Machine Learning Project report

Alex Pasquali – 626378 – a.pasquali5@studenti.unipi.it – MSc in Artificial Intelligence
Gaetano Antonicchio – 616685 – g.antonicchio@studenti.unipi.it – MSc in Data Science & BI

ML course (654AA), Academic Year: 2020/2021
Date: *25/01/2021*
Type of project: **A**

ABSTRACT

This report presents the implementation of an Artificial Neural Network simulator developed from scratch using `Python` without off-the-shelf libraries. We created an ensemble of 7 neural networks and selected its constituent models through a coarse-to-fine grid search using 5-fold cross validation. In the following sections we describe the key features of the final model and its validation and training schemas.

1. INTRODUCTION

This project is focused around developing an Artificial Neural Network capable to solve both classification and regression problems. The aim of the project was to pursue a rigorous method to select the best models and validate their results. To find the best configuration, we adopted a coarse-to-fine grid search with 5-fold cross validation. Then the models were tested on an internal test set, providing an unbiased estimate of the performance of the models. The correctness of the implementation was tested on the well-known "Monk's problem" [1]. Subsequently, the models were deployed as an ensemble and used for predicting the outputs of the ML-CUP20-TS dataset. The results of the model selection phase and the experiments are described in the following sections.

2. METHOD

The software was developed using **`Python`** programming language, the **`numpy`** library, and some minor utilities of **`sklearn`**, **`pandas`** and **`math`**. We implemented a fully connected feed forward Neural Network, that learns its synaptic weights through back-propagation using the Stochastic Gradient Descent (SGD) optimization algorithm.

We also implemented some variations, such as: different activation functions, L1 and L2 regularization, momentum, different weight update policies (online, mini-batch and batch) and learning rate schedules (constant, linear decay and exponential decay). The above-mentioned can be fine-tuned through different hyper-parameters configurations (listed in section 2.2). We describe how the best models were selected and their related performances on the ML-CUP20, both singularly and within an ensemble framework. No pre-processing was performed on the ML-CUP20 datasets.

2.1 Model Implementation

We implemented a class **`Network`** where the input dimension is specified by the parameter `input_dim`, while the number of hidden layers and the number of units per layer are specified by `units_per_layer` (tuple of integers). Through `acts` it is possible to select the activation function of each layer simply by the function's name. Two different weights initializations were implemented: *random uniform* and *fixed*, even though the latter was used only for debugging/testing purposes. Each instance of `Network` has also a list of **`Layer`** objects to store the weights and biases and to implement methods such as `forward_pass` and `backward_pass`, used during training, evaluation, and inference. The main methods of `Network` are `forward` (that given an input pattern, computes the output), `propagate_back` (that propagates back the error to update each layer's gradient), `compile` and `fit` to perform training, `evaluate`, and `predict` (to compute the outputs on a blind test). This implementation of a Neural Network can solve both classification and regression problems simply by changing the activation functions using a single parameter.
The `functions` module contains the implementation of all the activations and losses, as well as their derivatives. Moreover, it implements metrics such as the correct classification rate (with a non-decision area) and the Euclidean error, learning rate decays and regularizations.
The **`Optimizer`** class implements the SGD algorithm with backpropagation [2].

2.2 Hyper-parameters Overview

- **`input_dim:`** Indicates the dimension of the input layer.
- **`units_per_layer:`** Tuple of integers indicating the number of units for each layer (excluding the input layer).
- **`acts:`** Names of the activation functions. They can be either: *identity*, *sigmoid*, t*anh*, *relu* or *leaky_relu*.
- **`init_type:`** Indicates the type of weight initialization (*uniform* or *fixed*).
- **`init_value:`** Weights value in case of fixed initialization.
- **`limits:`** Bounds on the weights' values in case of random uniform initialization.
- **`lr:`** Learning rate (eta)

- **`lr_decay:`** Can be either *linear* or *exponential*. If it's `None` the learning rate will stay constant.
- **`staircase:`** In case of exponential decay, the learning rate decreases in a stair-like fashion if `staircase` is set to `True`.
- **`momentum:`** Momentum coefficient (alpha).
- **`lambd:`** Regularization coefficient.
- **`reg_type:`** Indicates the type of regularization, either *L1* (lasso) or *L2 (*ridge).
- **`batch_size:`** It can be either *full* or a number. The former indicates a "full batch" while the latter is used for mini-batch or online weights updates.
- **`metr:`** The name of the metric used to evaluate the performances of a model. We developed `bin_class_acc` (correct (binary) classification rate) and `euclidean` (Mean Euclidean Error).

