Machine Learning

Gianmarco Ricciarelli gianmarcoricciarelli@gmail.com
Stefano Carpita carpitastefano@gmail.com

ML - Academic Year: 2018/2019

February 14, 2019

Type of project: **A**

Github Page

Abstract

With this report we describe our type **A** project for the Machine Learning course. We experiment two datasets, namely MONK's and CUP, by building from scratch a neural network's implementation, which we've validated by searching for the best hyperparameters' combination via well-known validation techniques. Finally, for each one of the datasets, we collected the data describing the results.

1. Introduction

By chosing a type **A** project, we followed the goal of building and optimizing a *feedforward neural network model* by searching the best hyperparameters' combination in order to obtain the best performances on the MONK's and CUP datasets. The model we've built implements the standard *backpropagation algorithm* with *gradient descent*, as described in [1]. As optimization tool, we've searched the so-called hyperparameters space by using well-known validation algorithms like *k-fold cross validation* and *(random) grid search*, described in [1] and [2]. More technical details regarding the implementation can be found in section 2.

2. Methods

Our code base is written using the *Python* programming language, which was enhanced with numerical libraries like *NumPy* and *Pandas* to ease the computational cost involved in implementating popular machine learning algorithms. Following the philosophy of type **A** projects, we haven't used out-of-the-box implementations for the algorithms composing our framework. We now give a brief overview of the code's structure, discuss some implementation choices and present our neural network's initialization and training algorithms. Visit the project's *Github Page* if you want to check our project's code base.

2.1. Code overview

We can view our code base essentialy divided in two branches: the *neural network* branch and the *model selection and assessment* branch. The first one is composed by the code for implementing the neural network model, and all the code related to the building and training of such a model, e.g. the activation functions, the losses, the regularization metrics and so on, that is divided in several scripts in order to ease the implementation. The code base's second branch consist in the model selection and assessment related code, which is composed by several classes, each one implementing an algorithm like `GridSearch` and `KFold_Cross_validation` which are utilized for the model's validation, selection and assessment, and the scripts that build up the testing routines for the MONK's and CUP datasets. The second branch also is splitted in several scripts.

2.2. Implementation choices

Since we wanted to be able to take different paths during the building phase of our model, we enriched the first code base's branch by writing diffent activation functions, losses and regularization metrics. In the `activation` script we writed the code for the `sigmoid`, `relu`, `tanh` and `identity` activation functions. The `regularizers` script contains the code for implementing both the L^1 and L^2 regularization metrics, as described in [1], and the `losses` script contains the code for implementing popular loss function like, for example, the `mean_squared_error` and the `mean_euclidean_error`. All this metrics are applied in the `nn` script, which contains the code for implementing the neural network model, and hence the backpropagation algorithm and the training algorithm, described in section 2.3. The `NeuralNetwork` class, which is contained in this script, represents a neural network using the standard *backpropagation algorithm* with *gradient descent*, *standard momentum* and *regularization*, and follows the *minibatch* approach, as described in [1]. The `NeuralNetwork` class contains also various early stopping metrics, like the GL_ϵ and the PQ_ϵ which we implemented following [3]. More details about what particular combination of metrics was used during the experimental phase can be found in section 3.

2.3. Network's initialization and training

Our neural network can be initialized for pursuing either a *classification* task or a *regression* one. The initialization phase for the two tasks is the same. During the initialization phase, we pass to the network the design matrix `X` and target column vector `y`. Other than that we pass the number of neurons per hidden layer `hidden_sizes`, the maximum number of epochs `epochs`, the minibatch's size `batch_size`, the activation function for each layer `activation` and the hyperparameters for the learning rate, η, the momentum, α and the regularization, λ. In order to initialize the network's weights matrices we followed the *normalized initialization* technique, described in [1] and [4], e.g. the formula defining the network's weights in the uniform distribution taken in the range

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{m+n}},\ \frac{\sqrt{6}}{\sqrt{m+n}}\right],$$

in which m and n, respectively, represents the number of inputs and outputs for the layer. During the training phase we essentialy apply the *forward propagation* and the *backpropagation* phases of the standard *backpropagation* algorithm over a minibatch of examples taken from the training set. During each epoch of training we also keep track of the errors between the predicted values and target ones, in order to be able to plot the *learning curves* for the training session. At the end of each epoch the neural network's weights are updated following the standard backpropagation/momentum/regularization convention.

3. Experiments

Before delving into the details of the results we obtained by applying our models to the datasets, we provide some informations about the *preprocessing routines* and *validation schema* we decided to apply to the MONKS dataset, see section 3.2 for the details regarding the CUP dataset. Here are the steps we followed in order to reach the final states of our analysis.

