CS59000: Machine Learning for Natural Language Processing

HOMEWORK 2

Due: 23:59:59, Nov 14, 2017 on Tuesday

1 Programming: Named Entity Recognition

In this homework, you are required to write programs for named entity recognition for the given data set.

1.1 Introduction

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

An example:

```
Input (words) show flights from Boston to New York today
Output (labels) O O O B-dept O B-arr I-arr B-date
```

In this example, *B-dept* is the term where departure sources are, and *B-arr* and *I-arr* are the destinations, while *B-date* is the date. *B-* means the **beginning** of a term, *I-* means **inside** of a term, and *O* means **outside** of any predefined terms. As seen in this example, in NLP, named entity recognition problems are usually casted as sequence tagging problems.

To know more about named entity recognition (NER), please refer to the wikipedia website `https://en.wikipedia.org/wiki/Named-entity_recognition`.

In this homework you can use `NLTK`, `pytorch`, `numpy` and `scipy` as external python packages. Still, all generic packages are allowed. And also please do **not** use the GPU function because the grading environment `data.cs.purdue.edu` does not necessarily have one!

1.2 Models

You are required to implement one model: **Deep Maximum Entropy Markov Model** (DMEMM). DMEMM extends MEMM by using a neural network to build the conditional probability. The formulation for MEMM is as follows:

$$\begin{aligned} P(\boldsymbol{y}|\boldsymbol{x}) &= P(y_0)P(y_1|y_0, x_1)\dots P(y_T|y_{T-1}, x_1) \\ &= \prod_{i=1}^{T} P(y_i|y_{i-1}, x_i), \end{aligned}$$

where $P(y_i|y_{i-1}, x_i) = \dfrac{\exp\left(\boldsymbol{w}^\intercal \phi(x_i, y_i, y_{i-1})\right)}{\sum_{y_i} \exp\left(\boldsymbol{w}^\intercal \phi(x_i, y_i, y_{i-1})\right)}$.

If we use neural networks as probability generators, the model can be illustrated as follow, with : as a concatenation operation:

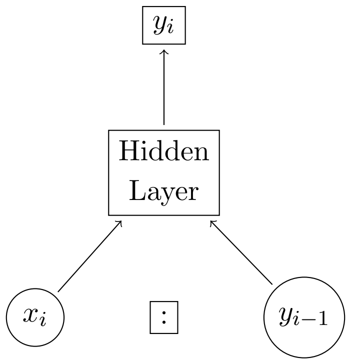

Figure 1: A simple Multi-layer Perceptron Model to generative probability at position T.

Thus, an embedding for each word and an embedding for each tag are required for a deep version of MEMM (DMEMM).

Also, in the inference, you need to implement the Viterbi algorithm (a dynamic programming algorithm) for decoding to find the best path. For the details of this algorithm, please refer to the course slides.

1.3 Features

You are required to use word embeddings of lower dimension as features. But to relieve your workload and to make your program run faster, you only need to use randomly initialized word embeddings. You also need to use tag embeddings for each tags as shown in the illustration. 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 postion with “1”, like $[0, 0, 0, \dots, 1, \dots, 0, 0, 0]$), or initialize them as low dimension vector similarly as word embeddings. You do no need to train word embeddings by yourselves. You can “fine-tune” your word embeddings and tag embeddings if you want (treat them as parameters).

1.4 Dataset

You will use the data set prepared by us for this homework. The data set has been pre-processed. And this time the whole data set including training, validation, testing is released. An example of how the original text look like has been shown in the introduction. There are a variety of different tags in this data set.

In the processed data, each word has been converted to its id in the word dictionary, so as each label is converted to its id in the tag dictionary. In this case, each sentence x is a vector of integers, so as its corresponding labels $\boldsymbol{y}$.

1.5 Model Tuning

You can exploit your model by exploring different hyper parameters to boost the performance:

- Adding a regularizer in your loss function.
- The coefficient of your regularizer.
- More complicated structure of neural networks, say, convolutional neural networks.
- The learning rate for gradient descent.
- The window size of words [previous word; current word; next word].
- The hidden layer size.
- The Dimension of word embeddings.
- The number of iterations (epochs) or using an "early stop" approach.
- Fine-tune the word embeddings (Treat word embeddings as parameters).
- Different optimization methods.
- Replace : concatenation operation with a bi-linear product operation, with an extra 3-D tensor parameter V ($\boldsymbol{x_i}^\intercal V \boldsymbol{y_{i-1}}$).
- Other brilliant ideas.

1.6 Tips

Some sample code can be found here: `http://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html`. 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.

1.7 Starter Code

The file "hw2.py" has been provided as the starter code for your implementation. If you are not comfortable with python class, you can use functions to replace it. The evaluation function has been written for your convenience, and the evaluation script is provided (Please include it in your submission folder for grading convenience).

1.8 Submission

Please follow the instruction given in HW0 for submission, including code and report. The report shall cover the problem, the algorithm, your experimental settings and the results. Please also follow the **naming requirements** we instructed. Your program must finish running within a reasonable time. Please make sure you tested your code on `data.cs.purdue.edu` before your submission.