Machine Learning Project Report

Giulia Ghisolfi, Lorenzo Leuzzi, Irene Testa
Master Degree in Computer in Science (Artificial Intelligence Curriculum), University of Pisa
g.ghisolfi@studenti.unipi.it, i.testa@studenti.unipi.it, l.leuzzi1@studenti.unipi.it
ML course (654AA), Academic Year: 2022-2023
Date: 22/01/2023
Type of project: **A** *(Python)*

Abstract

In this work we have implemented a feed-forward fully-connected Neural Network. We coded multiple variants of the Gradient Descent algorithm and tested them on the MONK's problems. To select the best model for the ML-CUP22 competition we performed a coarse-to-fine grained grid search over the hyper-parameters of the implemented Neural Network, using the k-fold cross validation technique. The final model we chose is an ensemble made by the 10 best hyper-parameters configurations.

1 Introduction

Our goal was to better understand the underlying principles and mechanisms of Neural Networks implementing and testing multiple variants of the Gradient Descent algorithm. We also aimed at approaching the ML-CUP22 task with a rigorous methodology, using advanced model selection and validation techniques. In particular we performed a coarse-to-fine grained grid search over the hyper-parameters with a k-fold cross validation technique and built and ensemble of the best 10 models.

2 Method

2.1 Implementation choices

We implemented a multi-layer, feed-forward, fully connected Neural Network to solve both classification and regression problems. Along with the standard Gradient Descent training algorithm we also implemented some of its variants. In particular, we coded: (1) different loss functions (Mean Square Error, Mean Euclidean Error and Log-loss); (2) different evaluation metrics (Mean Square Error, Mean Euclidean error, Log-loss and Accuracy); (3) different weights update policies (stochastic, mini-batch and batch); (4) different weights initialization policies (such as Xavier [2] and He [3] initialization methods); (5) different learning rate schedules (fixed and linear decay); (6) different stopping criteria (such as the early stopping technique); (7) different activation functions (identity, ReLu, leaky-ReLu, logistic, tanh, softplus and softmax); (8) l2 regularization; (9) momentum and Nesterov momentum. The above described features can be fine tuned through a set of hyper-parameters. Further details on their usage are provided in the `README.md` file in the project package.

2.2 Tools and libraries

The software was developed using the Python programming language. The following libraries were used: (1) `numpy` and `scipy` for numerical computations; (2) `matplotlib` to plot graphs; (3) `pandas` and `json` to handle `.csv` and `.json` files respectively; (4) `pickle` for the objects serialization; (5) `scikit-learn` for some minor utilities.

2.3 Software overview

The class `NeuralNetwork` implements a fully-connected Neural Network capable of solving both classification and regression tasks, according to the `classification` flag specified as a parameter of the class constructor. This class contains a list of `Layer` objects and exposes the main following methods: `fit()`, `predict()`, and `score()`. If the `classification` flag is set to `True` the above methods automatically handle target encoding to properly work with the activation function specified for the output layer, supporting both binary and multi-class classification tasks. The purpose of this design choice was to make the model easy to use.

The class `Ensemble` implements an ensemble of Neural Networks. Its constructor takes as arguments a list of dictionaries specifying the hyper-parameters for instantiating the `NeuralNetwork` constituent objects. For regression tasks the ensemble predictions are computed averaging the targets values predicted by its constituents models, while for classification tasks the ensemble predictions are the targets most frequently predicted by the ensemble's models.

The file `validation.py` contains the implementation of some utility functions to tune Neural Networks hyper-parameters and to apply validation techniques.

We also provided a basic command-line interface to easily replicate the experiments we have performed on the MONK's problems and on the CUP dataset. For its usage and installation see the `RADME.md` file in the project package.

2.4 Validation schema

We used the MONK's datasets as benchmarks to assess the correctness of our implementation, therefore we simply trained and tested the models on the already provided training and test sets.