1. Since the MONKS datasets' feature are categorical, that is, every feature's value represents a class, not a numerical value, we preprocessed the three datasets by writing a `monks_binarize` script for applying a *1-of-k encoding*, as recommended during the course, hence obtaining 17 binary input features.

2. As a supplementary preprocessing phase, we've applied a *symmetrization* to the matrix containing the dataset's values, in order to ease the training during the validation phase by having a matrix of values closer to the symmetric behavior of the sigmoid function, which is used for this dataset.

3. Since we've chosen to follow [2] for the hyperparameters' search during the validation phase, we first performed some *preliminary trials* in order to have a glimpse on the best intervals for searching our model's hyperparameters. During this trials we manually varied the model's hyperparameters, e.g. the learning rate, the momentum constant and so on, and observed the resulting *learning curves* (see appendix A for some other cases discovered during the analysis). For this part of the analysis we've used the 20% of the training set as validation set, and the remaining part for training the network.

4. We then deepen the search using the most interesting intervals discovered during the preliminary trials in the validation phase by using our implementation of the *(random) grid search* algorithm, in which we also used our implementation of the *k-fold cross validation* algorithm (which follows the standard approach of using a value of 3 for the k parameter).

That is, our validation schema for the MONKS dataset essentially consist in using the random grid search algorithm to investigate some random sampled "points" in the hyperparameters' space, evaluating the performances for each one of this points and finally selecting the best combinations of parameters based on the diffent metrics like *generalization error*, *accuracy*, *precision*, *recall* and *f1-score*.

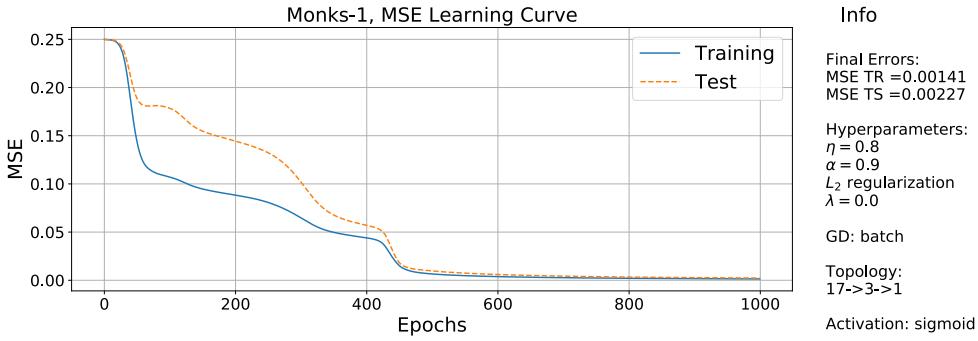
3.1. Monk's results

Task	Topology	Batch size	Activation	η	α	λ	MSE (TR - TS)	Accuracy (TR - TS) (%)
MONK 1	17 -> 3 -> 1	batch	sigmoid	0.8	0.9	0.0	0.03185 - 0.04871	97 % - 94 %
MONK 2	17 -> 3 -> 1	batch	sigmoid	0.5	0.9	0.0	0.00139 - 0.00148	100 % - 100 %
MONK 3	17 -> 3 -> 1	batch	sigmoid	0.6	0.9	0.0	0.02283 - 0.04771	97.8 % - 93.7 %
MONK 3 (reg.)	17 -> 3 -> 1	batch	sigmoid	0.6	0.9	0.01	0.05955 - 0.04820	93.3 % - 97.1 %

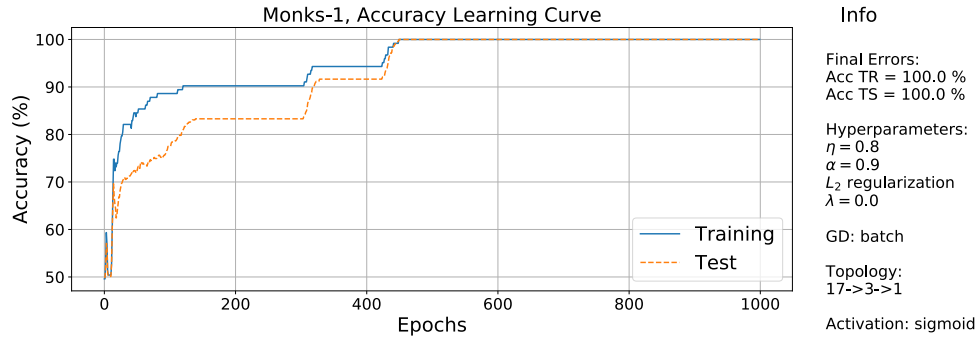
Table 1: *Average prediction results obtained for the MONKs tasks.*

3.2. Cup results

We now proceed to detail the results we obtained for the CUP dataset. As for the MONKS dataset, we've employed a *validation schema* consisting in an application of the *random grid search* algorithm in conjunction with the *k-fold cross validation* algorithm. As we've said before, we've chosen a standard value of 3 for algorithm's k parameter. In order to come up with final ranges for model's hyperparameters we've performed some preliminary tests on the dataset, as for the MONKS dataset.
In table 2 you can see the final ranges we selected for the random grid search algorithm for the CUP dataset.

