CS3105 Practical 2: Neural Networks

Nguyen Dang, Ian Gent, MunSee Chang

October 2021

This practical carries 50% of the coursework assessment for the module and is due at 21.00 on Monday November 22nd, 2021. (Please note that MMS deadlines are definitive and also subject to change so should be checked in case of doubt.)

The practical involves building a feed-forward Artificial Neural Network (ANN) for a multi-class classification task. It comes in four parts:

- Part 1: given a dataset and the starter code, you are asked to construct an ANN for multi-class classification, train it with the provided hyper-parameter values, report results and compare it with a baseline linear classifier model.
- Part 2: in this part, you will improve the prediction accuracy of the ANN built in Part 1 by varying the hyper-parameters' values. The required accuracy is at least 70%.
- Part 3: the dataset given in this part has input (numeric) features belonging to different scales and some missing values. Your task is to implement data preprocessing to make sure that your ANN's performance is on-par with the previous part.
- Part 4 (advanced): In this advanced task, you will analyse the importance of the input features on the prediction of your ANN model. We will use the data of Part 1 and the hyper-parameter setting chosen in Part 2 for this task.

REMEMBER:

- Your code submission and report MUST follow the instructions described in this specification.
- Please make sure that your submitted code can be compiled and run using the listed example commands on the school's lab machines, and that it produces the same results as written in your reports. **We will re-run your code on the lab machines to validate your reported results.**

Starter Code

This practical uses `MINET` [1], 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` [2], 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). One important note about `MINET` is that the library implementation uses *row-vector* instead of *column-vector*. This convention follows `pytorch` and several other deep learning libraries and is useful for practical reasons. For example, an instance in `MINET` is represented as a row of numbers, and a cell $W[i, j]$ in the weight

[1] `https://github.com/lephong/minet`
[2] `https://pytorch.org/`

matrix W between two layers represents the connection of node ith of the first layer and node jth of the subsequent layer. Therefore, a linear layer is calculated as $Y = XW + b$ rather than $Y = WX + b$.

`MINET`'s source code and all other necessary files (`jblas` library and a compiling script) are provided in the starter code. `MINET`'s javadoc is available at `https://lephong.github.io/minet/`.

The starter code is provided in `P2/StarterCode` on `studres`. Please use it for your submission. You can make change to the provided code, including `MINET`'s code, if necessary, but DO NOT remove any of the given print statements. The starter code includes the following components:

- `ANN.java`: a `java` class with methods for building/training/evaluating an ANN using `MINET`. Description of all functions of the class is provided in the file.
- `P2Main.java`: the main program of the practical, where you will load and process the data, build and train an ANN on the given data and report final performance.
- `build.sh`: a bash script containing commands for compiling your code on the School lab machine.
- `run.sh`: a bash script containing example commands for testing your code on all parts.
- `minet/`: a folder containing all source code files of `MINET`, used when compiling your code.
- `lib/`: a folder containing all `.jar` libraries needed, **please include any external libraries here**.
- `data/`: the datasets used for the practical.
- `settings/`: a folder containing settings for your experiment in `.json` format. You will need to add your own setting files for Part 1 and Part 2 here.

Note that some parts of the code in `ANN.java` and `P2Main.java` are missing. Your task is to fill them in (see the following sections for more details).

Part 1: Building an ANN

The dataset given for this part is already split into two sets: the training set and the test set (see `data/Part1/train.txt` and `data/Part1/test.txt`). This is a classification task with three output classes. The data files are in the format supported by `MINET`, i.e., the first line includes three integer values representing the number of input features, the number of output labels (always 1 for this practical), and each of the remaining lines represents one sample (input feature values separated by space, followed by a semi-colon and the output class index).

There are three tasks for this part:

1. The first task is to fill in the missing pieces of code in `ANN.java` and `P2Main.java` (marked as `//YOUR CODE HERE`) such that the resulting program can:
   - Load the training and test sets, and split the training set further into training and validation sets (for early stopping).
   - Read hyper-parameter values from a `.json` file, build an ANN and train it with the given data and setting (see `MINET`'s tutorial 1 [3] for an example on how to create an ANN).
   - Report results of the trained ANN on the test set.

