Neural Network Simulator

Dininta Annisa, Edoardo Federici
d.annisa@studenti.unipi.it, e.federici1@studenti.unipi.it
Master Degree in Computer Science
ML course (654AA), Academic Year: 2020/2021
Date: 24/01/2021
Type of project: **A**

Abstract

This short report will describe our implementation of an Artificial Neural Network using Python. We will use mini-batch Gradient Descent along with standard Backpropagation as the gradient computing technique. We will explain our choices on how we approached grid search, hyper-parameter tuning and cross validation. The ANN has been tested on the MONK datasets and on the CUP dataset.

1 Introduction

This project aims to develop a neural network simulator, specifically a feedforward multi layer perceptron. We implement a backpropagation algorithm to learn the weights with several versions of gradient descent: stochastic, batch, and mini-batch. The simulator is applied to MONK's classification problem as a benchmark and ML-CUP regression problem. The best configurations for both dataset are obtained using grid search and k-fold cross validation.

2 Method

This simulator is implemented in Python 3.9 and uses Numpy library (version 1.19.4) for linear algebra operations. Section 2.1 describes the implementation, including which parameters can be tuned. Section 2.2 describes how the dataset is preprocessed before the network is trained, while section 2.3 contains the model selection and evaluation schema.

2.1 Implementation Detail

The main class of this simulator is `MLPBase`, where the backpropagation algorithm is implemented by minimizing mean squared error (MSE). This class is a base class for `MLPClassifier` and `MLPRegressor`, each having different mechanism to predict new

data. For the classification task, the activation function in the output unit is sigmoid; thus, if the output is >0.5, it is classified as 1, otherwise 0. Meanwhile, in the regression task, the activation function in the output unit is linear: the prediction mechanism outputs directly the final result.

The simulator allows us to specify some parameters:

1. **Number of hidden layers** and **number of units per layer**, but the architecture is always fully-connected feedforward.
2. **Batch size**, if the value is 1, it means stochastic gradient descent is performed.
3. **Maximum iteration**, to stop training after a certain number of epochs.
4. **Activation function**, which can be either *linear*, *heaviside*, *sigmoid*, *tanh*, *relu*, *softplus*, or *leakyrelu*.
5. **Learning rate**, fixed for every iteration.
6. **Regularization parameter**, for L2 (ridge) regularization.
7. **Momentum coefficient**, to use momentum.
8. **Shuffle** and **seed**, to randomize the order of input data.

2.2 Preprocessing

We preprocessed the MONK dataset as it contains categorical attributes. We applied one-hot encoding to all the three tasks, obtaining 17 numeric attributes from the original 6.

2.3 Validation Schema

The final model used in MONK's classification is selected using k-fold cross validation with k = 5, which means the development data is split into 80% training set and 20% validation set with different combinations in 5 iterations. The test set is already available separately to estimate the final model's performance.

For ML-CUP problem, since the available test set is a blind test set, we decided to split the dataset into a development set (80%) and an internal test set (20%). The development set is used in model selection by means of k-fold cross validation with k = 4.

3 Experiments

3.1 MONK Results

The hyper-parameters of the chosen final model for the three MONK's problems are shown in Table 1, along with the MSE and accuracy of both training set and test set. The '#Units' column refers to the total hidden units in the only hidden layer. We used only one hidden layer as it shows good enough results. We also referred to existing work which is able to achieve 100% accuracy with only one hidden layer [1].
Eta is the learning rate, alpha is the momentum coefficient, and lambda is the regularization parameter.

Task	#Units	Eta	Alpha	Lambda	MSE (TR/TS)	Accuracy (TR/TS)
MONK1	4	0.05	0.4	0	0/0	100%/100%
MONK2	5	0.05	0.6	0	0/0	100%/100%
MONK3	7	0.01	0.4	0	0.065/0.030	93.44%/96.99%
MONK3+reg.	7	0.02	0.4	0.0001	0.065/0.027	93.44%/97.22%

Table 1: Average prediction results obtained for the MONK's tasks.

All four models show the best result when using *relu* as the activation function of the hidden units. The weights are initialized with random values with a uniform distribution in the interval [-0.7, 0.7]. We trained the network with stochastic, batch, and mini-batch gradient descent to verify the correctness of the simulator, but finding the best hyper-parameters is done with stochastic gradient descent since it requires less epochs to reach the same accuracy. The learning curve of the three tasks are shown in Figure 1, 2, and 3 respectively. For MONK 3, we also use L2 regularization as shown in Figure 4.

