ML 2023 Project Slides

Lipari Giuseppe & Carmine Vitiello.
MatrimonioIn3
Master degrees in Big Data Technologies and Artificial Intelligence
g.lipari@studenti.unipi.it
c.vitiello6@studenti.unipi.it

Date 14/01/2024

Type of project: **A**

Objectives

The goal of our project was to create a Multi-Layer Perceptron/Neural Network (MLP/NN) model simulator and apply it for didactical purposes.

We implemented the Gradient Descent algorithm in three versions: Stochastic, mini-batch and batch for a Didactical Neural Network (D-NN) with Backpropagation as a core component.

Implementation choices:

We used Python within VsCode and GitHub for versioning and teamwork. We tried to exploit various GD possible parametrization. We design the structure of the code to make simple future improvement. Also we modularize different cross validation run to avoid some of not useful gridsearch configuration. As know training and validation phases can take some hours depending to the number of the models and dimensions of datasets

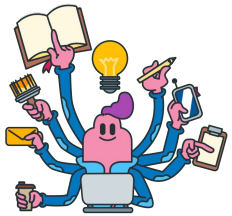
Project main classes:

- didacticNeuralNetwork: core class of the simulator
- kfoldCV: compute the K-fold cross validation
- gridSearch: compute the coarse and fine grid search and returns the candidate for the validation

Library:

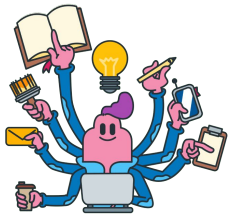
- Python Multiprocessing library to try time optimization for the plot drawing, we discover that can take few seconds each fold training
- Pandas to import and split the datasets
- Sci-kit sklearn for hot-encoding the monks datasets
- Numpy for the mathematical operations
- Matplotlib and matplotlib-latex-bridge for plotting and describe the model training in plot subtitle

Functions and Techniques implemented 1:



- Activation Function (Sigmoid, Tanh, Relu, Leaky Relu, Identity) with each derivatives
- Learning Eta linear decay
- GD Batch/on-line/mb
- Gradient Clipping to avoid overflow gradients
- Dropout regularization
- Regularization Tikhonov and lasso
- Early stopping on the Validation set variance during training within a threshold and patience
- Drawing plot of learning (Training and Metrics)
- Errors metrics : Mean Squared Error, Mean Euclidean Error, Mean Absolute Error, Root Mean Squared Error

Functions and Techniques implemented 2:

