CS597 Assignment 3: Language Modeling with Neural Networks

Introduction

Here you will implement and train a Neural Network LM instead of the Markov chain model used in HW 1. Your Neural Language Model (NLM) will be evaluated by a downstream task: the text classification you did in HW2. The primary goal is to give you hands-on experience with neural N-gram languages. Understanding how these neural models work will help you understand not just language modeling, but also the pipeline of a common NLP task. This assignment is also more "from scratch" than the previous ones, which will help you prepare for the final project. Make sure you have installed pytorch and numpy to work on this assignment.

Neural Language Model

In this LM task, you need to build a vocabulary and calculate an optimal word embedding for every word in this vocabulary by training a neural network. And you need to compute the loss function on some training data, then update the parameters and the word embeddings with backpropagation. Recall that in an n-gram language model in the HW2, given a sequence of words *w*, we want to compute the probability (**P**):

$$P(w_{k+1}|w_1, w_2, \ldots, w_k)$$

Where w_k is the k-th word of the length-k sequence. In this model, you should maximize the probability of the correct word w_{k+1} given the latent representation of the context words. Note that your k-grams will come from a corpus where the first k words are context features and the word w_{k+1} is the training label. Some useful background can be found in the class lecture notes slides 58-64, CSE597-Wk5-Mtg9-FFNs.pdf (and associated readings).

Task: N-Gram Neural Language Model

Modify the code in NML.py to implement a simple 3-gram language model. Make sure that your loss decreases during the training process, and the embedding of the word can be used in the downstream task (text classification) which you have finished in the HW1. These are the steps you need to complete for this task:

> **[15 Points] Step 1:** Modify the *Class NgramLM(nn.Module)* which is a definition of the NLM model.

> **[25 Points] Step 2:** Use the NgramLM class to implement the *def training()* function;

note that you should define the optimizer and the loss function first. You should experiment with different loss functions. You may need a DataLoader to enlarge the batch size.

[10 Points] Step 3: Finish the *def output (model, file1, file2)*. In this function, you need to copy the embedding vocabulary to disk in the format of a GloVe embedding file, called *embedding.txt*. Then it can be used in the downstream classification task. You are to conduct a controlled experiment where you compare a condition where words are initialized with random embeddings that have the same dimensionality as the GloVe lexicon, to a condition where you use the GloVe vectors. The random embedding vocabulary with the same words and the same embedding dimension with the n-gram embedding vocabulary, but the embedding vectors are randomly initialized, we call it *random_embedding.txt.*

[10 Points] Step 4: Conduct experiments on the text classification task. You are requested to test the performance differences between initializing with *random_embedding.txt, embedding.txt, and glove.6B.50d.txt* by the model from HW1 (which we give the formal answer in this project).

The zip file associated with this homework has three *py files (classifier.py, NLM.py, run.py), a glove folder with a *txt file with the GloVe vectors, and a data folder with three sizes of review texts and labels:

reviews_100.txt and labels_100.txt are small, so this can be used for development (dev set) and debugging.
reviews_500.txt and labels_500.txt are the dataset for this assignment (both NML part and downstream part), and you will be graded based on them.
reviews.txt and labels.txt are much bigger files than the others; they will be used to evaluate the bonus part.

Bonus: 10 points extra credit

It is very time consuming if NML is trained in a large dataset, so we only use 500 data examples in the dataset to train this NML, and test the classification task. However, there are several methods or tricks to overcome the training demands, such as enlarging the training batch size or sampling. If you can use the whole dataset to train the NML within the time limits (1hour) in a common laptop machine (without a GPU) and get an accuracy > 85% in the classification task, you will get extra credits (10 points).

Note that you can use the saving and loading in pytorch to conduct your experiments. Feel free to implement any other helpful functions, but other packages are excluded.

Questions [40 Points]

1. [20 Points] How did the choice of initialization of word embeddings affect training of the LM and/or performance of the embeddings in the HW1 classifier?

2. [20 Points] Explain your choice of loss function, based on a comparison with at least one other loss function.

Evaluation

Your submission will be evaluated on several axes:

Writeup: correctness of answers, clarity of explanations, etc.

Execution: your code should train and evaluate within the time limits (**1hour**) without crashing and give the required accuracy (Acc > **70%** on the classification task).

Deliverables

You should submit the following files to Canvas as a zip file which contains:

1. **[60 Points]**

   NLM.py. Do not modify or upload any other source files. Make sure that the following commands work without error, and in the allotted time limit of 1 hr, before you submit:

   pythonNLM.py

   ```
   python run.py --algo GLOVE --lr 0.001 --embed_size 50 --hidden_size 5 \
      --emb_file ./glove/glove.6B.50d.txt -- review_file ./data/reviews_500.txt \
      --label_file ./data/labels_500.txt
   python run.py --algo GLOVE --lr 0.001 --embed_size 50 --hidden_size 5 \
      --emb_file ./data/random_embedding.txt --review_file ./data/reviews_500.txt \
      --label_file ./data/labels_500.txt
   python run.py --algo GLOVE --lr 0.001 --embed_size 50 --hidden_size 5 \
      --emb_file ./data/embedding.txt -- review_file ./data/reviews_500.txt  \
      --label_file ./data/labels_500.txt
   ```

2. **[40 Points]** The answer pdf file. (See point breakdown above.)