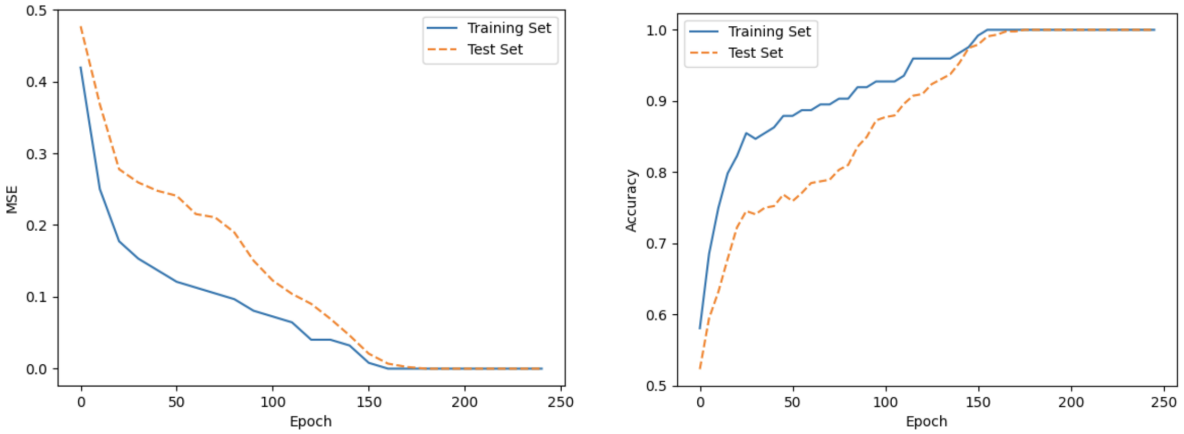

Figure 1: Plot of the MSE and accuracy for MONK 1.

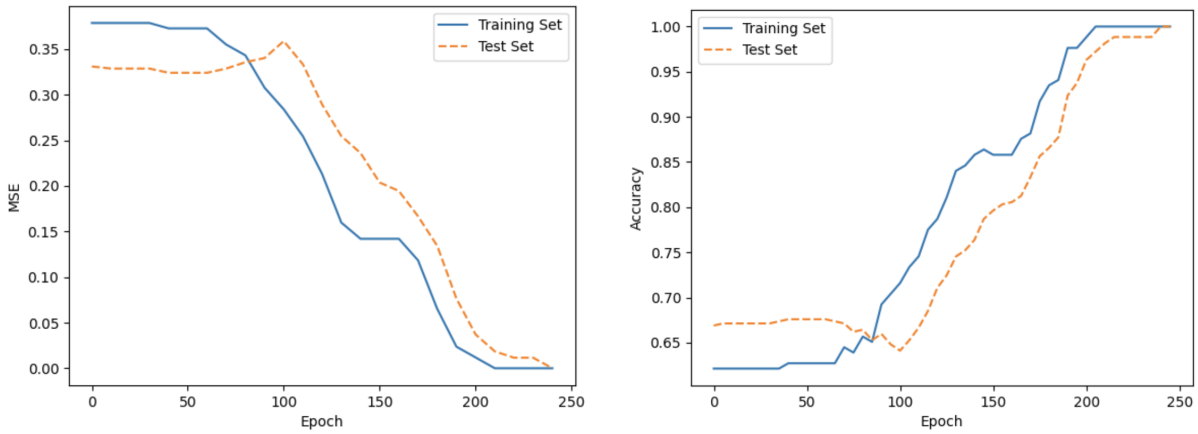

Figure 2: Plot of the MSE and accuracy for MONK 2.

