1 Coding Assignment: Deep Maximum Entropy Markov Model (40 Points)

1.1 Problem

In this homework, you are required to implement a Deep Maximum Entropy Markov Model (DMEMM) for the targeted sentiment task using the given dataset.

1.2 Dataset Description: Open Domain Targeted Sentiment

Named Entity Recognition is an NLP task to identify important predefined named entities in the text, including people, places, organizations, and other categories.

Targeted sentiment analysis aims to capture sentiment expressed towards an entity in a given sentence. The task consists of two parts, Named Entity Recognition (identifying the entity) and identifying whether there is a sentiment directed towards the entity. For the purpose of this homework, we will focus on doing them in a collapsed way (combining sentiment and named entity recognition into one label sequence, and trying to predict that sequence).

Here is an example of the Targeted Sentiment Task:

Input (words)	I	love	Mark	Twain
Output (labels)	O	O	T-POS	T-POS

In this example, the two T-POS tagged words are the target, a Named Entity with a positive sentiment (since I is expressing love towards the target, Mark Twain). In this case, T- means the term (or target) and O means outside of any predefined terms. In this homework, we just consider predicting if a word is a target with a sentiment (T-POS, T-NEG, T-NEU) or Other. As seen in this example, in NLP, targeted sentiment problems can often be casted as sequence tagging problems. For more details regarding the Targeted Sentiment Task, you can look at the following paper: Open Domain Targeted Sentiment

In this assignment, you have to build a Deep Maximum Entropy Markov Model for the targeted sentiment task. You are provided with a folder called data, that contains the train and test files. You must use the training set to build your model and predict the labels of the test set. **You must not use the test set while training the models**. Make sure that you maintain the output format (printing results) defined in Section 1.7 for your predictions.

1.3 Model

The Deep Maximum Entropy Markov Model (DMEMM) extends the Maximum Entropy Markov Model (MEMM) seen in class by using a neural network to build the conditional probability. The formulation for MEMM is as follows:

$$P(\mathbf{y}|\mathbf{x}) = P(y_0)P(y_1|y_0, x_1)\dots P(y_n|y_{n-1}, x_n) = \prod_{i=1}^{N} P(y_i|y_{i-1}, x_i)$$

where $P(y_i|y_{i-1}, x_i) = \dfrac{\exp(\mathbf{w}^T\phi(x_i, y_i, y_{i-1}))}{\sum_{y_i}\exp(\mathbf{w}^T\phi(x_i, y_i, y_{i-1}))}$

We can use the neural network as a probability generator for the function ϕ. In this case, the model takes y_{i-1} and x_i, and predicts y_i. Thus, an embedding for each word and an embedding for each tag are required for a deep version of MEMM (DMEMM). In addition to the neural network, for inference, you need to implement the Viterbi algorithm (using Dynamic Programming) from scratch for decoding and finding the best path. For the details of this algorithm, please refer to the course slides.

1.4 Features and Word Representations

You are required to use word and tag embeddings of lower dimension as features:

1. You are required to represent your text using ALL of the following models:
   (a) Randomly initialized embeddings (you must initialize them and train them in your code).
   (b) Pre-trained Word2Vec embeddings
   (c) Any contextualized word embedding layers of your choice provided by PyTorch (e.g. BiLSTM, Transformers,etc.). You are free to experiment with these layers, and to stack them to create a more expressive model. You can use either (a) randomly initialized embedding or (b) pre-trained Word2Vec embeddings as input to your model. The output vectors of this model will then be used as contextualized word representations.

   For Option 2 (pre-trained Word2Vec), you must not submit the Word2Vec model with your submission. Instead, we have created a directory with a Word2Vec model that is accessible by everyone via ssh to data.cs.purdue.edu. The Word2Vec model is located at `/homes/cs577/hw2/w2v.bin` on `data.cs.purdue.edu`. Please make sure that your code loads the Word2Vec model from this path and then creates the embeddings. The starter code will show you how to do this. **DO NOT** submit a Word2Vec model with your submission.