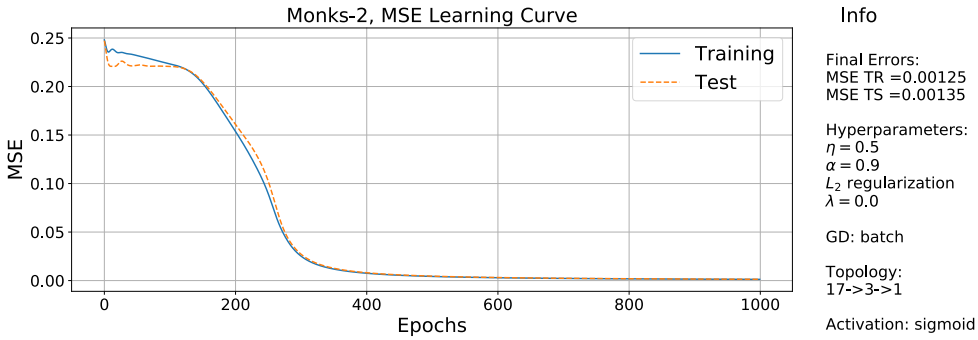


(a)

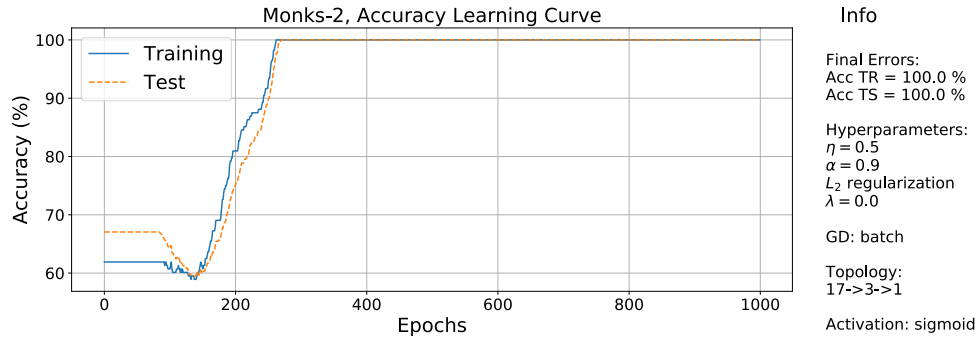


(b)

Figure 1: *Example of a final learning curve on MONKS 1.*

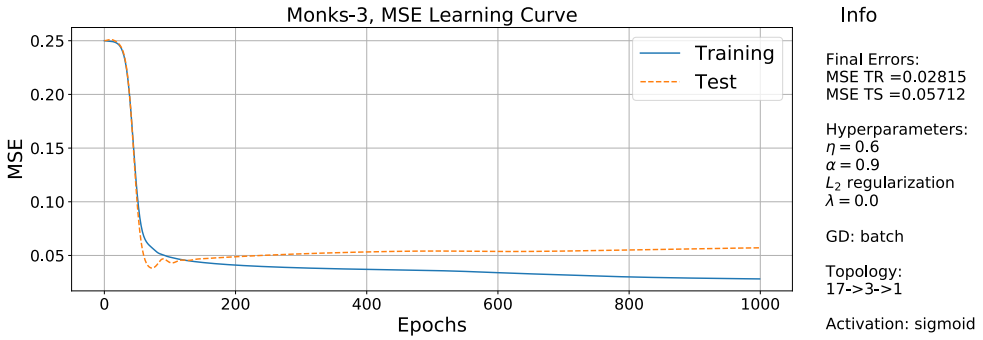


(a)

