29/01/2025

MACHINE LEARNING PROJECT

Students: Leonardo Bandiera Marlia, Irene Bini, Alberto Montanelli
Master degree curricula in:
Complex Systems, Fundamental Interactions, Data Analysis in Experimental Physics
Team name: BG_peppers
E-mails: l.bandieramarlia@studenti.unipi.it, i.bini3@studenti.unipi.it, a.montanelli@studenti.unipi.it

Project A

OBJECTIVES

- Our aim is to build a **multi layer perceptron** from scratch, able to perform binary classification and to solve regression problems
- The neural network is trained using **gradient descent algorithm** combined with **backpropagation**
- Once the perceptron is built, the focus shifted on improving it. Best fitting hyperparameters are found through **grid search** supported by **successive halvings** algorithm, **elastic regularization** stabilizes the training and **Adam** and **Nesterov Accelerated Gradient** optimizers are added

METHOD

- The implementation of the neural network has been made from scratch in **Python**, using `Numpy`, `Matplotlib` and `Pandas` as support libraries
- `Layer` class is the basic building brick of the network. Layers are constituted with their input dimension, output dimension and activation function (mainly `sigmoid` and `leaky_ReLU`). The weights are initialized extracting randomly from a uniform distribution depending from the fan-in and the biases are initially set to zero. A **forward and backward** methods are also implemented
- `NeuralNetwork` class associates the layer configuration to a regulizer and an optimizer respectively passed from classes `Regularization` and `Optimization`. Here, it is possible to choose the type of regularization and optimization and to tune their parameters
- `DataProcessing` class is designed to divide data in **training+validation set** (80%) and **test set** (20%). Afterwards, the training+validation data is set to be employed as hold-out validation or a **k-fold cv**

METHOD

- `ModelSelection` class allows the **training algorithm** to be performed on the previous setted data with **batch method** (online/mini-batch/batch), returning the training and validation loss for each epoch
- To evaluate the goodnees of the loss function, three stopping checks are realized, based on **early stopping, smoothness and overfitting**
- In order to find the best hyperparameters:
  1. a **coarse, hierarchical exploration** of the hyperparameters space is performed in order to eliminate non-promising areas and to fix the `opt_type`, thus eliminating `None` and keeping `NAG` and `adam`
  2. a **successive halvings** algorithm is implemented to find the most robust architectures, ranging from 1 to 3 hidden layers and from 16 to 256 units per layer
  3. a **final, fine, total exploration** of the grid is performed in order to fix the other hyperparameters, such as `learning_rate`, `batch_size`, `lambda`, `alpha` and `opt_type` itself

NOVELTIES

- **Elastic regularization**: prevents overfitting and balances Tikhonov and lasso approaches. Tikhonov allows enhanced numerical stability by reducing the magnitude of the weights while lasso helps selecting the most significant features squashing some weights to 0

$$RegTerm = \lambda[\alpha \cdot sign(w) + 2 \cdot (1 - \alpha) \cdot w]$$

- **Adam optimizer**: helps building a more resilient network, with less dependency from hyperparameters [1]
- **Nesterov accelerated gradient optimizer**: boosts convergence speed [2]
- **Successive halvings grid search**: in tandem with a more extensive, classic grid search, successive halvings algorithm allows for a more rapid skimming of non-promising hyperparameters configurations [3]

ADAM OPTIMIZER

- Adam stands for **adaptive moment estimation**: individual learning rates for different parameters are computed from estimates of **first and second moment of the gradients,** respectively `m` and `v`
- The aim is to minimize the expected value of the noisy objective function `f(θ)` with respect to its parameters `θ` (in our case the loss function, with parameters `w` and `b`)
- The timestep `t` is incremented for every epoch and for every batch
- The algorithm updates at every `t` the exponential moving averages of the gradient `m_t` (estimate of the mean) and the squared gradient `v_t` (estimate of the uncentered variance), whose decay is regulated by hyperparameters `beta_1` and `beta_2`, fixed respectively to 0.9 and 0.999. `epsilon` is fixed to 10^{-8}
- In doing so, the **learning rate is adapted for each parameter** and it is updated for every `t`. This is the key feature of the optimizer, that guarantees **robust convergence** and **some independence from the hyperparameters**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta(f(\theta))_t$$

