MACHINE LEARNING PROJECT
A.A 2023/2024
14/02/2024

- Gian Marco Gori, Francesco Luigi Moretti, Silvia Sonnoli
- Team Name: AMonk Us
- MSc in Data Analysis for Experimental Physics, MSc in Complex Systems
- Email: g.gori21@studenti.unipi.it, f.moretti14@studenti.unipi.it, s.sonnoli@studenti.unipi.it

- Project type A

OBJECTIVES

- We tried to perform some classification and regression tasks on MONK and ML-CUP datasets respectively.
- To do so, we implemented a Multilayer Perceptron, trained by gradient descent algorithm using backpropagation.
- We explored several configurations of hyperparameters through grid search, assessing their quality through MSE for MONK and MEE for ML-CUP; each configuration has been validated via 5-fold cross validation.

CONTRIBUTIONS

- The implementation has been made in the Python language, using Numpy, Matplotlib, Pandas and Seaborn libraries as a support.

- Layer class is intended to perform forward and backward propagation recursively. It also allows to set and get weights and biases.
  Vectorization in classes operations allows to run trials faster and in a more efficient way.
  Input class inherits from the Layer one and has special methods.

- NeuralNetwork class build the network given input and output layers. It performs training allowing to :
  - choose between online/mini-batch/batch,
  - set hyperparameters like η, λ and α,
  - decide if utilize Nesterov momentum, L1 regularization or Adam.

- grid_search and cross_validation functions allows to perform a k-fold grid search to find the best models.

CONTRIBUTIONS

- We tried different layer structures, varying both width and number of layers, as well as activation functions like sigmoid, hyperbolic tangent, Relu and leaky Relu.
- For the initialization of the weights we have decided to sample randomly the values from a uniform distribution between [-0.5, 0.5].
- We experimented early stopping, though we discarded it because we did not find significant differences in the results.
- We used Jupyter notebooks to study hyperparameters importance and produce significative charts.

MODEL NOVELTIES

We implemented the following novelties:

- Nesterov Momentum: we added the possibility to use Nesterov momentum approach. We have seen an improvement of the convergence speed and stability.
- Optimizer: we implemented the Adam optimizer; again we noticed a faster convergence and an improved stability.
- L1 regularization: we tried the Lasso approach in addition to L2 regularization; in this case we haven't found significant improvements.
- Normalization and standardization: we tried to apply these preprocessing methods; again we haven't found better results than without them.

MONKS SETTINGS

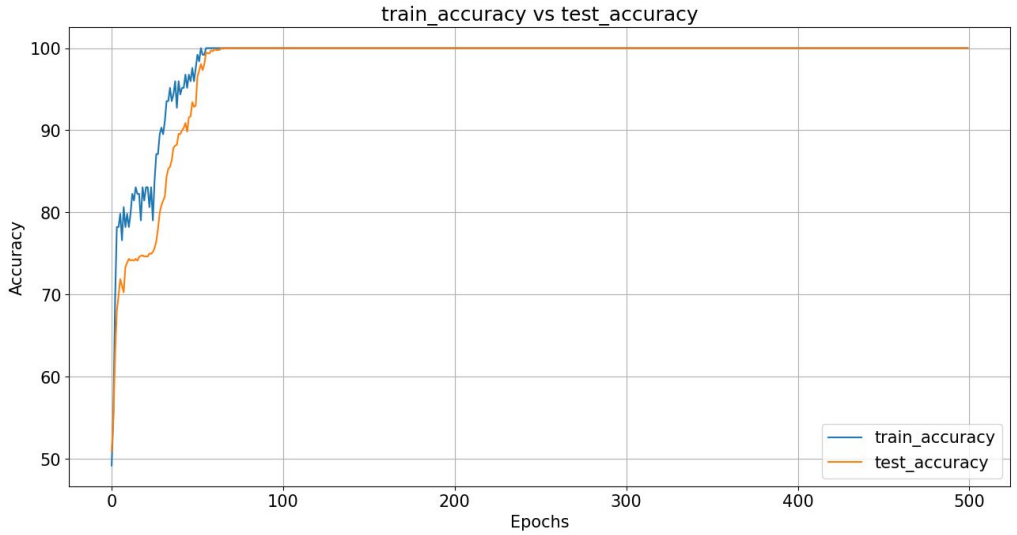
- We applied one hot encoding to the input data of all the MONK problems.
- For the hidden layer activation function we tried both Relu and sigmoidal functions, with the latter performing better.
- On the other hand, for the output layer we first used softmax activation function but, for binary classification, it requires two units in the output layer hence a major and useless complexity. Instead we preferred once again the sigmoidal function that similarly squashes the results between 0 and 1 so that we can think the output as a probability.
- Speaking of the loss function, we decided to use the Binary Cross Entropy instead of a more standard LSE, as the plateaus in the training criterion are less present with the log-likelihood loss function[1].

MONK RESULTS

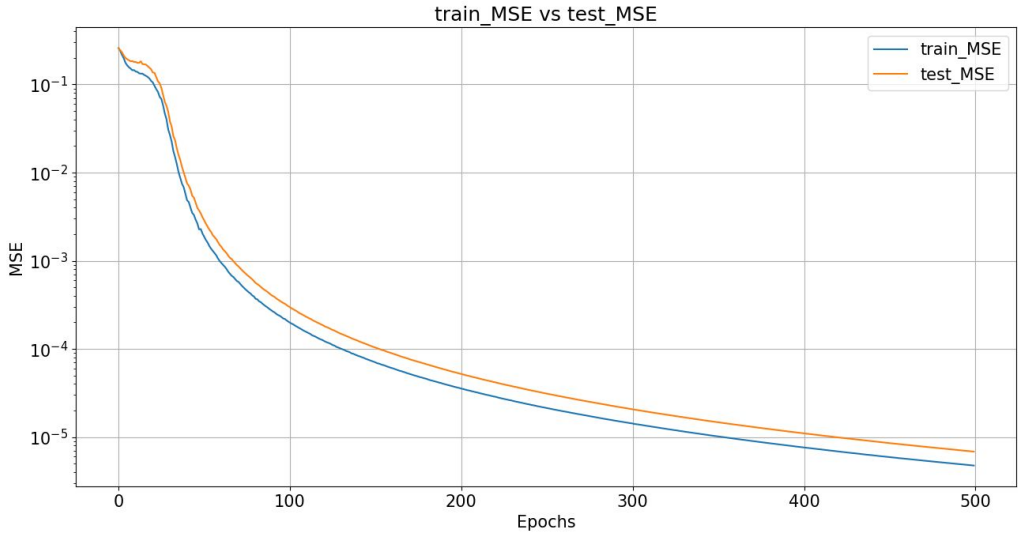
TASK:	#UNITS,NBATCH,ETA,LAMBDA, ALPHA:	MSE(TR/TS):	ACCURACY(TR/TS) (%) :
MONK1	4 sigmoidal units Online algorithm $\eta = 0.05$, $\alpha = 0.6$, $\lambda = 0$	TR: 4.24e-04 TS: 6.47e-04	TR: 100 TS: 100
MONK2	4 sigmoidal units Online algorithm $\eta = 0.05$, $\alpha = 0.6$, $\lambda = 0$	TR: 4.72e-05 TS: 5.32e-05	TR: 100 TS: 100
MONK3	4 sigmoidal units Batch algorithm $\eta = 0.3313$, $\alpha = 0.5$, $\lambda = 0$	TR: 3.99e-02 TS: 3.01e-02	TR: 94.26 TS: 96.53
MONK3 (reg.)	3 sigmoidal units Batch algorithm $\eta = 0.5522$, $\alpha = 0.4$, $\lambda = 0.001$	TR: 4.18e-02 TS: 3.03e-02	TR: 93.44 TS: 96.99

