CS5011: A4 - Learning
Question Classifier using Neural Networks

Assignment: A4 - Assignment 4

Deadline: 10th of May 2022

Weighting: 25% of module mark

Please note that MMS is the definitive source for deadline and credit details. You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

---

1 Objective

This practical aims to construct and use Artificial Neural Networks (ANNs) to build a question classifier. By doing the practical, you will get familiar with the main components of an ANN system and with a number of basic concepts of Deep Learning for Natural Language Processing (NLP) applications.

2 Competencies

- Build an ANN system for a classic NLP problem using Bag-of-Word approach.
- Understand the issue of sparse matrices when representing text input for ANNs and how to resolve it.
- Understand the concept of word embeddings and how to use a pre-trained word embeddings in an ANN system.

3 Practical Requirements

3.1 Introduction

Natural Language Processing (NLP) is an important area of Artificial Intelligence with several real-world applications. In recent years, Deep Learning has become a widely used approach for NLP problems due to its ability to gain superior performance on various NLP tasks compared to traditional NLP solving approaches. In this practical, we will look into a classic NLP problem - the question classification problem. Given a question, the goal is to identify what the topic of the question is, e.g., whether it is about food, or a place, or an animal, etc.

You will be asked to build an ANN system for the question classification problem and to try a number of techniques for improving the system. The practical consists of four parts:

- Part 1 (Basic): You will build a basic ANN-based question classifier using the Bag-of-Word approach and report the performance of your system on the given datasets.

- Part 2 (Intermediate): You will look into the issue of sparse matrices when representing text input for ANNs. You will implement an embedding layer for the input to resolve the sparsity issue, hence speeding up the computation of the ANN system built in the previous part.
- Part 3 and Part 4 (Advanced): In this part, you will make use of a pre-trained word embedding in your ANN system and investigate its impact on your ANN's performance.

3.2 Starter Code

This practical uses `MINET`, a MInimal neural NETwork library for Java written by Phong Le. `MINET` is an educational project for Java users to learn basic neural networks. Its structure is mainly inspired by the popular deep learning Python library `pytorch` [1], but is kept minimal so that the learner can easily understand the structure as well as playing with the library (e.g. adding layers, optimisers).

`MINET` source code and all data used for this practical are available on `studres`, which can be used as the starter code for the practical. `MINET` Javadoc can be found at `https://lephong.github.io/minet/`.

We also provide two bash scripts in the starter code:

- `build.sh`: command for compiling the code.
- `run.sh`: example commands for each part of the practical. Please make sure that your submission can be run correctly using this script.

3.3 Part 1

In this part, you will build an ANN question classifier for the TREC dataset [1] [2]. For ease of implementation, we already pre-processed the data for you. In particular, the input texts were tokenized (into words) and converted into word indices, and the output labels were converted into integers. The dataset was split into training, validation and test sets. We will use the one-hot encoding to represent the text inputs.

All data files for this part are available in `data/part1/`. They include:

- `vocab.txt`: the vocabulary. Each line contains one token. The line number (*starting from* 0) represents the index of the corresponding token. The first line (indexed 0) is reserved for unknown tokens. Note that to avoid data leakage, this vocabulary is extracted from the training data only. Only tokens that appear at least twice in the training set were extracted. For simplicity, the text was converted to lower case before the extraction.
- `classes.txt`: the output classes (question topics). Each line contains one class. The line number (*starting from* 0) represents the index of the corresponding class.
- `train.txt`, `dev.txt`, `test.txt`: the pre-processed training, validation (or development), and test sets. Each line corresponds to a sample and its label, i.e., a question and its topic. For example, the following line:
  `10 3 5 ; 2`
  represents a sample question with three words (taken from lines 10, 3 and 5 in `vocab.txt`) and has an output label class of 2 (taken from `classes.txt`).

Your task is to build a classifier for the given data and report the results of the training and evaluation process. The MNIST example provided by `MINET` (`minet/example/mnist/`) can

[1] `https://pytorch.org/`

[2] `https://cogcomp.seas.upenn.edu/Data/QA/QC/`

be used as a starting point for this part. Please make sure your program prints out the built network and results of the training/test process as in the MNIST example.

Regarding the hyper-parameters of the ANN model, you can start with the following setting:

- learning rate: 0.1; maximum number of epochs: 500; patience parameter: 10; batch size: 50; number of hidden layers: 3; size of the first hidden layer (the embedding layer): 100; size of each of the other hidden layers: 200; activation function for hidden layers: ReLU.

The following points should be considered in the evaluation:

- The training process and the performance of the trained ANN model on the test set.
- How do the hyper-parameters affect performance of the ANN? Remember to provide sufficient details to support your discussion.

3.4 Part 2

In this part, you will re-use the datasets and the ANN hyper-parameter setting in Part 1. Your task is to implement an Embedding layer to deal with the issue of slow matrix computation due to sparse input representation as discussed in the lectures. In particular, you will write a new class called `EmbeddingBag` (see `src/EmbeddingBag.java`) to represent the embedding layer in your ANN model, and the computation for both forward and backward phases should be implemented in an efficient way.

