COS 484: Natural Language Processing **(Due: 03/01/2024)**

Assignment #2

Instructors: Karthik Narasimhan

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 http://nlp.cs.princeton.edu/cos484. When you're ready to submit, please follow the instructions found here: http://bit.ly/COS_NLP_Submission

- This assignment contains 5 problems. 3 theory problems and 2 programming problems.
- We *highly* recommend that you typeset your submissions in LaTeX. Use the template provided on the website for your answers. If you've 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 (12:00pm Eastern)** on the due date mentioned above.
- As per the late day policy outlined on the course website, you have 96 allowed late hours (about 4 days) overall to use at your discretion throughout the semester. Once you you run out of late hours, 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.
- Assignment submission guidelines:
  https://princeton-nlp.github.io/cos484/assignments/logistics.pdf
- For the programming questions, you only need to complete the coding part specified in the `.ipynb` notebook. You need to compress the two `.ipynb` files for programming problem 1 and 2 into a single `.zip` file and upload this one file. Make sure the cells are properly executed and have the outputs printed.

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	0.1	0.1	0.6	0.1	0.0	0.0
-	0.0	0.45	0.3	0.0	0.05	0.0	0.2	0.0	0.0
O	0.15	0.05	0.05	0.1	0.1	0.05	0.05	0.25	0.2

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

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

Problem 2: Understanding word2vec

(5 + 5 = 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) 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) 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.

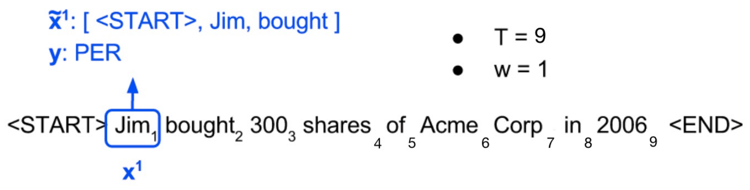
x̃¹: [<START>, Jim, bought]
y: PER
• T = 9
• w = 1
<START> Jim₁ bought₂ 300₃ shares₄ of₅ Acme₆ Corp₇ in₈ 2006₉ <END>
x¹

Figure 1: A sample input sequence.

Figure 1 shows an example of an input sequence and the first window ($\tilde{\mathbf{x}}_1$) from this sequence. Let $\mathbf{x} \overset{\text{def}}{=} \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ be an input sequence of length T and $\mathbf{y} \overset{\text{def}}{=} \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$ be an output sequence, also of length T. Here, $\mathbf{x}_t \in \mathbb{R}^{1\times|V|}$ is a one-hot (row) vector representing the word at the t-th index of the sentence, for a vocabulary V. 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. Therefore, $\mathbf{x}_t E$ denotes the row vector of the embedding matrix, which is the embedding of that word.

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 around $\mathbf{x}_t$ by concatenating w tokens each to the left and right of $\mathbf{x}_t$: $\tilde{\mathbf{x}}_t \overset{\text{def}}{=} [\mathbf{x}_{t-w}, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_{t+w}]$. $\mathbf{y}_t \in \mathbb{R}^{1\times C}$ is also a one-hot row vector representing the ground truth label for $\mathbf{x}_t$, where C is the number of labels.

For windows centered around tokens at the very beginning of a sentence, we add special start tokens (`<START>`) to the beginning of the window and for windows centered around tokens at the very end of a sentence, we add special end tokens (`<END>`) to the end of the window. For example, consider constructing a window around "Jim" in the sentence above. If window size were 1, we would add a *single* start token to the window (resulting in a window of [`<START>`, `Jim`, `bought`]). If window size were 2, we would add *two* start tokens to the window (resulting in a window of [`<START>`, `<START>`, `Jim`, `bought`, `300`]).

Therefore, each input instance is a vector of the same length and we can use a simple feedforward neural net to predict $\mathbf{y}_t$ from $\tilde{\mathbf{x}}_t$: As a simple but effective model to predict labels from each window, we will use a word embedding layer, a single hidden layer with a `Sigmoid` activation, combined with a `Softmax` output layer and the cross-entropy loss:

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

where $E \in \mathbb{R}^{|V|\times D}$ are word embeddings which are learned during training, $\mathbf{h}_t \in \mathbb{R}^{1\times H}$, and $\hat{\mathbf{y}}_t \in \mathbb{R}^{1\times C}$, where V is the vocabulary, D is the size of the word embedding, H is the size of the hidden layer, and C is the number of classes being predicted. Here $C = 5$ - 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 multiplication (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 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 $\langle \texttt{PER}, \texttt{PER}, \texttt{O}, \texttt{ORG}, \texttt{O}, \texttt{PER}, \texttt{PER} \rangle$. 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.

Link to notebook: Colab notebook.

Programming 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.

Link to notebook: Colab notebook.