ML Project

Authors: *Emanuele Angilè, Silvio Martinico, Vittorio Mussin*
Master Degrees: Mathematics, Computer Science (Big Data Technologies),
Digital Humanities (Language Technologies)
e-mails: e.angile@studenti.unipi.it, s.martinico1@studenti.unipi.it, v.mussin@studenti.unipi.it
ML course (654AA), Academic Year: 2022/2023
Date: 06/01/2023
Type of project: **A**

Abstract

In this report we describe our implementation of a feed-forward Neural Network with few hidden layers, which we tested both for classification (on *Monk* datasets) and regression purposes. The implementation (in `Python`) includes grid-search/random-search and K-fold Cross Validation for hyperparameters search and validation. Ensemble techniques are adopted on prediction phase. Performance was evaluated through the tests reported.

1 Introduction

The goal of this project is to implement, from scratch, a Neural Network with the aim of solving both classification and regression problems.

In order to deal with different kinds of task (classification or regression) and datasets (different sizes, different number of features, different data) we follow a rigorous method to select the best hyperparameters configuration: we explore the hyperparameters space through a grid search and we validate them thanks to the *K-fold Cross Valitation*.

The performances of the model for classification task were evaluated by tests on three different Monk datasets [1]. Regarding the regression, the model was tested on the given MLCUP-22 dataset.

2 Method

We implemented -in `Python`- a multi-layer, fully connected, feed-forward Neural Network that, over multiple epochs, learns the weights of the connections between its neurons through *Stochastic Gradient Descent* (SGD) algorithm and we used *Backpropagation* to compute the gradient.

In order to improve the performances of our Neural Network, we defined some additional features that we will describe later in this section.

2.1 Implementation

The Neural Network is defined by the class `MLP`. Its attributes are:

- `architecture`: number of neurons for each layer
- `activation`: activation function of hidden layers
- `weights_std`: initial weight's standard deviation
- `task`: classification or regression
- `xavier`: boolean variable for Xavier's weights initialization

We initialized the weights using two strategies:

- if `xavier = False`, we sampled the weights from a Gaussian Distribution $\mathcal{N}(0, \frac{1}{3})$ with a mean equal to 0 and a variance equal to $\frac{1}{3}$;
- if `xavier = True`, then, in a layer with n input units and m output units, we sampled the weights from a Uniform Distribution on the interval $\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right]$ as suggested by Glorot and Bengio.[2]

We set the bias terms of each units to 0. In the hidden layers, the possible activation functions are ReLu and sigmoid (specified by a parameter a), whereas for the output layer, we always used the sigmoid for classification and the identity for regression.

We used the SGD algorithm to perform the learning phase, with the mini-batch size determined by the parameter mb, and the learning rate determined by the parameter η. Additionally, we included the possibility of using a learning rate decay (specified by parameters $(\eta_0, \tau, \eta_\tau)$), determined by the boolean parameter `eta_decay`. In the algorithm, we incorporated momentum, specified by the parameter α, which adds a re-scaled portion of the gradient from the previous epoch.

We exploited the SGD algorithm to minimize the Mean Squared Error (MSE) cost function for classification task, or the Mean Euclidean Error (MEE) cost function for regression task.

Regarding regularization, we implemented *Tikhonov Regularization*, specified by the parameter λ, by adding the Euclidean norm of the vector of all weight and bias terms to the cost functions.

As stop condition for training we used the convergence of the cost function (or accuracy measure) over the epochs - implemented in the method `training`.

The validation was implemented in the function `cross_validation` through *K-fold Cross Validation*, together with the *grid-search* and *random search*, for the exploration of the hyperparameters' space.

For the delivery of the final model we take an ensemble over the best models, implemented in the function `ensemble`. The latter performs a double ensemble: it first takes the 5 best hyperparameters combinations obtained through the CV and, after training, it compute their average result; then, it repeats this first step five times and, in the classification case, it takes a vote, while it again compute the average in the case of regression. In this way, aside from averaging the best models, we avoid to get a bad result caused by an unlucky weights initialization.