The following points should be considered in the evaluation:

- How do you make sure that your implementation of the computation is correct?
- How does your ANN with the new implementation compare to the one used in Part 1?

3.5 Part 3

The weights of the embedding layer in your ANN model in the previous parts were initialised randomly. In this part, you will make use of pre-trained word embeddings for setting those weights. We will use the popular word embeddings GloVe [2] [3].

Similar to part 1, for ease of implementation, we have pre-processed the data in advance for you. The data for this part can be found in `data/part3/`. The list of files are the same as in Part 1. The only difference is in the `vocab.txt` file. Each line of this file now has a list of numeric values added. The added list is the pre-trained vector of GloVe for the corresponding word token. Note that the new vocabulary contains word tokens from the whole TREC data instead of only the training set. In real settings, the whole vocabulary of GloVe would be used because we cannot assume which words are used in our dataset. However, to reduce computation and memory cost, in this practical, we will simplify the setting by limiting the vocabulary to the given TREC data.

Your task is to extend the `EmbeddingBag` layer in Part 2 to allow the initialisation of the weight matrix for the embedding layer using GloVe, and discuss the impact of using pre-trained embedding on your ANN models in your report.

3.6 Part 4

(Please make sure you completed all previous parts before attempting this one)

There are various possible extensions for the implementation in Part 3. First, after initialising the weights of your ANN model, you can either let the ANN update all the weights during the training as usual, or freezing the embedding layer's weights and only train the rest, or combining both approaches. Second, you can try other word embeddings such as word2vec [4].

[3] `https://nlp.stanford.edu/projects/glove/`

[4] `https://code.google.com/archive/p/word2vec/`, see `GoogleNews-vectors-negative300.bin.gz`

You can also suggest and investigate your own extensions but they should be at least with a similar level of complexity.

The original TREC text data (before pre-processing) are available in `data/original/`, in case you want to use them for the extensions.

3.7 Notes on Parts 3 and 4

For Part 3 and Part 4, you may or may not see an improvement in performance when using pre-trained embedding in this practical. If you do not see improvement, please do not worry about it. The main aim of this practical is not about getting the best performance. Our main focus is to get ourselves familiar with the techniques under study while still keeping everything else as simple as possible. There are several factors that can impact the performance and choosing the right combination is normally important. However, investigating all of those options is out of the scope of this practical.

4 Code Specification

Your submission should make use of the starter code provided and must compile and run without the use of an IDE. Your system should be compatible with the version of Java available on the School Lab Machines (Amazon Corretto 11 – JDK11). Your source code should be placed under `src/` folder in the provided starter code. All non-standard external libraries must be included in the `lib/` folder. For all parts, make sure your code can be compiled with the provided `build.sh` script and it must run with the following command.

```
java A4Main <part1/part2/part3> <seed> <trainFile> <devFile> <testFile>
                <vocabFile> <classesFile>
```

You can add extra arguments at the end of the command line if desired.

We will re-run your programs on the school lab machines.

5 Report

You are required to submit a report describing your submission in PDF with the structure and requirements presented in the additional document `CS5011_A_Reports` found on `studres`. The report includes 5 sections (Introduction, Design and Implementation, Testing, Evaluation, Bibliography) and has an advisory limit of 2000 words in total. The report should include clear instructions on how to run your code. The evaluation section should include the discussion points listed in this specification for each part.

6 Deliverables

Please use the given StarterCode for your submission. Simply rename `StarterCode` with your student ID and submit the folder as a single ZIP file. Please do not alternate the structure of the folder.

The ZIP file must be submitted electronically via MMS by the deadline. Submissions in any other format will be rejected. Your ZIP file should contain:

1. A PDF report.
2. Your implementation as described in this specification.

7 Assessment Criteria

Marking will follow the guidelines given in the school student handbook. The following factors will be considered:

- Achieved requirements and quality of the implementations provided.
- Quality of the report, including analysis and insights into the proposed solutions and results.

Some guideline descriptors for this assignment are given below:

- For a mark of 7 to 11: the submission implements Part 1 correctly and the report is written adequately.
- For a mark of 11 to 13: the submission implements Part 1 correctly with a reasonable attempt on Part 2. The implementation, testing and evaluation are of good quality and the report is clearly written.
- For a mark of 13 to 16: the submission implements both Part 1 and Part 2 correctly. It contains clear, well-designed code with a very good evaluation and a high-quality report showing a good level of understanding of the requirements.
- For a mark of 16 to 18: the submission implements Parts 1, 2 and 3 correctly. It contains clear and welldesigned code together with an excellent report.
- For a mark of 18 to 20: the submission satisfies all requirement of the previous band together with one or two extensions described in Part 4 in high quality.

8 Policies and Guidelines

Marking: See the standard mark descriptors in the School Student Handbook

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_-Descriptors`

Lateness Penalty: The standard penalty for late submission applies:

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

Good Academic Practice: The University policy on Good Academic Practice applies:

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Nguyen Dang, Alice Toniolo, and Mun See Chang

cs5011.lec@cs.st-andrews.ac.uk

March 23, 2022

References

[1] Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.