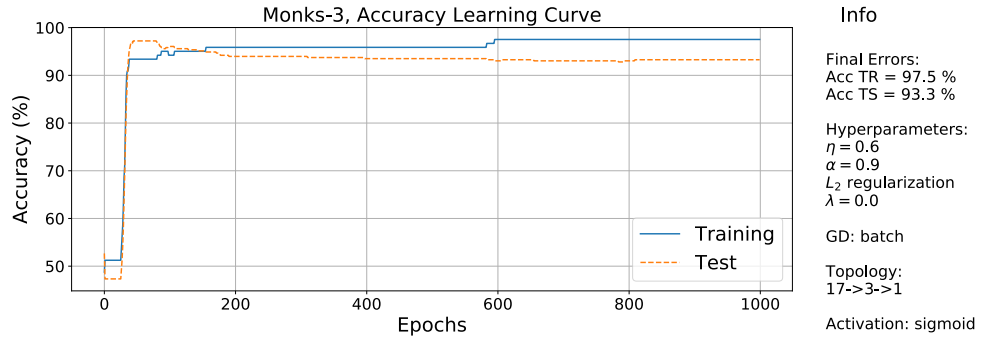


(b)

Figure 2: *Example of a final learning curve on MONKS 2.*

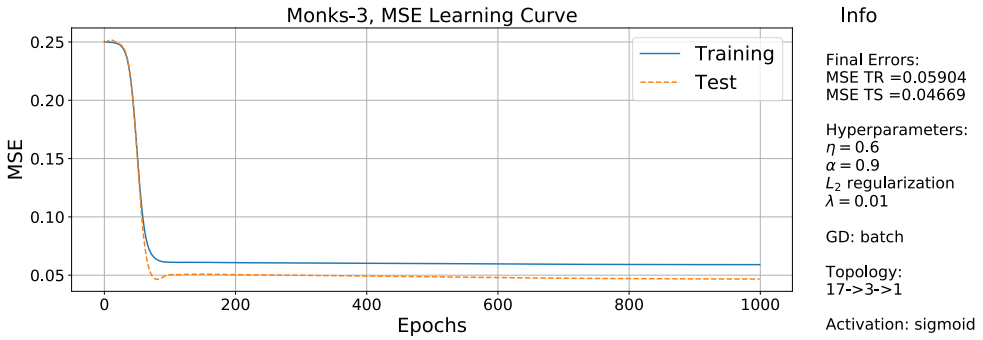


(a)

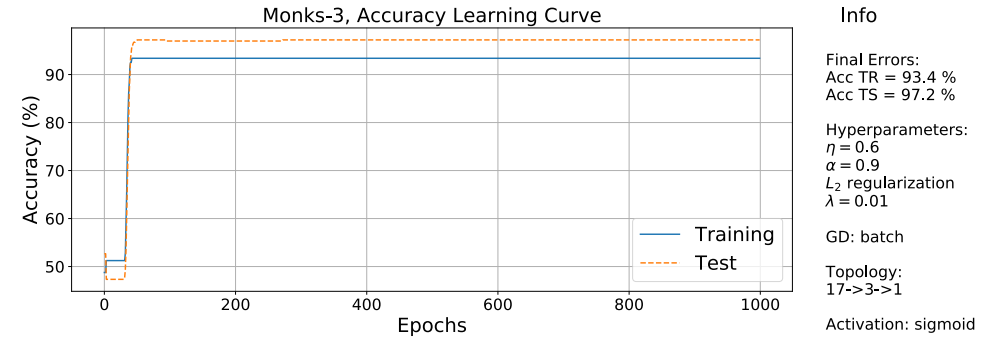


(b)

Figure 3: *Example of a final learning curve on MONKS 3 (without regularization).*



(a)



(b)

Figure 4: *Example of a final learning curve on MONKS 3 (with regularization).*

The validation schema for the Cup dataset it is summarized by the following steps.

1. The original training dataset has been splitted randomly in a *design set*, used for the model selection phase, and an internal *test set* used only for the final models assessment. We have chosen a small 5% sample size percentage for the test set (50 samples), because the main focus of the analysis is to perform a good model selection.

2. Preliminary 'babysitting'. The design set has been splitted into training and validation sets, with 70% and 30% percentages, in order to initially explore the behaviour of the neural network on the cup dataset, and to identify good parameter ranges. We tried to achieve overtraining by increasing the number of units and adjusting the learning rate and momentum, without using regularization approaches. We have chosen to use the *relu* activation function for the hidden layer, which is faster respect to the sigmoid, and more suitable for a regression task. We have observed that the learning time needed to achieve overtrainingwith the *batch* gradient descent approach, for a wide range of different number of units, requires at least 2000 epochs. We fixed the maximum amount of learning time to 3000 epochs.

