Machine Learning Project report

Gerlando Gramaglia – 530269 – g.gramaglia1@studenti.unipi.it
Domenico Profumo – 533695 – d.profumo@studenti.unipi.it
MSc in Artificial Intelligence

ML course (Code: 654AA), Academic Year: 2020/2021
Date: *25/05/2021*
Type of project: **A**

ABSTRACT

This report represents the implementation of an Artificial Intelligence Neural Network from scratch using Python programming language. We used k-fold cross validation, selected the 8 best models using a grid search and created an ensamble of them. In the following sections we describe all the components implemented in the project.

1. INTRODUCTION

The aim of the project was to create a Neural Network that learn from training data using the best configuration of hyperparameters. To finds the hyperparameters we use a grid search with 5-fold cross validation and then we tested the models on internal test set (used only for testing phase), during the training phase, models uses the classic backpropagation algorithm. The correctness of network implementation was tested on "Monk's problem" [1] and then we used ML-CUP20-TR to train models. We selected the 8 best models and deployed an ensamble to predict the outputs of ML-CUP20-TS. The Neural Network can resolve both classification and regression problem. In chapter 2 we describe all libraries we used, a software overview, validation schema. In chapter 3 we provide result experiments about Monk Problem and ML-CUP. In chapter 4 the conclusions.

2. METHOD

We implemented the software using Python programming language. We used some mathematics and data manipulation tools such as: numpy, sklen, pandas, matplotlib and math. No machine learning tool was used. We implemented a multi-layer feed forward neural network that learns through back-propagation algorithm using Stochastic Gradient Descent technique. The network can learn using different activation function, momentum, thikonov regularization, learning rate schedules (constant, variable), different weight update policies (online, mini-batch, batch), early stopping. In section 2.1 we describe our implementation choices and main classes. Section 2.2 lists all hyper-parameters and how to set them. In section 2.3 we describe validation schema used for ML-CUP.

2.1 Model implementation

The main class is `Neural_network,` it contains the type of problem (classification or regression), hyper-parameters (section 2.2) and a structure that contains the layers of network. Methods of `Neural_network` class are:

- `Fowarding` and `forward` for the "forwarding phase"
- `training` for training phase, it takes as parameters: number of epochs, training and validation set and weight update policy (expressed by hyper-parameter `batch_size`). It iterates for maximum number of epochs times (or less if early stopping is set) using `Fowarding` and `backpropagation` methods
- `backpropagation`

We used class `Layer` to support `Neural_network` class. An instance contains matrix of weights of the specific layer (one column for each layer's unit) and two matrices to support the training phase: once contains the input after a forwarding phase (one row for each training example, the number of rows depends on hyperparameter *`batch_size`* explained in next paragraph) and the other matrix contains gradient weights of precedent step. The last matrix is used only if a momentum policy is adopted otherwise it is set to a zero matrix. We implemented the utility `Function.py` that contains activation functions and its derivates, Mean Euclidean Error and Mean Square Error functions, weights initialization functions and others to support our network. In `C_k_fold.py` and `Hold_out.py` implements validation schemas k-fold and hold-out. Both use functions of `Model_Selection.py`, in this file we implemented a method that gets a grid of hyperparameters, creates a model for each combination of hyperparameters and return a list of best models. Instead of classic matrices, our neural network uses instances of class `Matrix_io` that facilitates use of data set matrices. `Read_write_file.py` contains functions for data manipulations of ML-CUP and Monk dataset.

2.2 Hyper-parameters

An instance of the `Neural_network` class gets type of problem (regression or classification) and a set of hyper-parameters:

- `units`: an array that contain the cardinality of units for each layer. Array's first element is the input's dimension and last element is the number of output layer's units. It is structured as follow:
- `[input layer's units, units of first hidden layer, units of second hidden layer, … , output layer's units]`
- `alfa`: momentum coefficient. It could be set to zero if you don't want to use momentum policy
- `V_lambda`: Thikonov regularization coefficient. It could be set to zero if you don't want to use thikonov regularization

