UNIVERSITY OF PISA, DEPARTMENT OF COMPUTER SCIENCE

A Neural Network Simulator for the Machine Learning Course

Marco Ponza

May 20, 2014

This report is about the project I developed for the machine learning course: a neural network simulator for classification and regression tasks. In detail I built a multilayer perceptron with one layer of hidden units. The neural network was tested on two different types of datasets.

1 INTRODUCTION

For this project I developed a neural network simulator and its own neural network from scratch. The neural network implemented is a multilayer perceptrons with one layer of fully connected hidden units. The method used for the training is the online back-propagation algorithm (with momentum and weight-decay), as seen during the machine learning course. All the software was completely developed using the C++ programming language.
The neural network was tested on two different datasets:

- Monk, using the simple validation for the model selection (on every Monk dataset);
- Loc, using the multifold cross-validation for the model selection.

After a brief description of the structure of the code, I will describe the results obtained by validating and testing the neural network on these datasets. Particular focus will be given to the model selection process.

2 Structure of the Code

2.1 DataSet

The DataSet class is the container of the Loc or Monk dataset. It allows to get the data pre-processed using the *1-of-k* coding or scaling all the data values between [0; 1]. It also allows to re-scale a value between [0; 1] to its original value (i.e. an output computed by the neural network).

2.2 NeuralNetworkSimulator

The NeuralNetworkSimulator is the only class which uses all the other components implemented. You can use this class to run the neural network on the Monk or Loc problem specifying the hyper-parameters or doing the model selection.

2.3 HyperParameters

This class includes all the configurable neural network parameters and it is used for instantiating the NeuralNetwork class. The application allows to specify:

- $task$, which can be Monk or Loc;
- $nNeurons$, the number of units of the hidden layer;
- $epochs$, the maximum number of epochs computable;
- η, the learning rate;
- α, the momentum parameter;
- λ, the weight-decay parameter.

2.4 Neuron

The Neuron class represents a single unit of the neural network. It can compute its delta value and update its weights. The activation function used by every neuron is the following:

$$\frac{1}{1+e^{-net_i(x)}}$$

where net is the net input to unit i.

2.5 Neural Network

This class is the core of the project because it is, obviously, the neural network used by the NeuralNetworkSimulator. The NeuralNetwork has two vectors of neurons: the hidden and the output layer. It can be trained using the online back-propagation algorithm and compute its results on a specific dataset. If a validation set is passed to the training function, an early-stopping function is used:

- if the MSE on the validation set at epoch $i + 100$ is higher than the MSE on the validation set at epoch i then the training is stopped;
- if the MSE on the validation set is 0 then the training is stopped.

2.6 Validation

The Validation class allows to perform the simple validation on a training data:

- the first 70% of the training data, the training set, is used for training the neural network;
- the next 15% of the training data, the validation set, is used for the validation process in order to choose the best hyper-parameters configuration with the lowest average MSE on 10 runs;
- the last 15% of the training data, the "stopping set", is used for computing the maxiumum number of epochs to use on the test set.

The validation procedure, for every HyperParameters specified, computes the average MSE on 10 runs of the neural network trained on the training set and using the early-stopping function on the validation set. At every run the weights of the neural network are random initialized. The maximum number of epochs computable is fixed to 1000. The best hyper-parameters configuration chosen is the one which has the lowest average MSE.
Then, with the best hyper-parameters configuration found, the neural network is trained on the training and validation set (85% of the training data) and the maximum number of epochs is determined by the early-stopping function using the stopping set as validation set. Also in this case the maximum number of epochs computable is fixed to 1000.
The simple validation was used on the Monk's problem.

2.7 CrossValidation

This class allows to perform the k-fold cross-validation on a training set for every HyperParameters specified. We can distinguish two cases:

1. when the early-stopping function is disabled: in this case the standard k-folds cross-validation is performed on every HyperParameters specified. The final MSE computed is the average MSE calculated on k folds;
2. when the early-stopping function is enabled: in this case the training data is splitted on k + 1 folds and, for every HyperParameters specified, the k-fold cross-validation is performed:
   a) the neural network is trained on k - 1 folds;
   b) the MSE is performed on the fold which was left out.