2.3 Validation Schema

The dataset was split into **development set** (90% of ML-CUP20-TR**)** and **internal test set** (10%). The former was used to perform model selection, while the internal test set was used to estimate the generalization capabilities of the chosen models. The adopted procedure consisted of a **5-fold cross validation** on all model configurations generated by a grid-search. To reduce the search space, we first performed a coarse grid search, trying different hyper-parameter configurations inserted by hand, so to have a grasp on the topology and parameters ranges that were more suitable for the task at hand. Subsequently, we selected the best models by looking at the metric reported in terms of *Mean Euclidean Error* and its standard deviation. Those models were then fine-tuned through a finer grid-search, applying a number of random perturbations to the parameters of the best models. The "base models" for these fine gird-searches were selected so that they are not too similar, indeed, they have a different number of hidden layers, activation functions, batch sizes, regularizations and learning rate decays. Subsequently, the best models coming out of these randomized fine grid searches were taken as constituent of and **ensemble**. The ensemble performs its predictions by averaging the predictions of its constituent models. Each of the constituent models was trained on the whole development set and then tested on the internal test set. This strategy is well-known for providing a good unbiased estimation of the error on unseen data, since the internal test set was never used in phase of model selection.
Lastly, the ensemble model was retrained on the whole ML-CUP20-TR dataset before being used for predicting the outputs of the blind test.

3. EXPERIMENTS

3.1 Monks

The inputs were transformed with a **one-hot-encoding** on all three Monks datasets. This led us to 17 binary inputs per pattern. The model's topology used for the Monk 1 and Monk 2 tasks consisted of 1 hidden layer with 4 neurons, whereas for the Monk 3, to better study the effects of overfitting and regularization, we used 1 hidden layer with 15 units. For all three tasks we used the *ReLU* activation function for the hidden layer, and the *Sigmoidal* for the output layer. The loss was computed using the *Mean Square Error* (MSE) on a full batch gradient descent over 500 epochs. We noticed that the datasets were sensitive to weight initialization, hence we opted for initializing the synaptic weights with values in range (-0.25, 0.25) for Monk 2 and (-0.1, 0.1) for Monk 1 and Monk 3. The scores reported in **Table 1** were obtained by averaging the results over 10 trials.

Task	Topology	eta	alpha	lambda	Loss (TR/TS)	Accuracy% (TR/TS)
MONK 1	(17, 4, 1)	0.76	0.83	0	5.5328e-05 / 0.000195	100% / 100%
MONK 2	(17, 4, 1)	0.8	0.8	0	5.092e-05 / 5.4184e-05	100% / 100%
MONK 3	(17, 15, 1)	0.8	0.8	0	3.339e-5 / 0.023113	100% / 94.9%
MONK 3 (L2 reg.)	(17, 15, 1)	0.8	0.8	0.0023	0.010676 / 0.0204675	99.09% / 95.13%
MONK 3 (L1 reg.)	(17, 15, 1)	0.8	0.8	0.0012	0.002904 / 0.02113	99.92% / 94.42%

Table 1. Average loss and accuracy obtained over ten trials on the Monks datasets.

Figure 1 and **Figure 2** show the loss and accuracy curves of one trial for all three Monks datasets. Inspecting the curves, we noticed that on average Monk 2 reaches convergence after 120 epochs, whereas Monk 1 requires 200 epochs. Monk 3 presents a clear overfitting: **Figure 2.A** shows that the training and test losses, without regularization, tend to diverge due to noise present in the dataset. We attempted to reduce the overfitting by using L1 and L2 regularizations. With *Tikhonov regularization* (L2), the test accuracy improves, at the expense of the performance on the training set, which decreases by 0.91%. However, this was a good trade-off since the model reported a better out-of-sample error. L1 regularization on the other hand, (on average) did not improve the performances of our model. We included only one learning curve per trial for each dataset because the curves of different trials were remarkably similar.