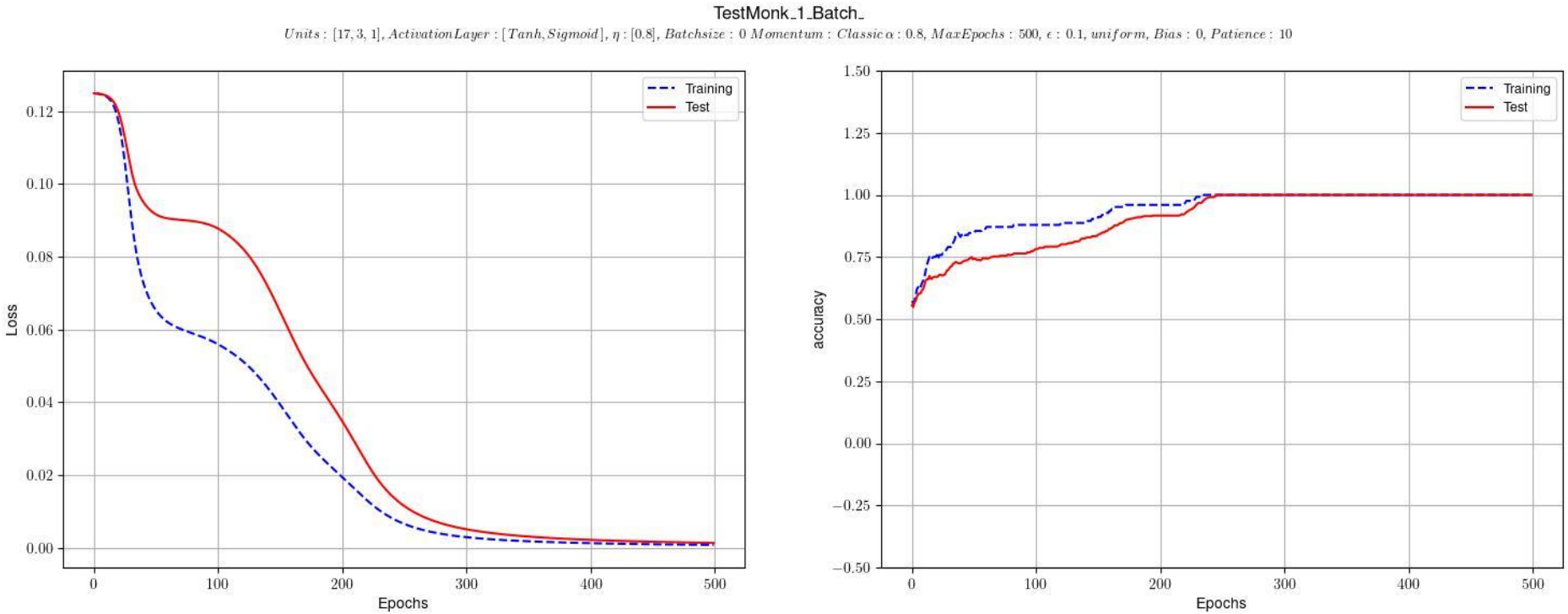


- Grid Search – Coarse e Fine versions
- Momentum Classic and Nesterov
- Shuffle batch option in Mini batch training
- Various weights initialization: Uniform, Random, Basic and Glorot(Tanh and Sigmoid layers) and He for ReLu types.
- Back propagation
- K-Fold Validation for model selection and Holdout for model assessment
- Loss function Squared Error and derivative(use in Backpropagation)
- Ensemble

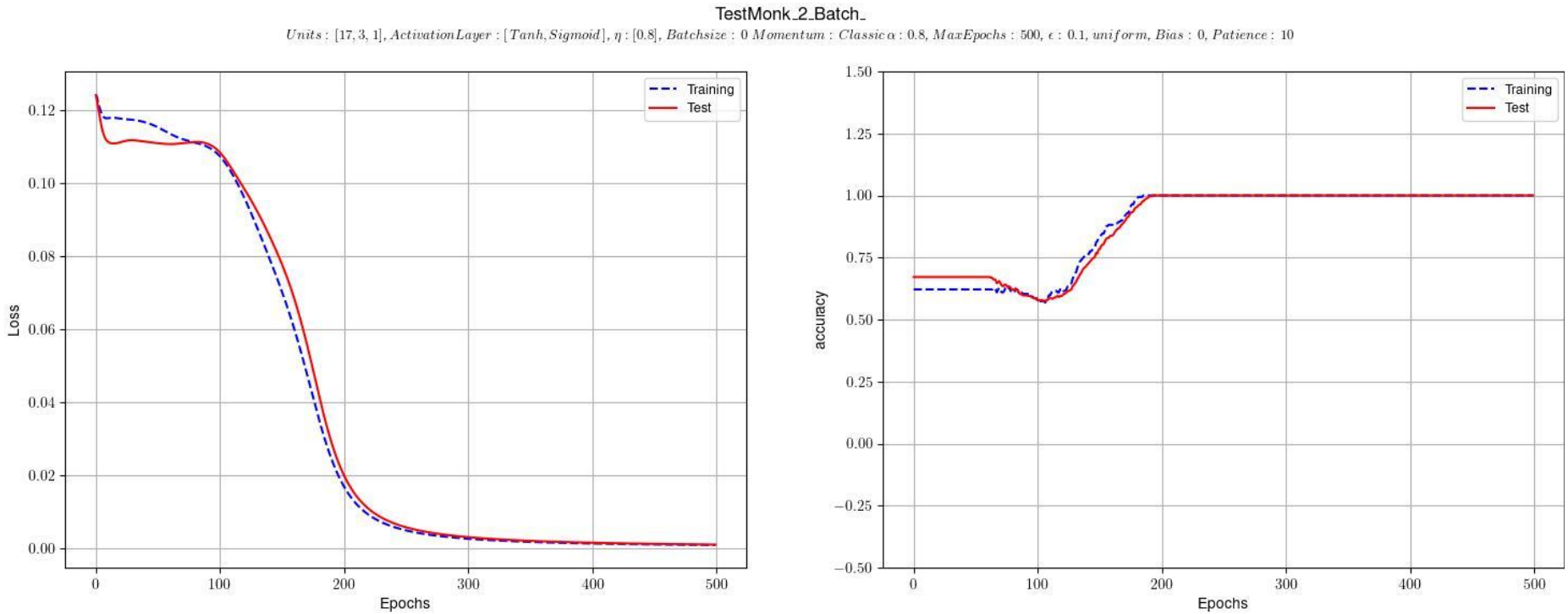
DataSet	Hyperparameters		MSE	Accuracy (%)
Monk 1	• NET: 17,3,1; • Activation functions: tanh,sigmoid; • Eta: 0.8, Batch; • Classic Momentum; • Momentum alpha 0.8;	• Max Epochs: 500; • Distribution:Uniform • eps = 0.1 • Bias init: 0	• TR: 0.0008107606711617572; • TS: 0.001349230631575087;	• TR: 100% • TS: 100%
Monk 2	• NET: 17,3,1; • Activation functions: tanh,sigmoid; • Eta: 0.8, Batch; • Classic Momentum; • Momentum alpha 0.8;	• Max Epochs: 500; • Distribution:Uniform • eps = 0.1 • Bias init: 0	• TR: 0.0007915448154647248; • TS: 0.0009257553235895012;	• TR: 100% • TS: 100%