Then a) and b) are repeated for a total of k runs and the final MSE is the average MSE computed on k folds. At every run (on every fold) the weights of the neural network are random initialized. The maximum number of epochs was fixed to 10000. The best hyper-parameters configuration chosen is the one which has the lowest average MSE.

At the end, with the best hyper-parameters configuration found, the neural network is trained on k folds and the maximum number of epochs is determined by the early stopping function using the k + 1 fold as validation set. Also in this case the maximum number of epochs computable is fixed to 10000.

The k-folds cross-validation was used on the Loc's problem.

2.8 FileManager

This class is used for managing all the reading and writing operations. In detail it allows to:

- read the Monk and Loc files;
- write the performance of the neural network (used by the plots);
- write the competition predictions to file.

3 The Monk's Problem

In the Monk's Problem:

- the neural network used for this task has 17 inputs (because of the preprocessing) and 1 output;
- all the datasets were preprocessed using the *1-of-k* coding;
- an input is classified as:
  - 1 if the output of the neural network is ≥ 0.5;
  - 0 otherwise;
- the simple validation was run on every datasets with all the combinations of these hyper-parameters:

Number of Neurons	η	α	λ
{3, 4, 5}	{0.05, 0.08, 0.1}	{0, 0.2, 0.5}	{0, 0.02, 0.05}

- in the tests:
  - the hyper-parameters configuration used, for every Monk's problem, is the one found by the simple validation;
  - the neural network was trained using all the data in the Monk's ".train" file;
- the points of the plots were generated every 10 epochs.

We will now show all the plots generated by running the neural network with the best hyper-parameters found during the model selection.

