Practical 2

Neural Networks

Nguyen Dang

University of St Andrews

Overview

This practical involves building *Artificial Neural Networks (ANNs)* for a *multiclass classification* task.

- Part 1: construct an ANN for a given dataset, compare results with a baseline (linear) model.
- Part 2: improve ANN performance by (manual) hyper-parameter tuning.
- Part 3: implement data-preprocessing (imputation of missing values & standardisation).
- Part 4: identify input features' importance.

Overview

This practical involves building *Artificial Neural Networks (ANNs)* for a *multiclass classification* task.

- Part 1: construct an ANN for a given dataset, compare results with a baseline (linear) model.
- Part 2: improve ANN performance by (manual) hyper-parameter tuning.
- Part 3: implement data-preprocessing (imputation of missing values & standardisation).
- Part 4: identify input features' importance.

You'll be given an **ANN library (minet)** and a **starter code** to work with.

minet

a *MInimal neural NETwork library* for Java, written by Phong Le:

- Github repo: https://github.com/lephong/minet
- Javadoc: https://lephong.github.io/minet/
- Tutorials: https://github.com/lephong/minet/tree/master/tutorial
- minet uses the Linear Algebra library jblas (http://jblas.org/) for matrix computations.
    - both minet and jblas are included in the starter code.

minet

minet is a *small educational project* for Java users to learn the basics of neural networks

- Its structure is mainly inspired by *PyTorch* (https://pytorch.org/).

minet

minet is a *small educational project* for Java users to learn the basics of neural networks

- Its structure is mainly inspired by *PyTorch* (https://pytorch.org/).
- The library is minimal and extendable, and efficiency is *not* the main focus.

minet

minet is a *small educational project* for Java users to learn the basics of neural networks

- Its structure is mainly inspired by *PyTorch* (https://pytorch.org/).
- The library is minimal and extendable, and efficiency is *not* the main focus.
- Important note: its implementation uses ***row-vector*** (we used *column-vector* in the lectures).

column-vector representation

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

row-vector representation

$$\mathbf{w} = [w_1 \; w_2 \; \dots \; w_n]$$

minet

minet is a *small educational project* for Java users to learn the basics of neural networks

- Its structure is mainly inspired by *PyTorch* (https://pytorch.org/).
- The library is minimal and extendable, and efficiency is not the main focus.
- Important note: its implementation uses ***row-vector*** (we used *column-vector* in the lectures).

column-vector representation

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ & & \dots & \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix}$$

training example 1

row-vector representation

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ & & \dots & \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

training example 1

A dataset with m training examples, each of which has n input features

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network

MNIST example: an ANN with one hidden layer, 1000 hidden nodes and ReLU activation function

```
int indims = trainset.getInputDims();  // get the dims of x (784)
int hiddims = 1000;
int outdims = 10;  // there are 10 digits (i.e. 10 categories)
Sequential net = new Sequential(new Layer[] {
        new Linear(indims, hiddims, new Linear.WeightInitXavier()),
        new ReLU(),
        new Linear(hiddims, outdims, new Linear.WeightInitXavier()),
        new Softmax()});
```

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network

MNIST example: an ANN with one hidden layer, 1000 hidden nodes and ReLU activation function

```
int indims = trainset.getInputDims();  // get the dims of x (784)
int hiddims = 1000;
int outdims = 10;  // there are 10 digits (i.e. 10 categories)
Sequential net = new Sequential(new Layer[] {
        new Linear(indims, hiddims, new Linear.WeightInitXavier()),
        new ReLU(),
        new Linear(hiddims, outdims, new Linear.WeightInitXavier()),
        new Softmax()});
```

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network
- **Sequential**: A sequential container, containing a sequence of layers

MNIST example: an ANN with one hidden layer, 1000 hidden nodes and ReLU activation function

```
int indims = trainset.getInputDims();  // get the dims of x (784)
int hiddims = 1000;
int outdims = 10;  // there are 10 digits (i.e. 10 categories)
Sequential net = new Sequential(new Layer[] {
        new Linear(indims, hiddims, new Linear.WeightInitXavier()),
        new ReLU(),
        new Linear(hiddims, outdims, new Linear.WeightInitXavier()),
        new Softmax()});
```

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network
- **Sequential**: A sequential container, containing a sequence of layers
- **Loss**: loss functions

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network
- **Sequential**: A sequential container, containing a sequence of layers
- **Loss**: loss functions
- **Optimizer**: Algorithms for updating the weights (currently *Stochastic Gradient Descent* only)

minet

Some important classes:

- **Dataset**: for loading and iterating over a dataset.
- **Layer**: represent a “layer” in a neural network
- **Sequential**: A sequential container, containing a sequence of layers
- **Loss**: loss functions
- **Optimizer**: Algorithms for updating the weights (currently *Stochastic Gradient Descent* only)

See minet’s tutorial 1 for a full example: https://github.com/lephong/minet/blob/master/tutorial/tutorial1.md

Starter Code

- `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.

Part 1: Building an ANN

Datasets are given in the starter code (data/Part1/train.txt and data/Part1/test.txt)

Task 1: build a fully-working ANN by filling in the missing code in the given starter code.