- `type_learning_rate`: it could be: `“fixed”, “variable”.`
- `learning_rate_init`: the initial learning rate that could change using variable learning rate formula [2] if the parameter `type_learning_rate` is `“variable”` otherwise it remains the same for all training phase.
- `function`: is a string that specify the activation function of hidden layers. All hidden layers use the same activation function. It could be: `{“sigmoidal”, “tanh”, “relu”, “leaky_relu”, “identity”, “linear”`}
- `fun_out`: is a string that specify the function of output layer’s units. It could assume the same values of `function`
- `type_weight`: is a string that specify initialization method of weights. Weights of each layer are initialized with the same method. Values of type_weight could be: {`“random”, “uniform”`}
- `early_stopping`: a Boolean variable to set early stopping policy
- `batch_size`: it could be an integer between 1 and size of training set. The value assigned to `batch_size` establish updating weights policy. `Batch_size` = 1 represents online version, `batch_size` = size of TR represents batch version and a value between 1 and the size of TR defines mini-batch version

2.3 Validation Schema

We divided ML-CUP20-TR in development set and test set. The development set contains 80% of ML-CUP20-TR while test set contains 20%. Development set was split in training set and validation set that contains 80% and 20% of development set. Hyper-parameters are inserted by hand and then we performed a grid search. Validation set is used to evaluate the generated model at each epoch and we kept epoch *t* in which we obtained the lowest error on validation set, this epoch *t* is used to stop retraining of the model. We also used early stopping to arrest the training phase when validation error start to increase according to Generalization Loss explained by Lutz Prechelt [3]. We selected the best models by looking at the metric reported in terms of Mean Euclidean Error on validation set and its standard deviation. Then we applied a random perturbation to the parameters of 8 best models, so we create 8 new models with these perturbated parameters and trained them from scratch. The final 8 best models are chosen from models with perturbated parameters and the 8 best models came out of previous grid search. The 8 best configurations are retraining on the entire development set and then ensembled creating an instance of the class `Ensable`. The ensamble performs its predictions by averaging the predictions of its constituent models. We used test set only to evaluate each of the 8 model and ensabled model, we had **never** used test set during model selection or other previous phases.

3. EXPERIMENTS

3.1 MONK'S RESULTS

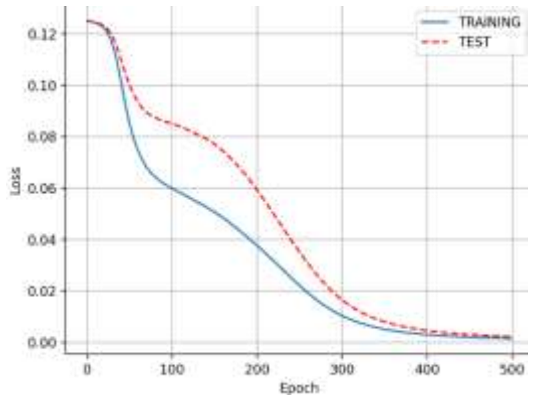
We have adopted the one-hot encoding technique to transform each input into 17 corresponding binary values. For all 3 monks we used a single hidden layer with 4 units, *ReLu* activation function for hidden units and *sigmoidal* for output units in all three monk problems, for monk 3 we also used *tanh* function with regularization with a weight decay lamda = 0.001 to obtain a best accuracy on test set. The loss was computed using the Mean Square Error (MSE) on a full batch gradient descent over 500 epochs, the functions were sensitive to the initialization of the weights and we chose its randomly between interval [-0.1,0.1]. Our best results are in Table 1. Notice that the errors and accuracies on training and test set presented in Table 1 are a mean of 5 trials: we tried 5 random start configuration for each parameter configuration and then we take the mean of MSE and accuracies.