3.2 CUP

To choose the best models we used the development set to perform first **coarse grid search** so to reduce the search space and find suitable hyperparameters values for our models. Then we proceeded with fine tuning through a random search. In total we tested around 2000 configurations which required approximately 30 hours to be completed. The hardware we used consisted of a Dual-Core Intel Core i5 2,7 GHz and a 4-Core Intel i7 2,8 GHz. For the coarse grid search, the values of each hyperparameter were inserted by hand and all the possible combinations were generated. However, their number was limited by the computing power at our disposal. We decided in fact to fix the hyperparameters that, in a preliminary analysis, proved to be less influent, such as `decay_rate`, `decay_steps` and `limit_step` (all related to the learning rate decay). Moreover, the activation functions we used, except for the output one that was linear, were only *TanH* and *Leaky ReLU* because it was not useful to have activation functions with only positive values (such as *ReLU* or *Sigmoidal*) since the target range was not limited to this case. The values tried in the coarse grid search are shown in **Table 2**. Due to time constraints, we performed a screening phase and decided to try the model with 1 hidden layer only with no regularization, while the model with 5 hidden layers of 8 units each generally did not provide good results, so it was tested with less combinations of hyperparameters. Furthermore, combinations that led to overflow were automatically discarded.

For each of the configurations we performed a **5-fold cross validation** on the **development set**, which returned the average over the 5 folds of *Mean Euclidean Error* and *Mean Squared Error* for both training and validation, as well as their respective standard deviations. The best models were selected considering their MEE and its standard deviation. We noted that different types of models provided us satisfactory results, so we exploited their diversity and opted for an **ensemble** approach [3]: we picked three of the best models such that they had distinct characteristics (number of layers, activation functions, eta, alpha, and others), and we ran three randomized fine grid searches, each of them on a number of random configurations obtained by perturbating the values of the base model's parameters (which were still considered as a combination in the fine grid searches). We equipped our `grid_search` function with the option to perform a random search if the parameter `coarse` was set to `False`. The values for `lr, lambd, 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 30.

Finally, we picked the best 7 models resulted from these fine grid searches and used them as constituents of the ensemble. Those, along with their respective training and validation MEE, are reported in **Table 4.** Since the validation error is not a good estimate of the generalization error, we retrained each model on the entire development set and estimated the generalization capabilities on previously unseen data by using the internal test set, which was rigorously kept separated from the development set during the whole model

selection phase. **Figure 3** shows the MSE and MEE curves of each model on the development and internal test set. It is possible to observe that model 1 is one of the most stable. In comparison, the curves of model 0 (which has the same depth and batch size as the previous one) resulted to be less smooth, probably due to the higher learning rate. Another interesting aspect is that, out of all the possible configurations, the *TanH* proved to be more effective than the *Leaky ReLU*. The ensemble was lastly retrained on the original training set (ML-CUP20-TR) before being used for predicting the outputs of the ML-CUP20-TS.

Hyper-parameters	Configuration tested
units_per_layer:	{(20, 2), (20, 20, 2), (20, 20, 10, 2), (8, 8, 8, 8, 8, 2)}
acts:	{leaky_relu, tanh}
lr:	{0.001, 0.0001}
lr_decay:	{None, "exponential", "linear"}
staircase:	{True, False}
momentum:	{0.5, 0.8}
lambd:	{0, 0.001, 0.0001, 0.00001}
reg_type:	{"l1", "l2"}
batch size:	{1, 200, 'full'}
epochs:	{150, 400, 700}
init_type:	{"uniform"}
limits:	{(-0.001, 0.001), (-0.1, 0.1)}

Table 2. *Hyperparameters* used in the grid-search. The role of each hyperparameter is described in Section 2.2. In this table we list the combinations tried in the grid search.
Note: the activation functions reported above were not used jointly, but they represent the alternatives for the hidden layers.

