Assignment 1: Text Classification with Neural Nets

CS678, Fall 2022

August 13, 2022

Academic Honesty: Please note that students should complete the assignment independently. While you may discuss the assignment with other students, **all work you submit must be your own!** In addition, if you use any external resources for the assignment, **you must include references to these resources in your assignment report.**

Goal: In this assignment, you will implement a binary sentiment classifier based on the feed-forward neural net (FFNN) architecture. This includes implementing the FFNN model and its training procedure. The implementation will be based on `PyTorch` (https://pytorch.org/). Other generic Python tools such as `numpy` are also allowed (if you are not sure, please consult the instructor).

PyTorch Setup

You will need to install PyTorch before starting this assignment. Follow the link (`https://pytorch.org/get-started/locally/`), you will be guided to an installation command. **You are recommended to install PyTorch within the anaconda environment that you previously created (executing first the conda activation command and then the PyTorch installation command).** The assignment is small-scale enough to complete using CPU only, so don't worry about installing CUDA or getting GPU support working unless you want to. However, the provided framework is based on CPU.

If you are new to PyTorch, you may find this tutorial useful: `https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html`.

1 Assignment Material Overview

Dataset You'll be using the movie review dataset of Socher et al. (2013). This is a dataset of movie review snippets taken from Rotten Tomatoes. The original dataset actually consists of full parse trees with each constituent phrase of a sentence labeled with sentiment (including the whole sentence). The labels are "fine-grained" sentiment labels ranging from 0 to 4: highly negative, negative, neutral, positive, and highly positive.

However, in this assignment, we will work on a simplified version of this dataset. Specifically, we only consider two classes – positive (1) or negative (0), and we will build a binary sentiment classifier. The data files given to you (under the `data` folder) contain newline-separated sentiment examples, consisting of a label (1 or 0) followed by a tab, followed by the sentence, which has been tokenized but not lowercased. The data has been split into a train, development (dev), and blind test set. On the blind test set, you do not see the labels and only the sentences are given to you. The framework code reads these in for you.

FFNN Model The FFNN classifier you will implement is similar to the "Deep Averaging Network" (DAN) model of Iyyer et al. (2015). Formally, given a sentence $s = (w_1, w_2, ..., w_{|s|})$, where w_i is a single word in this sentence and $|s|$ denotes the sentence length, you will first map each word to its embedding vector, denoted as $c_i \in \mathbb{R}^{d_e}$, where d_e is the size of the embedding. The DAN model will then calculate an average embedding vector $av \in \mathbb{R}^{d_e}$ for this sentence, and pass it through two feedforward layers:

$$\begin{aligned} av &= \sum_{i=1}^{|s|} \frac{c_i}{|s|}, \\ h &= ReLU(W_h \cdot av + b_h), \\ h_{out} &= W_{out} \cdot h + b_{out}, \\ pred &= softmax(h_{out}). \end{aligned}$$

Here, $W_h \in \mathbb{R}^{d_h \times d_e}, b_h \in \mathbb{R}^{d_h}$ are learnable parameters for the first non-linear layer (with activation function *ReLU*); $h \in \mathbb{R}^{d_h}$ represents the intermediate hidden units; the second linear layer is parameterized by two learnable parameters $W_{out} \in \mathbb{R}^{n_{cls} \times d_h}, b_{out} \in \mathbb{R}^{n_{cls}}$, where n_{cls} is the number of classes in the classification task (e.g., two in a binary classification problem), and it produces the *logits vector* $h_{out} \in \mathbb{R}^{n_{cls}}$. Note that this last linear layer has projected the hidden units to the class space (and hence has a dimension of $\mathbb{R}^{n_{cls}}$); however, this logits vector is not normalized. To get the final prediction probability distribution, you should pass h_{out} to a *softmax* function.

Framework Code The framework code consists of four `.py` scripts, as explained below:

- `sentiment_classifier.py`: This is the main class where you will start your code. It uses `argparse` to read in several command line arguments. It's okay to add command line arguments during development or do whatever you need, but **you cannot modify this file for your final submission**. For the existing arguments, you should not need to modify them. Please read the explanation of each argument in the source code. The main method loads in the data, trains the FFNN model, and evaluates it on train, dev, and blind test, and writes the blind test results to a file.
- `sentiment_data.py`: This file defines the `SentimentExample` object, which wraps a list of words with an integer label (0/1). It also includes the `SentimentExampleBatchIterator` class for "batch-ifying" the training data, but some blanks need to be filled.
- `models.py`: **This is the primary file you'll be modifying.** You will complete the implementation of the `FeedForwardNeuralNetClassifier` and its training method `train_feedforward_neural_net`.
- `evaluator.py`: This file contains the evaluation code, which calculates accuracy, precision, recall, and F1 of the tested examples. You should not need to modify this file.

2 Part 1: Implementing FFNN (60 points)

In Part 1, students need to complete all TODO's in the framework code, including implementing the FFNN model and its training method in `models.py`, and the batch iterator in `sentiment_data.py`.

Your submitted code should be able to be executed via:

```
python sentiment_classifier.py --n_epochs 10 --batch_size 32 --emb_dim 300
--n_hidden_units 300.
```

Please make sure that by executing these command lines the grader can launch your code and see your model outputs. The model performance should also be included in your submitted report. During development, you may set `--no_run_on_test` to disable the evaluation on test set.

A correctly implemented FFNN-based classifier should obtain around **0.77 accuracy on the dev set**. The training and evaluation should finish within 1-2 minutes if you use a recent laptop and less than 10 minutes or so in general.

3 Part 2: Exploration of FFNN (40 points)

The second part of this assignment asks you to explore the FFNN model from different perspectives:

Vector dimensions (10 points): The command line in Part 1 specifies the the embedding size as 300 and the hidden size as 300. Would increasing or decreasing the two sizes lead to different model performance? Students are encouraged to explore different choices of the two hyper-parameters, e.g., 50, 100, 300, or 500.

Optimization methods (10 points): The current framework code instructs students to implement an Adam optimizer with its default configuration. Is there a better optimization method? Should one pick a larger or smaller learning rate? Students are encouraged to explore a different optimization algorithm and trying different learning rates.

Embedding initialization (20 points): In Part 1, your have implemented a randomly initialized word embedding. Will the model achieve a better performance if you use a pre-trained GloVe embedding (Pennington et al., 2014)? For this exploration, students would need to implement loading and preprocessing the open-source GloVe embedding (`https://nlp.stanford.edu/projects/glove/`) as well as using it to initialize the `word_embeddings` of `FeedForwardNeuralNetClassifier`. The framework code has set up an optional argument `--glove_path` in `sentiment_classifier.py` such that the GloVe embedding would be invoked by executing:

```
python sentiment_classifier.py --n_epochs 10 --batch_size 32 --emb_dim 300
--n_hidden_units 300 --glove_path path/to/GloVe.
```

The same command line will be run for grading purpose as well.

Students should complete these explorations, showing results and analyses (i.e., your observations and interpretation) in the report.

4 Submission and Grading

You will upload your code and your report to Blackboard. In summary, your report should include (1) the final training and dev set performance of your trained FFNN model for Part 1; (2) results and analyses for Part 2. The grading will also be based on whether the grader could execute your code and successfully obtain expected model outputs within a reasonable time limit. Accuracy of your model on dev set and the blind test set for Part 1 will be checked.

Note that partial credit is awarded even for non-functional solutions. If you have even partially implemented functions, please submit them and we will score your submission appropriately.

References

[Iyyer et al.2015] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 1681–1691.

[Pennington et al.2014] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Socher et al.2013] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.