As for for the CUP task, instead, we shuffled the ML-CUP22-TR dataset and divided it in two parts: a Development Set (80%) and an Internal Test Set (20%). We did not perform any form of pre-processing. We conducted some preliminary trials on the Development Set to discard under-performing or irrelevant hyper-parameters values, evaluating the models on an hold out 20% of the Development Set. Then, we performed a coarse-grained grid search using a 3-fold cross validation technique on all the Development Set to determine the best values for the network's hyper-parameters, ranking them according to their average MEE over the folds. Afterwards, we performed a finer grid search, exploring smaller and denser ranges of hyper-parameters, using a 5-fold cross validation technique and ranking, again, the models according to their average MEE over the folds. We then selected the 10 best performing models as the constituents of an ensemble, whose performance had been compared with the one of the best 10 models. Since, the ensemble showed to perform better than all its constituents, we chose it as final model, trained it on all the Development Set and assessed its generalization capabilities on the Internal Test Set. Finally, we trained the ensemble on the whole ML-CUP22-TR dataset and used it to predict the targets for the blind test set.

2.5 Preliminary trials

For the MONK's benchmark problems we used similar architectures to the ones reported and tested in [5]. With such configurations we managed to achieve high accuracies on the test set.

For the ML-CUP22 dataset, we performed several trials to figure out the hyper-parameters combinations which gave worse results in order to reduce the search space and therefore speed up the successive grid searches. Details on the trials we have performed along with the obtained results are reported in Section 3.2.1.

3 Experiments

3.1 Monk Results

On all the MONK's problems we used one-hot encoding for the encoding the inputs, a single hidden layer with 4 units and a single output unit. We used the logistic activation function for both the hidden and output units, classifying each input as a '1' if the value of the activation function of the output unit was ≥ 0.5 and as a '0' otherwise. For all the problems we used MSE as the loss function, fixed learning rate and momentum coefficient $\alpha = 0.9$. For MONKS1 and MONKS2 problems we used batch size = 1 (stochastic), while for MONKS3 problem we used batch size = 4. We initialized the weights sampling their values from a uniform distribution in the range [-0.1, 0.1] for the first two problems, and in the range [-0.7, 0.7] for the third. In the third problem we also introduced the l2 regularization coefficient $\lambda = 5 \times 10^{-4}$.

Table 1 reports the Mean Square Error (MSE) and the Accuracy on the Training Set (TR) and on the Test Set (TS) averaged over 5 trials, along with the hyper-parameters we used for each of the three problems. Learning and Accuracy curves of a trial are shown in Figures 1, 2, 3 and 4.

Task	**Batch size**	**Initial lr**	λ	**MSE (TR/TS)**	**Accuracy (TR/TS)**
MONKS1	Stochastic	0.05	0	0.00320/0.00064	100%/100%
MONKS2	Stochastic	0.05	0	0.00040/0.00047	100%/100%
MONKS3	4	0.005	0	0.02289/0.04024	98.03%/94.49%
MONKS3 (reg.)	4	0.005	0.0005	0.06677/0.04758	93.44%/97.22%

Table 1: Performance results averaged over 5 trials for the MONK's datasets.

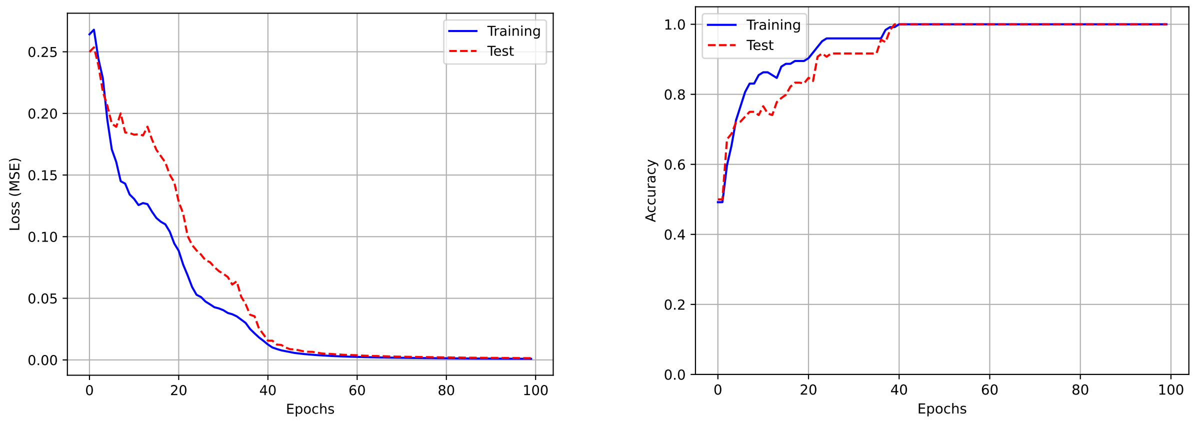

Figure 1: Loss and Accuracy curves over epochs on MONKS1 dataset.