   Note that you do not need to fill in `P2Main.preprocess_trainset()` and `P2Main.preprocess_testset()`, those are for Part 3.

[3] `https://github.com/lephong/minet/blob/master/tutorial/tutorial1.md`

2. The second task is to train an ANN with the provided example setting in `settings/example.json` and report results.

3. The third task is to build a linear classifier as a baseline to compare your trained ANN with. This can be done simply by creating an appropriate `.json` setting file. Please save it as `settings/linear.json`.

Code submission requirement: Include in your submission the filled-in starter code and the added `settings/linear.json` file. Make sure your code can run with the following example commands, and that your reported results can be re-produced on the School lab machines:

```
java -cp lib/*:minet:. P2Main data/Part1/train.txt data/Part1/test.txt 123
    settings/example.json
java -cp lib/*:minet:. P2Main data/Part1/train.txt data/Part1/test.txt 123
    settings/linear.json
```

Report submission requirement: The following points should be discussed in your report:

- What needs to be set in your `.json` file to make a linear classifier?
- How is the performance of the trained ANN compared with the linear model?

Part 2: Improving your ANN performance

In this part, you will improve the performance of your ANN on the dataset given in Part 1. The idea is to tune the hyper-parameter settings to achieve better performance. You should aim for a prediction accuracy of at least 70%. You are free to decide how to investigate the hyper-parameters. To make it simpler, we will assign fixed-values to the following three hyper-parameters: `batchsize=128`, `nEpochs=2000` and `patience=100`. You should leave those three hyper-parameters as they are and only investigate the remaining ones.

Code submission requirement: There is no code submission for this part. Please submit your final hyper-parameter setting as `settings/Part2.json`. Similar to Part 1, make sure that your program can re-produce the same results as written in your report on the School's lab machines. The example command for this part is as follows:

```
java -cp lib/*:minet:. P2Main data/Part1/train.txt data/Part1/test.txt 123
    settings/Part2.json
```

Report submission requirement: the following points should be discussed in your report:

- How did you use the given datasets for the tuning process?
- How did you come up with the chosen hyper-parameter setting? Which hyper-parameter values did you try? And which patterns did you observe based on the results? Note that we don't expect you to do a systematic search for the best hyper-parameter setting, as it can be quite costly, a manual tuning process should be fine. However, you should at least try various hyper-parameter settings and do some analysis. Another important point is that you do not need to try to achieve excellent performance, a prediction accuracy around 70% should be sufficient for this part.

Part 3: Data preprocessing

In practice, the data sometimes needs to be preprocessed before being used for a Machine Learning (ML) model [4]. In this part, you will implement two data-preprocesing techniques: data imputation and data standardisation.

Data imputation: Sometimes there are missing values in the input due to issues during data collection. We would need to fill in those values before giving the data to our ML models. One of the simplest techniques to impute those values is as follows. Consider a missing value x that belongs to an input feature V, we will calculate the mean of all the non-missing values of that feature and replace x with that value.

Data standardisation: Some Machine Learning models, including ANNs, are generally easier to train when the input features are *standardised*, i.e., each feature has its values centered around zero and with a standard deviation of one (see, e.g., `scikit-learn` page on data-preprocessing [5] for more details).

In this part, you will be given a dataset (see `data/Part3/`) whose input features belong to different scales and there are some missing values in the input. The missing entries are marked with a special value of 99999. Your task is to implement data-preprocessing (in this case, data imputation and standardisation) by filling in the methods `P2Main.preprocess_trainset()` and `P2Main.preprocess_testset()` in the starter code.