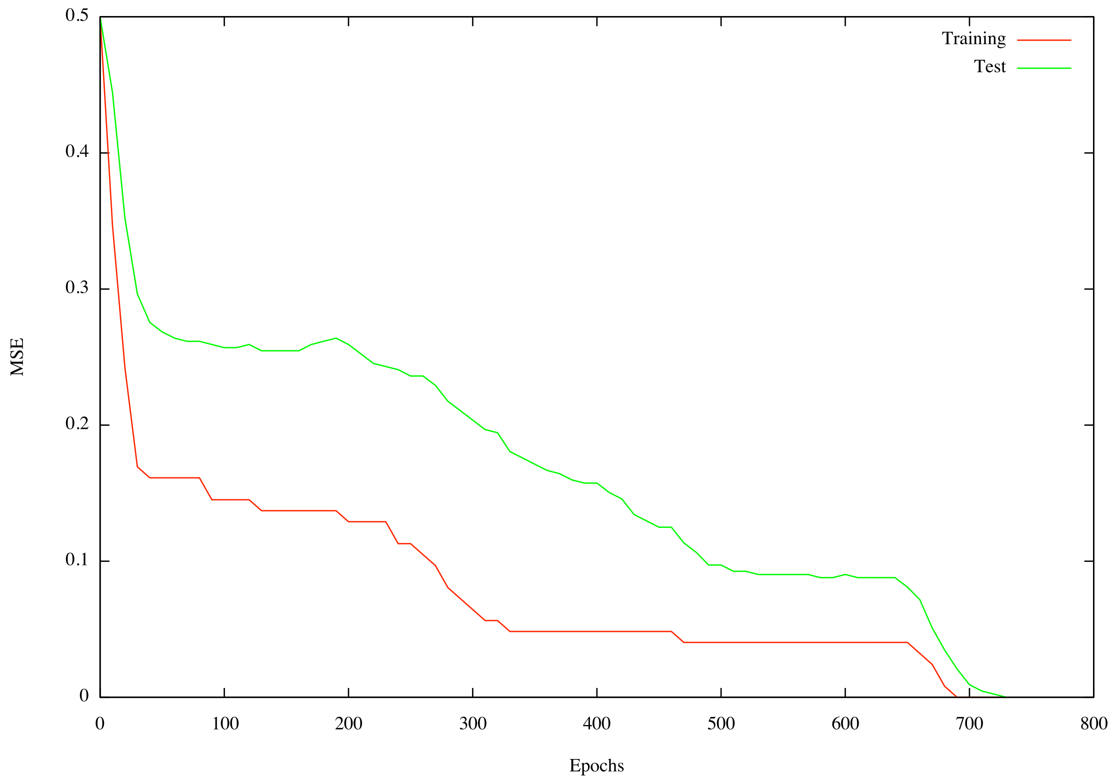


Figure 3.1: Neural network's learning curve on the Monk 1 dataset.

3.1 The Monk 1 Dataset

The best hyper-parameters configuration found by the validation process is the following:

Number of Neurons	Epochs	η	α	λ
3	800	0.05	0	0.02

where Epochs is the number of epochs computed by the early-stopping function on the stopping set (see section 2.6) during the validation process.
Despite the high value of the learning rate the behaviour of the neural network is stable and the MSE on the training and test set decreases very fast in the firsts 50 epochs (see figure 3.1). Although the maximum number of epochs computable is 800, the MSE on the training and test set is 0 after about 700 epochs: this is because, in the testing process, the neural network was trained on a higher number of data than the validation process.

3.2 The Monk 2 Dataset

The best hyper-parameters configuration found by the model selection is the following:

Number of Neurons	Epochs	η	α	λ
5	400	0.05	0.5	0

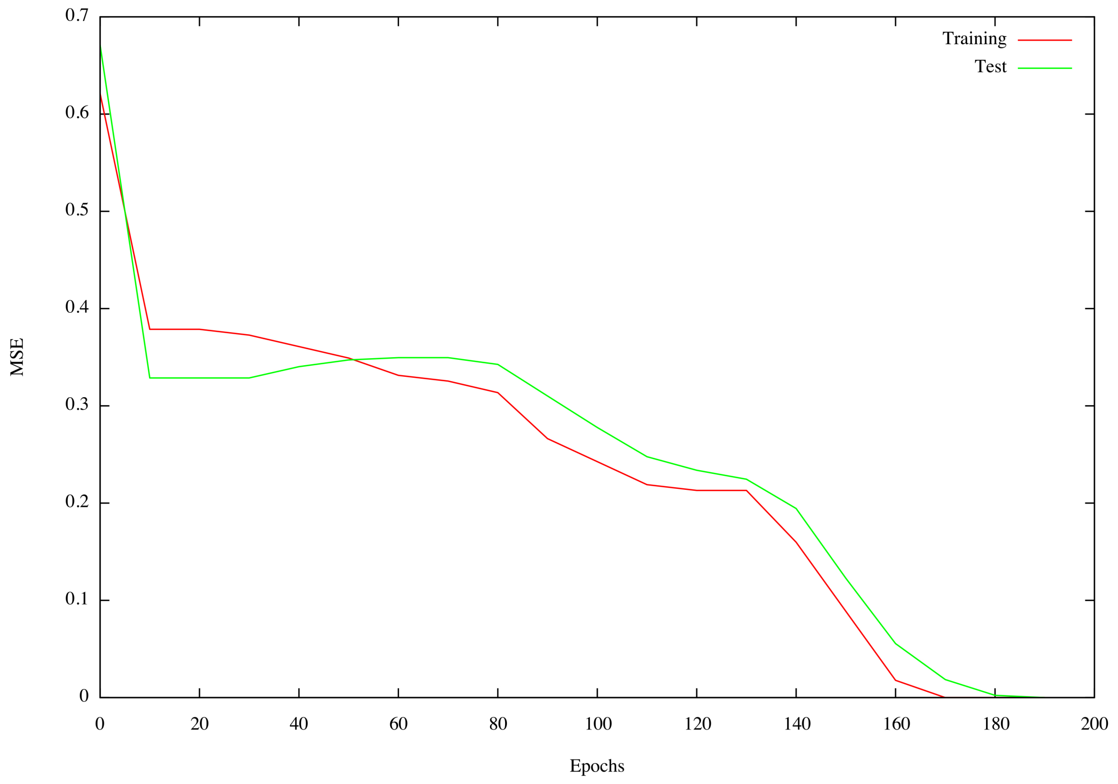


Figure 3.2: Neural network's learning curve on the Monk 2 dataset.

