Homework 04 – 05: Word Embeddings and Language Modeling

CS 6501-005 Natural Language Processing
20 points + 5 extra points

Deadline: Dec. 6th, 2019

1 Word Embeddings

In this section, we will use the same dataset as in our first homework on text classification. Specifically, in that dataset, we have four different files:

- trn-reviews.txt: the Yelp reviews in the training set
- trn-labels.txt: the corresponding labels of the Yelp reviews in the training set
- dev-reviews.txt: the Yelp reviews in the development set
- dev-labels.txt: the corresponding labels of the Yelp reviews in the development set

We are going to use this dataset to construct word embeddings and use these embeddings to further improve the classification performance

Preprocessing: convert all characters in both `trn-reviews.txt` and `dev-reviews.txt` into lowercases, then use **WordPunctTokenizer** from NLTK to separate punctuation from words[1]

Use the pretrained word embeddings from GloVe to construct sentence representations. You can download the pretrained word embeddings from

http://nlp.stanford.edu/data/glove.6B.zip

This pretrained word embeddings set contains 400K words with various embedding dimensions. For this problem set, we only the *50-D* embeddings.

For a given text, the simplest way of constructing a text representation is to compute the average of all the word embeddings in this sentence. For example, the representation of `I love coffee` is

$$\boldsymbol{s}_{\text{I love coffee}} = \frac{1}{3}(\boldsymbol{v}_{\text{i}} + \boldsymbol{v}_{\text{love}} + \boldsymbol{v}_{\text{coffee}}) \tag{1}$$

Following the same way, we can construct all the text representations in the training and development sets.

1. (2 points) Use the text representations and the `LogisticRegression` as in homework 01 to build a text classifier and report the classification accuracy on the development set. Please use the default setting of the classifier.

[1] `http://www.nltk.org/api/nltk.tokenize.html#nltk.tokenize.regexp.WordPunctTokenizer`

	# Sentences	# Tokens
Training	17,556	1,800,391
Development	1,841	188,963
Test	2,183	212,719

Table 1: Dataset statistics

2. (3 points) Recall that in homework 01, we use `CountVectorizer` to construct the feature vector for each text as $\boldsymbol{x}$. Now, with Eq. 1, we have another numeric form of the same text as $\boldsymbol{s}$. A simple way to get a richer representation is to concatenate these two numeric vectors together as

$$[\boldsymbol{s}, \boldsymbol{x}] \tag{2}$$

Build another text classifier with the concatenated representations as inputs and report the classification accuracy on the development set. Please use the default setting of the functions `LogisticRegression` and `CountVectorizer`.

3. (2 extra points) Please find the best hyper-parameter settings with the concatenated inputs and report the corresponding accuracy on the development set.

2 Recurrent Neural Network Language Models

In this section, you are going to implement a RNN for language modeling. To be specific, it is a RNN with LSTM units. As a starting point, you need to implement a very simple RNN with LSTM units, so please read the instruction carefully!

For this project, we are going to use the `wikitext-2` data for language modeling. I did some additional pre-processing on this dataset, therefore it is not exactly the same as the one available online.

In the data files, there are four special tokens

- <unk>: special token for low-frequency words
- <num>: special token for all the numbers
- <start>: special token to indicate the beginning of a sentence
- <stop>: special token to indicate the end of a sentence

Here are some simple statistics about the dataset

The **goal** of this part includes

- learn to implement a simple RNN LM with LSTM units
- get some experience on tuning hyper-parameters for a better model

I recommend to use `PyTorch` for the all the implementation in this section.

1. (3 points) Please implement a simple RNN with LSTM units to meet the following requirements:
   - Input and hidden dimensions: 32
   - No mini-batch or mini-batch size is 1
   - No truncation on sentence length, every token in a sentence must be read into the RNN LM to compute a hidden state, except the last token <stop>

- Use SGD with no momentum and no weight decay, you may want to norm clipping to make sure you can train the model without being interrupted by gradient explosion
- Use single-layer LSTM
- Use `nn.Embedding` with default initialization for word embeddings

Please write the code into a Python file with name `simple-rnnlm.py`. Please follow the requirements strictly, otherwise you will lose some points for this question and your answers for the following questions will be invalid. If you want to use some technique or a deep learning trick that is not covered in the requirement, feel free to use it and explain it in your report.

2. (3 points) **Perplexity.** It should be computed on corpus level. For example, if a corpus has only two sentences as following

$$\begin{aligned} &<\text{start}>, w_{1,1}, \ldots, w_{1,N_1}, <\text{stop}> \\ &<\text{start}>, w_{2,1}, \ldots, w_{2,N_2}, <\text{stop}> \end{aligned}$$

To compute the perplexity, we need to compute the average of the log probabilities first as

$$\begin{aligned} \text{avg} = &\frac{1}{N_1 + 1 + N_2 + 1} \{\log p(w_{1,1}) + \cdots + \log p(w_{1,N_1}) + \log p(<\text{stop}>) \\ &+ \log p(w_{2,1}) + \cdots + \log p(w_{2,N_2}) + \log p(<\text{stop}>)\} \end{aligned} \tag{3}$$

and then

$$\text{Perplexity} = e^{-\text{avg}}. \tag{4}$$

Please implement the function to compute perplexity as explained above. Submit the code with a separate Python file named `[computingID]-perplexity.py`.

Also, report the perplexity numbers of the simple RNN LM (as in `[computingID]-simple-rnnlm.py`) on the training and development datasets.

3. (3 points) **Stacked LSTM.** Based on your implementation in `[computingID]-simple-rnnlm.py`, modify the model to use multi-layer LSTM (stacked LSTM). Based on the perplexity on the dev data, you can tune the number of hidden layers n as a hyper-parameter to find a better model. Here, $1 \leq n \leq 3$. In your report, explain the following information

   - the value of n in the better model
   - perplexity number on the training data based the better model
   - perplexity number on the dev data based on the better model

   Submit your code with file name `[computingID]-stackedlstm-rnnlm.py`

4. (3 points) **Optimization.** Based on your implementation `[computingID]-simple-rnnlm.py` again, choose different optimization methods (SGD with momentum, AdaGrad, etc.) to find a better model. In your report, explain the following information

   - the optimization method used in the better model
   - perplexity number on the training data based the better model
   - perplexity number on the dev data based on the better model

   Submit your code with file name `[computingID]-opt-rnnlm.py`

5. (3 points) **Model Size.** Based on your implementation `[computingID]-simple-rnnlm.py` again, choose different input/hidden dimensions. Suggested dimensions for both input and hidden are 32, 64, 128, 256. Try different combinations of input/hidden dimensions to find a better model. In your report, explain the following information

   - input/hidden dimensions used in the better model

- perplexity number on the training data based the better model
- perplexity number on the dev data based on the better model

Submit your code with file name `[computingID]-model-rnnlm.py`

6. (3 points, extra) **Mini-batch.** Based on your implementation `[computingID]-simple-rnnlm.py` again, modify this code to add the mini-batch function. Tune the mini-batch size b as $\{16, 24, 32, 64\}$ to see whether it makes a difference. In your report, explain the following information

   - whether different mini-batch sizes make a difference. If the answer is yes, then
   - the best batch size
   - perplexity number on the training data based the better model
   - perplexity number on the dev data based on the better model

   Submit your code with file name `[computingID]-minibatch-rnnlm.py`