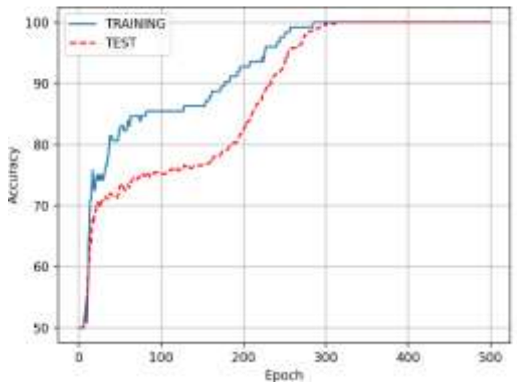
Task	#layer	eta	alfa	lamda	MSE (TR/TS)	Accuracy (TR/TS) (%)[i]
MONK 1	(17,4,1)	0.9	0.5	0	2.217329e-03/4.311596e-03	100%/100%
MONK 2	(17,4,1)	0.9	0.8	0	3,6787120e-04/3,881283e-04	100%/100%
MONK3	(17,4,1)	0.6	0.8	0	0.014721342/0.017172244	96.88%/ 95.78%
MONK3 (reg.)	(17,4,1)	0.9	0.8	0.001	0.02273535/0.0178323096	94.59%/97.17%

Table 1. Results of MONK's tasks.

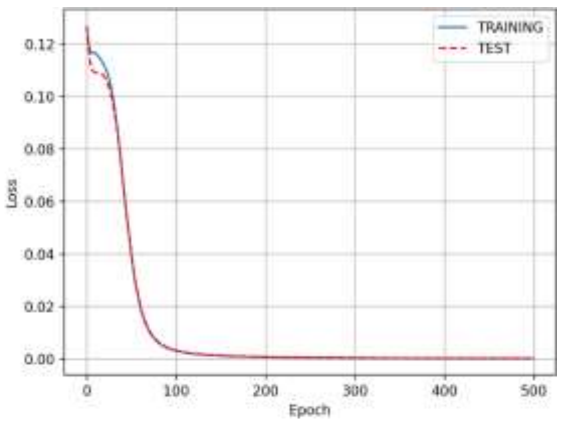
Figure 1 presents learning curve and accuracy of our best models that reach an accuracy of 100% for training and test for Monk 1 and 2 while the Monk3 have an accuracy of 96.88%/ 95.78% (Training accuracy/Test accuracy) while with regularization we obtained an accuracy of 94.59%/97.17%.