3. Early Stopping testing. The 'batch' approach allows to obtain smoother learning curves respect to minibatch/online and permits to apply easily an early stopping technique. We tested two approaches from [3]. The GL_ϵ approach stops the learning process when the generalization loss exceed a defined threshold ϵ, given as a percentage with respect to the minimum current value of the loss. The PQ_ϵ approach is similar, but take into account also the progress of the training error. For different hyperparameters configurations we have computed the ratio between the MEE error obtained with the two early stopping criteria,using a threshold fixed to $\epsilon = 1\%$, and the minimum validation error of the overtrained learning curve, reported in Fig. 5. The early stopping methods take action only after a minimum number of epochs, fixed to 500, in order to avoid initial oscillation of the learning curves. An example of early stopping computation is represented in Fig. 7. The Figs. 5, 6 show that the $GL_{\epsilon=1\%}$ method is more reliable, because it computes stopping points at positions located after the validation minimum, and gives MEE values closer to the minimum respect to PQ.

4. Initial grid search. The main focus of this phase is to find a restricted range for the number of units in the hidden layer, which is the prominent hyperparameter regulating the model complexity. We applied the search on a random grid with the hyperparameters reported in Fig 2 using a 5-fold cross validation. In this phase we haven't used a regularization penalization, but we used the $PQ_{1\%}$ early stopping criteria. We observed that for a number of hidden units less than 10 the MEE are high. For each hyperparameter combination we computed the mean and the standard deviation over the 5 folds, represented in Fig. 10. The average error decreases fast over the first 25 hidden units, and at the same time its standard deviation increases. The two quantities remains quite stable after 25 units.

5. Finer grid search. Considering the bias-variance trade off dilemma, we restricted a finer grid search to an interval for hidden units between 15 and 55, the zone where the average error is starting stabilizing and the variance is not yet at its maximum, as shown in Fig. 10. Furthermore, instead of using the early stopping method we added a $l2$ regularization in the range reported in Tab. 2. We fixed the maximum number of epochs looking at the stopping epochs of the initial grid, taking as value 2000 epochs, sufficients to include all the best results of the initial grid.

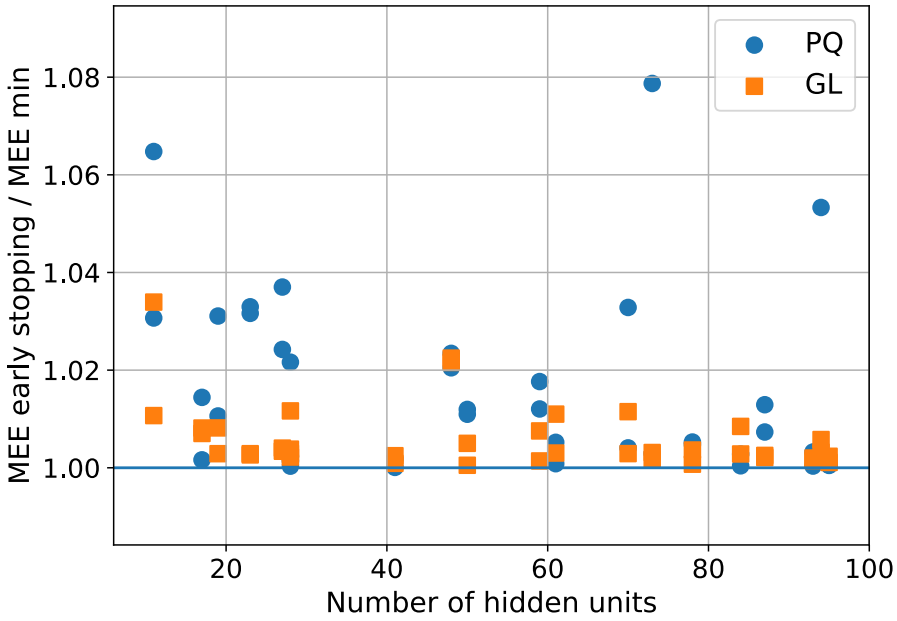


Figure 5: *Ratio between the MEE obtained with early stopping and the minimum validation MEE.*