2. You also need to use tag embeddings for each tag. For tag embeddings, you can use either bit-map embedding (a vector of the length as the total number of different tags, with all positions filled up with "0", except one position with "1", like $\begin{bmatrix} 0 & 0 & \dots & 1 & \dots & 0 \end{bmatrix}$, or initialize them as low dimension vector similarly as word embeddings and train them

1.5 Evaluation

Your code should output the Precision, Recall, and F1 score (see Sec 1.7). Here, we will describe how you can compute these metrics for doing this task.

Precision is defined as the percentage of named entities (targets) found by the learning system that are correct. Recall is the percentage of named entities (targets) present in the corpus that are found by the system. In this assignment, as the dataset is targeted sentiment, we are dealing with not only named entities, but also each entity has a sentiment. Thus, in the above definition, a named entity is correct only if it exactly matches with the gold standard span of the target mention and the corresponding sentiment. The F1-score is then the harmonious mean of precision and recall: (2 * precision * recall) / (precision + recall).

For an example on how to do this in the Named Entity Recognition Task (NER) which targeted sentiment task is an extension of, you can refer to the following link (accepted answer). In our case, we are looking at "exact match". In this example, since it is NER, only the entity has to match to be considered correct, but in our case the model must also correctly predict the sentiment: https://stackoverflow.com/questions/1783653/computing-precision-and-recall-in-named-entity-recognition

Please make sure to do your evaluation in this sequential way, otherwise you will be penalized.

Here are some examples to make sure your calculation is correct:

1. Ground truth: ['T-POS', 'O', 'O', 'O', 'O', 'O', 'O']
   Prediction: ['T-POS', 'T-POS', 'O', 'O', 'O', 'O', 'O']
   In this case, the first prediction is correct, the second prediction should have been 'O', but the model predicted 'T-POS', and everything else is correct. Here, we would have 1 True Positive and 2 Positives total (1 False Positive). The recall in this case would be 1.0, since it's TP/(TP+FN) = 1/1.

2. Experiment 1:
   Ground truth: ['T-NEG', 'O', 'O']
   Prediction: ['T-NEG', 'T-NEG', 'T-NEG']

Experiment 2:
Ground truth: ['T-NEG', 'T-NEU', 'O']
Prediction: ['T-NEG', 'T-NEG', 'T-NEG']

By our evaluation scheme, the first experiment would have higher recall (perfect) since the system found the only entity that was present and got the sentiment correct. However, both would have the same precision.

3. In this example, all 3 (precision/recall/F1) are the same (0.5): True: ['T-NEG', 'T-NEU', 'O'], Predicted: ['T-NEG', 'T-NEG', 'O']

   But in this one, they are not- the precision changes: True: ['T-NEG', 'T-NEU', 'O'], Predicted: ['T-NEG', 'T-NEG', 'T-NEU']

   In this one, the precision also changes: True: ['T-NEG', 'T-NEU', 'O'], Predicted: ['T-NEG', 'O', 'O']

1.6 Model tuning

You can improve your model in various ways. Here's a non-exhaustive list of ideas:

- Add a regularizer to the loss function. Here, you can also play with the coefficient of the regularizer function.
- Different learning/optimization methods
- Hyper-parameter tuning:
  - Learning rate
  - Hidden layer size
  - Word embeddings dimension size
  - Number of iterations (epochs) to train the model
- Experiment with different sequence embedding layers provided by the PyTorch library.