where Epochs is the number of epochs computed by the early-stopping function on the stopping set (see section 2.6) during the validation.
Because of the high value of the learning rate (see figure 3.2) the MSE on training and test set decreases fast in the first 10 epochs and slower in the next epochs. Also in this case the MSE on the training and test set becames 0 before the maximum number of epochs chosen by the model selection. This is because, as said before, in the testing process the neural network was trained on a higher number of data than the validation process.

3.3 The Monk 3 Dataset

The best hyper-parameters configuration found by the validation process is the following:

Number of Neurons	Epochs	η	α	λ
4	200	0.05	0.2	0

where Epochs is the number of epochs computed by the early-stopping function on the stopping set (see section 2.6) during the simple validation.
Because of the high value of the learning rate (see figure 3.3) the MSE on training and test set decreases very fast in the first 20 epochs. After about 70 epochs the MSE on the test set

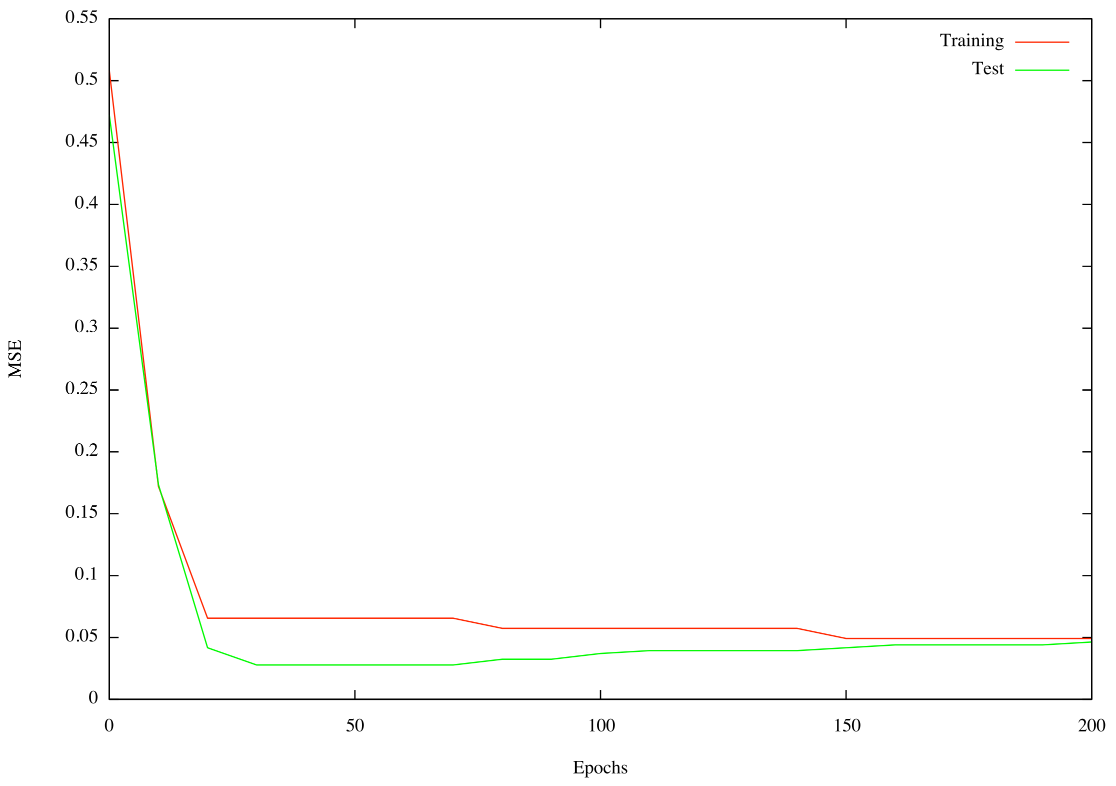


Figure 3.3: Neural network's learning curve on the Monk 3 dataset.

increases but, thanks to the maximum number of epochs computed by the model selection, the training is stopped at 200 epoch with, more or less, the same MSE on the training set.