2.1.1 Libraries

The following are the main libraries we exploited in the development of the project:

- `NumPy` to optimize matrix operations;
- `Pandas` to operate on datasets;
- `MatPlotLib` for visualization;
- `concurrent.futures` to parallelize the grid search.
- `sklearn` for preprocessing

2.2 Validation

The Validation was implemented through a 5-fold Cross Validation Schema and it was exploited, together with a grid-search, to select the best hyperparameters. This procedure consists in splitting the training dataset set into five parts (*folds*). After that, we used each time a different one of these five folds as validation set, while the remaining four folds were used as training set. Then, we trained the model with a specific configuration of hyperparameters on the training set and evaluated it on the validation set (this was done a specified number of times `cross_times` in order to avoid unlucky initializations).

After this process, we selected the five best models according to a score we computed: in the classification task, this score was given by the sum of the squares of mean and variance (among 5 folds and for `cross_times` times) of the MSE on validation set; in the regression task, the score was calculated as the sum of the squares of the mean and variance of the mean euclidean error (MEE) obtained on the validation set. It's important to note that a lower score indicates a better model.

The 5 best models were then ensembled, with a double ensemble, as we mentioned before in Section 2.1.

Regarding the ML-CUP-22 dataset, since we did not have a test set for assessing the generalization capabilities of our model, it was split into two parts: the development set (70%) and the internal test set (30%). The development set was used for the model selection, while the internal test set was used to evaluate the generalization capabilities of the chosen models on unseen data.

2.3 Preprocessing

For the ML-CUP-22 dataset, in order to hold off the magnitude of weights through multiple epochs, we performed a normalization.

Since the columns showed already a mean and variance near respectively to 0 and 1 while values were not normalized to stay in a unitary cube, we applied the Min-Max Scaler from `sklearn` to the input columns.

3 Experiments

3.1 Monk Results

The input was formatted through *one-hot encoding*; this resulted in 17 input neurons and one output neuron. All the Monk experiments were performed with a single hidden layer of just 3 neurons. Thus, the attribute `architecture` of the model was set to $(17, 3, 1)$.

Through the validation process, we found for each dataset the 5 best hyperparameters' combinations and then we ensembled them. In Table 1 we show, for each of the three dataset (Monk3 was tested both with and without regularization), the best hyperparameters we have found for the ensemble. See Appendix A for the exact combinations of hyperparameters. The maximum number of epochs was set to 600 but in many cases we had less epochs thanks to stopping criteria.

In Table 2 we can see the accuracies we obtained on each dataset, both on training set and test set.

In Figure 1, Figure 2, Figure 3 and Figure 4 we can see the learning curves and accuracies related to Monk datasets. In each plot the curves represent the average (accuracy or MSE) obtained by the five models corresponding to the five best hyperparameters combinations, each one with five different weights initialization, for a total of 25 models. The shadows represent the standard deviations, whereas the number of epochs was extended to the max number reached in the ensemble for the models that escaped the training earlier thanks to stopping criteria.

	MONK1	**MONK2**	**MONK3**	**MONK3 + reg.**
Architecture	$(17, 3, 1)$	$(17, 3, 1)$	$(17, 3, 1)$	$(17, 3, 1)$
Activation	ReLu	ReLu	ReLu, sigmoid	ReLu, sigmoid
η	0.7, 1	0.5, 0.75, 1	0.5, 0.75	0.5, 0.75, 1
λ	10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}	10^{-6}, 10^{-5}	10^{-7}, 10^{-6}	0
α	0.01, 0.05, 0.1, 0.15, 0.2	0.15, 0.1	0.05, 0.10, 0.15	0.05, 0.1
mb	25	25, 50	25, 50, 100	25, 50, 100
epochs	500	600	600	600
Xavier	/	/	False, True	False, True

Table 1: Best hyperparameters for Monk datasets

Task	**Accuracy (TR/TS)**
MONK1	100% / 100%
MONK2	100% / 100%
MONK3	94.26% / 96.3%
MONK3 + reg	94.26% / 96.76%

