ML project report

Simone Manti, mat. 566908, MSc in Mathematics (Curriculum Modellistico)
Luca Moroni, mat. 635966, MSc in Computer Science (Curriculum in Artificial Intelligence)
Maria Cassese, mat. 607577, MSc in Digital Humanities (Curriculum in Language Technologies)
Emails: s.manti1@studenti.unipi.it, l.moroni5@studenti.unipi.it, m.cassese4@studenti.unipi.it
ML course (654AA), Academic Year: 2021/2022
Date: December 31, 2021
Type of project: A

Abstract

The aim of this report is to present an implementation from scratch of an Artificial Neural Network simulator with `Python` without using additional ML libraries. In particular, in the following we describe in details the procedure (e.g. training and validation schemas, hyperparameters tested) used to choose the final model. We also build an implementation of Extreme Learning Machine from scratch and we compare the results of this two different classes of models.

Introduction

This project deals with an implementation from scratch of an Artificial Neural Network in order to solve the ML-CUP21 regression task. Firstly, we tested the correctness of our implementation on the "Monk problems" [Thr+91]. Then to find the best configurations we used a coarse-to-fine grid search with 5-folds cross validation. Finally we tested this models on an internal test set and at the end we used the ensemble tecnique to provide the final model to predict the output of the blind test set.

Furthermore we developed an implementation from scratch of an ELM (Extreme Learning machine) [HZS06] in order to solve in another way the task provided.

More details on these two implementations and the results of them are in the next sections.

Method

The software was developed in `Python` programming language, we also used `numpy` library and some minor utilities of `pandas`, `sklearn` (for preprocessing) and `math`.

We implemented a fully connected feedforward Neural Network, with classical back-propagation learning algorithm to update weights. We also implemented some variations, such as: different activation functions, L^2 and L^1 regularization, different weights update

policies (i.e. different mini batch size tested) and two different learning rate schedules (constant and linear decay). Moreover, we tested different architectures, more details in the further sections.

Furthermore, we implemented an Extreme learning machine with a direct method learning (i.e. by solving directly the linear system).

Even in this case, we implemented some variations, such as: dropout tecnique [Lai+20], L^2 regularization.

`StandardScaler` (for ML-CUP21) and `OneHotEncoder` (for MONK problems) preprocessing tecniques were performed.

Model Implementation

We implemented a class `Network` where we specified input dimension by `input_dim` and a vector of layers by `layers`. Each layer belongs to a class `Layer` in which we specified output dimension of the layer by `output_dim`, activation function of the layer by `activation` and probability distribution used to initialize the weights by `initializer`.

In `Layer` we implemented `initialize_weights` to initialize the weights, `forward_step` to implement a layer forward step and `backpropagate_delta` to implement a step of backpropagation algorithm.

In `Network` we defined `compile` to initialize the network, `forward_step` to perform a forward step of the network, `predict` to perform a prediction, `bp_delta_weights` and `apply_delta` for the implementation of backpropagation algorithm. We also defined `training` to perform training phase and `regularize` to add regularization.
We also implemented a class `RandomizedNetwork` which differs to the class `Network` by the fact that for the ELM we perform a `direct_training`.

The (only) hidden layer of an ELM belongs to a class `RandomizedLayer` in which we defined `drop_connect` that we used to implement **dropout** together with `bias_dropout` and part of `direct_training`.

In particular,ELM aims to reduce the computational time of the training phase due to the fact that does not use a gradient descent approach but a direct one. The approach used by this kind of this class of models consist in a random projection over a space of a very large dimension, this is done due to the cover theorem [Cov65]: the more is high the dimension of the projection the more is high the probability that the problem becomes linear separable in this space.

The model compute a linear regression over this higher dimensional space and this kind of problem can be solved directly by the normal equation approach.

Hyperparameters overview

- `units_per_layer`: vector of integers indicating the number of unit per layer (excluding input layer).

