EECS595: Natural Language Processing
Homework 2 Programming Assignment

Assigned date: 9/17/2021 ==== Due date: 10/1/2020

September 22, 2021

In this programming assignment, you will implement a part-of-speech (POS) tagger using a Recurrent Neural Network (RNN). **If you are new to deep learning coding, start early!**

1 Data Set

You are provided with 3 data files. **You will need to write code to preprocess them.**

1. `wsj1-18.training`. This file contains the training data where each word is annotated with a POS tag. You will train your models using this data.
2. `wsj19-21.testing`. This file contains the testing data. You will apply the trained model to predict POS tags for each word.
3. `wsj19-21.truth`. This file contains ground-truth POS tags for the testing data. You will compare the results returned by your model with the ground-truth tags to evaluate model performance. You will need to report the accuracy of your model.

The data files can be downloaded from the assignment directory on CANVAS. **Please do not share the data**, as this is part of the Penn Treebank, which is licensed by Linguistic Data Consortium. The format of the data is straightforward. Each word is followed by a tag. Punctuation marks are considered as words.

Note that in the testing data: `wsj19-21.testing`, infrequent words have already been replaced by the special token `UNKA`. During training, you should map words occurring less than 3 times in the training data to the special token `UNKA`.

2 RNN

For this assignment, the goal is to get you familiar with deep learning programming in PyTorch. **Please use the Python 3.8 and PyTorch 1.9 versions for consistency.** You are free to use any variations of RNN (e.g., LSTM, etc.) to implement your POS tagger. You are also free to use any help you find online. **However, you cannot directly copy the code from online resources; you need to write your own code.**

2.1 Hyper-parameter Tuning

In deep learning research, we typically experiment with several *hyper-parameters* when building models to ensure we achieve the best performance. **You must experiment with at least 3 different hyper-parameters**. For example, consider varying the following:

- Different word embeddings (dimension of embedding, pre-trained vs. learned)
- Size of the hidden layer (i.e., vector dimensions in the hidden layer)
- Nonlinear activation functions (e.g., ReLU vs. tanh)
- Types of optimizers (e.g., Adam vs. SGD)
- Training mini-batch size
- Optimizer learning rate
- Number of training epochs
- Dropout rate

Be sure to record how the testing accuracy changes with these hyper-parameters and note the best accuracy achieved, as you will need to report your experimental results. Read more about documentation requirements in Section 3.2.

2.2 Helpful Hints and Resources

A few notes and resources about RNN implementation:

- **New to PyTorch?** Justin Johnson's tutorials are great resources for beginners. See here and here. The latter is based in Google Colab, but the code will look mostly the same.
- **Sentence representation and word vectors.** A sentence is represented as a sequence of numbers where each number is an index to a word in the vocabulary. You can use lookup to get the pre-trained embedding for each word. Alternatively, you can use one-hot vectors and compute embedded representations using matrix multiplication, but this may be slower. Use of a pre-trained model is encouraged.
- **GloVe pre-trained word embedding.** The GloVe pre-trained word embedding can be downloaded from here, and comes in various vector sizes. For information on how to use GloVe embeddings in your code, please refer to this article.
- **Dealing with sentences with varied lengths.** Sentences have different lengths. You will have to use a fixed length. You can get the maximum length based on your training sentences. Once a length is determined, you will need to do padding if the input sentence is shorter than the length. To do the padding, essentially you just add "0" (index to a designated symbol for padding) to the end of your input sentence.
- **Data preprocessing.** You will need to spend some time on data preprocessing. A very nice tutorial on data preprocessing (e.g., padding, mask, etc.) can be found by following this link.

2.3 Code Expectations

Your code must be able to do the following:

1. Train the POS tagger and save it
2. Load the model file for testing
3. Apply the model to test data (given both the `wsj19-21.testing` and `wsj19-21.truth` files, and output the accuracy of the model on the test data)

You will find a skeleton code file `rnnpos.py` that imports some libraries you may need, handles command-line parameters, saving the trained model, and loading the model for testing. You need to provide code that trains and evaluates the model in the areas marked between "Your code starts here" and "Your code ends here"; some example lines are provided in these spaces, but they can be modified or removed. Some example code is provided, but you may add any other functions you need, and edit the `RNNTagger` model as needed. You should not need to modify the names or parameters of the `train`, `test`, or `main`, or the argument parsing code at the bottom.

For grading, your code must be able to handle the following commands.

To train your model, we will run the following command, which should save the trained model as `model.torch`:

```
$ python3 rnnpos.py --train --model_file=model.torch --training_file=wsj1-18.training
```

To test your model, we will run the following command, which will load the pre-trained model from `model.torch`:

```
$ python3 rnnpos.py --data_file=wsj19-21.testing --label_file=wsj19-21.truth --model_file=model.torch
```

This command should output: "The accuracy is ⟨*accuracy*⟩%", where ⟨*accuracy*⟩ is the testing accuracy of the model rounded to one decimal place, e.g., "93.7".

3 Submission Requirements

Your accuracy on the test data should be above 85% for full credit.

3.1 Submission of RNN Models

Submit the following:

- Your Python program implementing the RNN for POS tagging: `rnnpos.py`
- The trained model file for your **best** hyper-parameter configuration: `model.torch`

3.2 Submission of Documentation

Besides submitting your code and model as described above, you need to submit a report file named `Results.pdf`. This document should report the best testing accuracy of your RNN model and its corresponding hyper-parameter configuration.[1] You should list the hyper-parameters you experimented with, and provide the results from each configuration you tried.

[1]For grading, we will verify that your reported accuracy matches what your code outputs when we run testing with the submitted model file.