Table 2: Results on Monk datasets

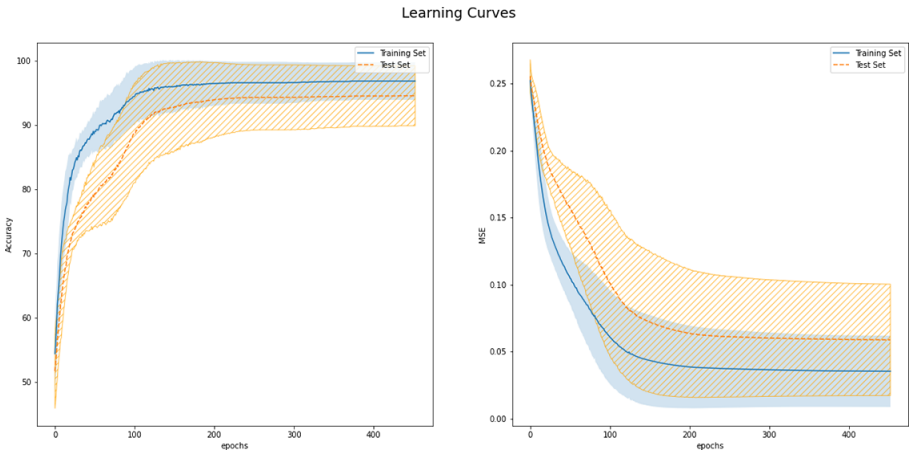

Figure 1: Accuracy and MSE for Monk1 dataset

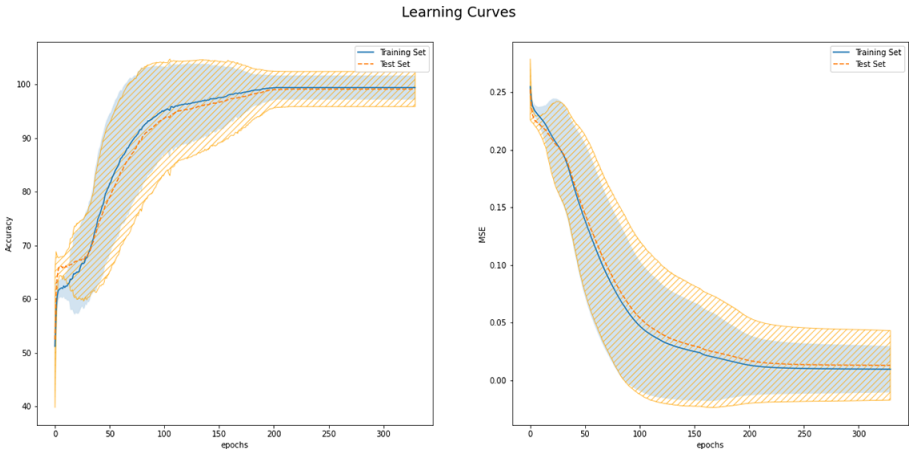

Figure 2: Accuracy and MSE for Monk2 dataset

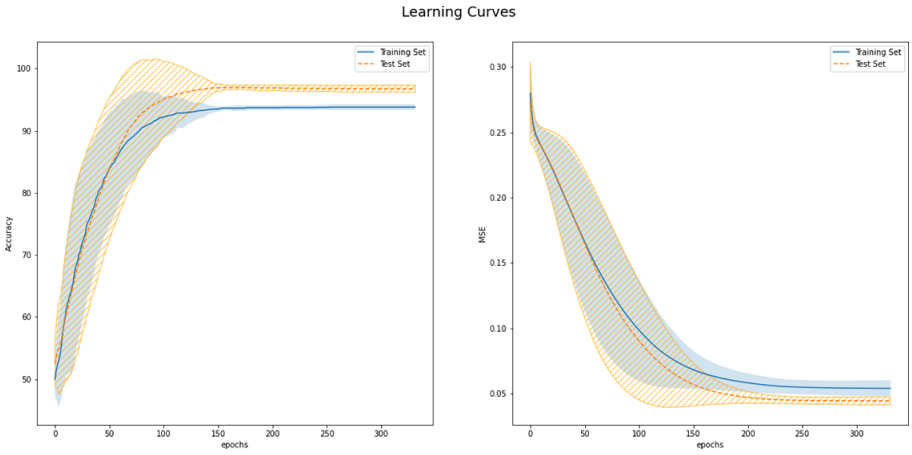

Figure 3: Accuracy and MSE for Monk3 dataset **with** regularization

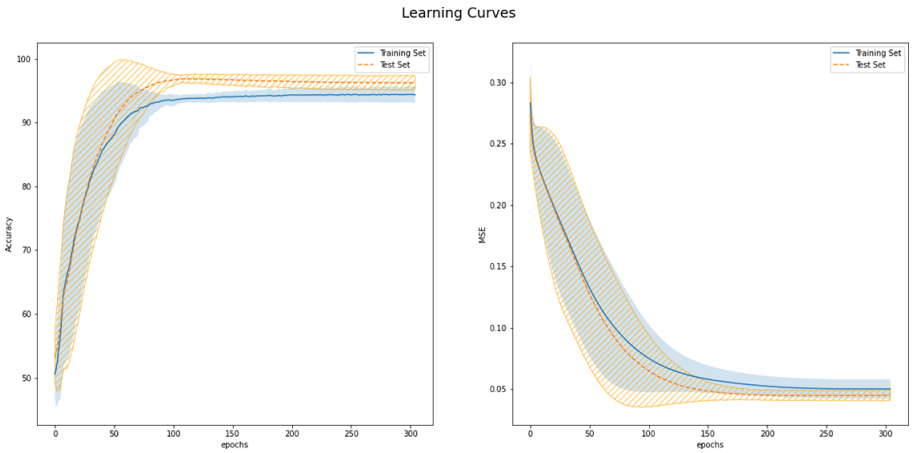

Figure 4: Accuracy and MSE for Monk3 dataset **without** regularization

3.2 Cup Results

We split the ML-CUP22-TR dataset into two parts of 70% and 30% of the dataset, respectively development set and (internal) test set, for the ensemble assessment. As validation schema - for candidates in the ensemble - we used 5-fold Cross-Validation, with `cross_times`=3 (number of iterations for each combination of hyperparameters).

To identify the most promising ranges and combinations of hyperparameters, the development set was first divided into two subparts, with 80% of the data being used as the training set and 20% of the data being used as the test set. This made it possible to evaluate the performance of different architectures and identify which ones are interesting and which ones are not. In fact, we manually attempted to train with various architectures and hyperparameters related to the training algorithm, and we plotted learning curves and results of the MEE to understand where to focus our search for optimal hyperparameters.

Once the most interesting ranges of hyperparameters have been identified, we observed that the percentage of good-performing hyperparameters combinations was not negligible. Then, we used a random search for a faster exploration. The random search we implemented draws a random sample (specified by the parameter `random_size`) from the overall hyperparameters grid, so that we can explore bigger grids while the computation time does not blow up.