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$

$$w^{new} = w^{old} - \eta\,\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}+\epsilon}}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla_\theta(f(\theta))_t{}^2$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}$$

$$b^{new} = b^{old} - \eta\,\frac{\widehat{m_t}}{\sqrt{\widehat{v_t}+\epsilon}}$$

NAG OPTIMIZER

- **Nesterov momentum** is a speed-up technique based on the **anticipation of the direction of descent**
- The update rules of weights and biases are added the gradient calculated in the predicted position of the aforementioned parameters, thus giving a boost in the convergence speed
- It combines the advantage of momentum of **reducing oscillations** with the enhanced **epoch-wise efficiency of gradient projection**, optimizing gradient descent itself
- Unlike Adam optimizer, NAG is extremely **dependent on the choice of hyperparameters**, but once an optimal set is found convergence is reached in much less epochs

$$w^{new} = w^{old} - \left\{\eta \left[\nabla_w \mathcal{L} \left(w^{old} + \mu \, \Delta w_{old}\right)\right] + \mu \, \Delta w_{old}\right\}$$

$$b^{new} = b^{old} - \left\{\eta \left[\nabla_b \mathcal{L} \left(b^{old} + \mu \, \Delta b_{old}\right)\right] + \mu \, \Delta b_{old}\right\}$$

