COS 484: Natural Language Processing **(Due: 03/03/2026)**

Assignment #2

Instructors: Tri Dao, 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 `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 by **11:59pm Eastern** on the due date mentioned above.
- As per the late-day policy outlined on the course website, you have 4 late days that you can use any time during the semester, with at most 3 late days per assignment. 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 should 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: Working With Colab. **The answers to the written questions proposed in the programming part should be answered in your Colab notebook.**
- **LLM usage policy**: You **may not** consult a Large Language Model (LLM) when working on the **Theoretical** part of this assignment. For the **Programming** Part only, you may use coding assistants (like GitHub Copilot, Cursor, etc.) for writing code. If you do use such assistants, include the prompts you used at the end of your 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 (14 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

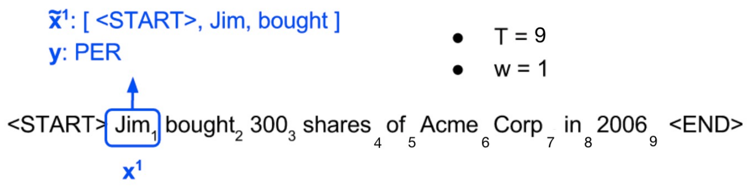
Table 2: Transition probabilities, θ. Rows denote tags y_j and columns are over y_{j+1}. Transition from null state represents the initial probability of a state, i.e. $\theta_{\emptyset \to +} = \pi(+)$.

(a) (5 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: Named entity recognition with neural networks (16 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.



$\tilde{\mathbf{x}}^1$: [<START>, Jim, bought]
y: PER
T = 9
w = 1
<START> Jim$_1$ bought$_2$ 300$_3$ shares$_4$ of$_5$ Acme$_6$ Corp$_7$ in$_8$ 2006$_9$ <END>
$\mathbf{x}^1$

Figure 1: 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 $\mathbf{y} \overset{\text{def}}{=} \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$ 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 $\mathbf{x}_t$: $\tilde{\mathbf{x}}_t \overset{\text{def}}{=} [\mathbf{x}_{t-w}, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_{t+w}]$. 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 3 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) (3 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 $2w+1$?

(c) (6 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.

Problem 3: Language modeling with recurrent neural networks (15 points)

In this problem, we will consider a recurrent neural network for language modeling. Given a sequence of one-hot input vectors $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n$ (see definition of one-hot vectors in A2P3), our RNN calculates the following at each timestep t:

$$\begin{aligned}\mathbf{e}_t &= \mathbf{x}_t E \\ \mathbf{h}_t &= \tanh(\mathbf{h}_{t-1} U^\top + \mathbf{e}_t W^\top) \\ \mathbf{y}_t &= \mathrm{softmax}(\mathbf{h}_t W_O^\top)\end{aligned}$$

Let us assume that $\mathbf{x}_t \in \mathbb{R}^{1 \times |V|}$ where V is vocabulary, $\mathbf{e}_t \in \mathbb{R}^{1 \times D}$, $\mathbf{h}_t \in \mathbb{R}^{1 \times H}$, and $\mathbf{y}_t \in \mathbb{R}^{1 \times |V|}$. As in prior homework, E is a word embedding matrix which is learned during training. However, now we have introduced an additional word embedding W_O for the output of the language model.

(1) (3 points) How many trainable parameters does our RNN model have?

(2) (5 points) Write down the expression for the gradient $\frac{\partial \mathbf{y}_3}{\partial E}$ in terms of the gradients corresponding to each computation of the RNN, using backpropagation through time, i.e. in terms of gradients of the form $\frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t}$, $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$, $\frac{\partial \mathbf{h}_t}{\partial \mathbf{e}_t}$, $\frac{\partial \mathbf{e}_t}{\partial \mathbf{E}}$.

(3) (3 points) In practice, it is common to enforce that the input and output word embeddings matrices are the same e.g. let $E = W_O$ (assume that $H = D$). Can you suggest a reason the weight tying might be beneficial?

(4) (4 points) A common issue with recurrent neural networks is that they exhibit vanishing gradients (i.e., going to zero) or exploding gradients (i.e., going to very large numbers) as the sequence length increases. We consider a simplified setting where this may occur.
Suppose that $H = D = 2$, $\mathbf{e}_t = 0$, for all timesteps t. Also suppose that for simplicity, instead of *tanh* we use *ReLU* - in other words, we have:

$$\mathbf{h}_t = \mathrm{ReLU}(\mathbf{h}_{t-1} U^\top),$$

where

$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0. \end{cases}$$

Can you give examples of nonzero $U \in \mathbb{R}^{2 \times 2}$ which are diagonal matrices of positive numbers, and for which the Jacobian $\frac{\partial \mathbf{h}_t}{\partial U}$

(i) has terms which all tend towards 0

(ii) has the most terms twhich tend towards infinity

as t becomes larger? You may assume that all values in $\mathbf{h}_0$ are positive.

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) (6 points) First, implement the data loading logic for HMMs. You may want to download the specified file and inspect the content to get a better understanding of its structure. You will also write logic for mapping words and tags to ids as well as handling unknown words.

(b) (10 points) Second, implement a bigram hidden markov model (HMM) and estimate its parameters using the training data. You also need to implement two decoding algorithms for HMMs: greedy decoding and Viterbi decoding (refer to the lecture slides for details).

(c) (8 points) Now, implement the training and evaluation logic for HMMs. Your evaluation should support using both decoding algorithms and report the following: accuracy, F-1 score, and confusion matrix of the predicted tags.

First run the HMM with greedy decoding. 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?

Then, run the HMM with Viterbi, report the accruacy and F-1 scores. Also observe the confusion matrix, what major differences do you observe compared to the previous matrix with greedy decoding?

Link to notebook: Colab notebook.

Problem 2: NER with Long Short-term Memory (25 points)

In the following programming problems, you are going to implement an LSTM model for the Named Entity Recognition (NER) task from the previous problem. Specifically, you will have to implement the following functionalities:

(a) (5 points) First, implement the data loading logic for LSTMs. This part may look similar to what you implemented in the previous problem, but the key difference is enabling padding sequences to the same lengths as input to the LSTM and masking out the padded tokens. Refer to the PyTorch documentation as necessary.

(b) (8 points) Then, implement the LSTM model using PyTorch modules, see the notebook for the specific modules you are allowed to use. Your code should support various initialization parameters, such as both unidirectional and bidirectional models.

(c) (6 points) Finally, implement the training and evaluation loops for the LSTM models, and run experiments with different hyperparameters. Your training and evaluation loop should use GPU for acceleration, and you can take advantage of PyTorch functions for cross entropy loss calculation and PyTorch optimizers.

(d) (6 points) Answer the questions at the bottom of the notebook by editing the cells.

Link to notebook: Colab notebook.