MONK RESULTS

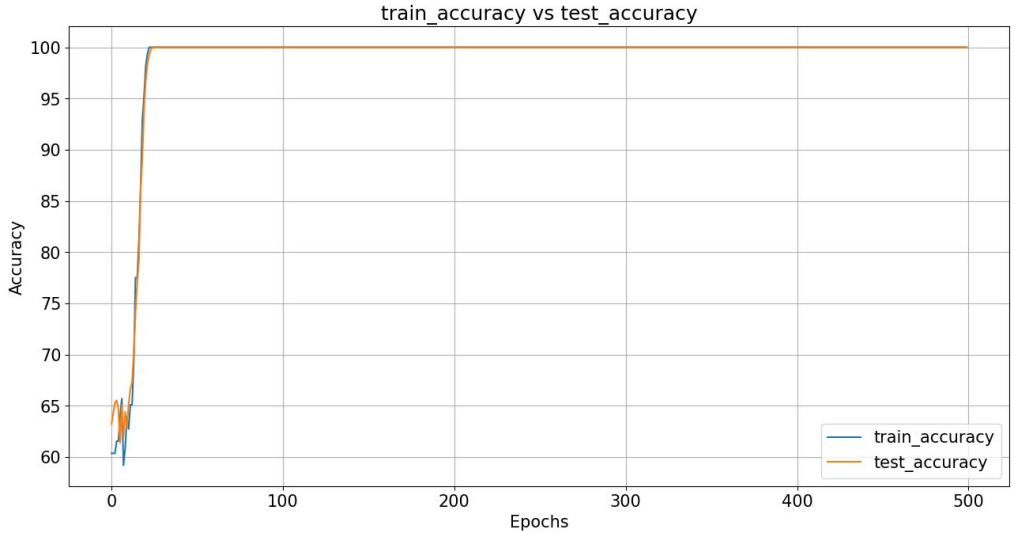
Monk1: Accuracy



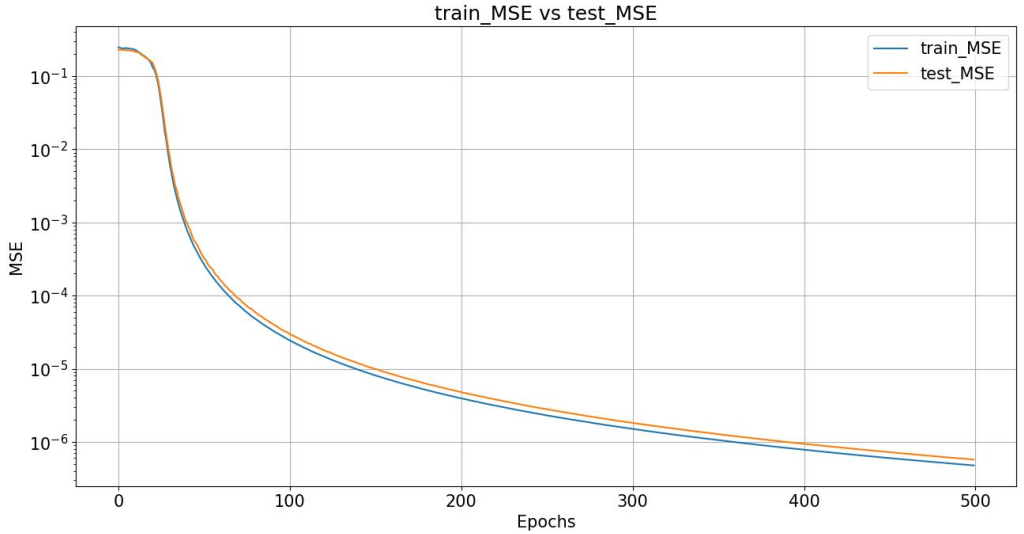
Monk1: MSE



Monk2: Accuracy

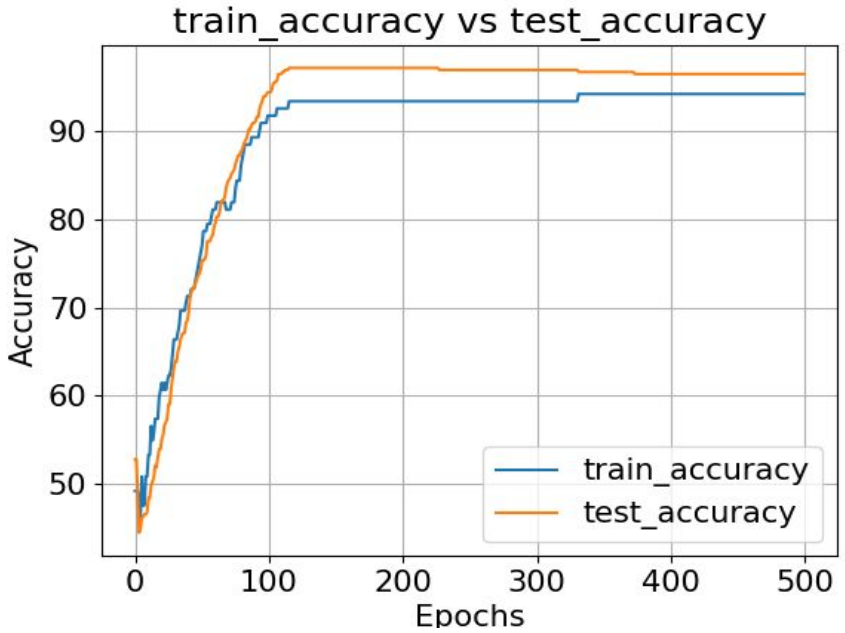


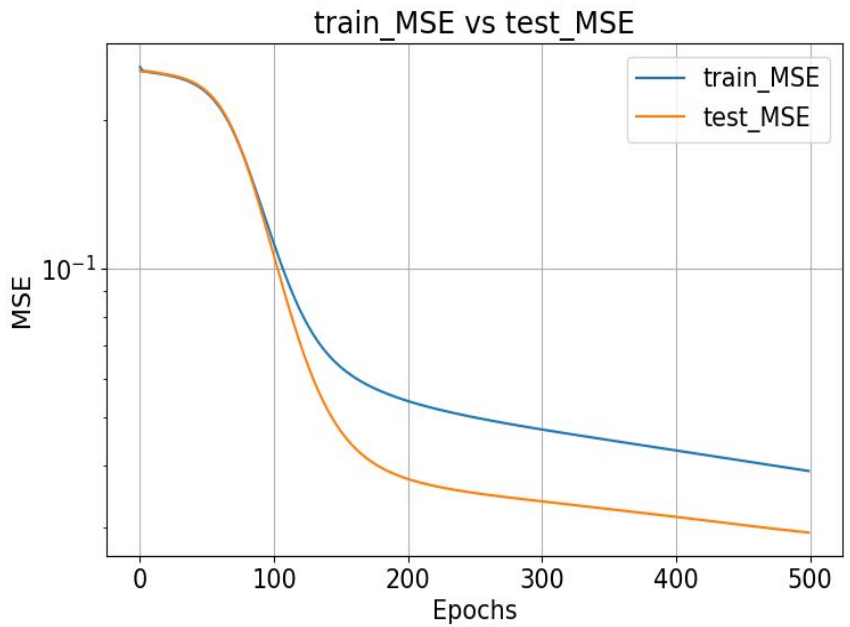
Monk2: MSE



MONK RESULTS

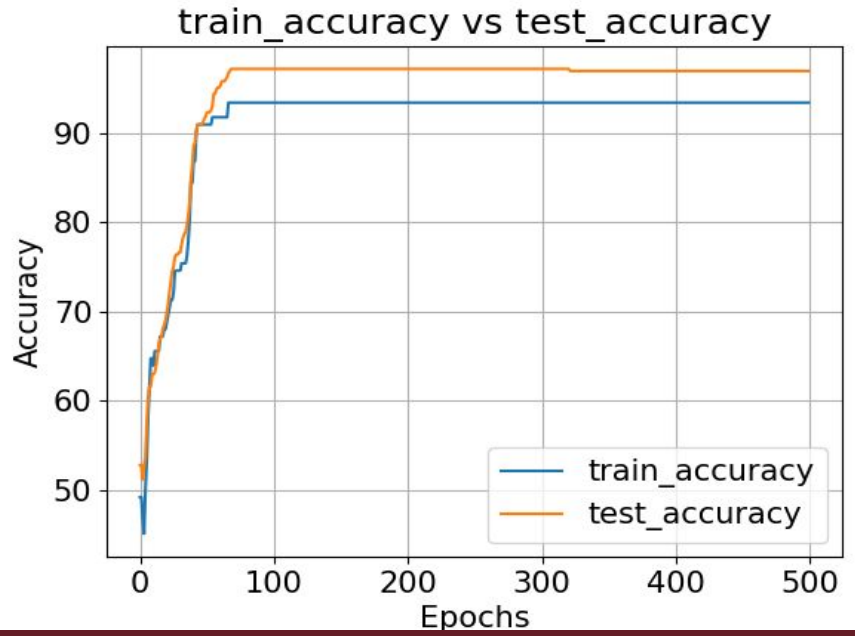
Monk3 :
Accuracy and MSE

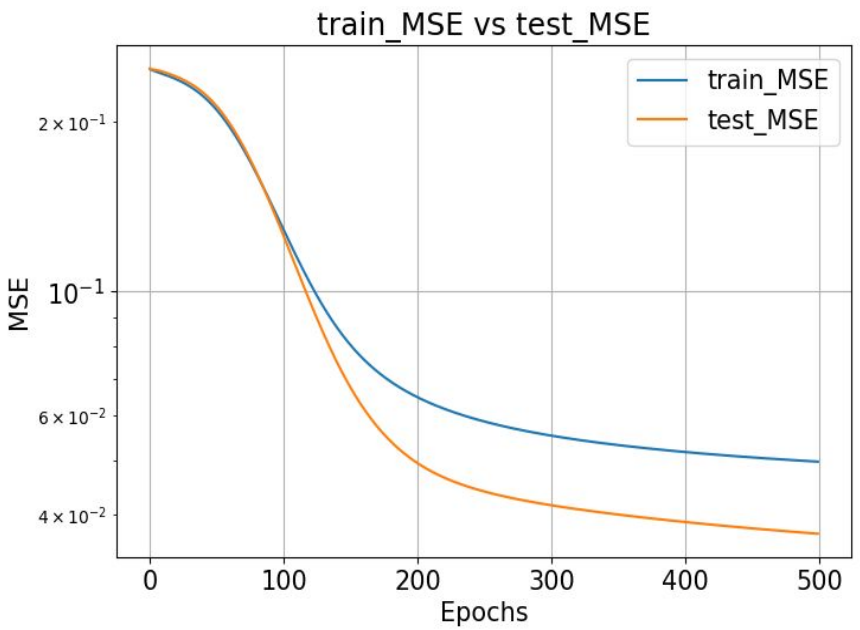




Monk3 with regularization :

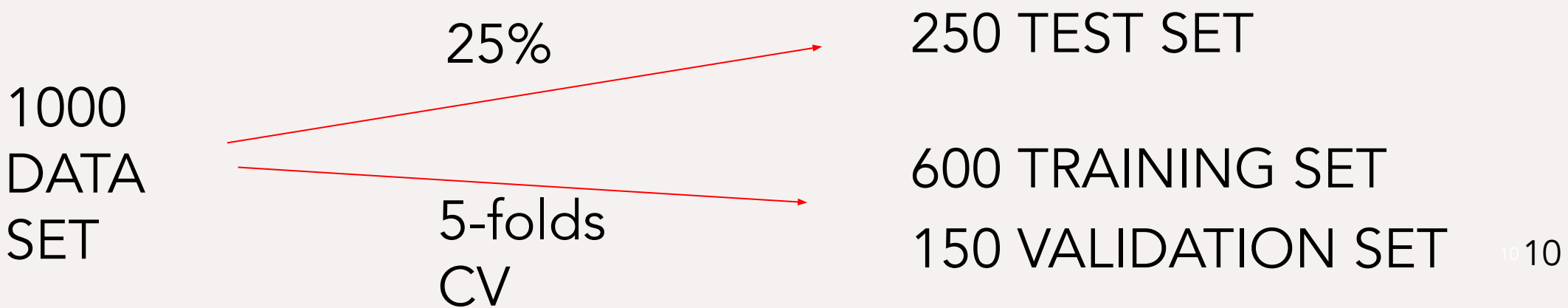
Accuracy and MSE





CUP VALIDATION SCHEMA: DATA SPLITTING

- As regards model selection and assessment, we both implemented hold-out model assessment, with a percentage of 25% for test set, and a k-fold cross validation for training and validation data set, with k = 5. After that we proceeded to have a final retraining with the best combination of hyperparameters.



CUP VALIDATION SCHEMA: MODEL SELECTION

- To start off, we performed our grid search on coarser-grained intervals of the various hyperparameters, like: $\eta \in$ [1e-04, 0.1], $\alpha \in$ [0.4, 0.9], $\lambda \in$ [1e-06, 0], eta_scale_batch-size = [None, linear, square root].
- Once found some best values for this first round, we then applied another grid search restricting our investigation around finer-grained neighbourhoods of such values. The resulting best configuration was then selected for the next phase of model assessment.
- After trying both minibatch and batch in a preliminary phase, we decided to use minibatch; we didn't use online algorithm as it was too unstable and produced poor quality plots (poor smoothness and local minima).
- We trained our models for 500 epochs, as it was a good compromise between the training "goodness" and the computational cost.
- It required about ~40 minutes for each configuration.

CUP RESULTS (EXPERIMENT)

- We experimented different kinds of architectures: at the beginning deeper models with less units, like for example 5-6 layers with 30 units each, were tried, but no significant improvements were reached.