- `weights_init`: distribution used to initialize weights.
- `loss`: metric used to evaluate empirical risk of the model.
- `learning_rate`: learning rate. In the case of decay $\neq$ None it indicates the learning rate at the first epoch.
- `alpha`: momentum coefficient.
- `lambda_reg`: regularization coefficient.
- `regularizer`: type of regularization used.
- `decay`: vector of two components, where the first one is the number of epochs where learning rate decays linearly, while the second one is the final learning rate achieved after the linear decay.
- `batch_size`: mini-batch size.
- `reg_type`: type of regularization used.
- `early_stopping`: number of epochs needed to early stop the training.
- `units`: number of units in the only hidden layer for ELM.
- `p_dc`: probability of dropout for hidden neurons.
- `p_d`: probability of dropout for hidden weights.

Validation schema

The dataset was splitted into **development set** (90% of ML-CUP21 dataset) and **internal test set** (10%). The first one was used for model selection phase, while the second one for the model assessment phase.

In the model selection phase, we performed a coarse-to-fine grid search in which we used a **5-fold cross validation**.

We performed the model selection phase using MSE (mean square error) and the model assessment phase using MEE (mean euclidean error) because MSE is generally more pessimistic than MEE: indeed, by Jensen's inequality [FMS96] $E_{MEE} \leq E_{MSE}$ (where E_{MEE} stands for the error using MEE, similarly for E_{MSE}).

Then, we selected the best models according to their MSE validation loss, we retrained them on the whole development set and at the end we took an **ensemble** of them to provide a final model.

Experiments

Monk Results

The input were preprocessed by **one-hot-encoding**: this led us to binary inputs of dimension 17. The model's topology used for Monk 1 and Monk 2 tasks consists in one hidden layer of 5 units, whereas for Monk 3 we used 1 hidden layer with 16 units. For all the three task we used *ReLu* activation function for the hidden layer and *sigmoid* activation function for the output.

We initialized the weights by taking them indipendently from a uniform distribution in $(-0.1, 0.1)$. The loss was computed using the *Mean Square Error* (MSE) on a full batch gradient descent over 500 epochs.

Table 1 summerizes all the results obtained.

Task	**Units**	**eta**	**alpha**	**lambda**	**MSE(TR/TS)**	**Accuracy%(TR/TS)**
MONK 1	(5,1)	0.75	0.75	0	$2.6 \times 10^{-4}/8.9 \times 10^{-4}$	100%/100%
MONK 2	(5,1)	0.9	0.9	0	$1.4 \times 10^{-4}/1.4 \times 10^{-4}$	100%/100%
MONK 3(no reg.)	(16,1)	0.8	0.6	0	$1.7 \times 10^{-2}/4.7 \times 10^{-2}$	98.6%/94.07%
MONK 3(L^2 reg.)	(16,1)	0.8	0.6	0.001	$3 \times 10^{-2}/3.4 \times 10^{-2}$	97.1%/96.4%

Table 1: Average loss and accuracy obtained over ten trials

Figure 1, **Figure 2**, **Figure 3** and **Figure 4** (in the Appendix at the end of the report) show the loss and accuracy curves of one trial for all the three Monk datasets. We noticed that in case of Monk 3 task we strongly need regularization, otherwise we fall in overfitting. Indeed, the results with L^2 regularization are better than the ones without (see **Figure 3** and **Figure 4**).

Cup Results

We divide this section in two parts.

1. The first one is related to the classical feedforward Neural Network CUP results.
2. The second one is related to the Extreme Learning Machine results.

Feedforward Neural Network

We choose the best models according to a **coarse-to-fine** grid search. To be more precise, due to time constraints we decided to start our model selection phase taking 3 values per hyperparameter and then we refined to the ones chosen.

In total we tested around 2200 configurations which required approximately 60 hours.The hardware we used consisted of a Intel Core i5-5200U 2.20GHz, Intel Core i7-10510U, 2.30 GHz, quelle di Luca.

