CS 388 Natural Language Processing
Homework 2: Part-of-Speech Tagging with LSTMs

Due: March 19, 2018

In this homework you will explore the application of Bidirectional Long Short Term Memory networks (BiLSTMs) to Part-Of-Speech (POS) tagging using data from the Penn Treebank. Follow the steps outlined below and then turn in any code you write electronically and a PDF document describing your methods, your experimental results, and your interpretation and explanation of these results. Specific information regarding how to submit is here.

Existing System

We are giving you a basic POS tagging BiLSTM system implemented in TensorFlow along with code for running experiments with Penn Treebank data.

The specific version of TensorFlow we are going to use for this assignment is r1.0. The best tools for learning TensorFlow are the tutorials on the TensorFlow website. The API doc for this version can be found here.

The code we have provided (in `/u/mooney/cs388-code/nlp/bilstm-pos/`) has 2 files `preprocess.py` and `pos_bilstm.py`, and `prerpcess.py` preprocess the WSJ data to generate the input and output vectors that are fed into the BiLSTM. `pos_bilstm.py` contains the implementation of the basic BiLSTM model for POS tagging.

Look at the following trace to see how to run the program. You basically need to provide 4 arguments -- directory path to the dataset (WSJ), directory path to the training directory, dataset split type and experiment type. The first argument is self explanatory. Training directory is the directory where the checkpointed models and the TensorFlow events file will be stored. Dataset split type indicates the way you want to split the dataset. If the dataset split type is standard, we use the standard splits for the WSJ dataset, else we use 80:10:10 split for train, dev and test. The experiment type may be train or test. In train mode, we train our model storing it in the training directory, whereas in the test mode, we load the most recent model from the training directory and check its performance on the held-out test set. Ideally, you should continue training until the dev set accuracy does not dip in a sustained manner. However, due to computational reasons, you can prematurely terminate the training if the dev set accuracy appears to have converged.

We are going to use Tensorboard to visualize the training progress. For each experiment, we want you to generate Tensorboard plots for loss, accuracy and OOV (out of vocabulary) accuracy for both the training and validation data (6 plots in total). An example of these plots can be seen here and here. Note that the attached images do not contain plots for OOV accuracy. You will have to write code for computing the OOV accuracy and then add it to the Tensorboard plots. For instructions on running Tensorboard, look at the following file. By default, Tensorboard will visualize the plots on port 6006 of the local host. If you are running the experiments on the CS system, you will have to use port forwarding to visualize the results on your laptop.

Installing TensorFlow

If you plan to use the CS machines for running experiments, you can install TensorFlow in your local user space using the following command.

```
pip install --user tensorflow
```

This will install TensorFlow r1.0 CPU version. If you have access to a machine containing GPUs (and root privileges), you can look up instructions for installing the GPU version here.

Your Task

Your task is to extract orthographic features from each word, change the architecture of the LSTM to support utlizing these as additional input features, and run experiments examining the effect of these features on predictive and run-time performance.

First, write code to extract the following orthographic features from each word in an input sentence: capitalized, contains a common English suffix (e.g. `-ing`, `-s`, and `-ly`), contains a hyphen, and starts with a number. To find a good set of suffixes to detect, just search the web to find a site with a good table of common English suffixes.

There are two ways these orthographic features can be added to the biLSTM. One is to include them in the existing input layer as additional one-hot features together with the one-hot features indicating the specific word. Another way is to include them as features in the final POS classification layer of the network (i.e. the current hidden layer and the orthographic features are used to predict the current POS tag). Augment the existing BiLSTM code to support both of these approaches.

Rerun the previously discussed experiment on the WSJ data, utilizing these orthographic features, and trying both of the input architectures described above.

Getting Started

To get you started here is a basic outline of what you need to do.

1. Modify the preprocessSingleFile function to add orthographic features in addition to word id for every word. This should be pretty straightforward

2. Use the compute_accuracy function in pos_bilstm.py to compute the OOV accuracy. Hint: If you can think of an appropriate mask, then you can directly call this function to compute the OOV accuracy. look at function get_mask to understand how mask is working to get probabilties in given code.

3. Modify the create_graph function in pos_bilstm.py. Depending on the way you are adding the features to your model, you will have to change lstm_input or outputs. To include the features in the input layer one way to do it is to concatenate the embedding vector for the word (got from the get_embedding function) with the one hot vector for the prefix features and the suffix features. So if you are creating both Prefix features and Suffix features, the total input size would then be Embedding Size + len(prefix feature size) + len(suffix feature size). Please try to figure out yourself how to add features to the output.

Note: Just to make it clear, we are using an embedding in this assignment. This is just a simple linear embedding and you may learn more sophisticated embedding techniques later on. The basic reason for using embeddings is computational tractability.

Using Condor

We encourage you to run your longer experiments using the department's Condor pool. An overview of the system is available here. Be aware that your jobs may be terminated by Condor if they are competing for resources and plan ahead for this if you choose to use Condor.

Report

Your report should briefly describe your implementation and the experiments you ran, including a nicely formatted table(s) of results, with training and test accuracy (both overall and just for OOV items) and run time. Include a discussion of your results that answers at least the following questions, where possible, explaining the results in terms of underlying algorithmic properties.

- Overall, how does adding orthographic features affect the accuracy (both overall and OOV) and runtime of the BiLSTM and why?
- How does changing the approach to adding these features (input vs. classification layer) affect the accuracy (both overall and OOV) and runtime of the BiLSTM and why?