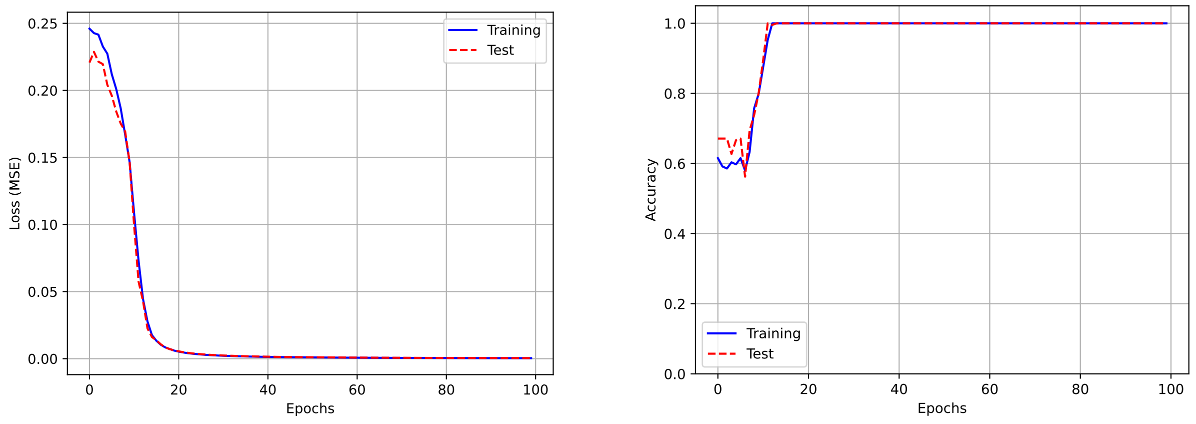

Figure 2: Loss and Accuracy curves over epochs on MONKS2 dataset.

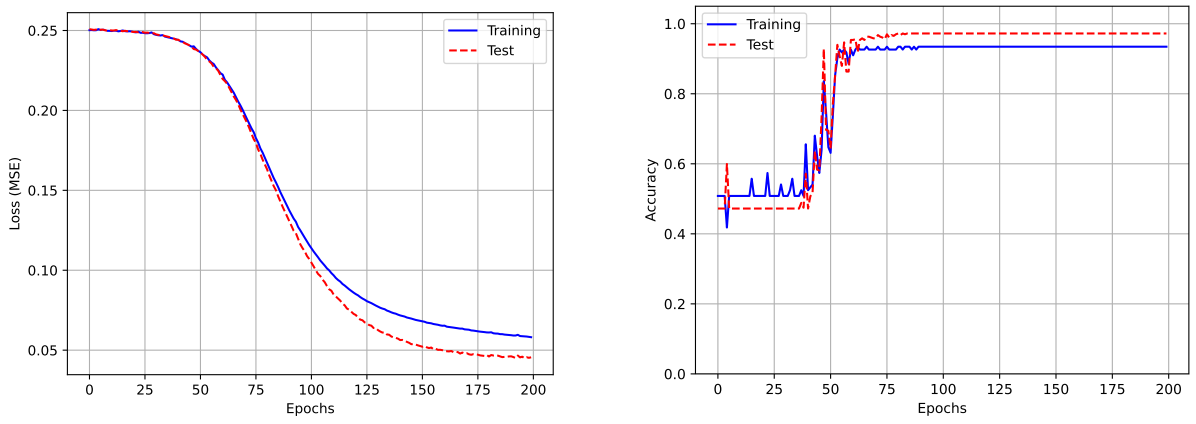

Figure 3: Loss and Accuracy curves over epochs on MONKS3 dataset without regularization.