We also decided to parallelize grid search in order to safe time computing.

Before implementing the grid search, we performed a screening phase to get a first glance of the search space. We narrowed the search space by choosing the best range of hyperparameters for the selected models, removing values that led to a loss explosion, or made the loss descent too slow. We tested three architectures of different depths: two consisting of three hidden layers and one consisting of two hidden layers, with the total number of neurons per configuration ranging from a minimum of 30 in the smallest architecture to a maximum of 80. We have noticed that by using deeper network configurations we get comparable results to those obtained with smaller configurations, so for our task it is not necessary to use a particularly deep network.

After the screening phase, we decided to fix some hyperparameters such as the distribution of initialization weights, the regularizer (L^2) and the number of epochs (500). In particular, we chose L^2 regularizer over L^1 regularizer because L^1 norm tends to induce sparsity, which we wanted to avoid since our architectures are shallows. We set the Gaussian distribution as the initialization parameter for the weights because we noticed that at the performance level there was no difference with respect to the uniform distribution and in Gaussian distribution we have the advantage of not having to define the initialization bound for the weights (in other words, we have one less hyper-parameter).

Once the linear activation function was set on the output layer, in the first step we manually tried different combinations of the activation functions (sigmoid, tanh, relu). Based on the results obtained, we chose to use a combination of alternating sigmoid and relu functions in all the configurations.

For each configuration we performed a 5-fold cross validation on the development set. Finally, we choose a final model taking the ensemble of the 7 best models: 2 "classical feedforward Neural Network" for every architecture (1 with constant learning rate and 1 with linear decay) and 1 ELM.

Hyper-parameters	**Configuration tested**
`units_per_layer`	{(50, 20, 10, 2), (30, 10, 10, 2), (20, 10, 2)}
`weigths_init`	{"`Gaussian`"}
`learning_rate`	[min = 0.01, max = 0.3]
`alpha`	[min = 0.1, max = 0.3]
`lambda_reg`	[min = 0.00001, max = 0.0003]
`epochs`	{500}
`batch_size`	{32, 64, 128}
`early_stopping`	{20}
`learning_rate_decay`	{(100, 0.01), (150, 0.01) (200, 0.01), (250, 0.01)}

Table 2: Hyper-parameters used for the grid search for classical NN. For some hyper-parameters we just put the range of variation

Figure 5, **Figure 6**, **Figure 7**, **Figure 8**, **Figure 9**, **Figure 10** and **Table 5**

summerizes all the results of the best models selected. We remark that all the models selected seems pretty stable in term of MSE and MEE error.

Extreme Learning Machine

In a first phase we noticed that the model was always in overfitting, so we decided to implement some types of regularization tecniques. The first is the Tikhonov approach to the least square problem adding a λ parameter to reduce the norms of the weights learned at the last level, after that we have noticed that the overfitting problem still persisted. Then we implemented another regularization approach based on dropout tecniques [Lai+20]

Another point to specify: we reasoned about the possibility to learn a bias in the last layer (because this is not done in the original paper). So, in the model selection phase we have taken into account two kind of ELM: one that learn the bias in the last layer and one that does not.

Also in this case, we performed a **coarse-to-fine** grid search. Unlike the Feedforward case, we decided to refine just one time because we noticed that the results did not change significantly.

Table 3 summarizes all the hyper-parameters used for model selection and all the configuration tested

Hyper-parameters	**Configuration tested**
`units`	$\{200, 400, 600, 800, 1000\}$
`lambda_reg`	$\{0, 0.01, 0.1, 1, 10, 100\}$
`p_d`	$\{0, 0.2, 0.4, 0.6, 0.8, 1\}$
`p_dc`	$\{0, 0.2, 0.4, 0.6, 0.8, 1\}]$

Table 3: Hyper-parameters used for the grid search for classical NN. For some hyper-parameters we just put the range of variation

Table 6 shows the results of the best models selected. We remark that the results does not change significantly with or without bias.