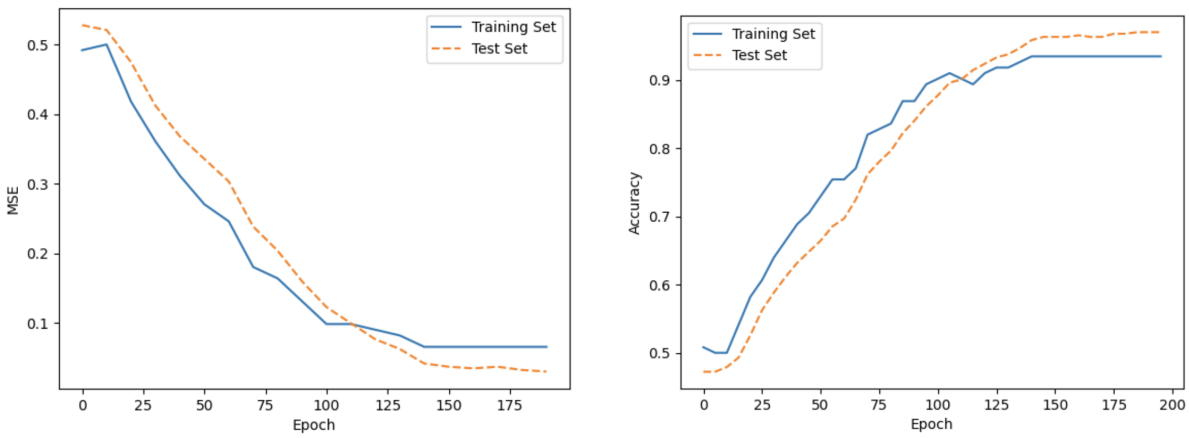

Figure 3: Plot of the MSE and accuracy for MONK 3.

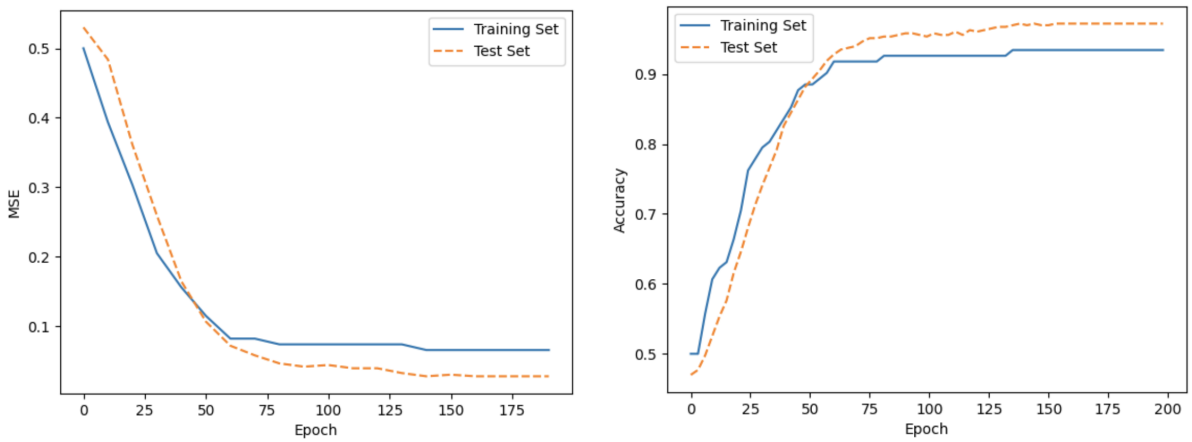

Figure 4: Plot of the MSE and accuracy for MONK 3 with L2 regularization.

3.2 CUP Results

The neural network is trained with 80% of the available data (1,219 samples) with k-fold cross validation (k = 4), while the remaining 20% (305 samples) is used as internal test set. Initially, we trained using random parameters with wide gaps and plotted the learning curve to obtain admissible range of parameters. We observed that the learning curve is unstable when the learning rate is >0.1 and smoother when the activation function in the hidden units is *logistic* function. It is also smoother for full-batch gradient descent, but since it takes longer computation time, we decided to use the batch version with batch size = 10.

After the screening phase, we performed a grid search with hyper-parameters range values as in Table 2. Each value is randomly generated with uniform distribution within the specified range. The 10 best performing configurations are reported in Table 3.

Hyper-parameter	**Values**
Learning rate	[0.01, 0.05]
Momentum	[0.5, 0.8]
L2 regularization	[0.0001, 0.0003]
Initial weight	[-0.7, +0.7]
Number of units per hidden layer	[3, 20]
Number of hidden layers	2

Table 2: Variable hyper-parameters and their respective possible values to be explored with grid search.