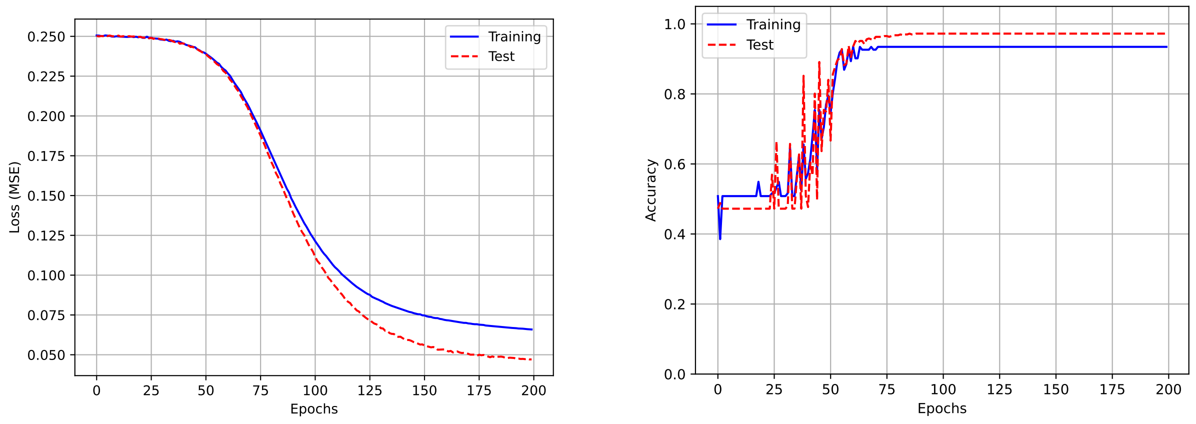

Figure 4: Loss and Accuracy curves over epochs on MONKS3 dataset with l2 regularization.

3.2 Cup Results

As already described in Section 2.4, we splitted the dataset in two disjoint sets: the Development Set (80%) and the Internal Test Set (the remaining 20%). The first had been used to perform model selection, while the latter to assess the generalization capabilities of the final chosen model. We decided to use MSE as the loss function minimized by the Neural Networks during the training phase and to use MEE to evaluate their performances. We also chose to use a Neural Network with two output units, one for each of the targets of the CUP dataset, with the identity as activation function. Such choice has been made after having noticed that the two targets in the Development Set seemed to be moderately negatively correlated according to the Pearson correlation coefficient, whose value was of -0.52.

3.2.1 Preliminary trials

Initially, we conducted some preliminary trials to figure out, and therefore discard, under-performing or irrelevant hyper-parameters combinations. To do so, we reserved 20% of the Development Set to test models, measuring the MEE between their predictions and the expected outputs. At first, we tried to grasp the maximum number of epochs for which the models show to improve their performance, therefore we run different models configurations fixing the maximum number of epochs to 700. These experiments revealed us that the measured MEE never showed to significantly decrease after epoch 500. We then tried different network architectures and discarded models with a single hidden layer or with too few units per layer which appeared to be under-performing, together with models with more than 3 hidden layers, whose fit required higher computational times and seemed not to significantly improve performances. In this screening phase we also managed to detect and therefore discard some hyper-parameters combinations which led to overflow. Furthermore, we decided to fix some hyper-parameters values which seemed to be less influential, such as: (1) the activation function of the hidden layers; (2) low momentum coefficients; and (3) the weights initialization method, for which we decided to use a variant of the Xavier method firstly proposed by Glorot and Bengio [1, 2]. We also decided to fix the batch size to 128, which appeared to us as a fair compromise between the time needed to train the models and the performance improvement smaller batch sizes generally provide. Lastly, to avoid overfitting, we decided not to use early stopping but to use, instead, l2 regularization.

3.2.2 Grid search

After the above described preliminary trials, we conducted a coarse-grained grid search over 1407 possible configurations using a 3-fold cross validation on the Development Set. Such a validation schema has been chosen in order to reduce the bias in the weights initialization and to avoid the estimation of the performances to be dependent on the training-validation splitting. We chose to use no more than 3 folds to limit the time needed to complete the validation cycle. This first grid search helped us to figure out the best intervals or values for the hyper-parameters to fine tune later. Indeed, ranking the models according to their average MEE over the folds we were able to identify and discard some hyper-parameters values which led to worse results.

We then performed a finer-grained grid search over 1119 combinations using, this time, a 5-fold cross validation, enlarging the ranges of some hyper-parameters in the directions they showed to perform better and/or using smaller steps.

The hyper-parameter values explored in the coarser and finer grid searches are shown in Tables 2 and 3 where, for the sake of clarity, we also reported some hyper-parameters that in the preliminary trials we decided to fix.

The τ parameter, whose value is relevant only when linear decay learning rate schedule is used, represents the maximum epoch after which the learning rate is kept fixed at 1/10 of its initial value.