- So we tried shallower models, but with more units like, 2-3 layers with 40-50-70 units each, finding better numerical results and less overfitting.
- Different kinds of activation functions were used: ReLu, Leaky Relu, ELU, but we have found that the classic Tanh activation was the best both in term of speed convergence (tanh as a bigger derivative so the learning is faster), both in stability.

12

Architecture ([act. func., #units]	Hyperparameters: η, λ, α, BatchSize, Nesterov, Optimizer	MEE Mean Std.Dev.
1 layer of 200 tanh units	η = 0.001, λ = 1e-06, α = 0.6 BatchSize= 16, Nesterov = True, Optimizer = True	TR = 0.64 ± 0.04 VL = 0.95 ± 0.01
1 layer of 25 leaky relu units 1 layer of 30 leaky relu units 1 layer of 10 leaky relu units	η = 0.009, λ = 0.1 , α = 1.2 BatchSize= 150, Nesterov = False, Optimizer = False	TR = 1.17 ± 0.08 VL = 1.40 ± 0.06
3 layers of 70 tanh units	η = 0.0005, λ = 5e-05, α = 0.8 BatchSize= 60, Nesterov = False, Optimizer = False	TR = 0.75 ± 0.01 VL = 0.99 ± 0.07
2 layers of 50 tanh units	η = 0.003, λ = 0.0, α = 0.9 BatchSize= 150, Nesterov = False, Optimizer = False	TR = 0.61 ± 0.02 VL = 0.94 ± 0.05
1 layer of 15 sigmoidal units 1 layer of 20 sigmoidal units	η = 0.1, λ = 0.0, α = 0.9 BatchSize= 150, Nesterov = False, Optimizer = True	TR = 1.10 ± 0.13 VL = 1.46 ± 0.11

Some of the results of our grid search over the MLcup dataset

CUP: BEST RESULT

- We chose the final model examining the results of the nested grid search algorithm (models with best validation average, discarding the ones with high variance), though ignoring some configurations with poor "curve quality".
- So, we focused on configurations with two layers and Tanh activation function and preferred the best trade-off between a configuration that would give us smooth curves, and one with reasonable results in term on MEE.
- The best hyperparameters we found are the following: η=0.001, λ=3e-05, α=0.9, batchsize=150, Nesterov=True, Adam=False.
- Once chosen the best hyperparameters, we retrained the model on the whole dataset, except, of course, the internal test set.