#Units	**Eta**	**Alpha**	**Lambda**	**MEE (TR)**	**MEE (VL)**	**MEE (TS)**
16, 17	0.02418	0.60313	0.0001096	2.70550	2.98861	3.23757
17, 19	0.03781	0.51418	0.0001720	2.71151	2.99717	3.16263
20, 15	**0.02418**	**0.65178**	**0.0001096**	**2.71822**	**3.00579**	**3.14738**
16, 17	0.02418	0.65178	0.0001096	2.76744	3.02773	3.20450
9, 13	0.02634	0.50323	0.0001053	2.76809	3.03415	3.30256
9, 13	0.01782	0.50323	0.0001053	2.81362	3.03455	3.02616
10, 17	0.04329	0.73335	0.0001454	2.75068	3.03474	3.09858
15, 15	0.02387	0.50991	0.0001258	2.74562	3.03949	3.02145
14, 9	0.03781	0.71333	0.0001199	2.77061	3.05305	3.20132
17, 19	0.03781	0.71333	0.0001199	2.79957	3.05546	2.91358

Table 3: Best models obtained by grid search, sorted by ascending MEE of VL. The final model is highlighted with bold text.

To select the most apt model for the blind test set, we picked the configuration with the best trade-off between MEE and learning curve's smoothness. Even though the first

two models in Table 3 have the lowest MEE for the validation set, their learning curve is a bit unstable; thus we selected the third configuration, highlighted in bold.

To test whether the spikes we observed in the learning curves of those configurations were due to the low batch size, we ran the same model with the same hyper-parameters but using batch size = 200. It greatly improved the learning curve's smoothness, as can be seen in Figure 5 and 6. However, since we did not include the batch size in the grid search, we felt uneasy to select this configuration as the final model, so our final model is still trained with 10 as the batch size. This observation is in line with Machine Learning theory, since an higher batch size will lead to more accurate gradients. A lower batch size allows the weights values to be more volatile, leading to a lower convergence but helping escaping local minima.

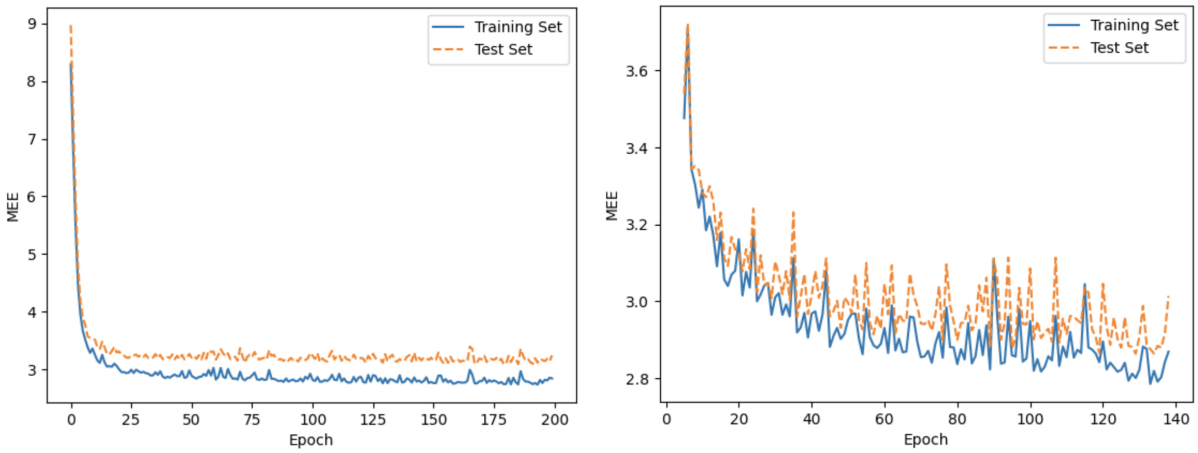

Figure 5: Plot of the MEE of the final model with batch size = 10; original plot (left) and zoomed-in version (right).

CPU	MONK (seconds)	CUP (minutes)
2.1 GHz Quad-Core AMD Ryzen 5 3500U	30	10-13
2 GHz Quad-Core Intel Core i5	10	5-6

Table 4: Time required to train the datasets on our CPUs.

In Table 4, we show our hardware resources and the time necessary to train one configuration of the model both on the MONK dataset and on the CUP. It took the same time to train the model on each of the MONK datasets. The model is not particularly fast and it showed especially during the grid search phase, where we tested 150 different configurations: it took several hours to complete.