Table 4 reports the hyper-parameters values of the 10 best performing models with the averaged MEE over the 5 training and validation folds (TR and VL respectively), and the MEE computed on the Internal Test Set (TS). The results on TS have been reported here for completeness purposes, but they have not been taken into account to make any decision regarding the selection of the final model.

Hyper-parameter	**Values**
Hidden layer size	[30, 30], [30, 60], [60, 30], [30, 30, 30]
τ	200, 500
λ	0.0001, 0.00025, 0.0005, 0.001
α	0.75, 0.8, 0.85, 0.9
Nesterov	True, False
Learning rate	Fixed, Linear decay
Initial learning rate	0.2, 0.1, 0.01, 0.005, 0.001, 0.0005, 0.0001
Maximum epochs	500
Batch size	128
Output layers activation function	Identity
Hidden layers activation function	Logistic
Loss function	Mean Square Error
Evaluation metric	Mean Euclidean Error

Table 2: Hyper-parameters explored with the first (coarse-grained) grid search.

Hyper-parameter	**Values**
Hidden layer size	[60, 30], [30, 30, 30]
τ	500
λ	0.005, 0.001, 0.0005, 0.00025, 0.0001, 0.00005
α	0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9
Nesterov	True, False
Learning rate	Fixed, Linear decay
Initial learning rate	0.12, 0.08, 0.06, 0.05, 0.04, 0.01, 0.008, 0.006, 0.004, 0.008, 0.002
Maximum epochs	500
Batch size	128
Output layers activation function	Identity
Hidden layers activation function	Logistic
Loss function	Mean Square Error
Evaluation metric	Mean Euclidean Error

Table 3: Hyper-parameters explored with the second (fine-grained) grid search.

3.2.3 Final chosen model and discussion

Among the 10 best models, whose hyper-parameters are reported in Table 4, no one seemed to significantly out-performed the others, but they all show similar performances in terms of MEE averaged over the validation folds, with a standard deviation ranging between 7.65% and 9.42% of the corresponding measure. For this reason, along with the fact that these 10 models have sufficiently distinct characteristics in terms of hyper-parameters, and they all showed smooth learning curves (see Figures 6 and 7 in the Appendix A), we decided to build an ensemble of them.

To compare the performances of the ensemble model with the ones of the best 10 models, we averaged the predictions made by each model on the validation folds and computed the MEE between such averaged predictions and the expected targets, obtaining a value of 1.41846. This result showed that the ensemble performed significantly better than all its constituents models – as already evidenced in several empirical studies (see e.g. [4]) – therefore, we chose it as final model. To assess the generalization capabilities of the ensemble we trained it on the Development Set, performing 5 trials for each of its 10 constituents models (for a total of 50 models) and averaging the predictions made by all of the models on the Internal Test Set. The MEE achieved by the ensemble on the Development Set and Internal Test Set is reported in Table 5. Since the Internal Test Set has never been used during the model selection phase, such value should give a fair estimation of the error of the final model on unseen data. The learning curves of the ensemble's models for MSE and MEE over the Development and Internal Test Set, are shown in Figure 5.

Model	Hidden layer size	λ	Learning rate	Initial lr	α	Nesterov	TR Score	VL Score	TS Score
Model 0	[60, 30]	0.00010	Linear decay	0.06	0.75	False	1.26840	1.43072	1.39423
Model 1	[60, 30]	0.00050	Linear decay	0.06	0.8	False	1.31034	1.43192	1.36736
Model 2	[60, 30]	0.00010	Linear decay	0.05	0.85	False	1.17854	1.43260	1.42724
Model 3	[60, 30]	0.00025	Linear decay	0.06	0.8	False	1.27271	1.43293	1.36388
Model 4	[30, 30, 30]	0.00010	Linear decay	0.06	0.7	False	1.28893	1.43343	1.41352
Model 5	[60, 30]	0.00050	Linear decay	0.08	0.8	True	1.22127	1.43510	1.38581
Model 6	[60, 30]	0.00050	Linear decay	0.06	0.85	False	1.27149	1.43592	1.38528
Model 7	[60, 30]	0.00025	Linear decay	0.12	0.65	False	1.22188	1.43631	1.40841
Model 8	[60, 30]	0.00025	Linear decay	0.05	0.85	False	1.21986	1.43657	1.36364
Model 9	[60, 30]	0.00025	Fixed	0.008	0.9	True	1.30741	1.43671	1.39379

Table 4: Best models hyper-parameters and MEE averaged over the training folds (TR) and validation folds (VL) of the 5-fold cross validation, along with the MEE achieved by them on the Internal Test Set (TS) after having trained the models on the whole Development Set.