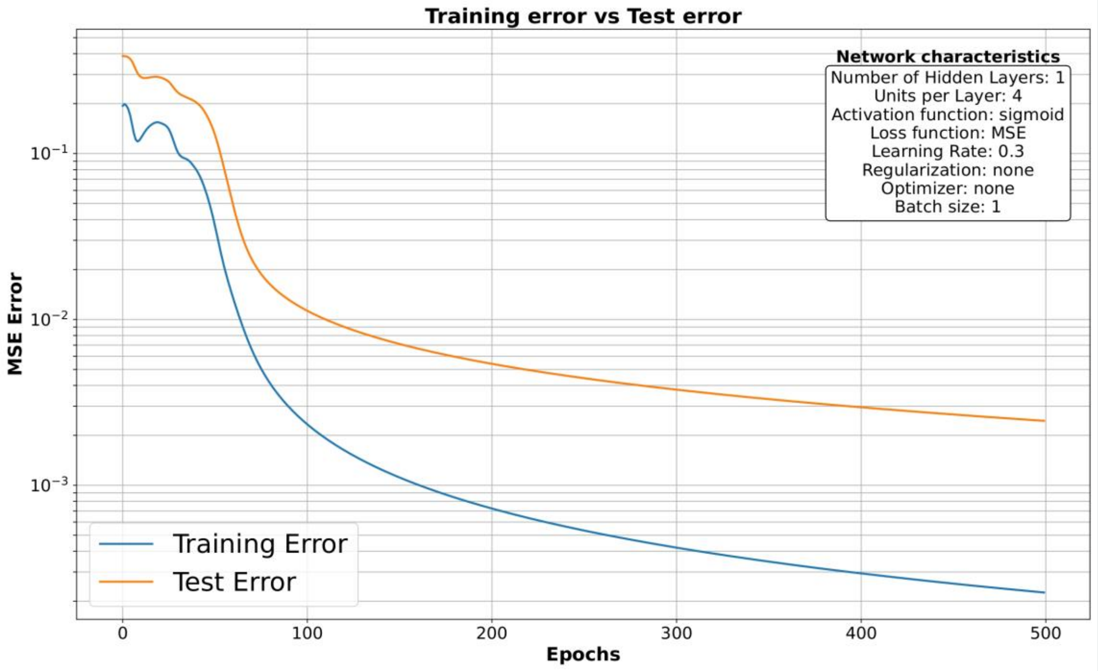
MONK EXPERIMENT

- The performance of the network is evaluated on the Monk dataset in order to assess whether it functions properly
- **One-hot encoding** of Monk data to make it readable for the neural network
- The neural network architecture was fixed to **1 hidden layer** with **4 sigmoidal units**
- In order to calculate the **accuracy** of the network, the outputs of each epoch are pushed to 0 or 1 if they are respectively under or above 0.5 in the sigmoid function. However, this treatment has not been applied to backpropagation in order to have a non-conditioned weights update
- No regularization or optimization was needed for Monk 1 and Monk 2, while for Monk 3 regularization is taken into consideration to help better convergence
- **Mean Squared Error** is used as loss function

MONK RESULTS

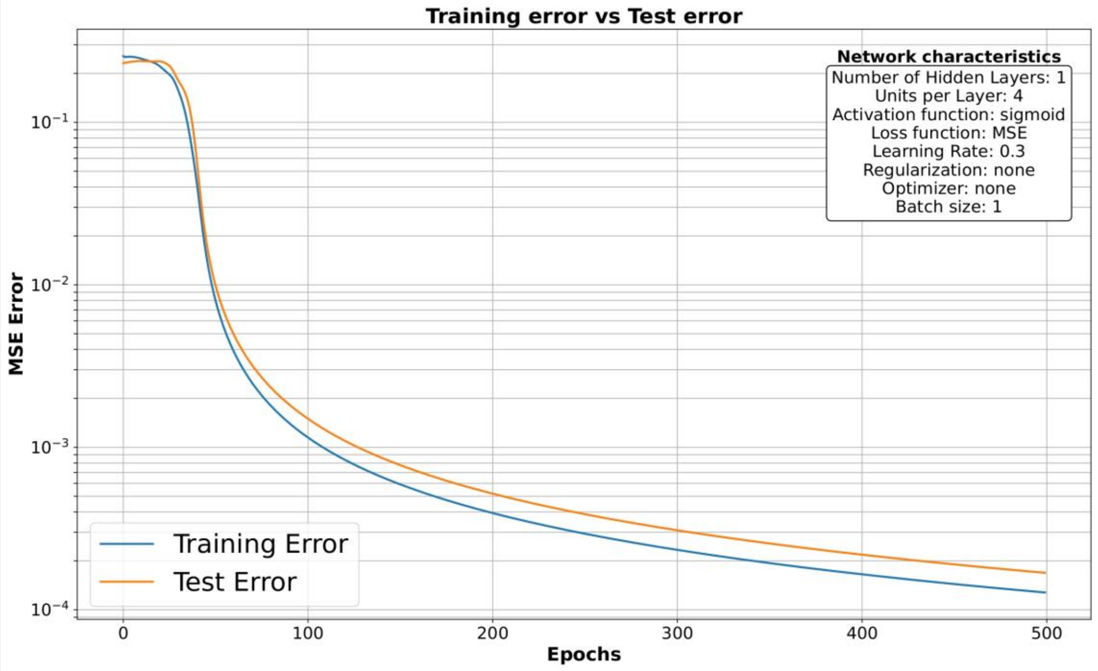
Monk task	Architecture and parameters	MSE (TR/TS)	Accuracy (TR/TS) [%]
Monk 1	1 hidden layer, 4 sigmoid units, online algorithm, $\eta = 0.3$	TR: $1.02 \cdot 10^{-4}$ TS: $1.33 \cdot 10^{-3}$	TR: 100 TS: 100
Monk 2	1 hidden layer, 4 sigmoid units, online algorithm, $\eta = 0.3$	TR: $1.28 \cdot 10^{-4}$ TS: $1.69 \cdot 10^{-4}$	TR: 100 TS: 100
Monk 3	1 hidden layer, 4 sigmoid units, online algorithm, $\eta = 0.01$	TR: $5.22 \cdot 10^{-2}$ TS: $4.11 \cdot 10^{-2}$	TR: 95.08 TS: 96.06
Monk 3 (reg.)	1 hidden layer, 4 sigmoid units, online algorithm, $\eta = 0.01$, reg_type = elastic, $\alpha = 0.5$, $\lambda = 10^{-5}$	TR: $6.19 \cdot 10^{-2}$ TS: $4.41 \cdot 10^{-2}$	TR: 93.44 TS: 97.22