4 The Loc's Problem

In the Loc's Problem:

- the neural network used for this task has 5 inputs and 2 outputs;
- the dataset was preprocessed scaling the values between [0; 1];
- for the sake of simplicity the MSE was computed without re-scaling the outputs, while the MEE was computed re-scaling the outputs;
- since MSE and MEE have a very similar behaviour, the k-folds cross-validation computes only the average MSE on k folds;
- the first 70% of the data, called "modeling data", in the "LOC-UNIPI-TR.csv" file was used for the model selection;

- the last 30% of the data, the testing data, in the “LOC-UNIPI-TR.csv” file was used for the testing process;
- the points of the plots were generated every 10 epochs.

4.1 Model Selection

4.1.1 Choosing the Learning Rate

In order to choose a good learning rate I performed some runs. In detail:

- the first 70% of the modeling data was used for the training;
- the last 30% of the modeling data was used as validation set.

The results obtained from the runs are the following:

- $\eta = 0.05$, this is the case which reached the best MSE. The learning is fast but, with an high number of neurons, the MSE starts to increase and the learning is stopped by the early-stopping function before the end of the epochs;
- $\eta = 0.005$, the MSE reached is slighty higher than the previous case, but the learning ends, also with an high number of neurons, at the epoch 10000;
- $\eta = 0.0005$, the learning is slow and the MSE computed after 10000 epochs is very high.

The final learning rate value was chosen to 0.005 since it was the best compromise: the learning is stable and the MSE reached is good also with different number of neurons.

4.1.2 Cross-Validation

Since the cross-validation process takes a lot of time:

1. firstly, I performed a 5-folds cross-validation in order to have a smaller set of hyper-parameters configurations to use on another cross-validation with more folds;
2. secondly, I performed a 10-folds cross-validation, using the results from the previous step, in order to determine the hyper-parameters configuration with the lowest average MSE. In the end, the best hyper-parameters configuration chosen was the best hyper-parameters configuration found by this step.

5-FOLDS The 5-folds cross-validation was run with all the combinations of these hyper-parameters:

Number of Neurons	η	α	λ
{10, 15, 20}	0.005	{0, 0.1, 0.3, 0.5, 0.8}	{0, 0.001, 0.003, 0.005, 0.008}

in order to reduce the number of hyper-parameters configurations used by the next step. The early-stopping function was enabled. The best 3 hyper-parameters configurations found by the 5-folds cross-validation are:

Number of Neurons	Epochs	η	α	λ	MSE
15	10000	0.005	0.5	0.001	0.039991
20	10000	0.005	0.5	0	0.0397374
20	10000	0.005	0.8	0.001	0.0409936

As we can see from the results of the 5-folds cross-validation, although the early-stopping function was enabled, the number of epochs chosed was 10000.

10-FOLDS The 10-folds cross-validation was run with the 3 best results from the 5-folds cross-validation. This step is used only for determining the best hyper-parameters configuration to use in the test without changing the hyper-parameters configurations chosen by the previous step. The early-stopping function was disabled. The results are:

Number of Neurons	Epochs	η	α	λ	MSE
15	10000	0.005	0.5	0.001	0.0406109
20	10000	0.005	0.5	0	0.0374404
20	10000	0.005	0.8	0.001	0.041517

At the end of the model selection, the best hyper-parameters configuration chosen by the 10-folds cross-validation is the following:

Number of Neurons	Epochs	η	α	λ
20	10000	0.005	0.5	0

4.2 TESTING

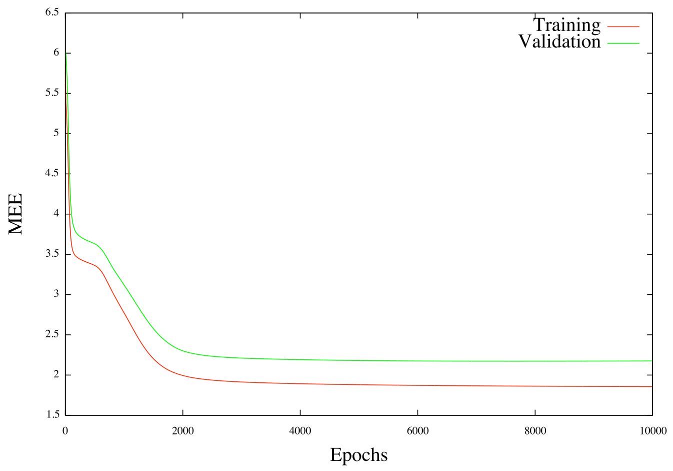
In the testing process the neural network:

- used only the best hyper-parameter configuration found by the last step of the model selection;
- was entirely trained on the modeling data;
- was tested using the testing data.

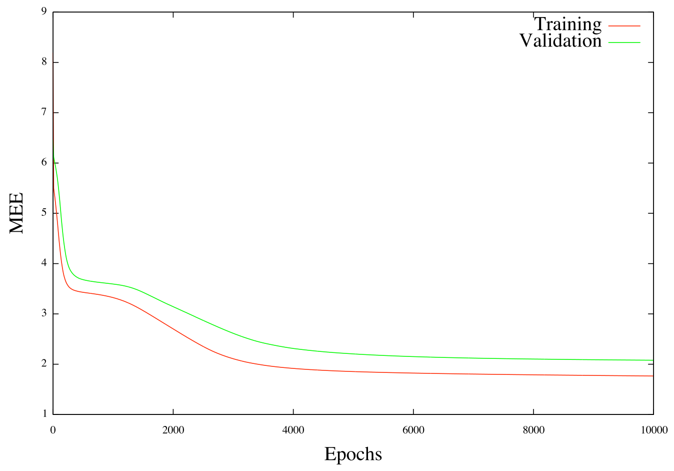
Seeing figure 4.2 the learning curve is good and the neural network behaviour is stable. In detail the MEE decreases quite fast in the firsts 5000 epochs and very slowly afterwards. The final average MEE computed on the test set by running 10 times the neural network is 1.95493.

5 THE AA1 CUP: BLIND TEST

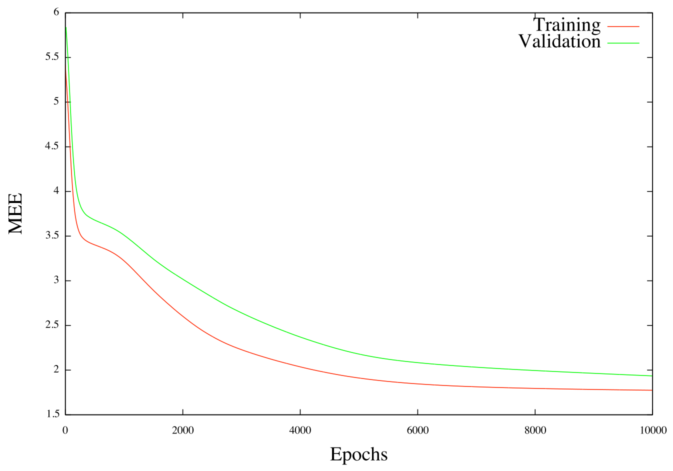
In order to generate the blind test predictions, the neural network was trained on all the data in the "LOC-UNIPI-TR.csv" file. The hyper parameter configuration used was the same used during the testing process.