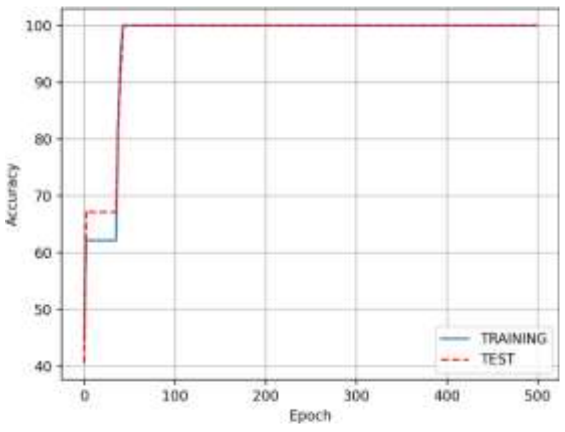
A. MONK 1



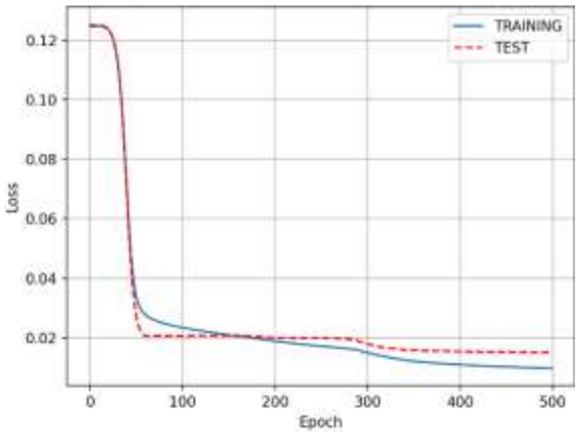


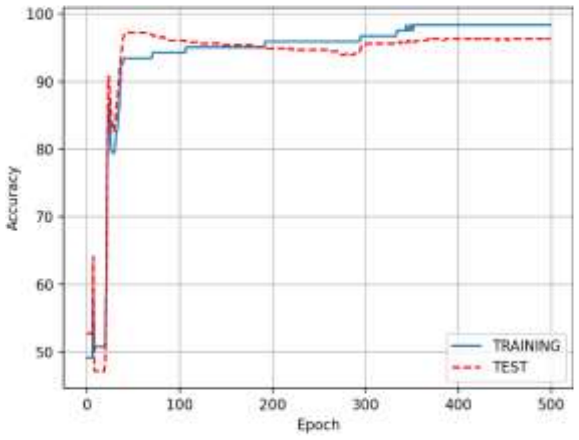
B. MONK 2



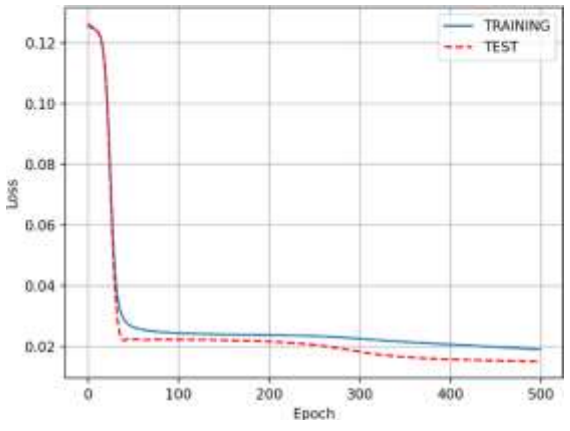


C. MONK 3





D. MONK 3 (Reg)



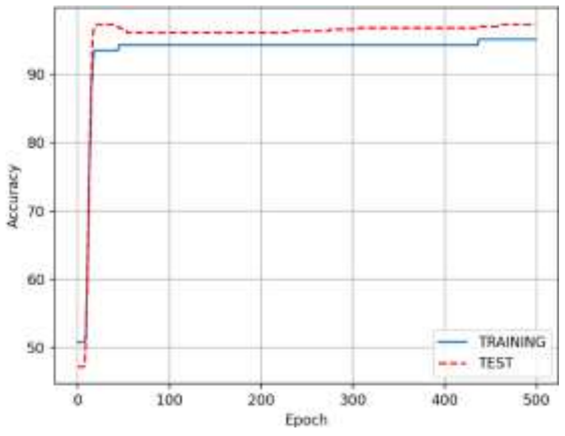


Figure 1: Monk's MSE (left) and accuracy (right) graphics over 500 epochs

3.2 CUP

We started with a preliminary analysis of hyper parameters to understand the behaviour of the network on data set, and then we selected best hyper parameters to perform the final grid search. An interval of all parameters used are reported in Table 2. In total, we tested about 1500 different configurations respecting the general rules [1]. The hardware components used are AMD Ryzen 5 3600 4,20 GHz Six-Core processor which completes the model selection phase approximately in 30 hours. In the preliminary analysis we noticed that more than 50 units led to the instability of the learning curve, while a high learning rate (more than 0.1) led the learning curve to diverge. We also implemented a *variable learning rate* [2] but we obtain better result with a fixed learning rate. Another important property that allowed us to avoid overfitting was the *Early Stopping* technique (of chapter 2.3) that arrest the training when Generalization Loss exceeds the threshold of 2%, in this case we returned the best model found so far otherwise the training go ahead until maximum epoch (that we set to 500). This technique allowed to accelerate and control the model selection process. We realized that many *layers* were not very functional, the models we found had a good error with one or two layers. Very high *hidden units* such as 100 or more units did not bring good results and sometimes the model was overfitted, indeed we chose not to exceed 50 units. We adopted only regularization L2 during training phase and a low value was sufficient to stable learning curves. Since we are in the case of regression we have chosen a linear activation function for output layer, while for hidden layer functions we tried *tanh, sigmoidal, relu, leaky_relu*; preferring *tanh, sigmoidal* for stability and results of the models. As we said, we perform a **5-fold cross validation** which returned the mean over the 5 folds of Mean Euclidean Error and Mean Squared Error for both training and validation and their respective standard deviations. Also, for each of 5 folds we tried 5 random starting configuration and we take the mean error of them. The best models were selected considering their MEE and its standard deviation. We used 8 best models came out of grid search to create 8 models with random perturbation of hyperparameters, the values for *learning_rate*, *lambda*, *momentum*, were randomly generated from a normal distribution centered at their base value, instead, the *batch_size* was generated by randomly varying its value within a range of 15 and each layer within a range of 3 units. The final 8 best models are chosen from models with perturbated parameters and the 8 best models came out of previous grid search. These models are retrained on all development set until epoch t in which we obtained the lower validation error. In Table 4 we represented 8 best models selected. After we performed an ensemble of the 8 models which results on development set and unseen internal test set are reported in Table 3. We used internal test only at the end of model selection phase and we used it only to provide an error evaluate on final model fit on development set. Figure 2 shows learning curves of 8 best models on training and validation set.