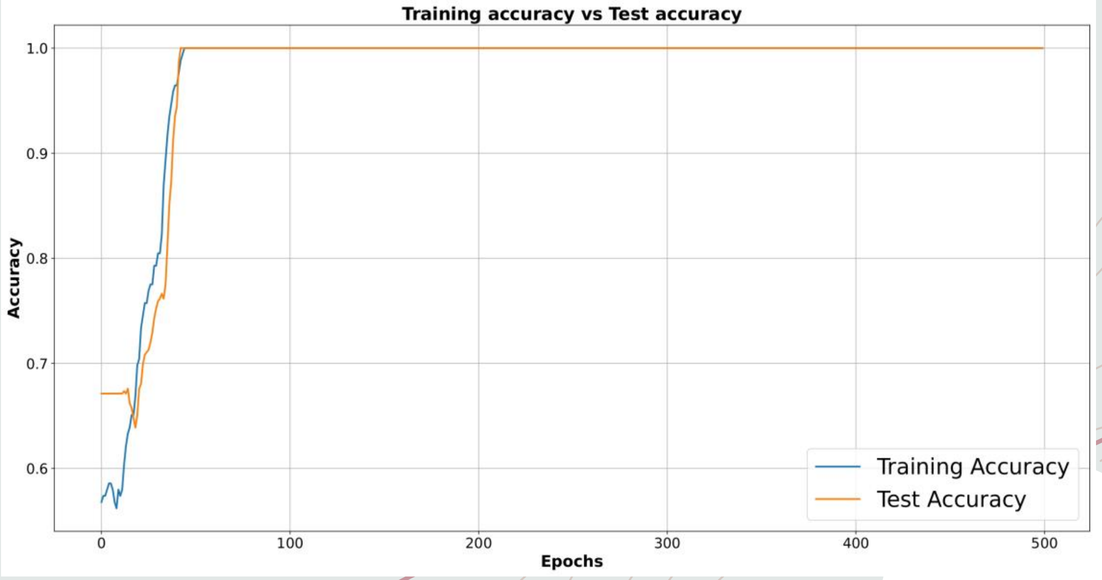
MONK 1


Training error vs Test error
Network characteristics
Number of Hidden Layers: 1
Units per Layer: 4
Activation function: sigmoid
Loss function: MSE
Learning Rate: 0.3
Regularization: none
Optimizer: none
Batch size: 1
MSE Error
Epochs
Training Error
Test Error