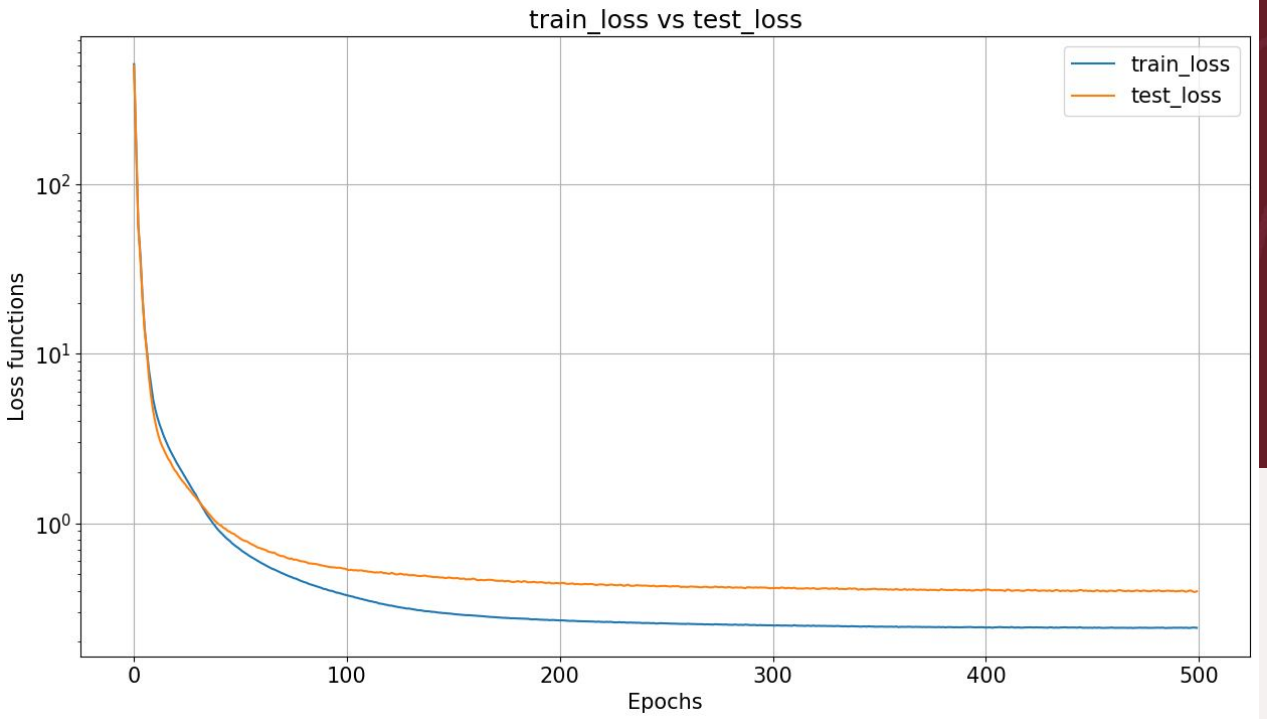
CUP: BEST RESULT

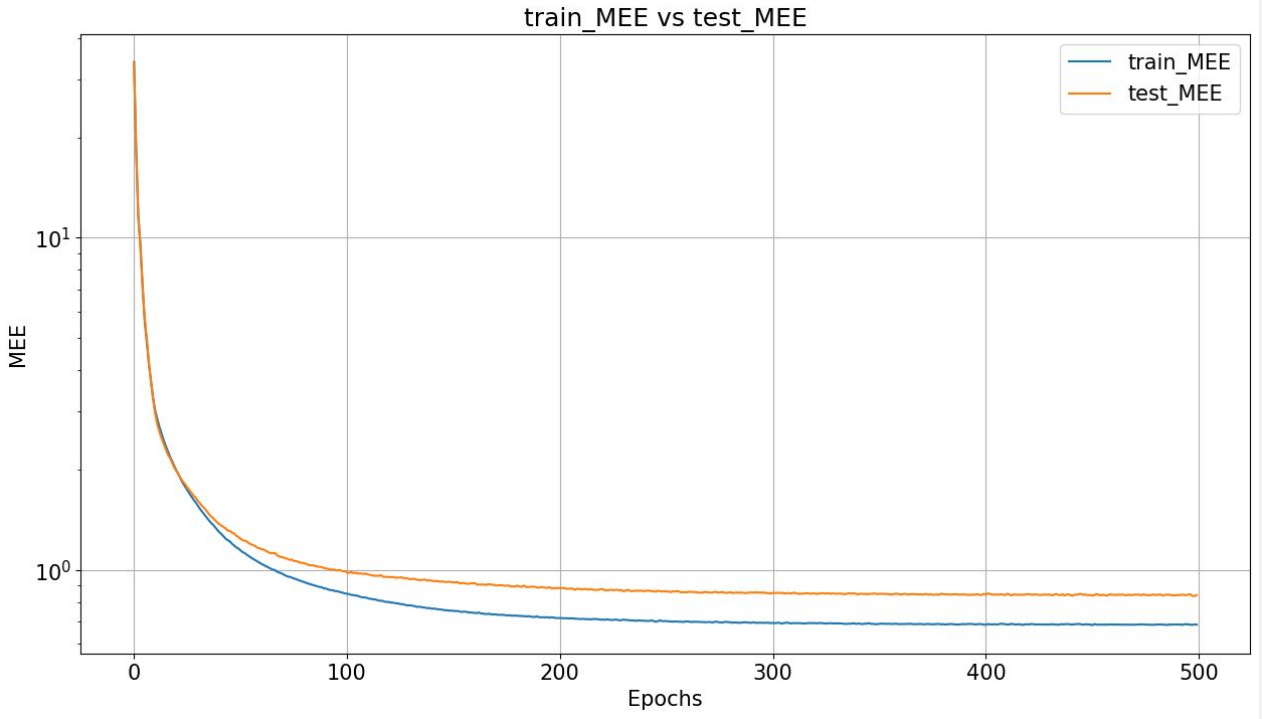
We obtained the best results with a model presenting 2 hidden layers having each 70 tanh units.

Cross validation results for this configuration:
Training MEE: 0.692 ± 0.006
Validation MEE: 0.932 ± 0.065

After retraining with entire training set we got:
Training MEE: 0.683
Test MEE: 0.838

On the right training and test curves during retrain.





L2 VS L1 REGULARIZATION

Ridge regularization

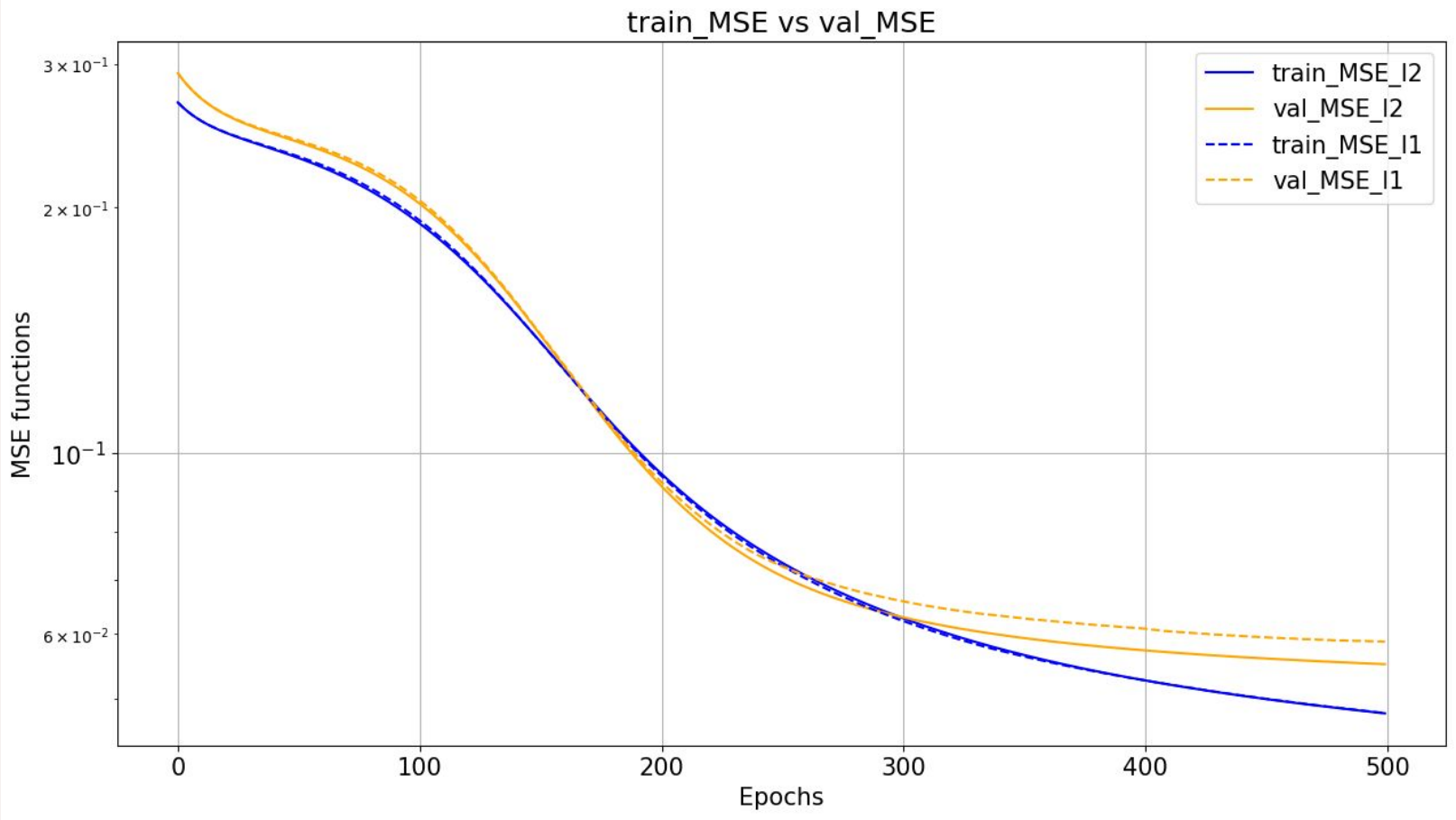
$$Loss(w) = \sum_{p=1}^{l} \left(y_p - x_p^t w\right)^2 + \lambda ||w||_2$$

