Deep learning for part of speech tagging

CSCI 544, Spring 2021: Assignment 3
Due Date: April 29th, 4pm.

Introduction

The goal of this assignment is to get some experience with deep learning, in particular PyTorch. You will be implementing a bidirectional long short term memory (LSTM) network for part of speech (POS) tagging. The TAs have created a solution using the setup described below. Your trained model will be evaluated based on its performance on an unseen test set in comparison to the TA solution.

We have uploaded the data for this assignment to Vocareum. There is training data ("train.txt" file), development data ("dev.txt"), and test data. The test data cannot be seen by you. Unlike Assignment 1, when you submit your code you won't see your performance on the development data. This is because the Vocareum support team has informed us that submission scripts are not designed for computationally intensive tasks. During grading, we will train a model based on your submitted code and then using this model and your tagging code we will generate tags for the test data and compute macro-F1 score (see below).

For instructions on how to install PyTorch see https://pytorch.org; there is an option to install a CPU only version. Note that Vocareum has the 1.3.1 version of torch installed.

Bidirectional LSTM using PyTorch

You will write two programs: **poslearn.py** will learn a bidirectional LSTM model from labeled data, **postag.py** will use the model to tag new data.

poslearn.py will be invoked in the following way:
>python poslearn.py train_data_path dev_data_path
and output a model file; you can use whatever format you want for the model including pickle. The train_data_path and dev_data_path should include the train data and dev data files respectively (not just the folders where the files are located), e.g., "python poslearn.py $ASNLIB/publicdata/train.txt $ASNLIB/publicdata/dev.txt". The reason that poslearn.py has the dev data as input is because you can use the dev data to evaluate your model during training. After each training epoch you can save a new model only if it performs on the dev data better than the previously saved model. Evaluation should be based on macro-F1 score, which is the average of the F1 scores for each POS tag that appears in the training data. In order to calculate the macro-F1 score you are allowed to use libraries such as scikit-learn.

In order to process input data, you need to read them with "utf-8" encoding, transform each token and POS tag to lowercase and separate tokens from POS tags. So for the input sentence "I/PRP guess/VBP ./." "I", "guess", and "," are tokens and "PRP", "VBP", and "," are POS tags.

postag.py will be invoked in the following way:
>python postag.py validation_data_path
e.g., "python postag.py ./validation.txt", and output the file "tagged_data.txt". It is important that you use this filename ("tagged_data.txt") for the output of your tagging code. Here "validation.txt" should be a file with tokens but without any POS tags. For example, if the validation file contains the sentence

“I guess ,” then the “tagged_data.txt” file would contain something like “i/prp guess/vbp ,/,” or “I/PRP guess/VBP ./.” The validation file will contain tokenized sentences. So it won’t have sentences like “I guess, this is good.” but “I guess , this is good .” instead. When the TA scripts calculate macro-F1 everything will be converted to lowercase. When you generate the “tagged_data.txt” file do not change the order of the sentences because the “tagged_data.txt” file would be compared with the ground truth file. For grading the unseen test data will be used.

Below we provide information about how the TA solution works. After reading the training data, two dictionaries are created, one for tokens and one for POS tags. The dictionary of tokens only uses the 1000 most frequent tokens in the training data and also has extra entries for padding tokens and unknown tokens. The value for the padding entry could be 0 and the value for an unknown token could be 1. Likewise for POS tags. It is possible that a POS tag that appears in the dev data (or test data) is not included in the training data. Padding is not needed for the POS tags. When the dev or test data are read each token is assigned the corresponding value from the tokens dictionary extracted from the training data unless it is an unknown token. Likewise for POS tags.

The next step is to create matrices for the tokens and the POS tags with dimensions number_of_sentences x maximum_length_of_sentence. As the maximum length of a sentence we recommend 100. An additional matrix is created storing the length of each sentence. This is done for both the training and the dev data. Then tensors for the tokens, tags, and lengths matrices are created. The TA solution uses the PyTorch DataLoader for loading the samples and with “sample_type” defined as “sequential”. The batch size used is 100.

Using 1-hot encoding did not result in good performance. So the TA solution uses embeddings (1 layer):
self.embedding = nn.Embedding(num_tokens, emb_size, padding_idx = 0)
where emb_size = 100

For the LSTM the nn.LSTM function is used with input size and hidden size equal to emb_size. Also, the bidirectional option is selected. The TA solution uses linear layers, Adam optimizer, cross entropy loss, and a learning rate equal to 0.001. The TA solution also uses pad_sequence to stack a list of tensors along a new dimension and pads them to equal length. The TA solution trains the network for 20 epochs.

Note that it is not required that you follow exactly the TA solution. For example, you could use other types of embeddings. Everything is allowed as long as it can run on Vocareum without requiring any extra installation.

What to turn in and Grading

You need to submit the following on Vocareum:

- **All your code**: poslearn.py, postag.py, and any additional code you created for the assignment. This is required and the academic honesty policy (see rules below) applies to all your code. You also need to submit the model that you trained.

Your grade is based on the performance (macro-F1 score) of your code **on the test data** compared to the TA solution. If your performance is the same or better then you receive full credit (100 points). Otherwise, your score is proportional to your relative performance:
score = 100 points * your_macro_F1_score / TA_solution_macro_F1_score

You need to submit your solution by the deadline of April 29th, 4 pm. After that any submissions will receive 0 credit.

Other Rules

- You can use any external libraries you want as long as your code runs on Vocareum without requiring any additional installation.
- Questions should go to Piazza first, not email. This lets any of the TAs or the Professor answer, and lets all students see the answer.
- When posting to Piazza, please check first that your question has not been answered in another thread.
- DO NOT wait until the last minute to work on the assignment. You may get stuck and last minute Piazza posts may not be answered in time.
- This is an individual assignment. DO NOT work in teams or collaborate with others. You must be the sole author of 100% of the code and writing that you turn in.
- DO NOT use code you find online or anywhere else.
- DO NOT post your code online or anywhere else.
- DO NOT turn in material you did not create.
- All cases of cheating or academic dishonesty will be dealt with according to University policy.

FAQ

a) **How will the TAs/Professor grade my HW?**
We have written scripts which will run on Vocareum. Since we automate the process of grading, make sure that you write your code to match the output format "exactly". If you do not do this there will be a penalty.

b) **I found a piece of code on the web. Is it ok to use it?**
No! We run plagiarism checks on the code you write. The plagiarism detector is a smart algorithm which can tell if you've copied the code from elsewhere/others.
Instead please contact TAs/Professor during the office hours with your questions.

c) **Vocareum terminates my code before it finishes, but it finishes executing on my computer.**
This usually means that your implementation is inefficient. Keep an eye out for places where you iterate. Run time issues can be solved by using efficient data structures (Hashmap, Hashset, etc.).

d) **My code works on my computer but not on Vocareum. Do I get an extension?**
No, the deadline is automatically enforced by the system. Submit your code incrementally to make sure it functions on Vocareum.

e) **When are the office hours ?**
Please refer to the course website: https://kgeorgila.github.io/teaching/cs544-spring2021/index.html and Blackboard for the Zoom links.