The grid contained approximately 11000 combination of hyperparameters, but we decided to set the `random_size` equal to 200.

In Table 3 we can see the explored ranges of hyperparameters, while Table 4 and Table 5 contain the best-performing hyperparameters. For a more detailed description of the best results of the grid search see Appendix B.

Architecture	$(9, 10, 20, 2)$, $(9, 20, 10, 2)$, $(9, 15, 15, 2)$
Activation	sigmoid
a (sigmoid)	$[0.8, 1.2]$
η	$[0.1, 0.3]$
λ	$[10^{-6}, 10^{-4}]$
α	$[0.1, 0.3]$
mb	$[50, 170]$
Xavier	True - False
η decay	True - False
τ (η decay)	$[25, 75]$
epochs	$[180, 300]$

Table 3: Ranges of explored hyper-parameters in random search

η	λ	α	mb	η decay	τ (η decay)	**epochs**
0.1, 0.2	10^{-6}, 10^{-5}, 10^{-4}	0.1 0.2	100, 130	True, False	50, 75	180

Table 4: Best hyper-parameters (training)

Architecture	**Activation**	a (sigmoid)	**Xavier**
$(9, 10, 20, 2)$, $(9, 20, 10, 2)$, $(9, 15, 15, 2)$	sigmoid	1, 1.2	True - False

Table 5: Best hyper-parameters (network)

From the random search we selected the best-performing hyperparameter combinations for the ensemble, and we evaluated the ensemble valuing both the MEE and the shape of the learning curve, aiming at smooth curves. We tested multiple ensemble models on the internal test set and we ultimately chose the one that provided the best results so far. The hyperparameters are in Table 6, while the resulting MEEs are in Table 7 and the learning curve in Figure 5.

Finally, we trained this final model on the entire training set that we initially split and we obtained an MEE equal to 1.4634. We then used it to compute the output on the blind test set.

