-hot vectors)

(b) (6 points) HMMs depend on the count statistics of the training data to generate predictions. However, there may be other clues that are useful for NER. For example, words that are single characters followed by a period ("K.") might represent an inital in someone's name. While HMMs cannot deal with these types of features in general, MEMMs are much more flexible in feature representations. For the MEMM model, add *at least* 6 new features to `regex_features_2` to improve model performance. You should be able to improve overall test accuracy to around 88% or higher.

Link to notebook: Colab notebook.