Conclusion

Table 4 summarizes the performance of the 6 best models and their ensemble. The prediction on the ensemble is computed as the mean of the predictions of its costitutent models on the blind test set. We are confident that, according to our estimate, our model performance on the blind test set will be around **1.2344559645778954**.

model	Development set MEE	Internal test set MEE
1	1.0731221799853017	1.2204596138367696
2	1.183255456414312	1.3180401937911286
3	1.1188596837160816	1.2597745298999619
4	1.059619989881865	1.18541039503219
5	1.0394754697397353	1.1922660683982798
6	0.986556837584323	1.2419209455380742
7 (ELM)	0.98603504949318	1.2233200055488644
ensemble	**1.0638463809735426**	**1.2344559645778954**

Table 4: MEE on the development set and internal test set for the 6 best models and their ensemble

Acknowledgments

We agree to the disclosure and publication of our names, and of the results with preliminary and final ranking.

References

[Cov65] Thomas M Cover. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition". In: *IEEE transactions on electronic computers* 3 (1965), pp. 326–334.

[Thr+91] Sebastian B Thrun et al. *The monk's problems: A performance comparison of different learning algorithms*. Tech. rep. 1991.

[FMS96] Nicola Fusco, Paolo Marcellini, and Carlo Sbordone. *Analisi matematica due*. Liguori, 1996.

[HZS06] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications". In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.

[Lai+20] Jie Lai et al. "BD-ELM: a regularized extreme learning machine using biased dropconnect and biased dropout". In: *Mathematical Problems in Engineering* 2020 (2020).

Appendix

model	units	act functions	learning rate	alpha	lambda	decay	batch size	Avg TR MSE	Avg VL MSE
1	(50,20,10,2)	(”relu”, ”sigmoid”, ”relu”, ”linear”)	0.15	0.125	0.00005	/	32	1.53	2.04
2	(50,20,10,2)	(”relu”, ”sigmoid”, ”relu”, ”linear”)	0.15	0.225	0.0001	(250, 0.005)	64	1.76	2.15
3	(30, 10, 10, 2)	(”relu”, ”sigmoid”, ”relu”, ”linear”)	0.075	0.23	0.00003	/	32	1.67	2.04
4	(30, 10, 10, 2)	(”relu”, ”sigmoid”, ”relu”, ”linear”)	0.4	0.1	0.000075	(250, 0.01)	32	1.77	2.10
5	(20,10,2)	(”relu”, ”sigmoid”, ”linear”)	0.2	0.175	0.000025	/	32	1.63	1.99
6	(20,10,2)	(”relu”, ”sigmoid”, ”linear”)	0.4	0.225	0.00005	(250, 0.01)	64	1.79	2.17

Table 5: Hyper-parameters and performance of the six best models for the Feedforward Neural Network in term of training and validation MSE

model	trainable bias	units	lambda	act functions	P_d	P_dc	Avg TR MSE	Avg VL MSE
1	None	1000	(”sigmoid”, ”linear”)	10	0	0	1.54	2.05
2	Yes	1100	(”sigmoid”, ”linear”)	9	0	0.5	1.52	2.05

Table 6: Hyper-parameters and performance of two best RBF models on Cup’s dataset in term of training and validation MSE

Dataset	model	trainable bias	units	lambda	act functions	P_d	P_dc	Avg TR MSE	Avg VL MSE
Monk1	1	None	1000	(”sigmoid”, ”linear”)	0	0	0.6	1.00E-23	0.08
Monk1	2	Yes	800	(”sigmoid”, ”linear”)	0.015	0	0.5	0.000001	0.074
Monk2	3	None	1000	(”sigmoid”, ”linear”)	0.01	0.4	0.2	0.12	0.21
Monk2	4	Yes	1000	(”sigmoid”, ”linear”)	10	0	0.8	0.13	0.21
Monk3	5	None	800	(”sigmoid”, ”linear”)	1	0	0.2	0.006	0.06
Monk3	6	Yes	500	(”sigmoid”, ”linear”)	0.005	0.3	0.05	0.03	0.08