Training MEE	Internal Test MEE
1.24344	1.35823

Table 5: MEE computed on the Training and Internal Test for the ensemble model after a full training on the whole Development Set.

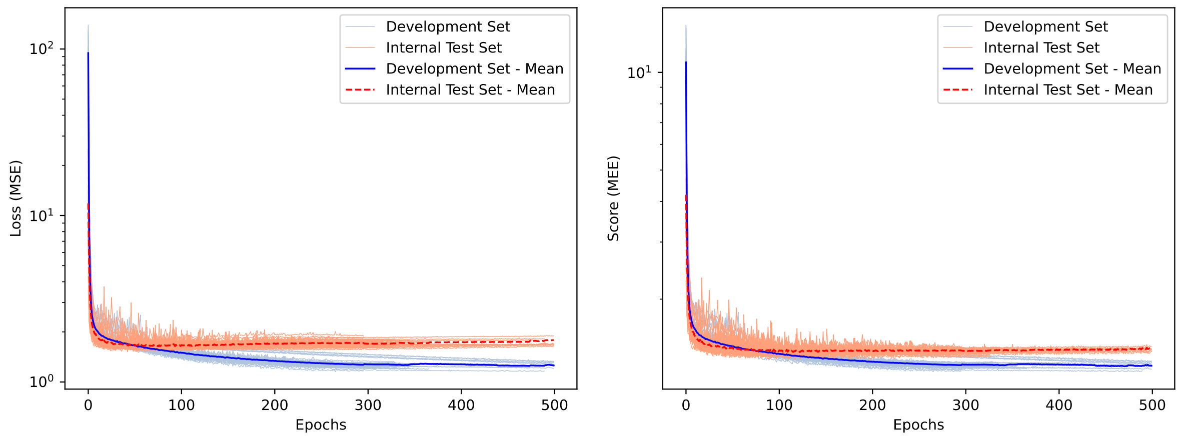

Figure 5: Learning curves of the ensemble's models showing, for each epoch, the Loss (MSE) and the Score (MEE) on a logarithmic scale over the Development and Internal Test Set. The thick curves have been plotted averaging, for each epoch, the losses of the ensemble constituents models. Since some of the models stopped their training before the fixed maximum epoch 500 because the MSE loss on the Development Set showed no significant improvement, the tail of the ticker curves is the average of a subset of the ensemble's models curves.

3.3 Computational time and hardware resources

The time required on average for training a model on the MONKS1 and MONKS2 datasets with 100 epochs, batch size 1 (stochastic) and the architecture described in Section 3.1 is around 5 seconds while the time needed for the MONKS3 dataset with the same architecture, 200 epochs and batch size 4 is around 8.4 seconds.

The average time required to train a model on the CUP Development Set with a maximum of 500 epochs and a batch size of 128 is around 78 seconds. The training of the final chosen model for the CUP task took 3375 seconds (56 minutes and 15 seconds).
The above times were measured on a Dual-Core Intel Core i5 @ 1.6 GHz.

To perform the coarse and fine grained grid searches on the CUP dataset we used two Intel Core i5-8210Y @ 1.60GHz, a Intel Core i5-8250U CPU @ 1.60GHz and we also dispatched part of the computation on the cloud computational environment offered by Kaggle Notebook platform (https://www.kaggle.com/).

4 Conclusion

We trained the ensemble made up of 50 models on the whole ML-CUP22-TR dataset and consequently used it to predict the targets of the ML-CUP22-TS dataset. We estimate that the MEE on the blind test set of our final model will be near the value we computed on the Internal Test Set, thus we expect it to be around 1.36. The predictions that our ensemble made for the blind test set are stored in the file `Ensemblers_ML-CUP22-TS.csv`. Our team name is **Ensemblers**.

Acknowledgments

The project gave us the opportunity of deeply understanding some theoretical aspects seen during the ML course – such as the Backpropagation algorithm and the way the values of the hyper-parameters interact – through a challenging yet stimulating hands-on experience. We faced various obstacles during the model selection phase, mainly due to the limited time and computational resources, but we managed to overcome them through careful experimentation and analysis, always backed up from the theory. Overall, this project served as a valuable learning experience which gave us also the opportunity to sharpen our soft skills of team-working and problem solving.