Hyper-parameters	**Values**
hidden_units (*units and layer*)	[50, 50], [20,20], [50],[30],[12]
batch_array (*batch size*)	[16, 32, 128]
epochs	[150, 500, 700]
learning_rate_init (*initial learning rate*)	[0.002, 0.003156, 0.005, 0.007]
alfa (*momentum*)	[0.5, 0.64, 0.8]
v_lamda (*regression*)	[0, 0.00001, 0.0000001]
fun (*activation function*)	["sigmoidal", "tanh", “relu”, “leaky_relu”]
weight (*type of weight*)	[“uniform”, “random”]
early_stopping	[True, False]
type_learning_rate (*learning rate fixed/variable)*	[“fixed”, “variable”]

Table 2. List of all hyperparameters tried to perform grid search

MODEL	DEVELOPMENT SET MEE	TEST SET MEE
MODEL 1	2.80261426227361	2.8216767844770
MODEL 2	2.75099004785974	2.8574945666912
MODEL 3	2.71236245449497	2.8455872972948
MODEL 4	2.66552699834967	2.9256225349512
MODEL 5	2.76055649323910	2.8336593949378
MODEL 6	2.65065685010568	2.9319330420984
MODEL 7	2.84152487158949	2.9371625829850
MODEL 8	2.78443725256822	2.8800808884867
ENSEMBLE	**2.66521807136338**	**2.78317293394569**

Table 3. Mean Euclidean Error on the development set and internal test set of the 8 best models and their ensemble.

4. CONCLUSIONS

Through the ensemble technique we choose the final model by the average of the outputs of the best models with the lowest *Mean Euclidean Error* on the internal test set (never used in the model selection phase). The final model allowed us to obtain the blind test data that we saved in the CUPpiccino_ML-CUP20-TS.csv file. According to our estimates the value of the MEE on the blind test should be about **2.783172933945697**.

ACKNOWLEDGEMENTS

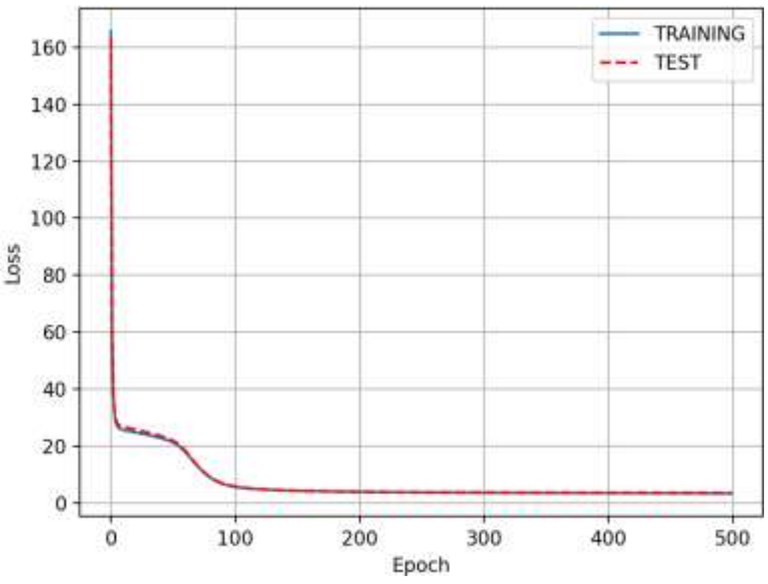
We are very satisfied with this experience that has allowed us to put into practice all the aspects concerning the construction of a neural network with the various model selection and model assessment techniques that we studied during the course. Teamwork also allowed us to manage all parts of the implementation and the various tests in an efficient and coordinated way. Hope that this experience will help us to open the way in the world of machine learning. We agree to the disclosure and publication of our name, and of the results with preliminary and final ranking.