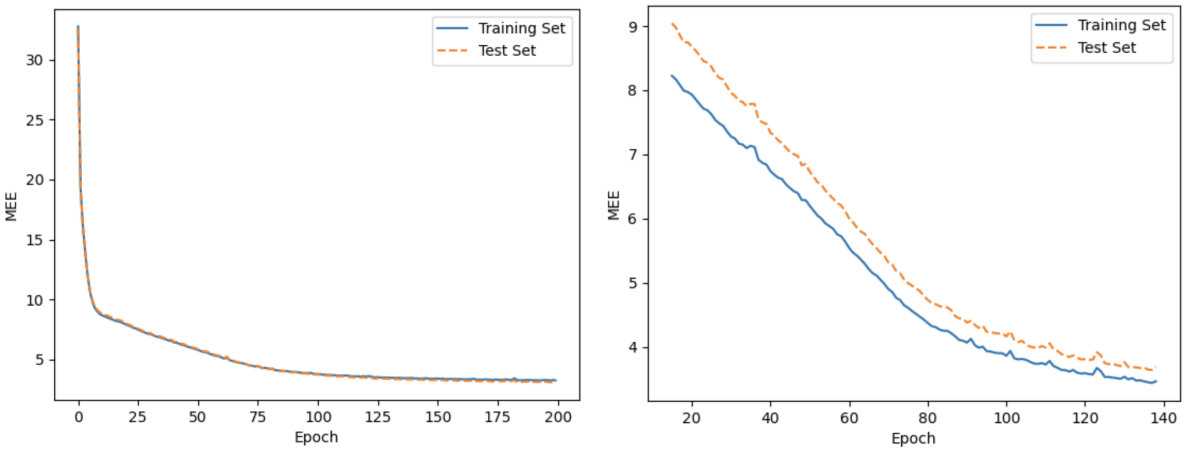

Figure 6: Plot of the MEE of the final model with batch size = 200; original plot (left) and zoomed-in version (right).

4 Conclusion

The MONK dataset was essential in showing how much the L2 regularization parameter helps addressing a configuration that would otherwise overfit.

The model would have benefited from a deeper exploration of the hyper-parameter space, especially regarding the hidden layer size, and from another approach to model selection to validate more soundly our claims, but due to time and hardware constraints we were not able to implement those improvements.

Nonetheless, it has been an important didactic resource to help us understand what it means to design and develop a Machine Learning model and how each moving part is connected to each other.
The blind test result is available in the file `thumbfish_ML-CUP20-TS.csv`.
Our group is called thumbfish.

Acknowledgments

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking.

References

[1] S. Thrun, Jerzy Bala, Eric Bloedorn, Ivan Bratko, J. Cheng, Kenneth De Jong, Sašo Džeroski, Scott Fahlman, Doug Fisher, R. Hamann, Kenneth Kaufman, S. Keller, I. Kononenko, J. Kreuziger, T. Mitchell, P. Pachowicz, Yoram Reich, Haleh Vafaie, and J. Wnek. The MONK's Problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Pittsburgh, PA, January 1991.

Appendix A

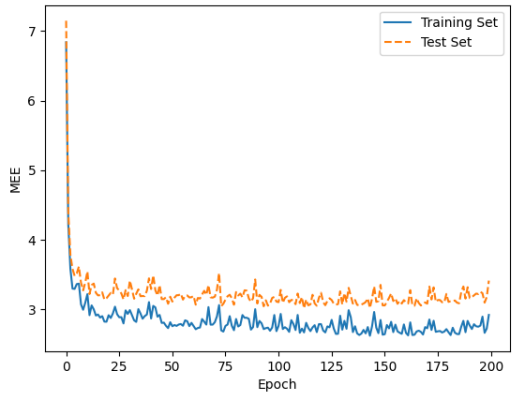

Figure 7: Plot of the MEE of one particularly bad configuration (the second one in Table 3) found with grid search, batch size = 10

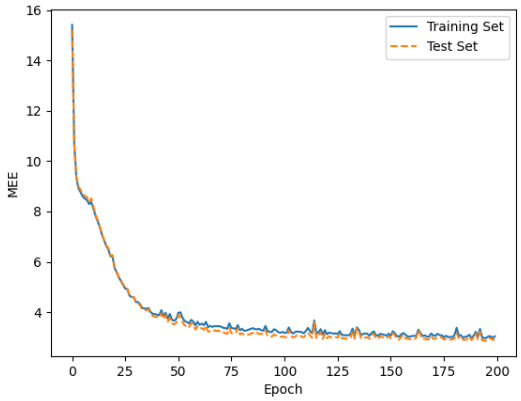

Figure 8: Plot of the MEE of one particularly bad configuration (the second one in Table 3) found with grid search, batch size = 200