Table 7: Hyper-parameters and performance of the RBF models on MONK’S datasets in term of training and validation MSE

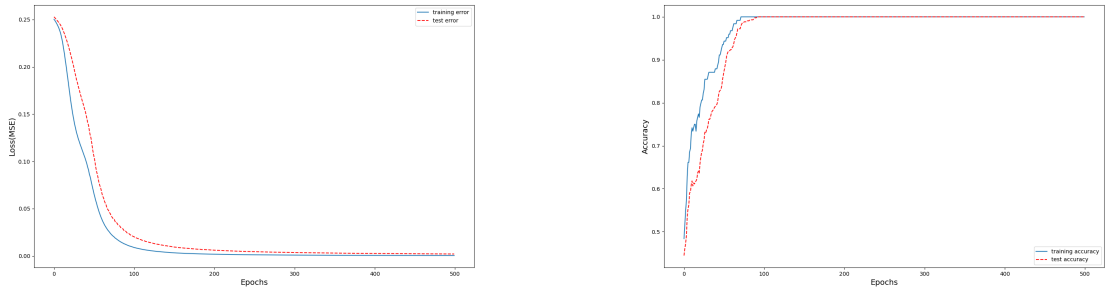

Figure 1: Monk 1

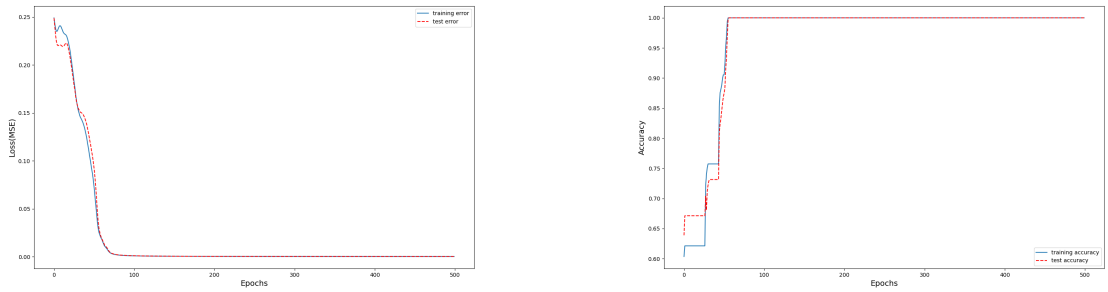

Figure 2: Monk 2

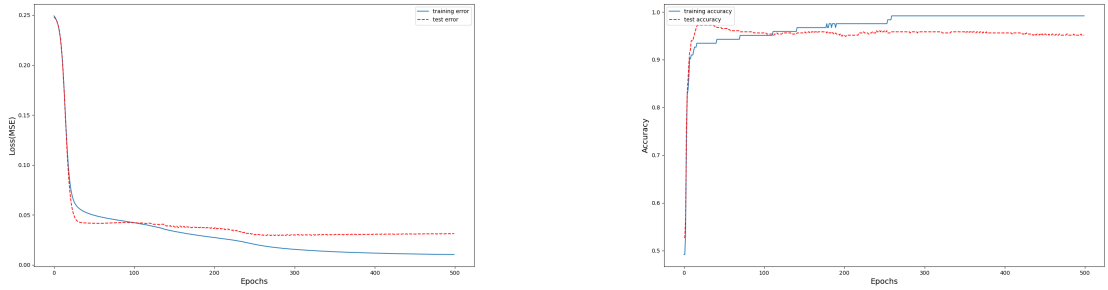

Figure 3: Monk 3 (no reg)