DataSet	Hyperparameters		MSE	Accuracy (%)
Monk 3 No Reg	• NET: 17,3,1; • Activation functions: tanh,sigmoid; • Eta: 1.0, Batch; • Classic Momentum; • Momentum alpha 0.8;	• Max Epochs: 1000; • Distribution:Uniform • eps = 0.075 • Bias init: 0	• TR: 0.017533809709355527; • TS: 0.02789459136709849;	• TR: 96% • TS: 93.51%
Monk 3 Reg	• NET:17,3,1; • Activation functions: tanh,sigmoid; • Eta: 0.8, Batch; • Lasso 0.0008; • Classic Momentum; • Momentum alpha 0.8;	• Max Epochs: 1000; • Distribution: Glorot • eps = 0.1 • Bias init: 0	• TR: 0.02112224388365535; • TS: 0.022829093539841878;	• TR: 95.08% • TS: 95.14%

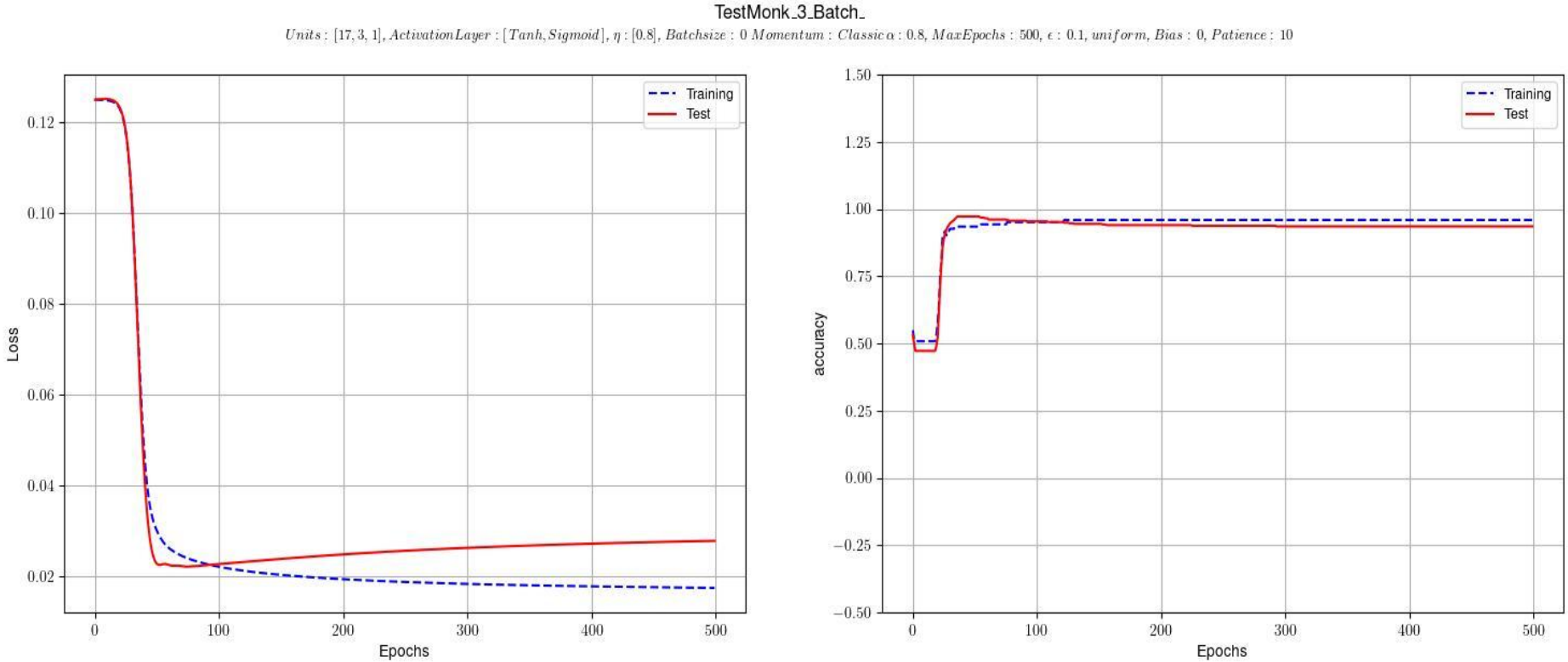
Plot Monk 1 MSE / Accuracy



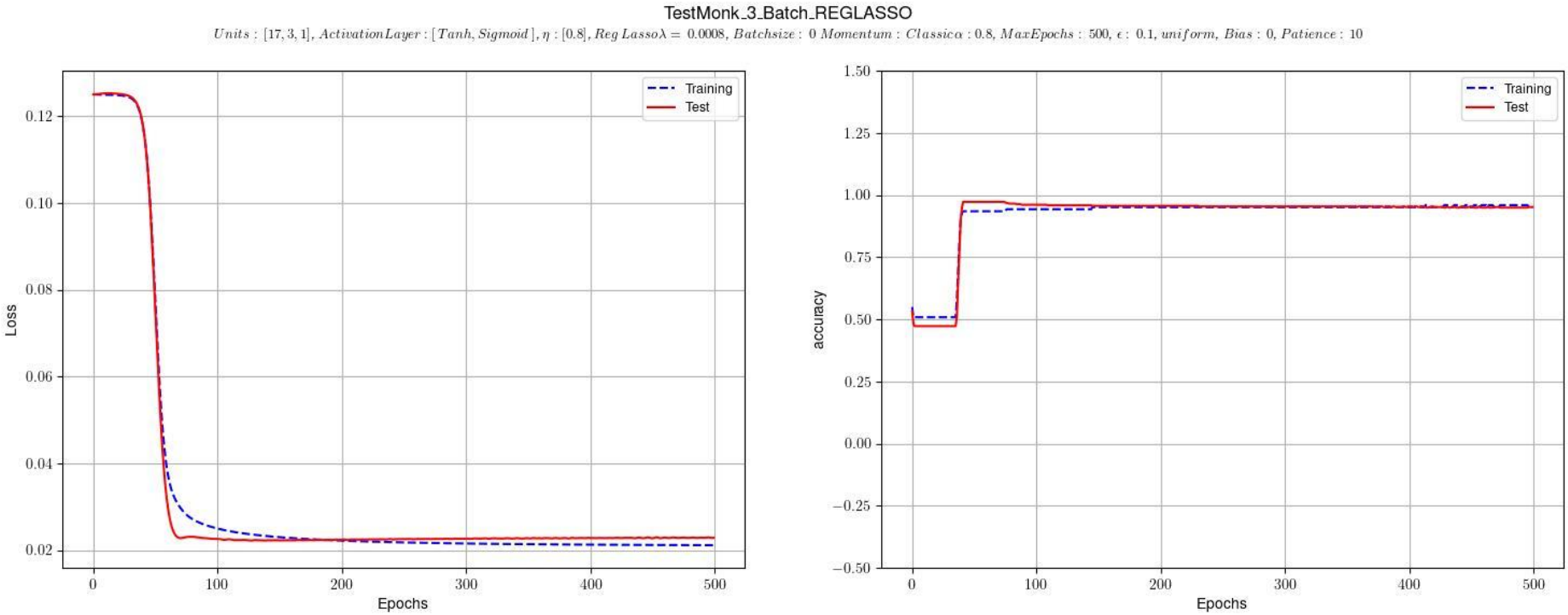
Plot Monk 2 MSE / Accuracy



Plot Monk 3 No Regularization MSE / Accuracy



Plot Monk 3 MSE / Accuracy



CUP Validation schema: data splitting

- The Training Cup dataset was splitted in a 75% as design set and a 25% for in house model assessment test set.
- Model Selection was made with a 5-Fold cross validation
- Model assessment with Hold-out
- Re-Training phase was made in all TR with models without early stop e with 75% of TR and 25% VL for Early stop model.

CUP Validation schema: model selection

We implements two different type of grid search: coarse and fine.
Coarse explores the combinations of many hyperparameters: Network topology, combination of activation functions, learning rates, different momentum with different alphas, differents regularization approach with its lambdas, batch dimensions, tau for learning decay, early stops thresholds and patience, type of initializations for weights and bias, gradient clipping option with threshold value and dropout option with probability.

Fine search is made starting by the hyperparameters of the model winner of the coarse selection.
The new grid search is made using tre new hyperparameters values adding and subtract the 20% of the winner. It is made for eta, momentum, lambda, gradient clipping, learning decay, eps for initialization, batch dimension, but if the winner it's a minibatch model and max number of epoch it not early stop was used.
For each selection the winner is the model with the lower MEE between the folds.

We started with small combinations of hyperparameters for catch a good starting range of them. So we split the grid search for differents cases like batch, minibatch, Early stops, with or without dropout (dropout need more units for good regularization). This was made for avoid grid search combination like high eta values for minibatch or viceversa.