Task 2: train an ANN with a given example hyper-parameter setting (settings/example.json).

Task 3: train a linear classifier as a baseline model, and compare with results of task 2.

Part 1: Building an ANN

Datasets are given in the starter code (data/Part1/train.txt and data/Part1/test.txt)

Task 1: build a fully-working ANN by filling in the missing code in the given starter code.

Task 2: train an ANN with a given example hyper-parameter setting (settings/example.json).

Task 3: train a linear classifier as a baseline model, and compare with results of task 2.

The following questions need to be answered 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

We use the same dataset as in Part 1.

We will improve the performance of our ANN by tuning its hyper-parameters

- You only need to focus on a subset of the hyper-parameters, the rest should be fixed as in part 1.
- No need to a systematic search for the tuning, but you should at least try various settings and report any observed patterns.
- A prediction accuracy of ~70% should be sufficient.

Part 2: Improving your ANN performance

We use the same dataset as in Part 1.

We will improve the performance of our ANN by tuning its hyper-parameters

- You only need to focus on a subset of the hyper-parameters, the rest should be fixed as in part 1.
- No need to a systematic search for the tuning, but you should at least try various settings and report any observed patterns.
- A prediction accuracy of ~70% should be sufficient.

The following questions should be answered 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?

Part 3: Data Preprocessing

In practice, the data often needs to be preprocessed before being used for a Machine Learning model

Datasets given in the starter code (data/Part3/train.txt and data/Part3/test.txt)

- Missing values: some values in the input matrix are missing (your task: ***imputation*** of missing values)
- Input features belong to different scales (your task: ***standardisation*** of input data)

Part 3: Data Preprocessing

In practice, the data often needs to be preprocessed before being used for a Machine Learning model

Datasets given in the starter code (data/Part3/train.txt and data/Part3/test.txt)

- Missing values: some values in the input matrix are missing (your task: ***imputation*** of missing values)
- Input features belong to different scales (your task: ***standardisation*** of input data)

The following questions should be answered in your report

- How did you implement data-preprocessing for the given training and test sets?
- 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)

We will re-use the datasets and hyper-parameter setting of Part 2.

Your task: analyse the input features and identify the (un)important ones.

- You can use any methods and libraries, just remember to include the .jar library file in the lib/ folder

Part 4: Feature Importance Analysis

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

We will re-use the datasets and hyper-parameter setting of Part 2.

Your task: analyse the input features and identify the (un)important ones.

- You can use any methods and libraries, just remember to include the .jar library file in the lib/ folder

The following questions should be answered 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.

Code Submission Requirement

Use the starter code for your submission. You can make change to the code if necessary, but do not comment out the given print statements.

Code Submission Requirement

Use the starter code for your submission. You can make change to the code if necessary, but do not comment out the given print statements.

Make sure that your code can run on the School's lab machines *with the given example commands in the spec* (Parts 1, 2 and 3) or with your submitted running instructions (Part 4 only).

Make sure that your reported results can be *reproducible on the School's lab machines*.

Code Submission Requirement

Use the starter code for your submission. You can make change to the code if necessary, but do not comment out the given print statements.

Make sure that your code can run on the School's lab machines *with the given example commands in the spec* (Parts 1, 2 and 3) or with your submitted running instructions (Part 4 only).

Make sure that your reported results can be *reproducible on the School's lab machines*.

See the spec for more details on submission requirements.

Report Requirement

The typical report structure with Design, Testing and Evaluation is NOT required for this practical. You can go straight to answering the questions for each parts.

Report Requirement

The typical report structure with Design, Testing and Evaluation is NOT required for this practical. You can go straight to answering the questions for each parts.

The ADVISORY limit is 2200 words, or 5 pages (tables, plots & bibliography excluded).

Report Requirement

The typical report structure with Design, Testing and Evaluation is NOT required for this practical. You can go straight to answering the questions for each parts.

The ADVISORY limit is 2200 words, or 5 pages (tables, plots & bibliography excluded).

Your report must be in `pdf` format.

Include the check table on the FIRST page of your report.

Report Requirement

The typical report structure with Design, Testing and Evaluation is NOT required for this practical. You can go straight to answering the questions for each parts.

The ADVISORY limit is 2200 words, or 5 pages (tables, plots & bibliography excluded).

Your report must be in `pdf` format.

Include the check table on the FIRST page of your report.

Do not include screenshots of your code or your program's output unless it is necessary.

If you include plots in your report, remember to give clear captions explaining what each axis (and color, if applicable) represents.

Some other important points

Please use lab machines!

- But do not use host servers for computation (just for login to lab machines): https://systems.wiki.cs.st-andrews.ac.uk/index.php/Lab_PCs

Full practical spec and starter code are on studres

- https://studres.cs.st-andrews.ac.uk/CS3105/Practicals/P2

Similar to Practical 1, FAQ will be provided on Teams.

Deadline: 9pm UK time Nov 22nd, 2021

- MMS deadline is definitive.
- Lateness policy: https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Practical is worth 50% of coursework for module → 30% of whole module mark