4. CONCLUSIONS

The predictions of the ensemble were computed using the average of the outputs of its constituent models on the blind test and were saved on the file **pastaforgetters_ML-CUP20-TS.csv.** We are confident that, according to our estimate, our model performance on the blind test will be around 2.87434414 (in terms of MEE).

model	Development set MEE	Internal test set MEE
model 0	2.62572901	2.851231694
model 1	2.84072486	2.990059987
model 2	2.51288648	2.783022344
model 3	2.50061058	2.739867993
model 4	2.35971248	2.735106998
model 5	2.95708481	2.964919811
model 6	3.16456303	3.056200117
ensemble	2.70875875	**2.87434414**

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

ACKNOWLEDGEMENTS

We found this experience to be very interesting and useful for truly understanding how neural networks work. Developing a network by ourselves gave us a deeper insight into the theoretical aspects studied during the course. Having to code a network, test it and experiment with it clearly stimulated our critical thinking and reasoning. We acknowledge that this was a key experience, essential to prepare us to face the future challenges of work environment.

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

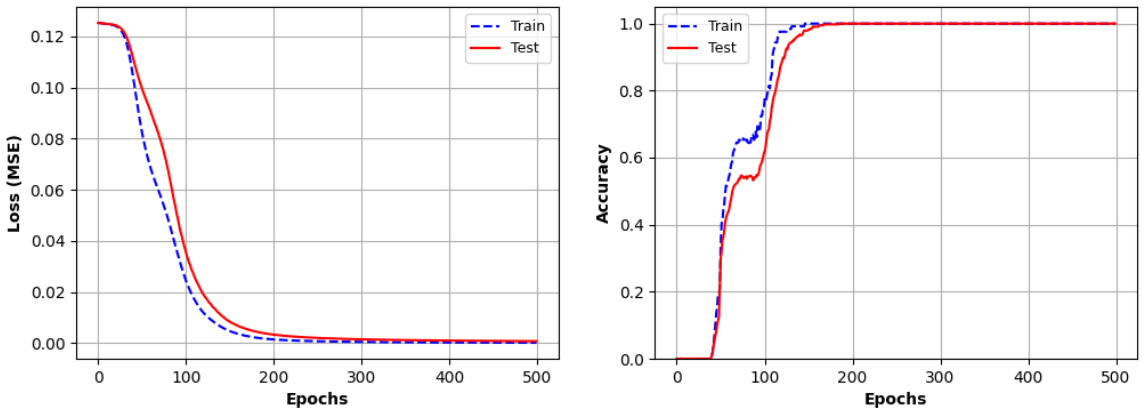
References

[1] S.Thurn, J. Bala, E. Bloedron, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Paphowicz, B. Roger, H. Vafaie, W. Van de Velde, W. Wenzel, J. Wnek and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, **1991**.

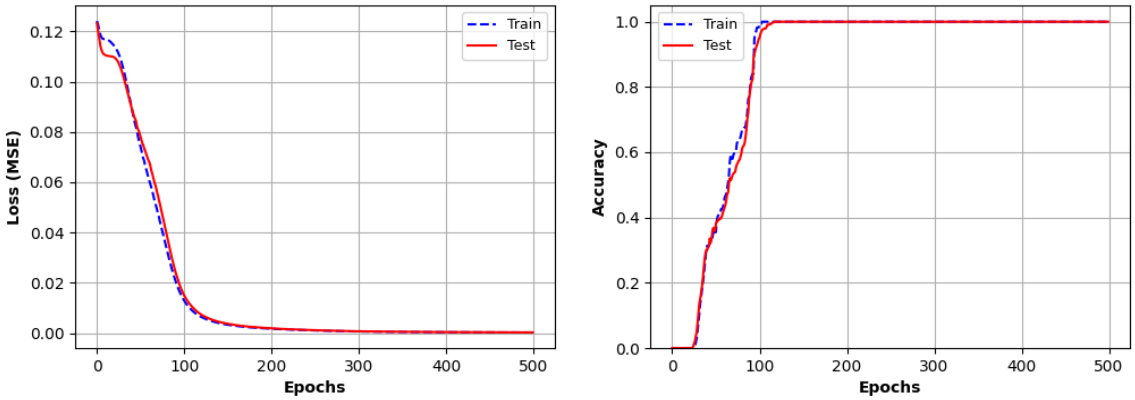
[2] A. Micheli: Derivation of the Back-propagation learning based algorithm, **2021**

[3] Richard Maclin. Popular ensemble methods: An empirical study. 11, 12 **1999**. doi: 10.1613/jair.614.