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

References

[1] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[4] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[5] Sebastian Thrun. The monk's problems: A performance comparison of different learning algorithms. Technical report, 1991.

A Appendix

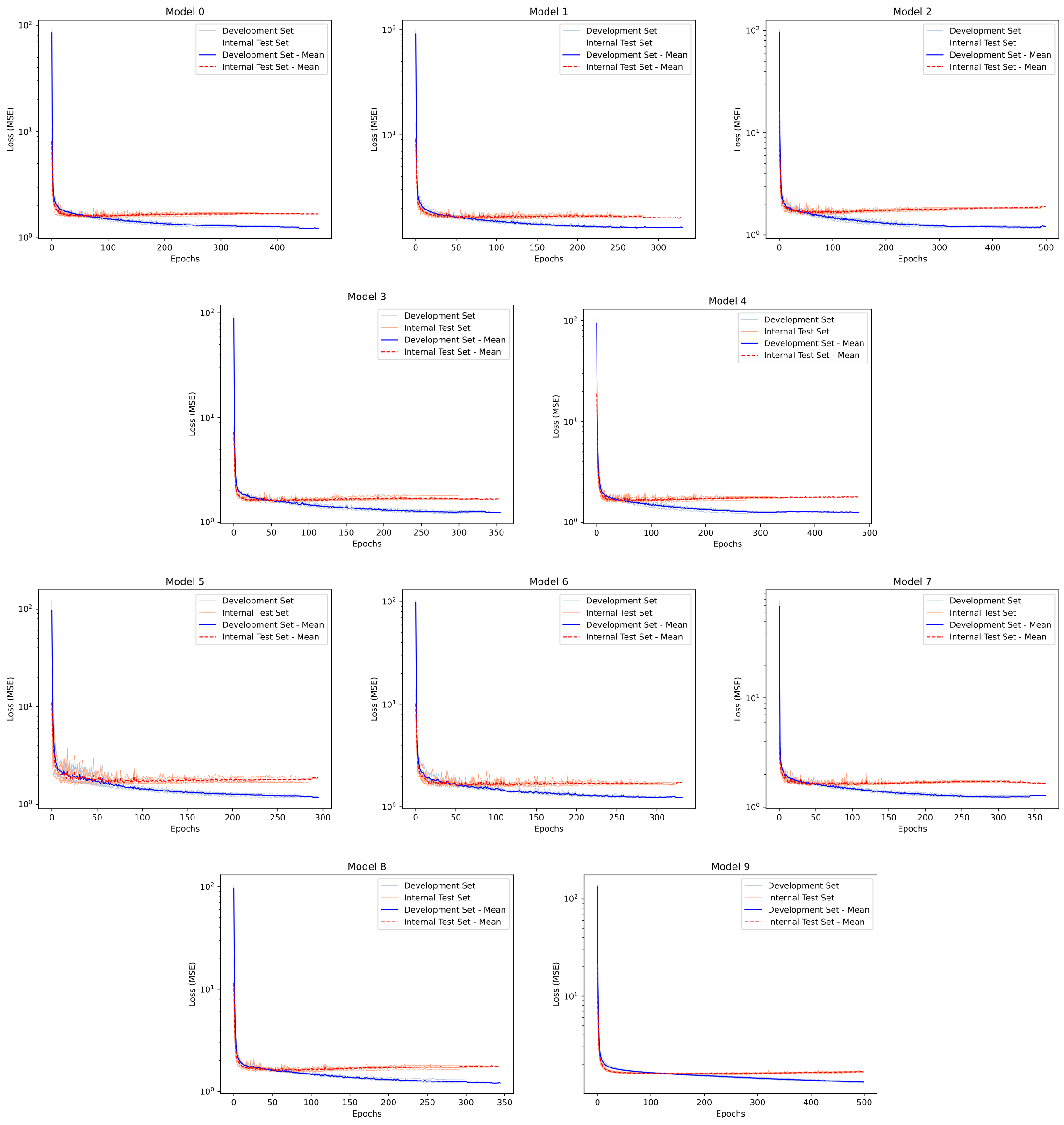

Figure 6: Learning curves of the ensemble showing, for each epoch, the Loss (MSE) on a logarithmic scale over the Development and Internal Test set. All of them are quite smooth. The smoothest curve is the one of model 9, the only one with a fixed learning rate schedule; the less smooth curve is the one of model 5, probably because of the combination of the learning rate schedule, the initial learning rate and the momentum coefficient together with the Nesterov momentum method.