$$w_{new} = w + \eta \Delta w - 2\lambda w$$

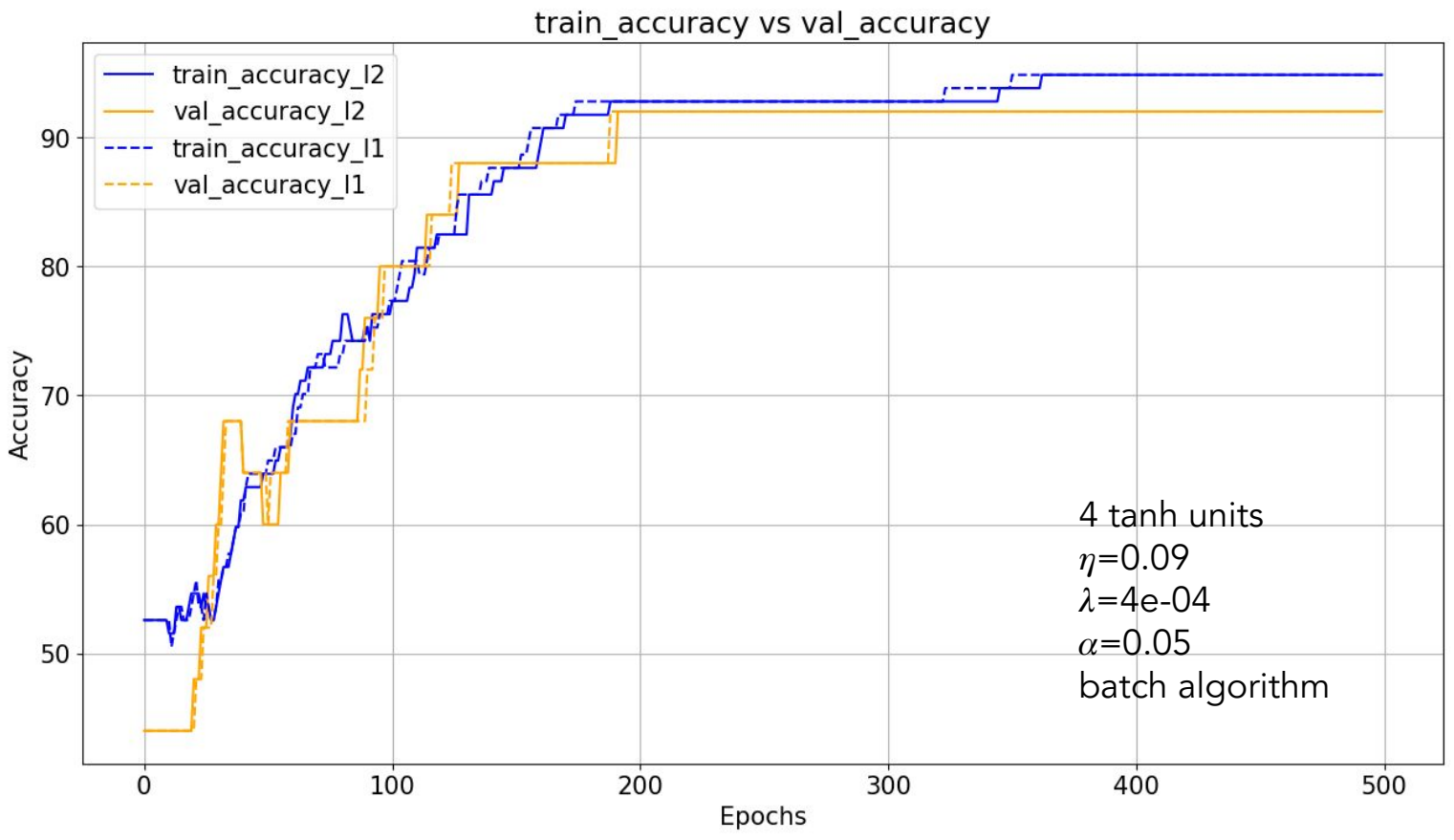
Lasso regularization

$$Loss(w) = \sum_{p=1}^{l} \left(y_p - x_p^t w\right)^2 + \lambda ||w||_1$$

$$w_{new} = w + \eta \Delta w - \lambda sgn(w)$$



L2 VS L1 REGULARIZATION

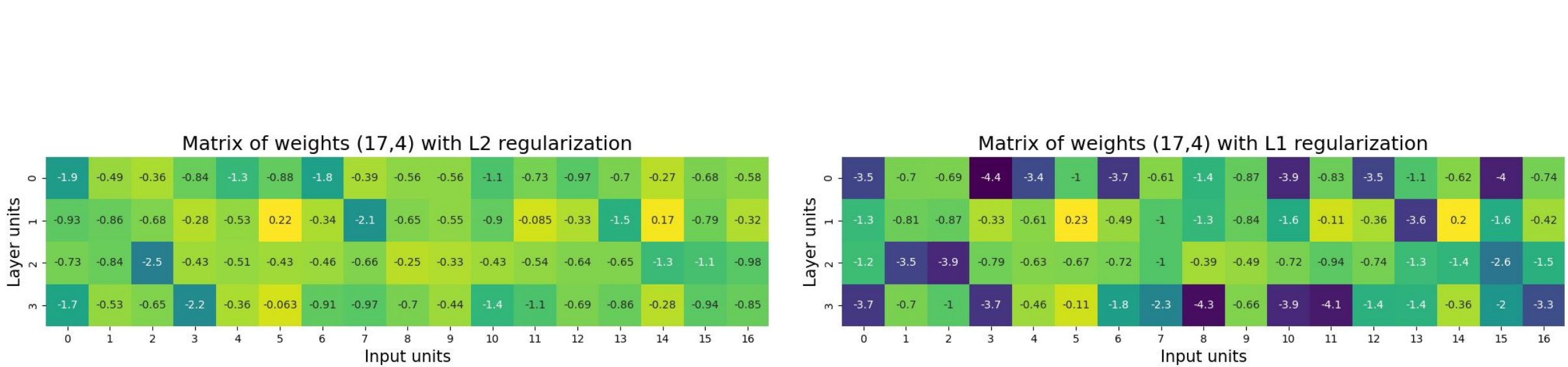


In this image are shown training and validation accuracy of two identical models with same initial weights, one with L2 regularization and one with L1 regularization.