Code submission requirement: Include in your submission the filled-in starter code. Make sure that your program can re-produce the same results as written in your report on the School's lab machine. The example commands for this part are as follows:

```
java -cp lib/*:minet:. P2Main data/Part3/train.txt data/Part3/test.txt 123
    settings/Part2.json 0 # without pre-processing
java -cp lib/*:minet:. P2Main data/Part3/train.txt data/Part3/test.txt 123
    settings/Part2.json 1 # with pre-processing
```

Report submission requirement: The following points should be discussed in your report:

- How did you implement data-preprocessing for the given training and test sets? Remember that the test set is never revealed until after the training is finished, i.e., information of the test set should not be used for any steps except the very final evaluation.
- Compare performance of the your ANN with and without data-preprocessing. You can re-use the same hyper-parameter setting as the final one used for Part 2.

Part 4: Feature importance analysis

It is strongly recommended that you finish at least the first two parts before attempting this one.

For this part we will re-use the datasets and hyper-parameter settings of Part 2. The data is artificial and was generated using the `scikit-learn` library. The input features may or may not all be important for the prediction of your best found ANN model. Your task is to identify the unimportant ones or to discover that they are all important. You are free to use any methods and libraries that you like.

Code submission requirement: Include your code and any external libraries for this part in the submission, with clear instructions on how to run and re-produce your results.

Report submission requirement: The following points should be discussed in your report:

- Which method(s) did you use for your analysis? How did you implement them?
- What are your findings? Discuss the results in details.

[4] see, e.g., `https://scikit-learn.org/stable/modules/preprocessing.html#` for implementation of a number of commonly used data pre-processing steps

[5] `https://scikit-learn.org/stable/modules/preprocessing.html#standardization-or-mean-removal-and-variance-scaling`

Hand-in and Report requirements

Report

Your report must be in `pdf` format. Consider the following points when writing the report:

- DO NOT include screenshots of your code or your program's output unless it is absolutely needed.
- The *ADVISORY limit* is **2200 words, or 5 pages**. This does not include tables, plots and bibliography. Try to make your report's content precise and straight to the points. Answer the questions suggested for each part one by one. You can also include any further discussion if you think they are interesting and worth mentioning.
- Include the following check table on the FIRST page of your report.
- The typical report structure with Design, Testing and Evaluation is not required for this practical. You can go straight to answering the questions of each parts. Of course you can also discuss issues you have encountered if necessary.
- If you include plots in your report, remember to give clear captions explaining what each axis (and color, if applicable) represents.

	Tasks	Done	Notes
Part 1	Missing code filled in and the program works properly	yes/no	
	Linear classifier setting submitted	yes/no	
	Results reported	yes/no	
Part 2	Expected performance achieved and chosen hyper-parameter submitted	yes/no	
	Tuning process and results reported	yes/no	
Part 3	Data pre-processing implemented	yes/no	
	Results reported	yes/no	
Part 4	Analysis done and reported	yes/no	

Submission

Please structure your submission as follows:

```
studentID
      report.pdf
      P2Main.java
      ANN.java
      build.sh
      run.sh
      data/
      minet/
      lib/
      settings/
```

Remember to include all external `.jar` libraries in folder `lib/`. Make sure your code can run properly using the provided example commands in `run.sh`.

Submit the whole folder as a single `<studentId>.zip` file to MMS. Before submitting, remember that the University policy on Good Academic Practice applies, see: `https://www.st-andrews.ac.uk/students/rules/academicpractice/`

Your source code should be well-commented whenever possible. Where any code from other sources is used, you must provide a clear description of which code is yours. You need to state if your program works fully or if there are non-working aspects. If the latter is the case, you should provide clear details.

Lateness: Late work will be penalised by MMS, see: `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

Mark Banding

- Submissions that completes Part 1 excellently without any attempts on other partst are likely to get marks up to 11.
- Submissions that completes Part 1 and Part 2 excellently without any attempts on other parts are likely to get marks up to 14.
- Submissions that completes Part 1, 2 and 3 excellently without any attempts on other parts are likely to get marks up to 17.
- Submissions that completes all four parts excellently are eligible for marks up to 20.