Architecture	**Xavier**	α	λ	mb	η	epochs	η **decay**	τ	a
$(9, 15, 15, 2)$	False	0.1	10^{-6}	100	0.1	210	True	75	1.2
$(9, 15, 15, 2)$	False	0.1	10^{-6}	130	0.2	210	True	75	1.2
$(9, 20, 10, 2)$	False	0.1	10^{-6}	100	0.2	210	True	50	1.2
$(9, 15, 15, 2)$	False	0.1	10^{-6}	130	0.2	210	False	/	1.2
$(9, 20, 10, 2)$	True	0.1	10^{-5}	100	0.2	210	False	/	1.2

Table 6: Final hyper-parameter combinations for the ensemble

MEE (internal TS)	MEE (TR)
1.5523	1.4776

Table 7: Results for (internal) test set and training set

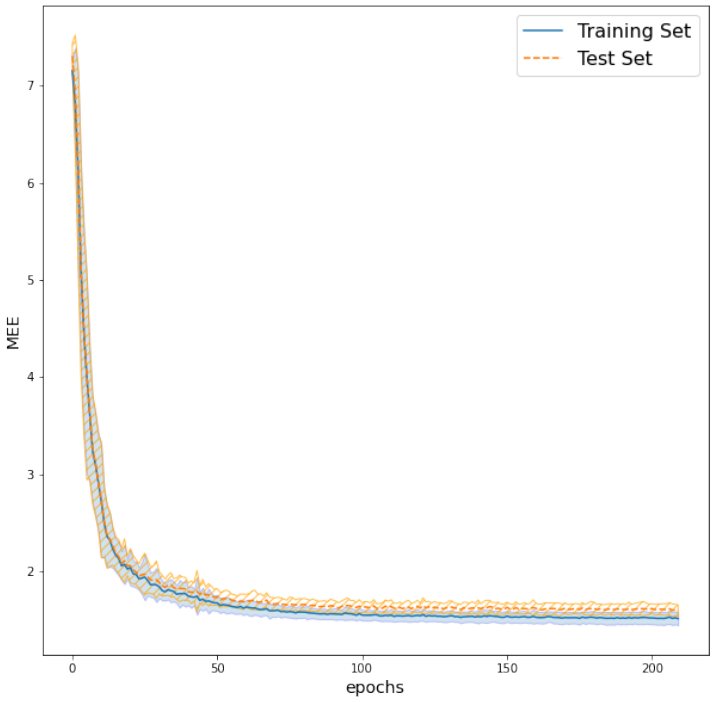

Figure 5: MEE for (internal) test set and training set

3.3 Discussion

When we began our manual attempts, we immediately noticed that the ReLU activation function had issues with the magnitude of the gradient, while the sigmoid function did not. The cause could be found in the first iterations of the SGD algorithm. In fact, we observed that the outputs of the newly initialized neural network were in a neighbourhood of 0 and the labels of the dataset had instead absolute values near 20 or 30. This implied that the differences between the outputs and labels that appear in the backpropagation algorithm formula had a large value, which shifts the weights away from having a mean of 0, especially in the early training epochs. In some sense, it was as if we were not correctly initializing the weights.

Then, we wondered why sigmoid was so well-behaved while ReLU was not. The answer was in the derivatives of the two functions: the ReLU derivative has values of 0 or 1 and it did not help to decrease the magnitude of the difference. Sigmoid, on the other hand, with $a = 1$ has a derivative less than or equal to $\frac{1}{4}$ and this term helped to slow down the increase of the gradient in each hidden layer. In conclusion, we found that using a small learning rate (≈ 0.001) was necessary when using the ReLU activation function. However, this would have slowed

convergence, so we decided to only investigate the sigmoid case in the final cross-validation.

We also started to investigate the optimal size of the mini-batch to use. After initial experiments, we immediately noticed that a size of $10 - 25\%$ of the dataset results in a faster convergence, while using a size of $50 - 70\%$ required three times as many epochs as the smaller mini-batch to achieve the same result. This also helped us to save computing time during the cross-validation and, in general, during the training phases.

Another issue we faced was the lack of smoothness in the learning curves. To address this we selected lower ranges for η and α *momentum* and we introduced an `eta_decay` to gradually decrease the initial learning rate, without negatively impacting the MEE scores.

After the cross-validation, we noticed that the worst results were given by the choice of high values (0.25-0.3) of η. Despite using the sigmoid function, an `eta` value too close to 0.3 (and above) was difficult to manage during the initial epochs, as previously mentioned.

What surprised us was instead the almost complete absence of overfitting cases. We believe that the decision to use relatively simple architectures, with only two hidden layers, is a major contributing factor to this behavior. However, it is likely that the implementation of stopping conditions and Tikhonov regularization in the training algorithm played a crucial role as well.