Some of Model Selection hyperparameters 1

Topology of the Net	[10,20,15,3], [10,10,8,3], [10,20,20,20,3], [10,10,10,10,3]...
Activation functions	[Leaky ReLu, Leaky ReLu, identity], [tanh, tanh, identity]...
Eta values	0.0001, 0.004, 0.008
Steps and final etas for learning decay	(1000, 0.0004), (1000, 0.00005), Without
Type of Regularization and lambda values	(tikhonov, 0.0001), (tikhonov, 0.01), (Lasso, 0.0001), Without
Batch dimensions	Batch, 1, 100, 50, 150
Momentum and alpha	(Classic, 0,7), (Nesterov, 0,5)

Some of Model Selection hyperparameters 2

Max epochs	500, 1000, 2000
Eps for weights initialization	Initialization: Glorot for Tanh and He for Relu Uniform in eps, -eps range for identity: 0.01, 0.1, 0.2
Gradient Clipping thresholds	1, 5, 10, without
Dropout probability	0.5, 0.7, without
Bias	0, 0.1
Early stop	Not enabled, or enabled with thresholds 1.e-10, 1.e-8, 1.e-12 and patience: 10, 20, 30

Best Models Batch / Mean metrics with 5 folds	**Training**	**Validation**	**MEE**
NET: [10, 15, 3], activation functions: [tanh, identity], eta: 0.008, momentum: [classic, 0.7], epochs: 2000, eps: 0.01, distribution: glorot, early_stop: True, patience: 10, threshold_variance: 1e-10	1.575222968804	2.38973942183	0.12650819093405
NET: [10, 15, 8, 3], activation functions: [leakyRelu, leakyRelu, identity], eta: 0.01, tau: [500, 0.004], g_clipping: 5, momentum: [classic, 0.7], epochs: 2000, eps: 0.0125, distribution: glorot, early_stop: True, patience: 10, threshold_variance: 1e-10	1.617646720129	2.26980991260	0.13055449868741

Best Models Batch / Mean metrics with 5 folds	**Training**	**Validation**	**MEE**
NET: [10, 15, 3], activation functions: [leakyRelu, identity], eta: 0.004, momentum: [classic, 0.7], epochs: 2000, eps: 0.01, distribution: glorot, early_stop: True, patience: 10, threshold_variance: 1e-10	2.05568540058315	2.592658957331966	0.13787906918083606
NET: [10, 15, 3], activation functions: [tanh, identity], eta: 0.004, momentum: [classic, 0.7], epochs: 2000, eps: 0.01, distribution: glorot, patience: 10, threshold_variance: 1e-10	2.13810553871876	3.0259311817028496	0.1450125802713949

Best Models mini Batch / Mean metrics with 5 folds	**Training**	**Validation**	**MEE**
NET: [10, 15, 3], activation functions: [leakyRelu, identity], eta: 0.005, dim_batch: 45, momentum: [classic, 0.7], epochs: 500, eps: 0.00125, distribution: glorot, early_stop: True, patience: 10, threshold_variance: 1e-05	1.66176309470606	2.2943094270308078	0.1249202050410519
NET: [10, 15, 3], activation functions: [leakyRelu, identity], eta: 0.00625, dim_batch: 50, momentum: (classic, 0.7), epochs: 500, eps: 0.001, distribution: glorot	1.65611069846429	2.3854479523932945	0.12613501094308896

Best Models mini Batch / Mean metrics with 5 folds	**Training**	**Validation**	**MEE**
NET: [10, 15, 3], activation functions: [leakyRelu, identity], eta: 0.005, g_clipping: 3, dim_batch: 45, momentum: [classic, 0.7], epochs: 500, eps: 0.00125, distribution: glorot, early_stop: True, patience: 10, threshold_variance: 1e-05	1.86862645012745	2.498702059705555	0.13828967820016616
NET: [10, 15, 3], activation functions: [leakyRelu, identity], eta: 0.00625, g_clipping: 3, dim_batch: 50, momentum: (classic, 0.7), epochs: 500, eps: 0.001, distribution: glorot	1.86978837329472	2.500108696025802	0.14267394925155413

Efficense of the training algorithm HW

We used timit python annotation for generates files, containing information such as the function's name, arguments, and execution time. By combining data from these files, the goal is to compare and identify the most time-consuming methods. The timing data is split based on different configurations and epochs, and a weighted mean is computed for specific functions to facilitate analysis. The objective is to pinpoint and optimize the performance of expensive methods.

During the execution of the software, there were significant differences in timing results. In reality, machines that have the i7-12700H and 7 5700X take half as many seconds to run the same Python program as the others.

We used these Cpu components in our machines:

- Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz 2.60 GHz (O.S Windows 10)
- 12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz (O.S Windows 11)
- AMD Ryzen™ 7 5700X @ 3.4GHz (O.S Windows 11)
- AMD Ryzen™ 3 3200G @ 3.6 GHz (O.S Ubuntu)

Efficense of the training algorithm SW

To enhance plot creation efficiency, the multiprocess capability of Python is employed to parallelize the research process for models and the generation of their corresponding charts.