Model	Hidden Layer	Learning rate	Activation function	Batch	Momentum	Lambda	weight	MEE TR	MEE VL
Model 1	[23, 23]	0.0032385058601875545	sigmoidal	131	0.6698997855293465	9.99866508508231e-07	random	2.84391202	3.0210297
Model 2	[33]	0.0022115423468592674	sigmoidal	19	0.39375050330941497	1.0000845150074158e-06	random	2.82919330	3.0395802
Model 3	[22, 21]	0.002957904828850339	sigmoidal	31	0.47870823379607513	9.978696007042747e-07	random	2.91362582	3.0634515
Model 4	[30]	0.003156	tanh	128	0.8	1e-06	random	2.87407300	3.0779709
Model 5	[20, 20]	0.003156	tanh	16	0.64	1e-06	random	2.90943699	3.0681806
Model 6	[30]	0.003156	sigmoidal	16	0.64	1e-06	random	2.86199126	3.0729336
Model 7	[20, 20]	0.003156	sigmoidal	16	0.64	1e-06	random	2.94866282	3.0873294
Model 8	[31]	0.006207524101468444	sigmoidal	125	0.6793953609585252	1.0000301827654278e-06	random	2.80920963	3.0299804

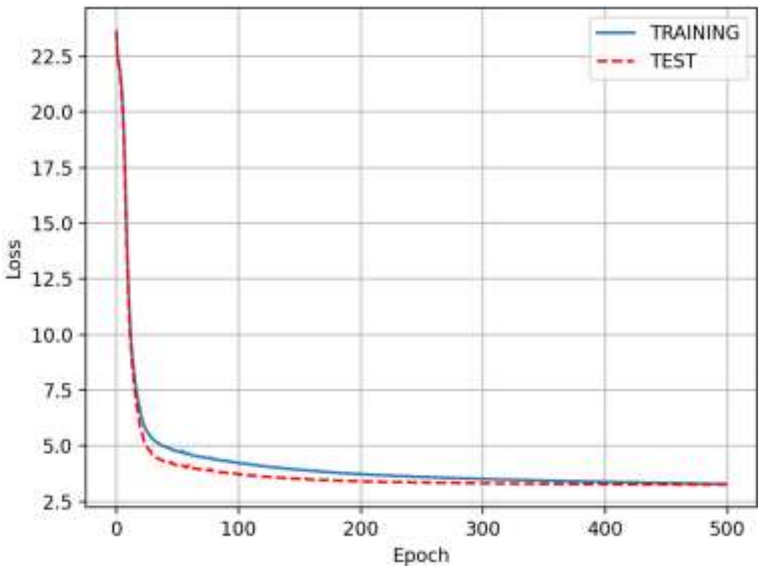
Table 4. *Hyperparameters* and performance of the eight best models in term of MEE