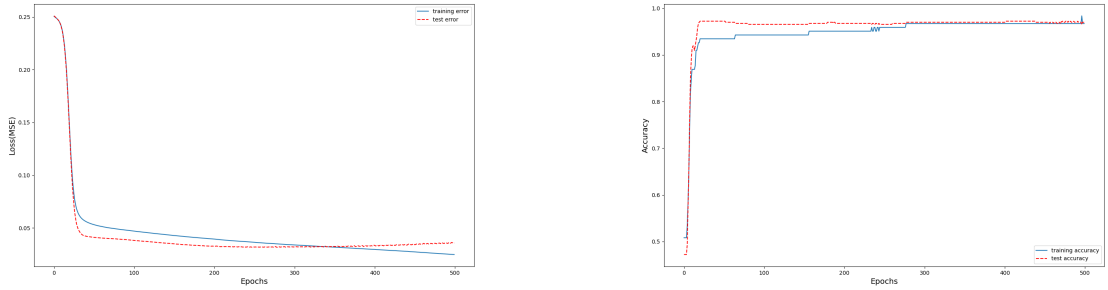

Figure 4: Monk 3 (L^2 reg)

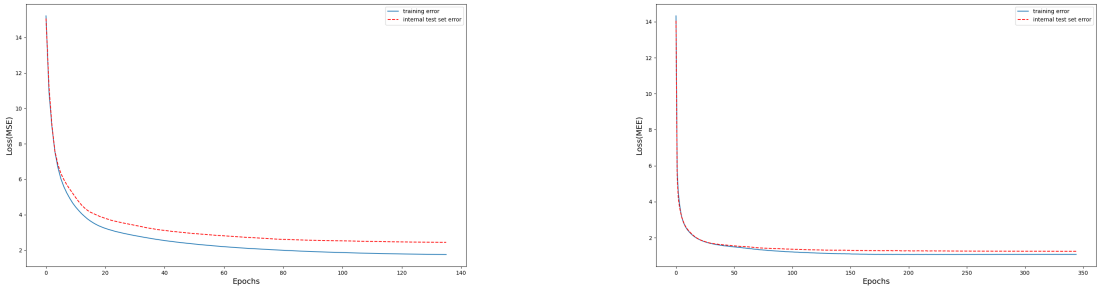

Figure 5: Model 1

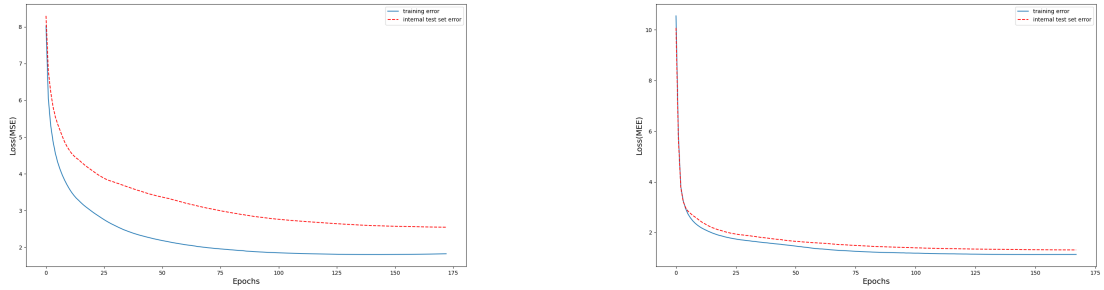

Figure 6: Model 2

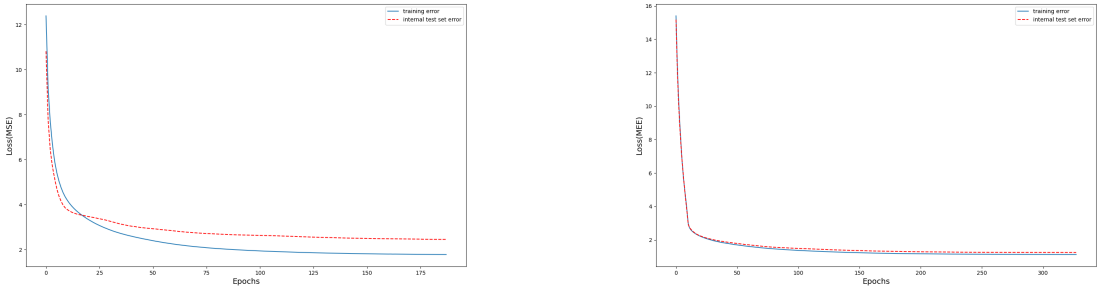