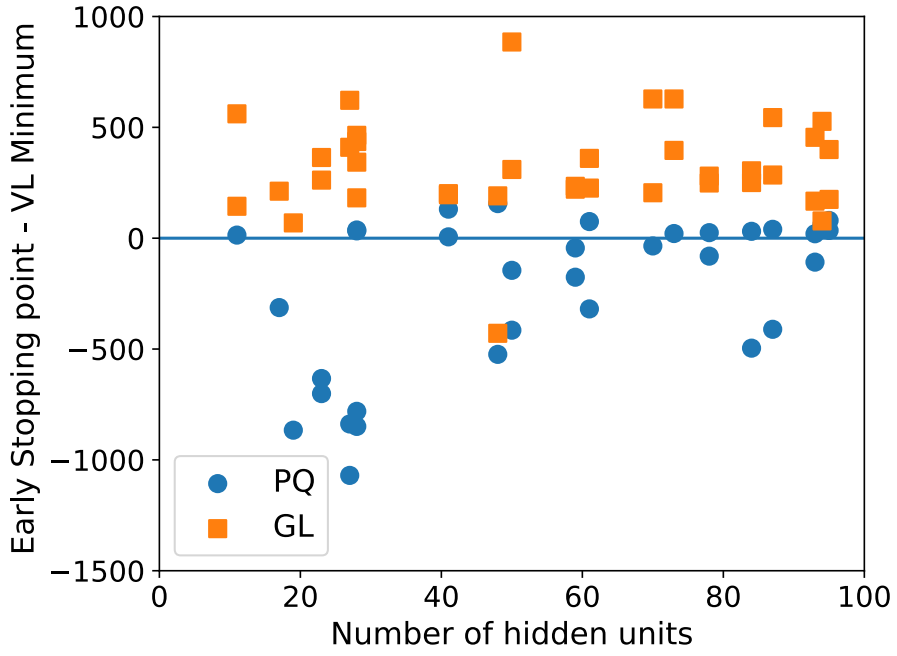


Figure 6: *Difference between the epochs stopping point and the minimum of the validation curve.*

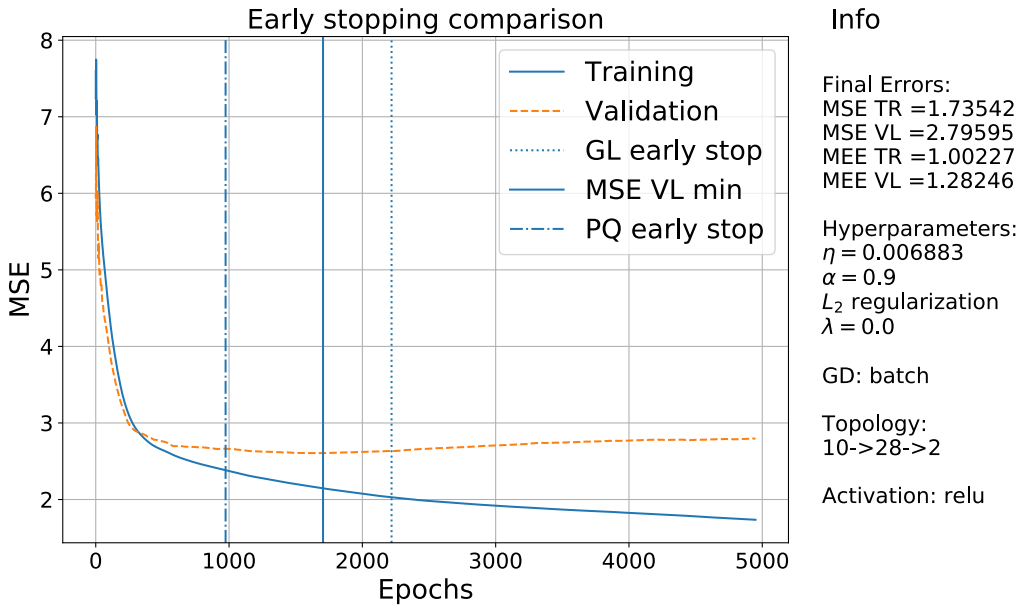


Figure 7: *Early stopping example. The vertical lines represent the early stopping points computed with $PQ_{1\%}$ and $GL_{1\%}$ methods and the point of minimum MSE on the validation curve.*