We performed a 5-folds CV with same random seed (to ensure the same training for every fold):

L2 validation accuracy: 93 ± 4
L1 validation accuracy: 93 ± 4

L2 VS L1 REGULARIZATION



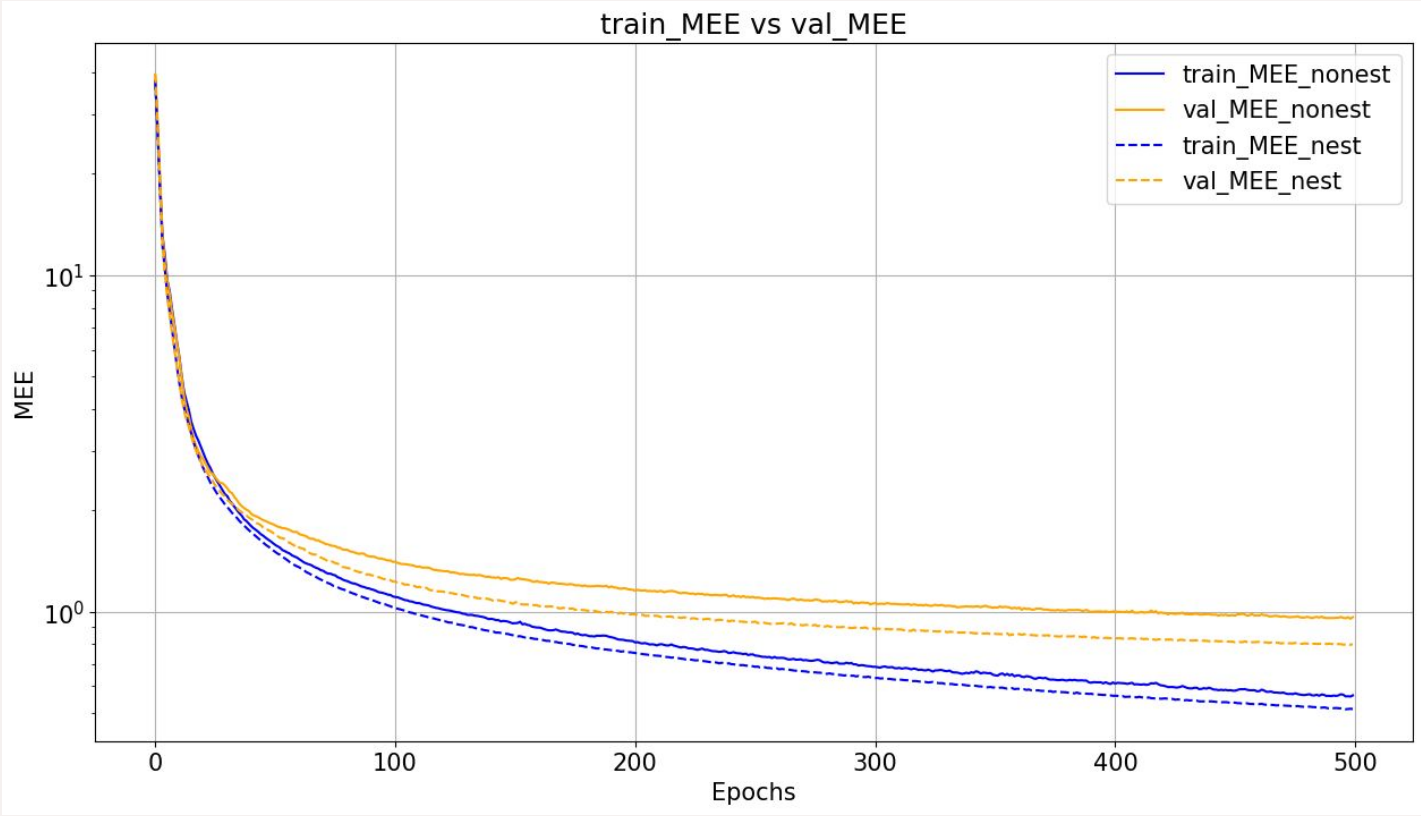
These images show how Lasso regularization tends to shrink some of the features weights to 0. To clarify the differences between the two matrices, we decided to plot the log10 of the absolute values of every element.

NORMAL MOMENTUM VS NESTEROV

Nesterov Momentum[2] accelerates convergence by considering the gradient of the loss function not at the current position but at an adjusted position. This anticipatory update can result in:

- Faster convergence
- Reduced oscillations
- Improved stability

This image shows training and validation MEE of two identical models with same initial weights, one with normal momentum approach and one with Nesterov momentum approach.