1.7 Running The Code

Your code should be runnable with the following command:

```
python main.py --train_file data/twitter1_train.txt --test_file \
        data/twitter1_test.txt --option 1
```

In this command, your code should train a Deep MEMM using the file `data/twitter1_train.txt`, test it `data/twitter1_test.txt`, and use Randomly Initialized Word Embeddings. Option 2 would correspond to using Pre-trained Word2Vec embeddings, and Option 3 would correspond to using your word contextualization model. For example, for Pre-trained Word2Vec, we would run:

```
python main.py --train_file data/twitter1_train.txt --test_file \
        data/twitter1_test.txt --option 2
```

Your code should output the Precision, Recall, and F1 Score. You may use a library to calculate this, but make sure you it follows the implementation in Sec 1.5. For example (note that these are just random numbers):

```
Precision: 51.89
Recall: 45.05
F1: 48.11
```

You must also save the predictions in the same format as already done in the starter code. You can examine the starter code to understand the formatting of the data. Note that the predicted tags should be saved in the same way as the true tags, as the code already does.

1.8 Packages

For this assignment, you will be using the PyTorch package for every task. Make sure that you do not use GPU. You must implement the Viterbi Dynamic programming algorithm yourself (no packages allowed for this part). If you choose, you can run your code locally. We suggeest you set up a conda environment with Python >= 3.8.10, PyTorch >= 1.9.0, and gensim to read the Word2Vec embeddings.

1.9 Tips

Some sample code can be found here. But there is a difference between CRF and MEMM: MEMM is locally normalized while CRF is globally normalized. Thus, you do not need to run the forward algorithm in the training to calculate the partition function, but only the Viterbi algorithm in the inference for decoding.

1.10 Time limit

There is no strict time limit for this assignment. However, your code should run in a reasonable amount of time (hours, not days).

2 Conceptual questions

Now, answer the following questions based on your implementation of the DMEMM.

1. State one hyper-parameter you tuned for the neural network you used to model ϕ. How did tuning these hyper-parameters change the result? Explain with learning curves. You may have tuned more than one hyper-parameter. In this section explain only one. For this question, you are not allowed to use embedding type as a hyper-parameter. (4 Points)
2. Evaluate your best neural network configuration with learning curves. Plot your train, validation, and test set performance on the same plot as the epochs increase and training progresses. What does this tell you about the model? (4 points)
3. Evaluate the difference between the three types of word representations used (randomly initialized embeddings, Word2Vec, and embedding+contextualization model).
   (a) How do they change the result? Explain with learning curves. Be sure to evaluate the accuracy of the neural network models with each word representation type. (4 points)
   (b) Why would you want to use one over the other? Be sure to state the advantages and disadvantages of each. (4 points)
4. If the model is only allowed to use the embedding of the current word when making the prediction over it, what would he model performance be? Explain. (4 points)

2.1 Submission instruction

2.1.1 Conceptual part

Upload your answers to the conceptual questions as a **typed** to Gradescope. For your pdf file, use the naming convention `username_hw#.pdf`. For example, if your username is `smith234`, then your file would be `smith234_hw2.pdf`. To make grading easier, please start a new page in your pdf file for each question. Hint: use a `\newpage` command in LaTeX after every question ends. For example, for HW2, use a `\newpage` command after each of the questions. After uploading to gradescope, mark each page to identify which question is answered on the page. (Gradescope will facilitate this.)

2.1.2 Coding Part

You need to submit your codes via Turnin. Log into `data.cs.purdue.edu` (physically go to the lab or use ssh remotely) and follow these steps:

- Place all the files in a folder named `username_hw#`. For example, if your username is `smith234`, then your folder would be named `smith234_hw2`. This naming convention is important. If the folder is not named correctly, there's no way to identify whose submission it belongs to, and **your work might not be graded**.
- Change directory to outside of `username_hw#` folder (run `cd ..` from inside `username_hw#` folder)
- Execute the following command to turnin your code:

  ```
  turnin -c cs577 -p hw2Spring2021 username_hw#
  ```

- To overwrite an old submission, simply execute this command again.
- To verify the contents of your submission, execute this command:

  ```
  turnin -v -c cs577 -p hw2Spring2021.
  ```

  Do not forget the `-v` option, else your submission will be overwritten with an empty submission.