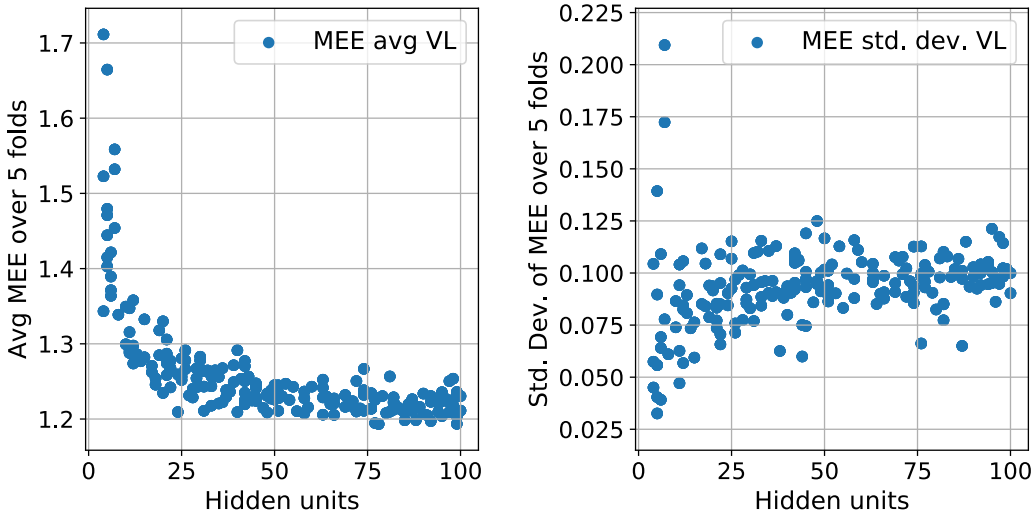


Figure 8: *Initial grid search results. On the left the average MEE for each hyperparameter combination over the 5 folds. On the right the MEE standard deviation.*

6. Final model selection. In Tab. 3 are reported the three best model, chosen by considering the minimum average MEE over the five fold, for the finer grid. The final three models has been re-trained on the whole design set, and assessed on the internal test set, held out before the beginning of the analysis. The re-training has been repeated for ten trials, the average MEE measure are reported in 3, with also the computing time estimated during the final re-training on an *Intel i5-7200U 2.5GhZ* CPU.

7. Cup Model. The model choosen for the cup it is Model 2. We have chosen this model because at almost equal performance on the validation 5-fold sets it has higher flexibility with respect to the other two models.

Hyperparameter	Initial Grid	Finer Grid
η	$[0.004, 0.02]$	$[0.004, 0.02]$
α	0.9	0.9
λ	0.0	$[0.0003 - 0.003]$
ϵ	1.0 %	-
hidden sizes	$[2, 100]$	$[15, 55]$
max epochs	3000	2000

Table 2: *Hyperparameters ranges for the random grid search algorithm.*

	MEE_{TS}	MEE_{TR}	MEE_{VL}	Hyperparameters	Training Time (s)
Model 1	1.145	1.110	1.208	$\eta = 0.018\, units = 20\, \lambda = 0.0014$	2.9
Model 2	1.085	1.062	1.202	$\eta = 0.020\, units = 43\, \lambda = 0.0018$	6.7
Model 3	1.098	1.084	1.216	$\eta = 0.019\, units = 36\, \lambda = 0.0020$	5.9

Table 3: *Best 3 models, with average errors on training, validation and test set and computing time on a single retraining on the* 95% *design set. All the three models use a momentum* $\alpha = 0.9$.