The average epoch value tends to be close to the upper limit of the hyperparameter interval. For instance, if the maximum value is 2000, the mean of the five folds will fall between 1800 and 2000, with a closer proximity to 2000 than the lower bound.

The activation functions play a crucial role in execution timing. The sigmoid function stands out as the most time-consuming, resulting in higher execution times. The overarching aim is to identify such bottlenecks and optimize them for improved overall performance.

Difference between the best models 1

 vs 

The construction of the model involved a planning phase where we defined intervals for the hyperparameter values. The system generated various candidates with all possible combinations using the grid search algorithm.

Our role was to select the right interval, then train and validate the models with 5-fold cross-validation.

At the conclusion of the network's operation, we evaluated the most effective learning curves and accuracy charts to isolate the hyperparameters of the winning model and establish new bounds for a new grid search start. Ultimately, we iterated through this training flow until we were satisfied with the results, based on metrics such as the average error of training and validation, average Euclidean error, and others.

Difference between the best models 2

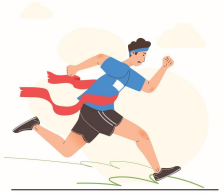
 vs 

This process was duplicated for two main configurations: batch and mini-batch. We aimed to avoid combining incorrect hyperparameters, thereby preventing the creation of unnecessary configurations. Indeed, there are only a few types of hyperparameters that can be similar or identical in both configurations. For example, in a mini-batch, it is necessary to have fewer eta values or stronger regularization methods than in a batch.

We implemented a constant to set the upper bound for the maximum number of best models, aiming to control the number of models that the system return as winner. After obtaining the set of the best models, the program uses the ensemble to blend their results.

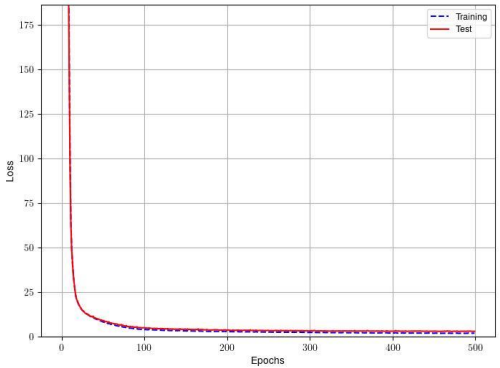
Finally, we use 25% of the TR Cup dataset that is not included in the training or validation to model assessment. This is necessary because we do not have target information in the blind cup dataset to compare with the output.

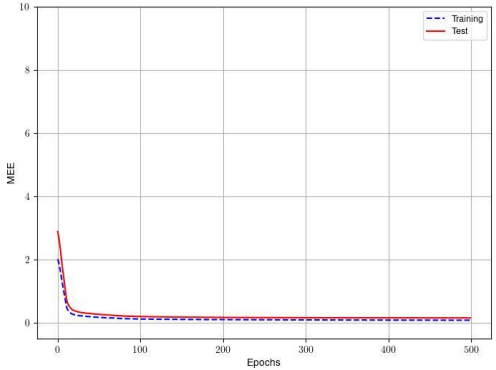
The best single model is…



- NET: [10, 15, 3],
- activation functions: [leakyRelu, identity],
- eta: 0.005,
- dimension batch: 45,
- momentum:[classic, 0.7],
- epochs: 500,
- eps: 0.00125,
- distribution: glorot,
- early stop: True, patience: 10, threshold variance: 1e-05

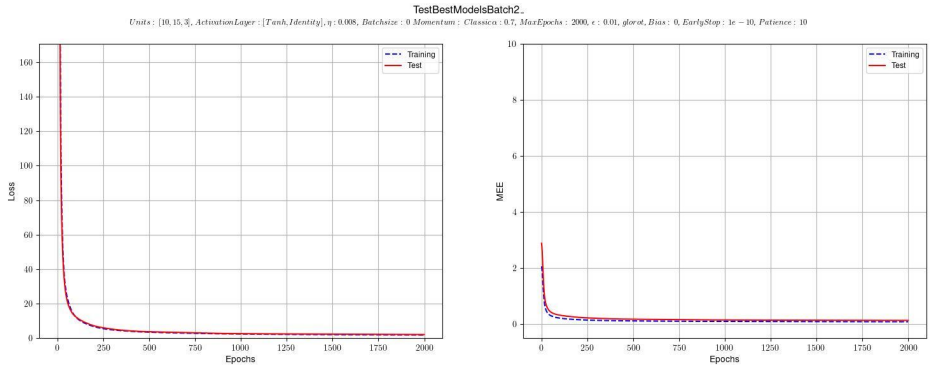
- Mean train: 1.661763094706064,
- Mean validation: 2.2943094270308078,
- Mean mae: 0.9003557566731688,
- Mean rmse: 1.2357324133233611,
- Mean epochs: 500.0,
- Mean Euclidean Error: 0.1249202050410519