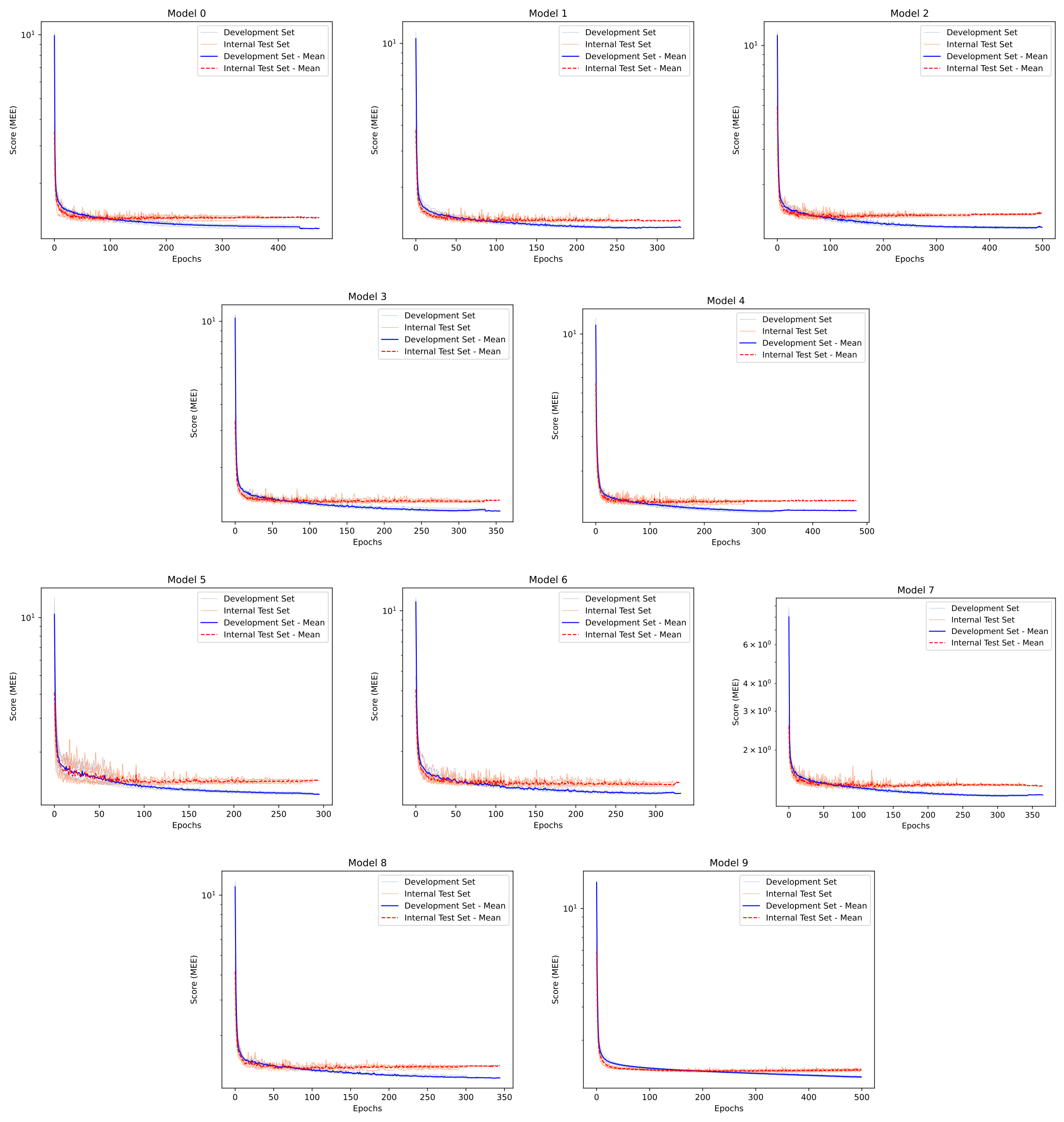

Figure 7: Learning curves of the ensemble showing for each epoch the Score (MEE) on a logarithmic scale over the Development and Internal Test Set.