GRAPH ML CUP

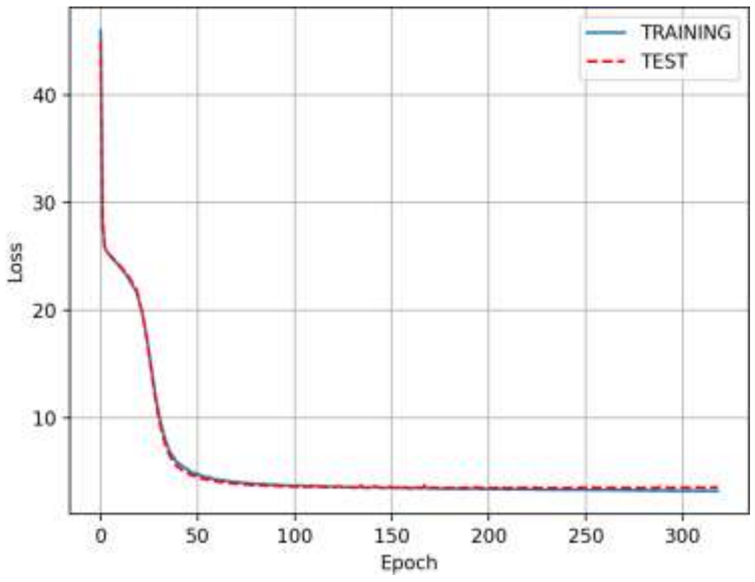
MODEL 1



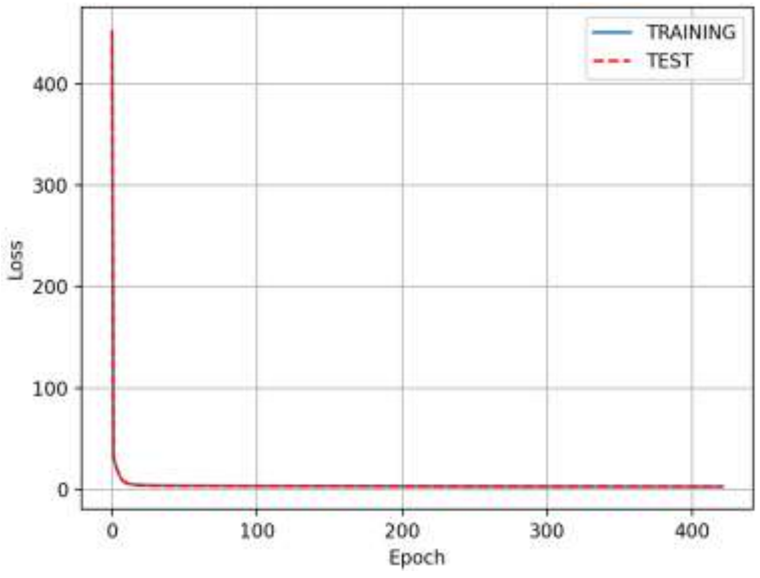
MODEL 2



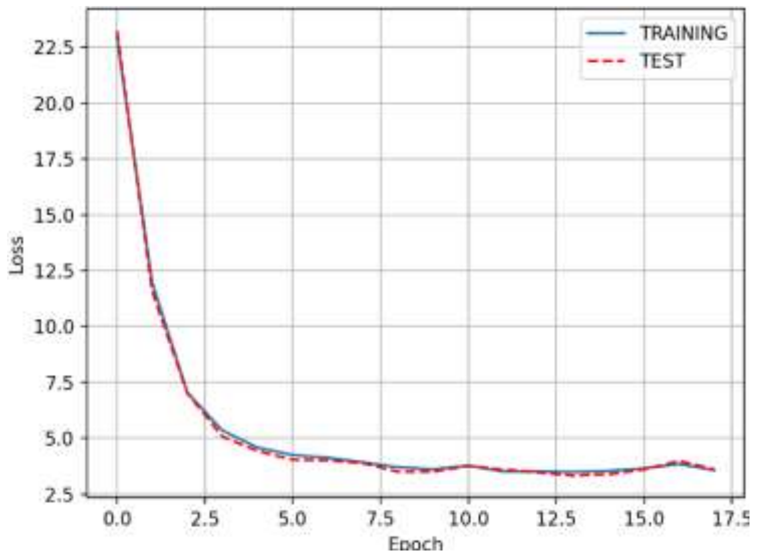
MODEL 3



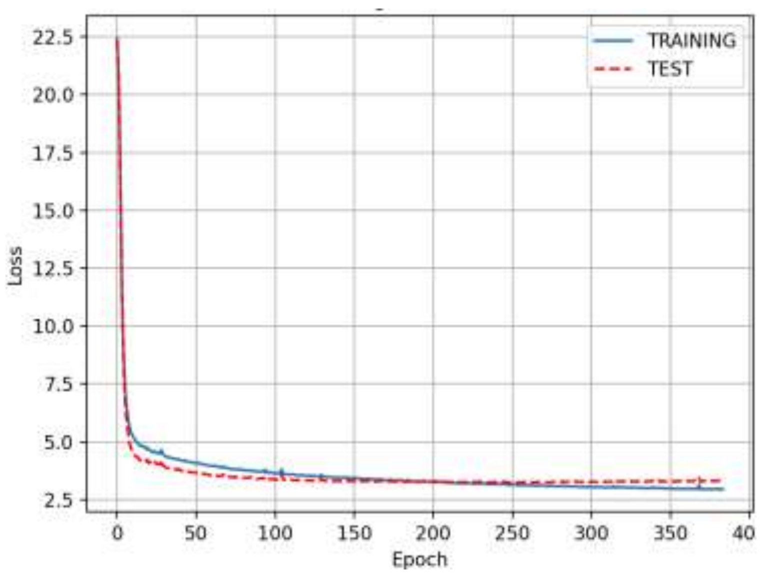
MODEL 4



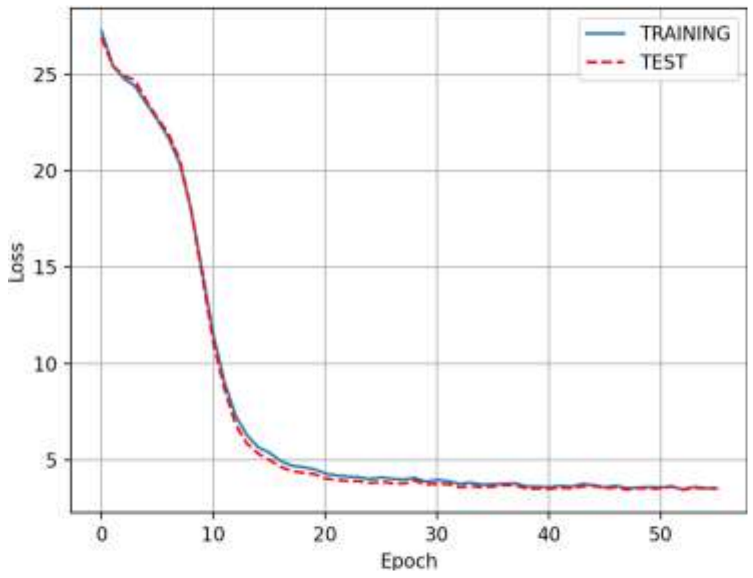
MODEL 5



MODEL 6



MODEL 7



MODEL 8

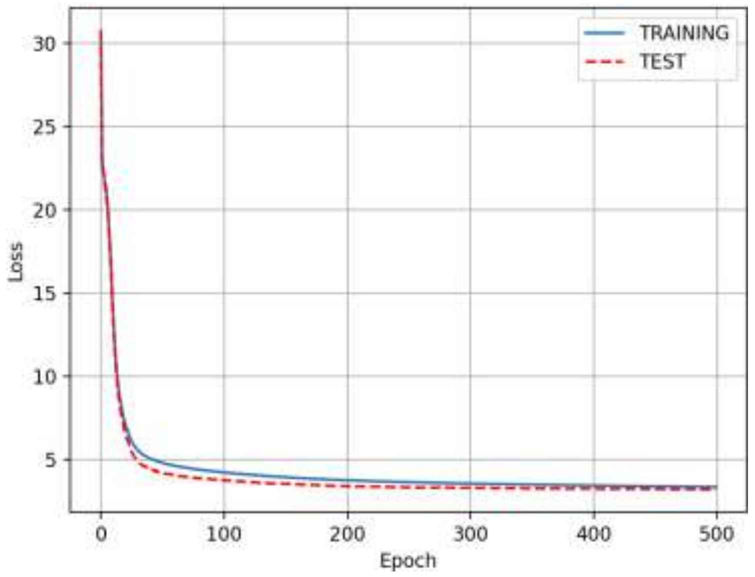


Figure 2 Learning curves of 8 best model for ML CUP

REFERENCES

[1] The MONK's Problems - A Performance Comparison of Different Learning algorithms" by S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang has been published as Technical Report CS-CMU-91-197, Carnegie Mellon University in Dec. 1991

[2] Section 8.3.1 DL book

[3] Early Stopping – But When? Lutz Prechelt Fakult¨at f¨ur Informatik; Universit¨at Karlsruhe D-76128 Karlsruhe; Germany prechelt@ira.uka.de, 28 March 2002

[4] Practical Recommendations for Gradient-Based Training of Deep Architectures Yoshua Bengio Version 2, Sept. 16th, 2012