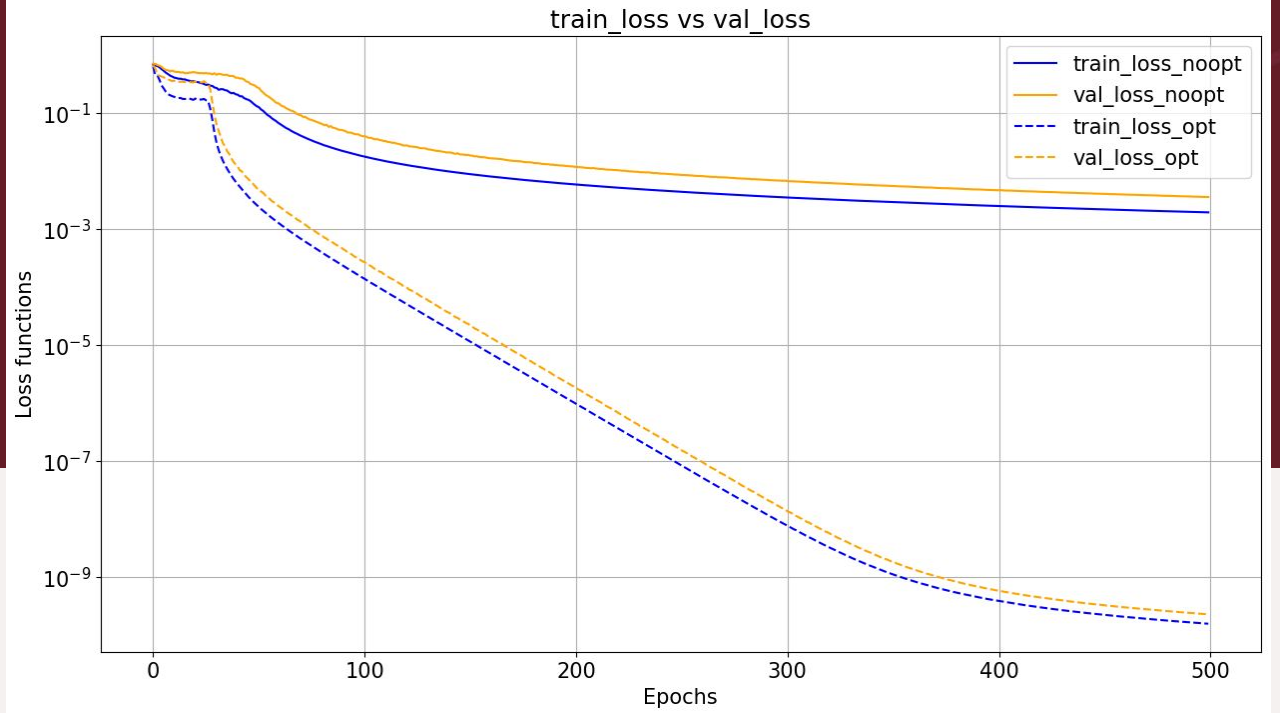
ADAM OPTIMIZER

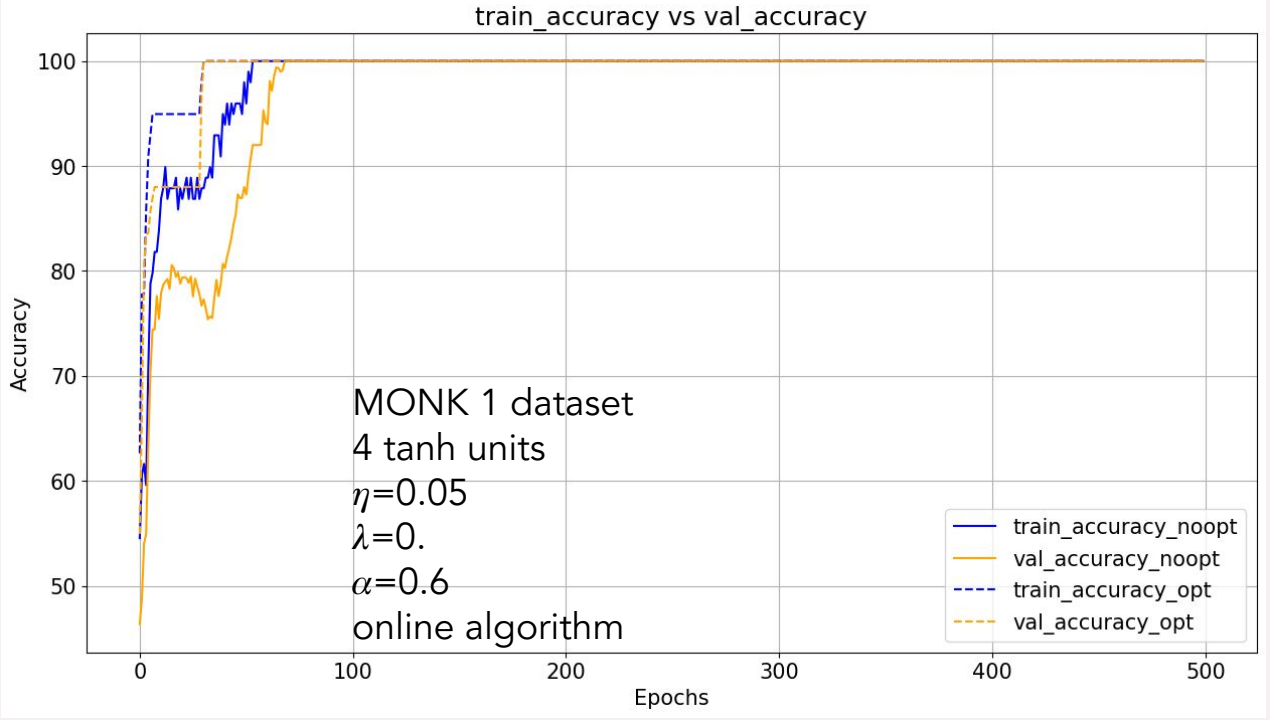
Adam[2] adapts the learning rates for each parameter individually based on the historical gradients. It maintains two moving averages for each parameter: the first moment and the second moment:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

This image shows loss and accuracy of two identical models with same initial weights, one uses Adam optimizer and one doesn't. It is clear that the optimizer helps obtain a faster convergence.



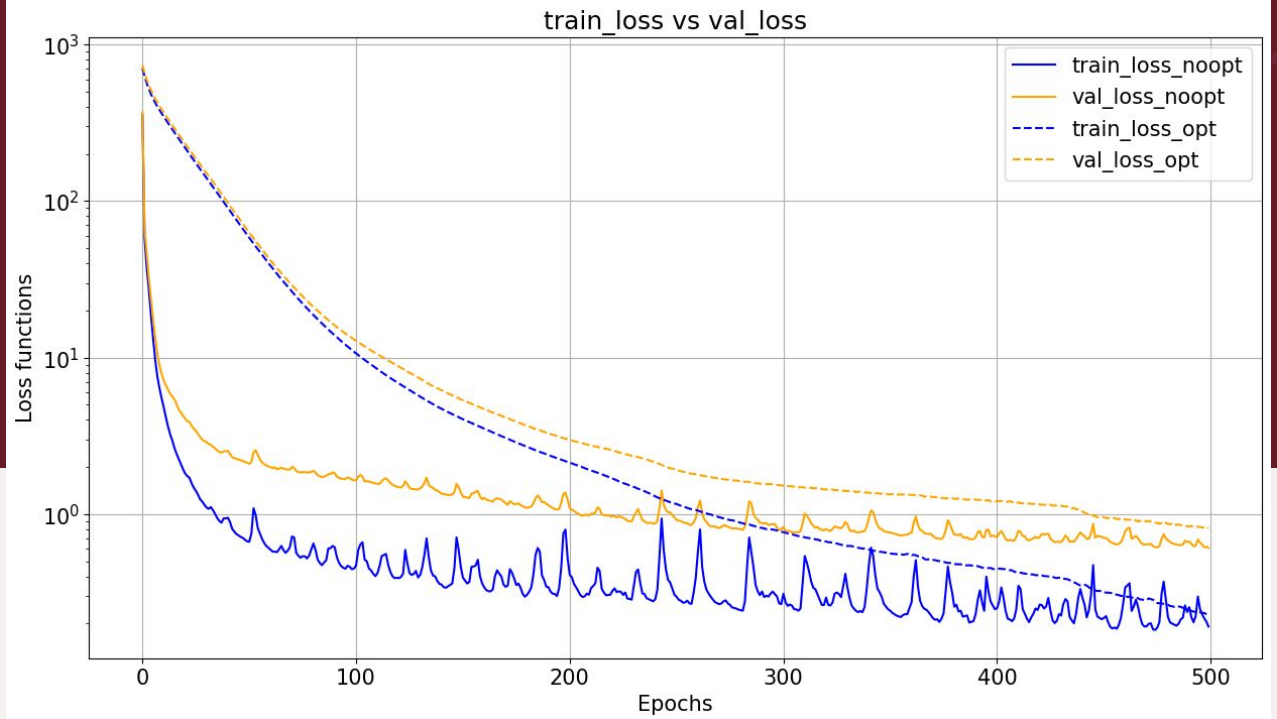


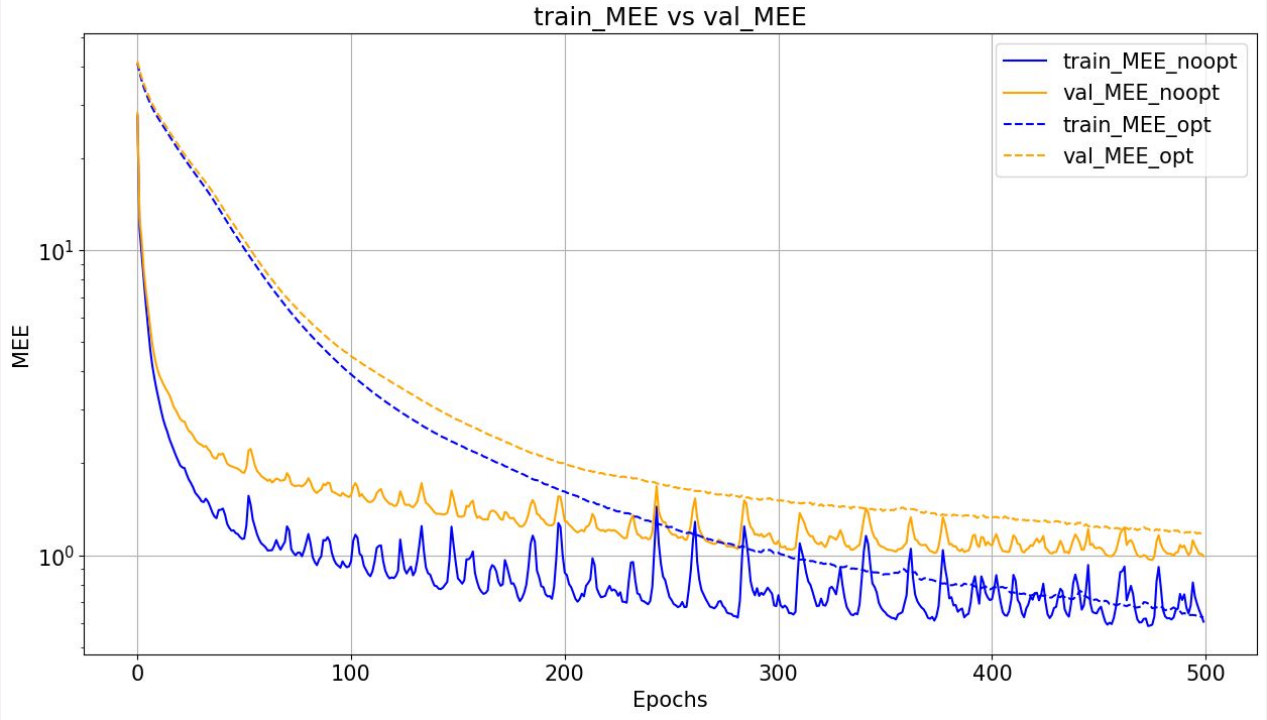
ADAM OPTIMIZER

In general Adam optimizer make the training curves more stable and requires less hyperparameter tuning.