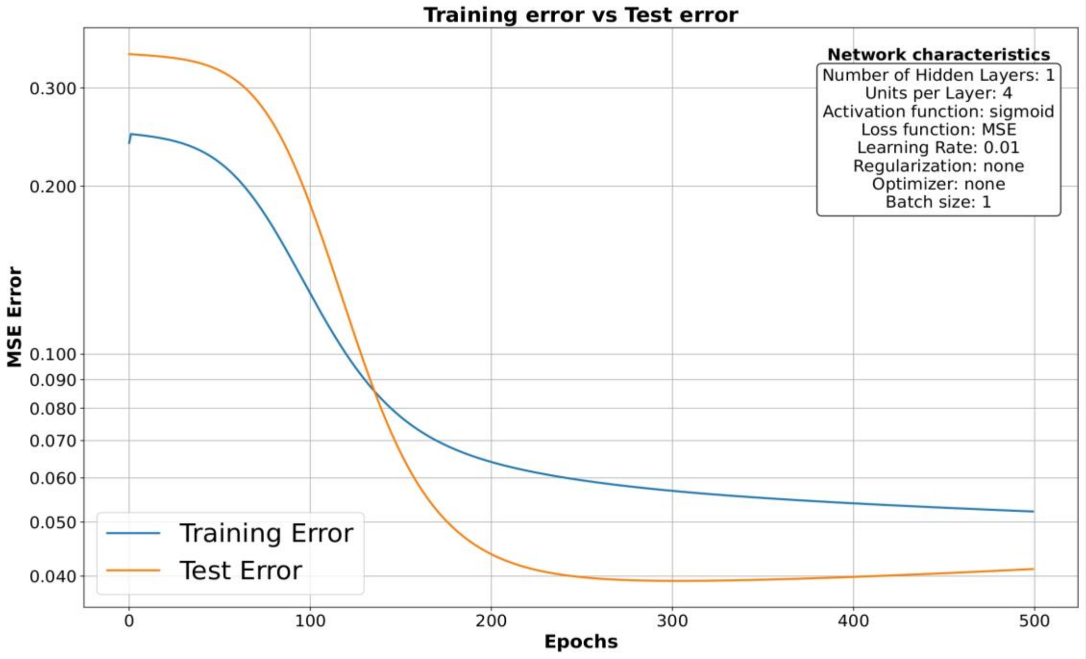

Training accuracy vs Test accuracy
Accuracy
Epochs
Training Accuracy
Test Accuracy

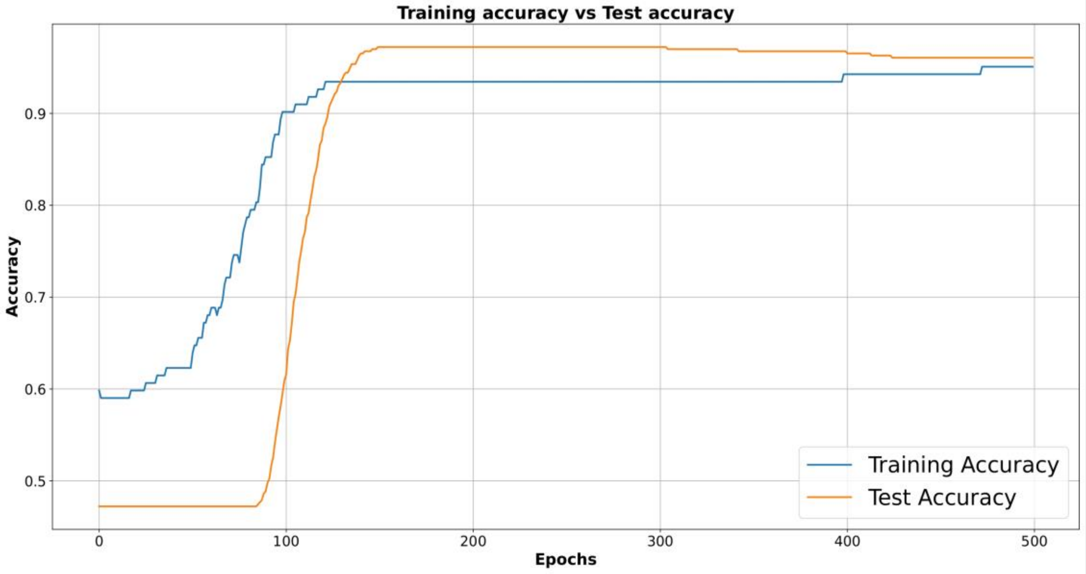
MONK 2


Training error vs Test error
Network characteristics
Number of Hidden Layers: 1
Units per Layer: 4
Activation function: sigmoid
Loss function: MSE
Learning Rate: 0.3
Regularization: none
Optimizer: none
Batch size: 1
MSE Error
Epochs
Training Error
Test Error