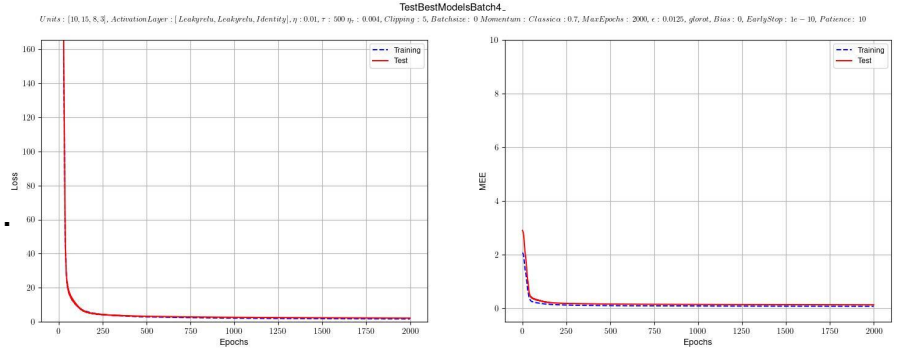


Plots Cup Batch

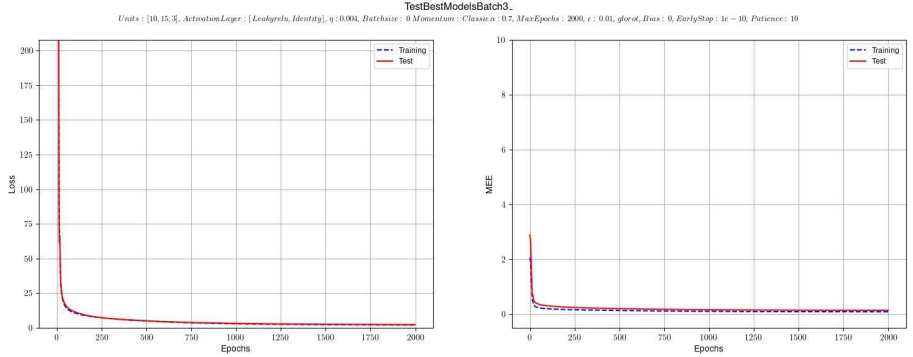
1.



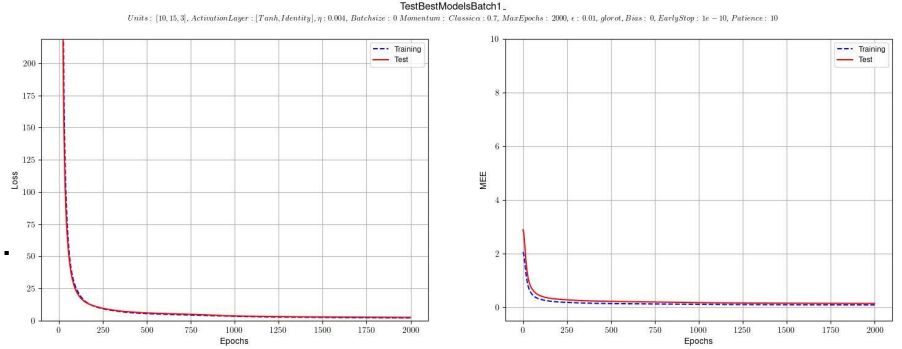
2.



3.

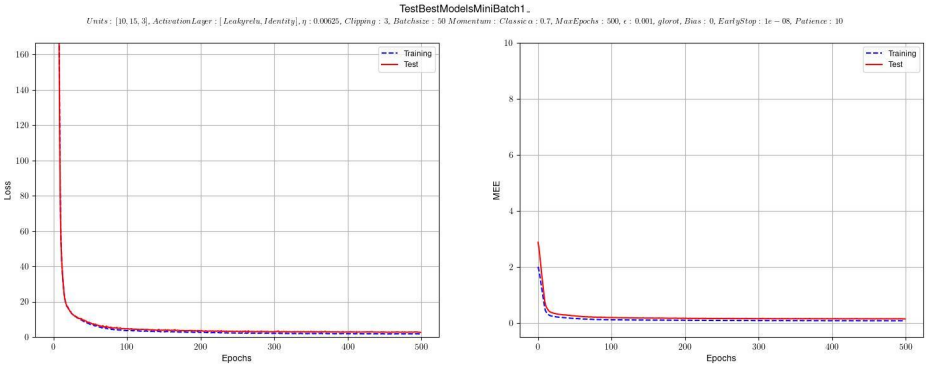


4.