4 Conclusion

Thanks to this project we now have a more concrete comprehension of the functioning of a Neural Network and of the the importance of its various features. Furthermore, we understood the importance of both model selection and model assessment and the links between various hyperparameters. Finally, we were able to appreciate the importance of working in a team where each member has a different background and makes their own personal contribution.

The result of the blind test is stored in the file `EnSeMble_ML-CUP22-TS.csv`.

Acknowledgments

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

References

[1] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[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.

Appendix A - Monk

MSE (VL)	**std (VL)**	**Acc. (TR)**	**Activation**	η	λ	α	mb	**Xavier**
0.0479	0.048	97.5%	ReLu	1	10^{-5}	0.05	25	/
0.0542	0.0529	97.4%	ReLu	1	10^{-7}	0.2	25	/
0.1187	0.0548	95.3%	ReLu	1	10^{-7}	0.15	25	/
0.0417	0.0559	98.45%	ReLu	0.7	10^{-4}	0.01	25	/
0.0875	0.0573	96.25%	ReLu	1	10^{-6}	0.1	25	/

Table 8: Best hyperparameters combinations for MONK1

MSE (VL)	**std (VL)**	**Acc. (TR)**	**Activation**	η	λ	α	mb	**Xavier**
0	0	100%	ReLu	0.5	10^{-6}	0.15	25	/
0.0015	0.0066	100%	ReLu	1	10^{-6}	0.15	25	/
0	0	100%	ReLu	1	10^{-5}	0.1	25	/
0.0015	0.0066	100%	ReLu	1	10^{-5}	0.1	50	/
0.0015	0.0066	100%	ReLu	0.75	10^{-6}	0.1	25	/

Table 9: Best hyperparameters combinations for MONK2

MSE (VL)	**std (VL)**	**Acc. (TR)**	**Activation**	η	λ	α	mb	**Xavier**
0.0771	0.0303	94%	sigmoid	0.5	10^{-7}	0.1	25	True
0.0792	0.0525	94.6%	ReLu	0.75	10^{-6}	0.05	100	False
0.0854	0.0308	94.8%	ReLu	0.75	10^{-6}	0.1	100	False
0.0688	0.0422	94.6%	sigmoid	0.75	10^{-6}	0.15	25	True
0.0667	0.0464	93.5%	sigmoid	0.5	10^{-6}	0.15	50	False

Table 10: Best hyperparameters combinations for MONK3

MSE (VL)	**std (VL)**	**Acc. (TR)**	**Activation**	η	λ	α	mb	**Xavier**
0.0792	0.032	94.9%	ReLu	1	0	0.05	100	False
0.0729	0.032	94.5%	sigmoid	0.75	0	0.1	25	True
0.0979	0.033	94.4%	sigmoid	0.5	0	0.05	25	False
0.0896	0.033	96.5%	ReLu	1	0	0.1	50	True
0.0792	0.037	94.2%	ReLu	0.5	0	0.1	100	False

Table 11: Best hyperparameters combinations for MONK3 without regularization

Appendix B - ML-CUP

MEE (VL)	**std (VL)**	**MEE (TR)**	**Architecture**	**Xavier**	α	λ	mb	η	**epochs**	η **decay**	τ	a (f_σ)
1.59	0.09	1.56	[9, 15, 15, 2]	False	0.1	1e-06	100	0.1	180	True	75	1.2
1.58	0.09	1.53	[9, 15, 15, 2]	False	0.1	1e-06	130	0.2	180	True	75	1.2
1.58	0.07	1.55	[9, 20, 10, 2]	False	0.1	1e-06	100	0.2	180	True	50	1.2
1.55	0.07	1.5	[9, 15, 15, 2]	False	0.1	1e-06	130	0.2	180	False	/	1.2
1.54	0.07	1.5	[9, 20, 10, 2]	True	0.1	1e-05	100	0.2	180	False	/	1.2
1.59	0.07	1.55	[9, 20, 10, 2]	False	0.1	1e-05	100	0.1	180	False	/	1
1.57	0.06	1.66	[9, 20, 10, 2]	False	0.2	0.0001	100	0.2	180	False	/	0.8
1.56	0.08	1.52	[9, 10, 20, 2]	False	0.2	1e-05	100	0.1	180	False	/	1.2

Table 12: Best grid search results for ML-CUP, note that every model uses the sigmoid as activation function.