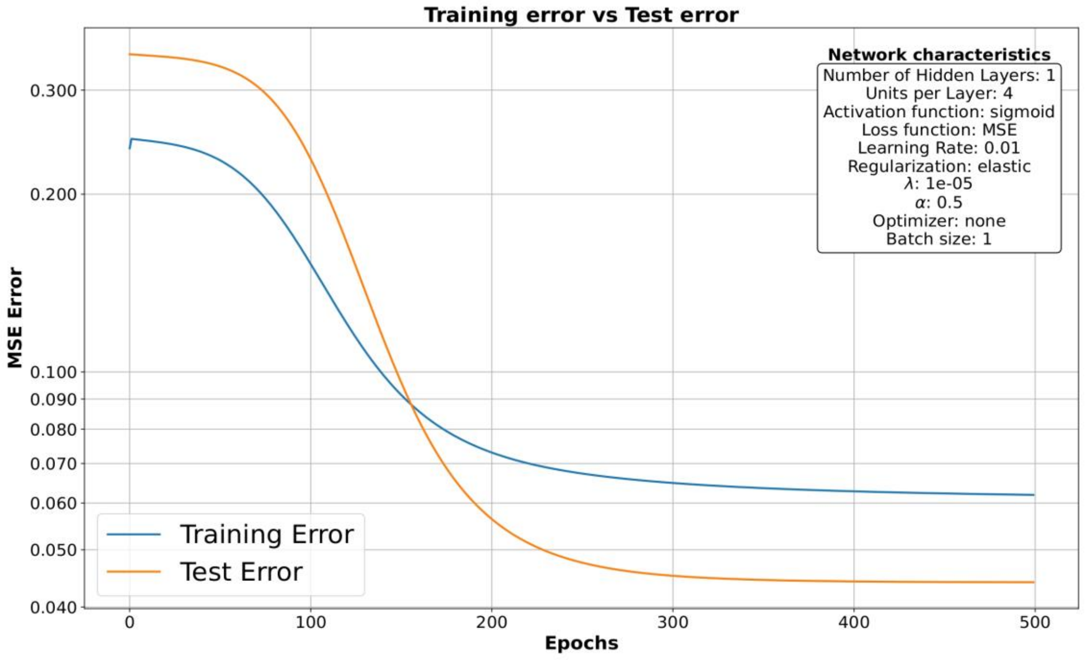

Training accuracy vs Test accuracy
Accuracy
Epochs
Training Accuracy
Test Accuracy