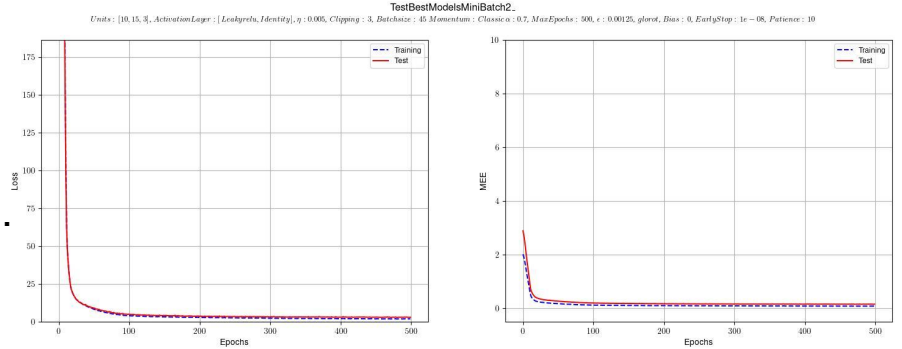


Plots Cup Mini Batch

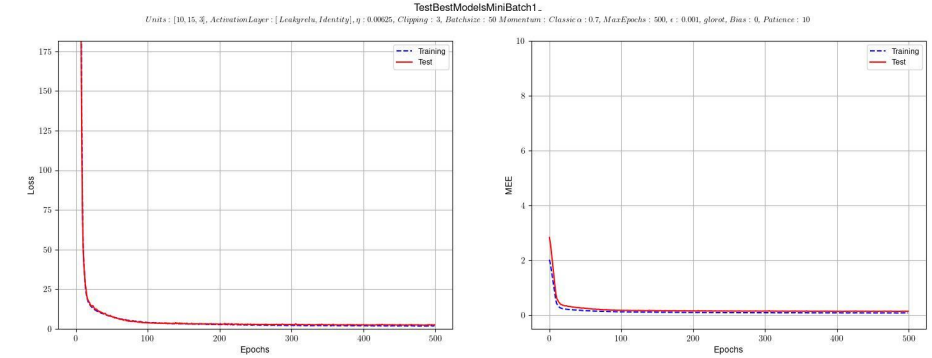
1.



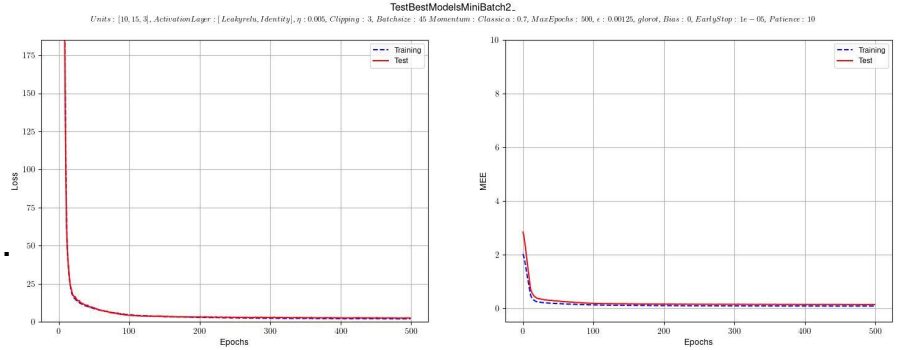
2.



3.



4.

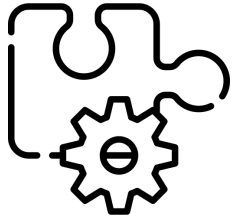


Conclusions

Working on the project let us play around with different hyperparameters setups, tweaking how the models are built, adjusting settings, and trying out various technique. It was like a virtual playground for learning about machine learning and neural networks, making the Machine Learning journey more hands-on and enjoyable.

Blind Test Results: MatrimonioIn3_ML-CUP2023-TS.csv

Appendix - Gradient Clipping and Dropout



Our goal was to moderate the error values for batch and mini batch, which is why we implemented these two features last.

To prevent the error values from overflowing, gradient clipping was implemented and used. At times, the network didn't end the training because it went too far from the minimum point with an error too big that caused the maximum memory value of an array with float to be 64 bits.

Dropout was implemented to avoid mini batch errors and decrease 'zig-zag' behavior caused by an excessive ETA value, but it was not used frequently due to time shortage to improve results.

Bibliography

1. A. Micheli, University of Pisa, 2023-2024, Machine Learning course by Micheli, Retrieved from https://elearning.di.unipi.it/course/view.php?id=524
2. Understanding the difficulty of training deep feedforward neural networks, Xavier Glorot, Yoshua Bengio ; PMLR 9:249–256
3. K. He, X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification", Proc. IEEE Int. Conf. Comput. Vis. (ICCV), pp. 1026-1034, Dec. 2015.