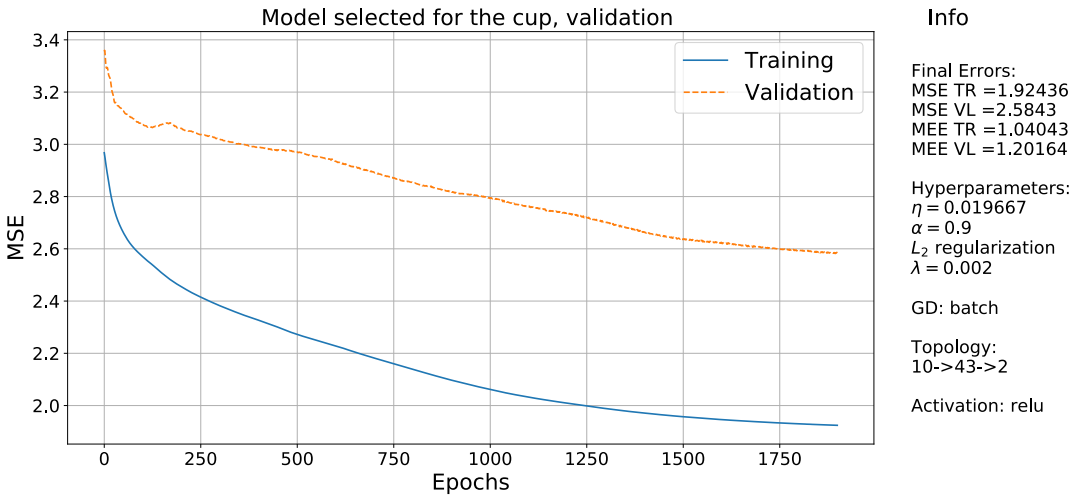


Figure 9: *Model 2, selected for the cup, training and validation curve.*

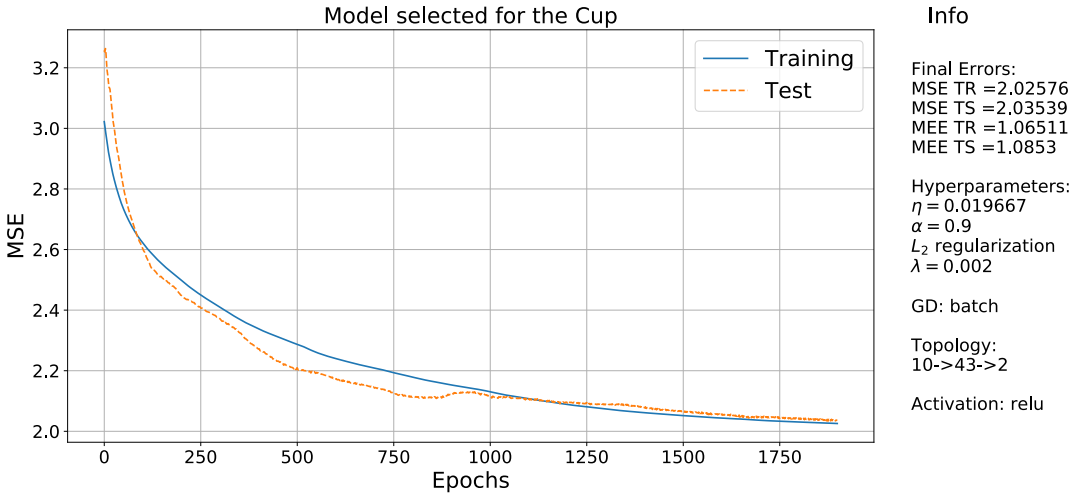


Figure 10: *Model 2, selected for the Cup, training and test curve.*

4. Conclusions

The test on the Monks dataset have permitted us to check the implementation of a simple artificial neural network, and to explore the effect of the hyperparameters on the learning curves.
The limited time and computing resources has pushed us to think more about reasonable choices or shortcuts to speed up the searching for a fine model selection for the Cup dataset, showing us how necessary is the integration between artificial and human intelligence.
The results of the Cup dataset are in the file carpricc_ML-CUP18-TS.csv, carpricc.

References

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*, MIT Press, 2016.

[2] James Bergstra and Yoshua Bengio. *Random Search for Hyper-parameter Optimization*, J. Mach. Learn. Res. 13, pp. 281-305, 2012.

[3] Prechelt Lutz. *Early Stopping - But When?*, Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg, 2012.

[4] Xavier Glorot and Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249-256, 2010.

Appendices

A. Preliminary trials' for MONKS dataset

In Table 4 you can see some interesting results discovered during the preliminary trials for the MONKS dataset.

Task	Topology	Batch size	Activation	η	α	λ	MSE (TR - TS)	Accuracy (TR - TS) (%)
MONK 1	17 -> 10 -> 1	1	sigmoid	0.1	0.9	0.0	$2e^{-05}$ - $3e^{-05}$	97 % - 94 %
MONK 1	17 -> 3 -> 1	10	sigmoid	0.8	0.9	0.0	0.00018 - 0.00026	100 % - 100 %
MONK 1	17 -> 3 -> 1	batch	relu	0.5	0.5	0.0	0.0 - 0.0	100 % - 100 %
MONK 2	17 -> 3 -> 1	1	sigmoid	0.1	0.5	0.0	0.00012 - 0.00013	100 % - 100 %
MONK 2	17 -> 3 -> 1	10	sigmoid	0.6	0.9	0.0	0.00026 - 0.00027	100 % - 100 %
MONK 2	17 -> 3 -> 1	10	relu	0.1	0.5	0.0	0.0 - $1e^{-05}$	100 % - 100 %
MONK 2	17 -> 10 -> 1	batch	relu	0.5	0.5	0.0	$3e^{-05}$ - $6e^{-05}$	100 % - 100 %
MONK 3 (no reg)	17 -> 3 -> 1	1	sigmoid	0.1	0.9	0.0	0.00073 - 0.07279	100 % - 91.4 %
MONK 3 (no reg)	17 -> 3 -> 1	10	sigmoid	0.5	0.9	0.0	0.0249 - 0.05952	98 % - 93.5 %
MONK 3 (reg)	17 -> 3 -> 1	10	sigmoid	0.4	0.9	0.001	0.01537 - 0.03656	99 % - 96.5 %
MONK 3 (reg)	17 -> 3 -> 1	batch	sigmoid	0.7	0.9	0.001	0.03826 - 0.05106	96 % - 93.7 %
MONK 3 (reg)	17 -> 3 -> 1	batch	relu	0.5	0.5	0.01	0.04034 - 0.03242	94 % - 96.5 %

Table 4: *Preliminary cases discovered during the analysis of the MONKS dataset.*