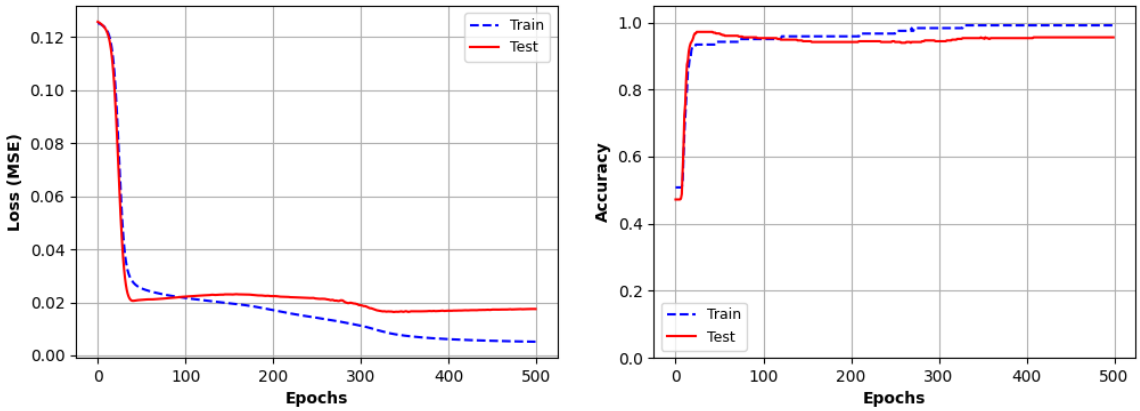
APPENDIX:



(A) Monk 1

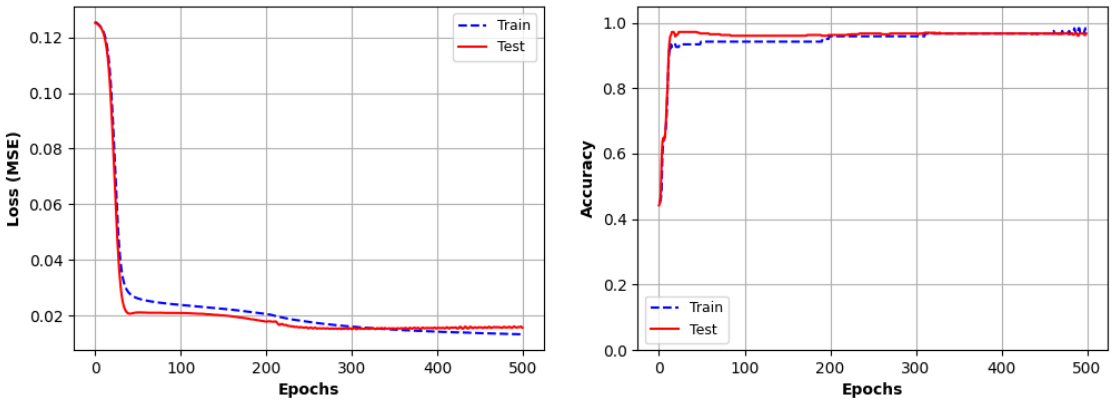


(B) Monk 2

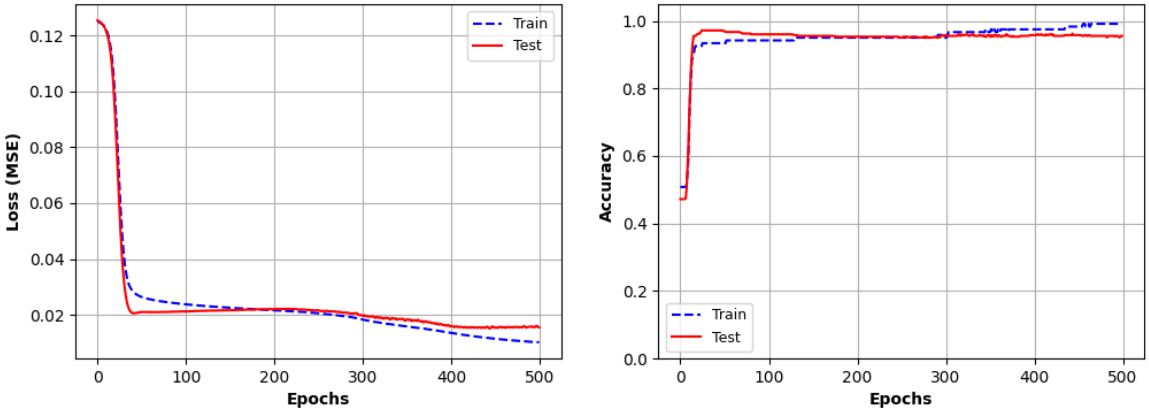


(C) Monk 3 (no regularization)

Figure 1. Learning curves for all three Monks' dataset. Monk 3 here is presented without regularization.



(**A)** Monk 3 (L2 regularization)



(**B)** Monk 3 (L1 regularization)

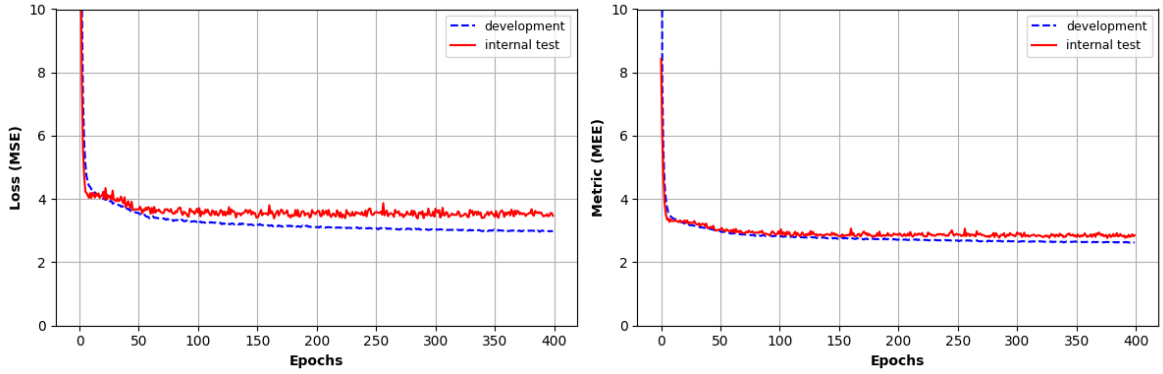
Figure 2. Learning curves for Monk 3 with L2 and L1 regularization.

model	Avg Train MEE	Avg Validation MEE	topology	activation function	batch size	eta	learning rate scheduler	staircase	alpha	lambda	reg_type
model 0	2.592547728	3.044978591	(10, 20, 2)	(tanh, identity)	16	0.002143519386	None	/	0.6	0	None
model 1	2.717161808	3.0496815297	(10, 20, 2)	(tanh, identity)	16	0.001	None	/	0.41992896013	0	None
model 2	2.460841015	2.87303076	(10, 20, 20, 2)	(tanh, tanh, identity)	100	0.01	None	/	0.6	0	None
model 3	2.51957026	2.9899078	(10, 20, 20, 2)	(tanh, tanh, identity)	100	0.0089121407692	None	/	0.6	0	None
model 4	2.305994365	2.929978	(10, 20, 20, 2)	(tanh, tanh, identity)	91	0.0071859582386	None	/	0.79774281122	0	None
model 5	2.852249912	3.05764814	(10, 20, 20, 2)	(leaky relu, leaky relu, identity)	200	0.005655844946	exponential	True	0.77093492968	1e-05	12
model 6	3.263625912	3.37747532	(10, 20, 20, 2)	(leaky relu, leaky relu, identity)	200	0.005	exponential	True	0.6	1e-05	12