(a) $nNeurons = 10$, $\alpha = 0.8$, $\lambda = 0.003$



(b) $nNeurons = 15$, $\alpha = 0.5$, $\lambda = 0.001$



(c) $nNeurons = 20$, $\alpha = 0.5$, $\lambda = 0$

Figure 4.1: Neural network's learning curves changing the hyper-parameters configurations. The learning rate was fixed to 0.005.

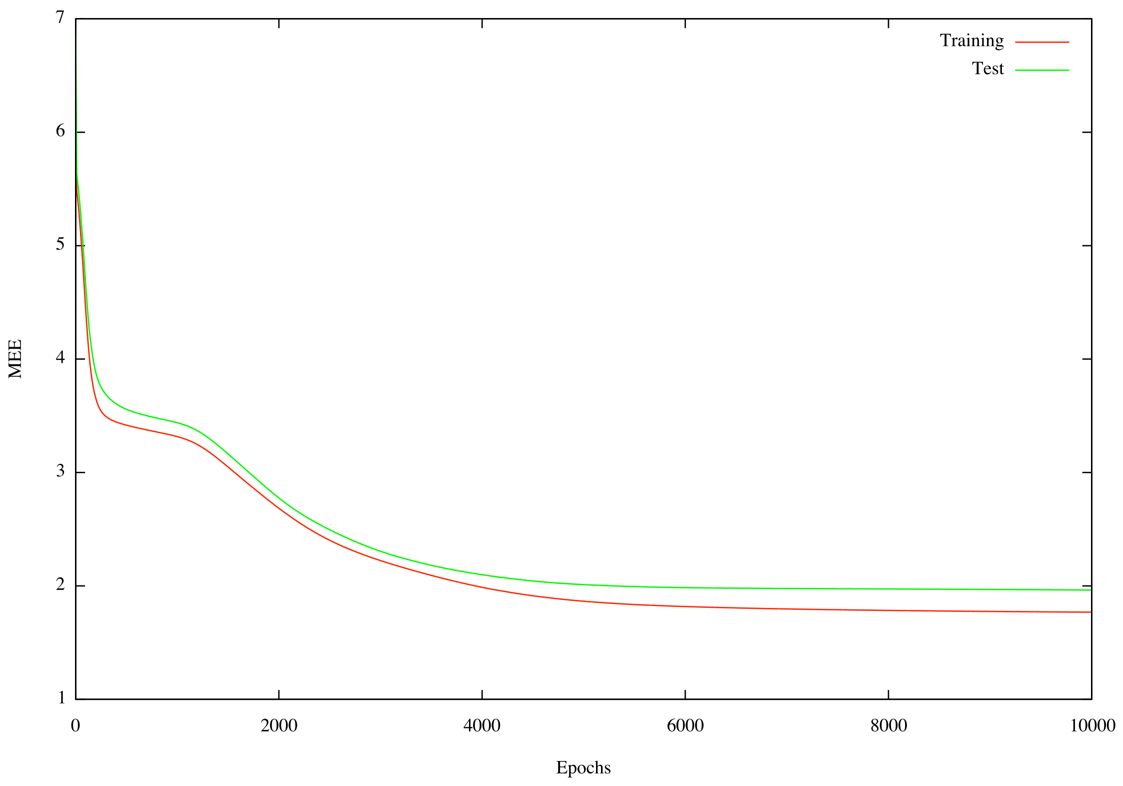


Figure 4.2: Neural network's learning curve on the Loc test with the best hyper-parameters configuration chosen by the model selection.

6 Conclusions

In this project I learned how to build a feed forward fully connected neural network using the theory from the machine learning course and my programming skills. Developing the Neural Network Simulator I understood the importance of the model selection process and the right use of the data (training, validation and test).
Although the the back propagation algorithm is quite easy to understand, debugging its implementation is time consuming.
Fortunately, I didn't find particular problems developing the model selection, with the exception of how to choose the number of epochs. This problem was solved using the early-stopping function on a dataset which was not used during the model selection process (see sections 2.6 and 2.7).
Another problem was the running time: on my old computer the model selection (especially the k-folds cross-validation) takes a lot of time.