In this images is shown how Adam optimizer can reach a ‘stable’ convergence even with a poorly choice of hyperparameters.

Performing some trials with Adam optimizer we found out that it’s convenient to set a higher eta in order to better exploit its adaptive capacity.

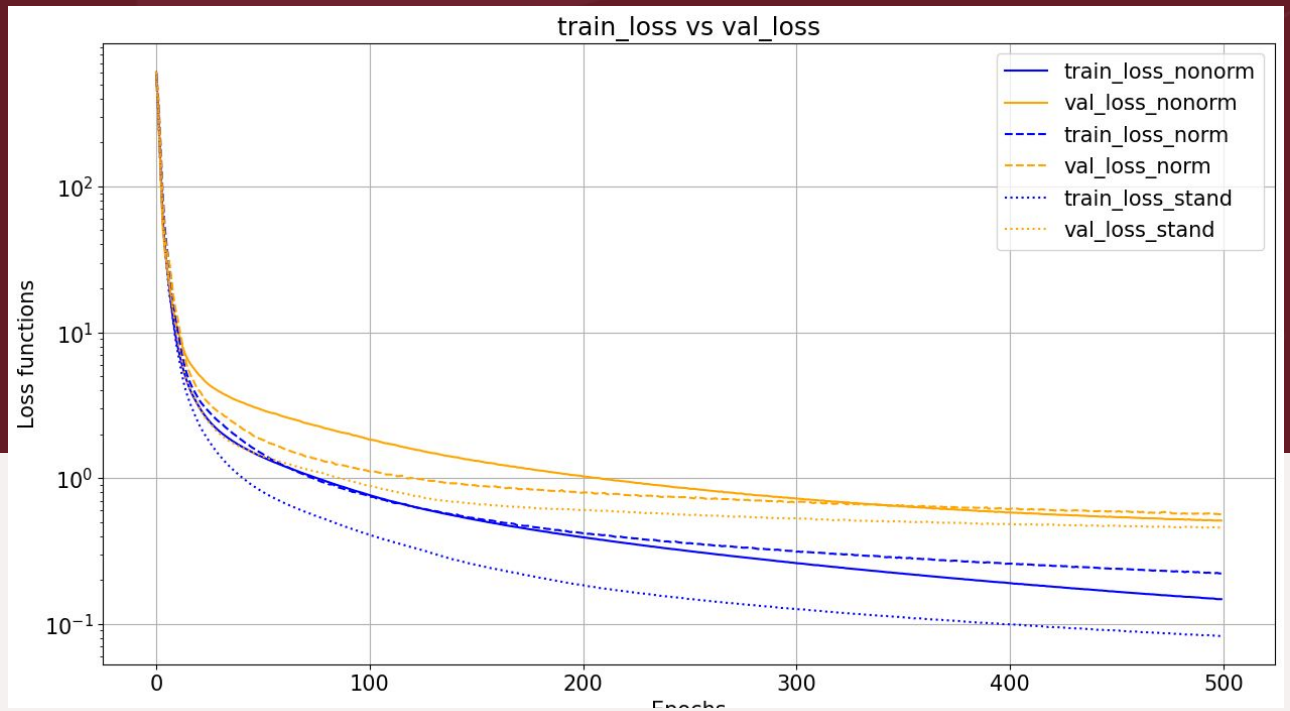


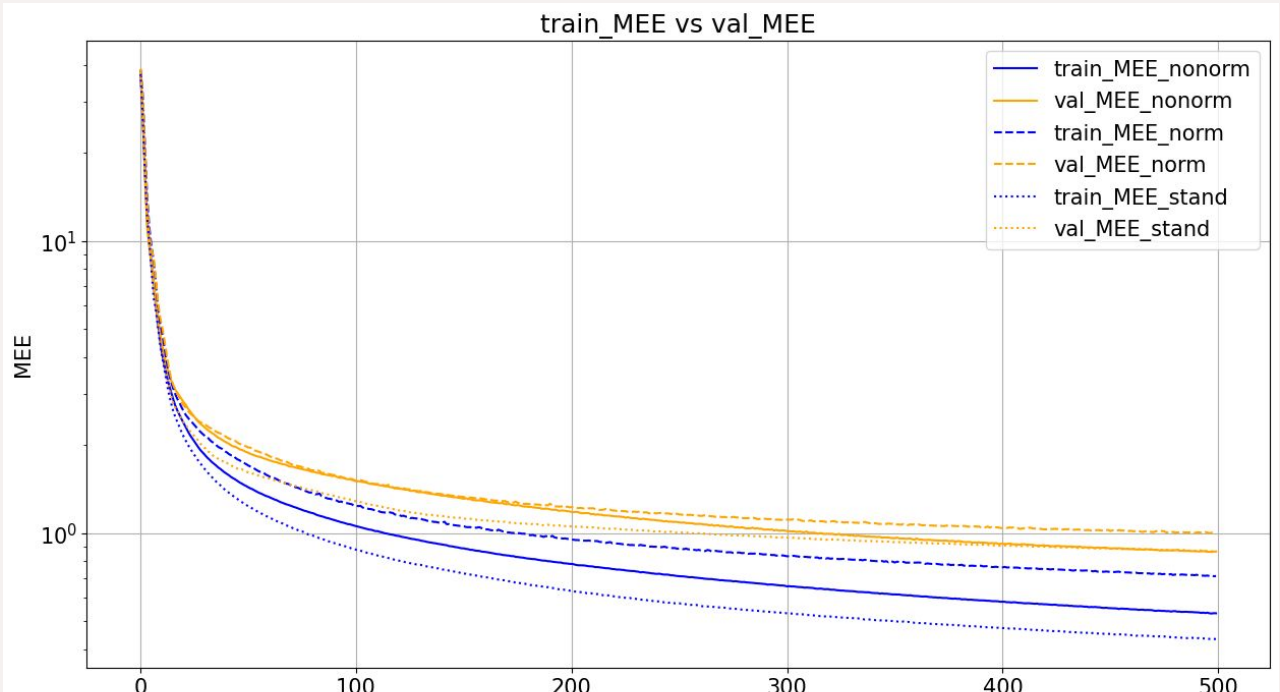


INPUT PREPROCESSING

Preprocessing techniques such as normalization/standardization, could be useful to reduce overfitting/obtain better numerical results.

In particular, looking at MEE plot, we can see how the normalization has some positive benefits on reducing the overfitting, on the other hand, despite a major overfitting, results obtained without preprocessing or with standardization are slightly better.





DISCUSSION

- Although in the end we didn’t use the Adam optimizer in the final model, it provided useful insight about optimization and we understood why it is one of the best algorithm.
- Unfortunately, we didn't have enough time to delve into the topic of parallel computing. Certainly, by utilizing GPU, we could have reduced execution times and covered the hyperparameter space more comprehensively.
- On the other hand, the Nesterov momentum technique helped us remarkably, providing more efficiency in convergence speed.
- The final result present some trace of overfitting, but in the end we retain that the model has reached a good level of generalization and hopefully a good MEE value.

23

CONCLUSIONS

- In the end we have definitively learn how to perform a grid search and how to treat and split the data for a general learning purpose.
- Last but not least we had the opportunity to implement from zero a neural network, giving us a chance to realize concretely what we have seen only as theory during the lectures.
- Since the execution times of the grid search were extensive, it was necessary to plan the parameter grids effectively. This allowed us to gain a better understanding of how hyperparameters function.
- Blind Test Results: AMonkUs_ML-CUP23-TS.csv

24

BIBLIOGRAPHY

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

[1]. X. Glorot, Y. Bengio: Understanding the Difficulty of Training Deep Feedforward Neural Network. Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia La guna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9. Copyright 2010 by the authors.

[2].L.Bottou, F.E.Curtis,J.Nocedal: Optimization Methods for Large-Scale Machine Learning. arXiv:1606.04838v3 [stat.ML] 8 Feb 2018

[3]. D. P. Kingma and J. Ba: Adam: A Method for Stochastic Optimization.
The paper was presented at the International Conference on Learning Representations (ICLR) in 2015.

APPENDIX A : Binary Cross Entropy for the Monks