Figure 7: Model 3

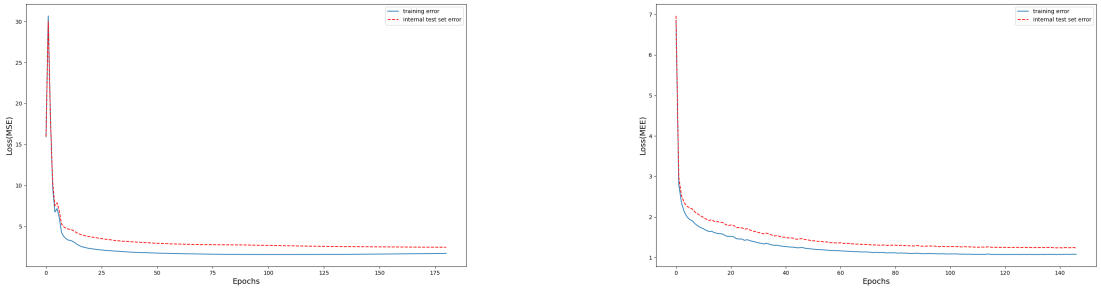

Figure 8: Model 4

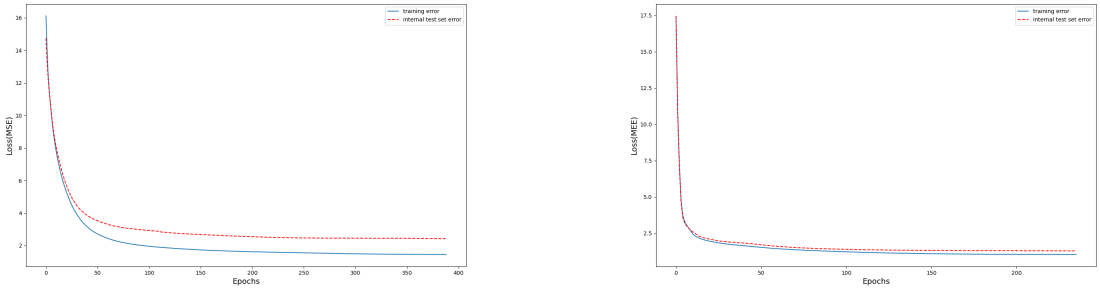

Figure 9: Model 5

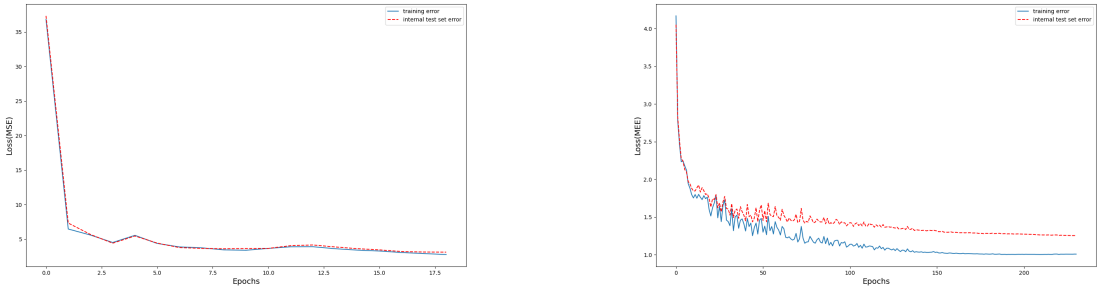

Figure 10: Model 6