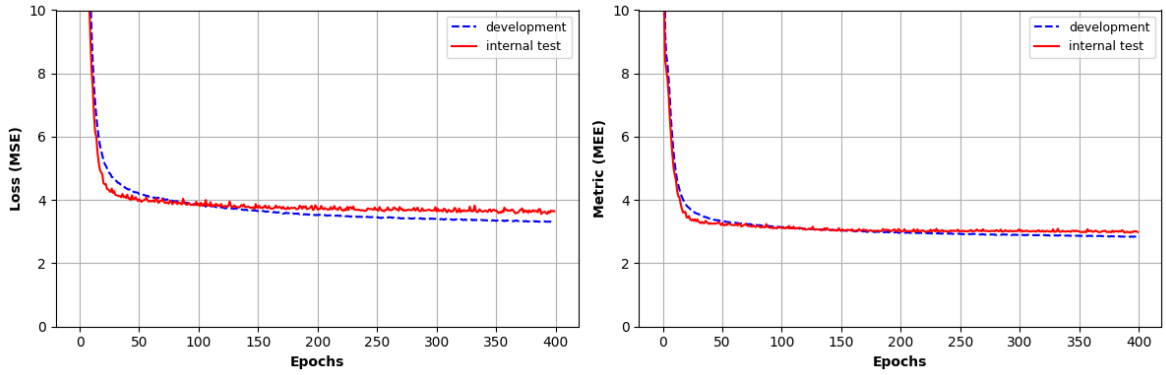
Table 4. *Hyperparameters* and performance of the seven best models in term of validation MEE

Figure 3. Ensemble models:

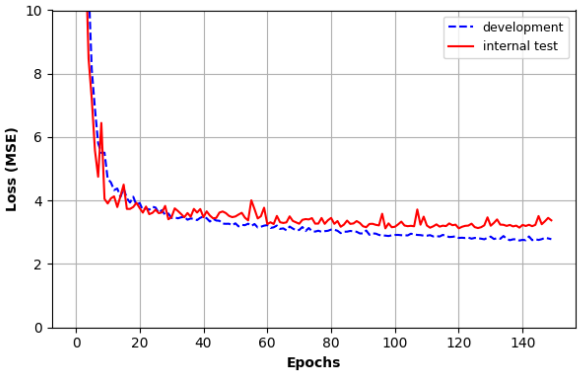
(a) Model 0

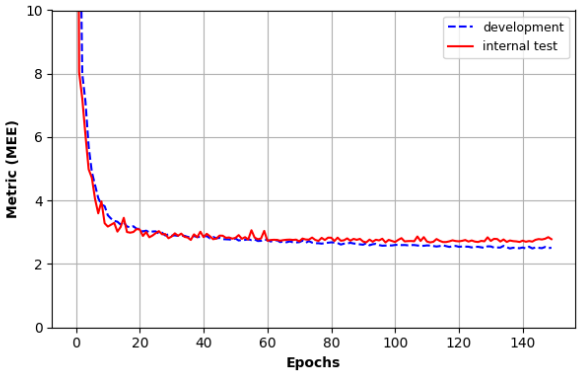


(b) Model 1

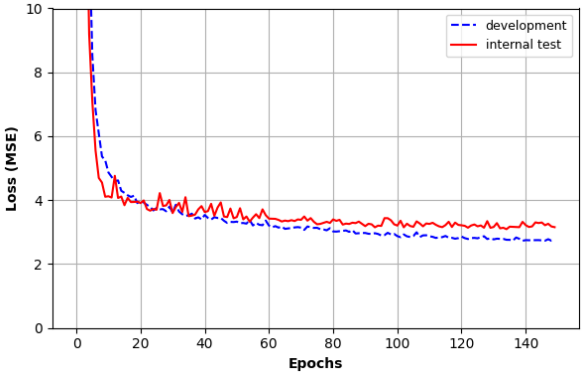


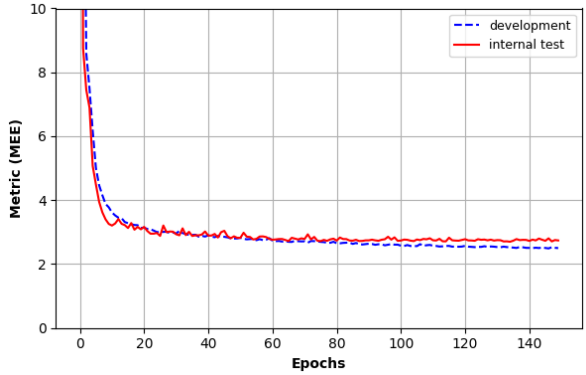
(c) Model 2



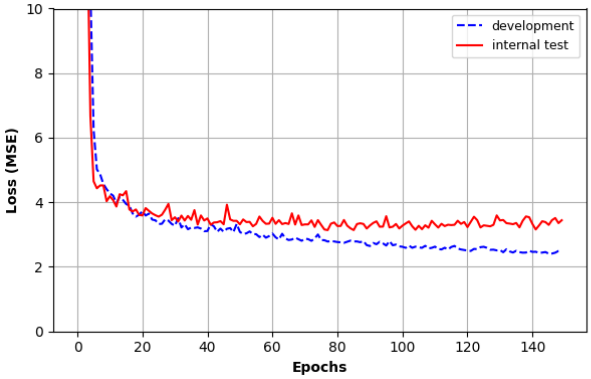


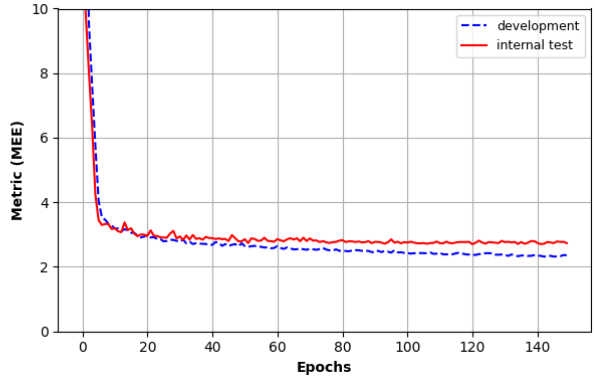
(d) Model 3



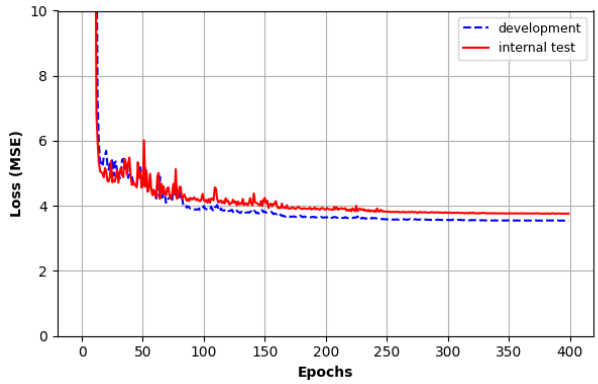


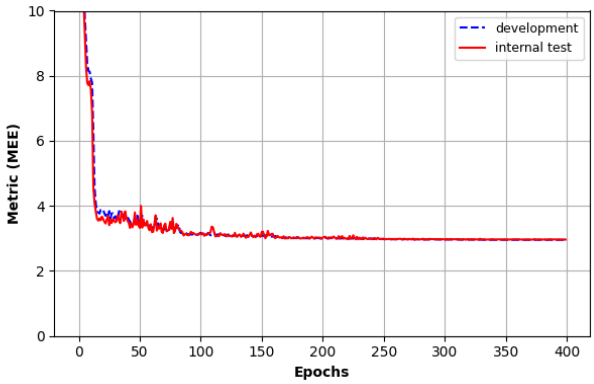
(e) Model 4





(f) Model 5





(g) Model 6