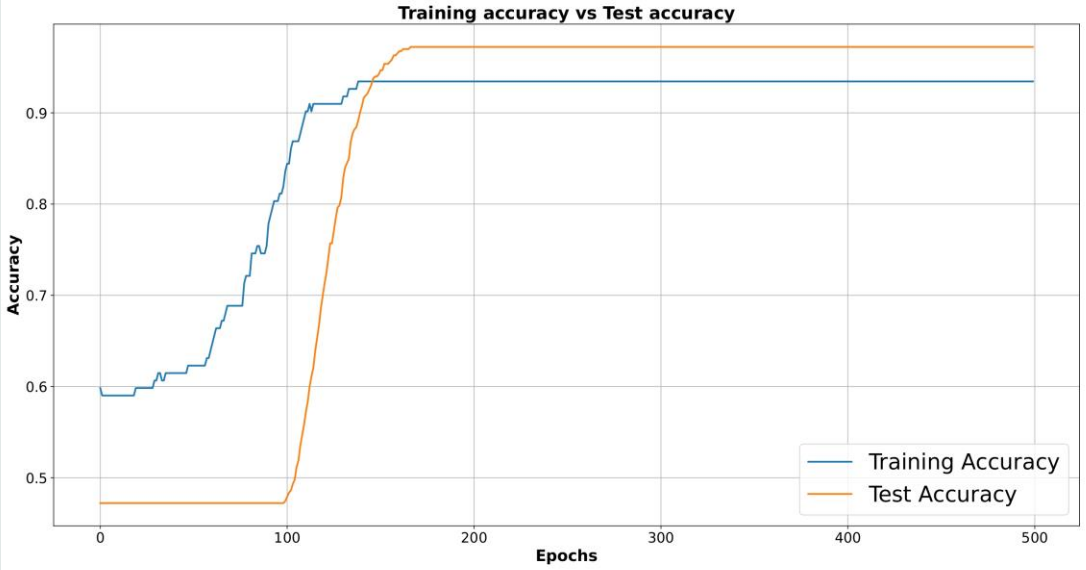
MONK 3





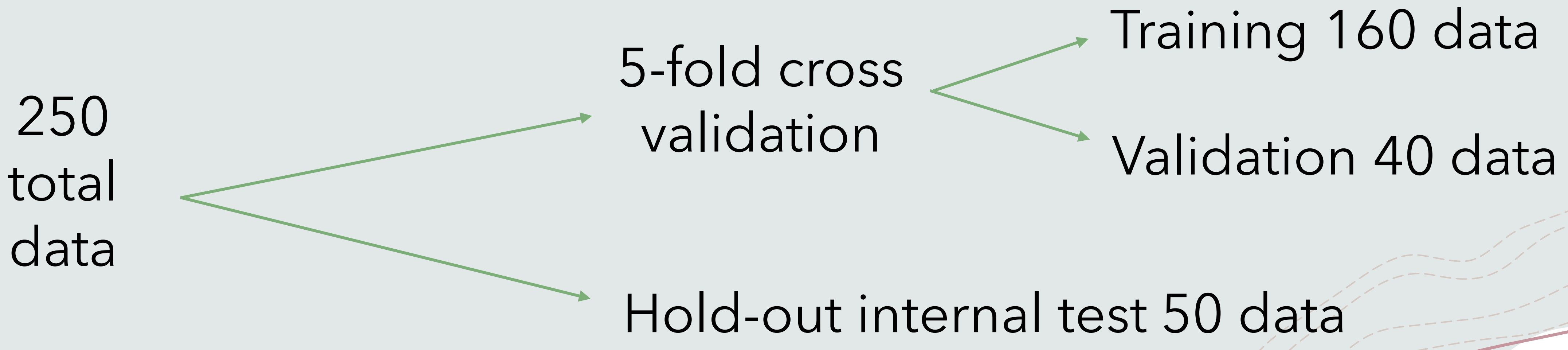
MONK 3 REG





CUP VALIDATION SCHEMA: DATA SPLITTING

- The dataset in split in **two sets**: train+validation set, being 80% of the original dataset, and test set, the remaining 20%
- The available data are 250. Therefore, at the end the internal test set is composed by 50 data, the validation set is composed by 40 data and the training set by 160 data
- In the model selection phase, a **5-fold cross validation** is used in order to find the best configuration of hyperparameters
- Once the best combination of hyperparameters was found, **retraining** was performed using the training set combined with the validation set as the new training set and the internal test set as the test set



CUP VALIDATION SCHEMA: MODEL SELECTION

- The chosen hyperparameters are: number of hidden layers, number of neurons for each layer, `learning_rate`, `activation_function`, `batch_size`, `lambda`, `alpha` and `opt_type`
- In order to find the best combination, 3 phases are planned out. The first one consists of a **hierarchical exploration** of the hyperparameter space finalized to eliminate non-promising areas
- Different types of architectures are tested, including deeper models with 4-5 hidden layers and shallower models with 1-2-3 hidden layers, varying the other hyperparameters
- This process was iterated for all three types of optimizers (`NAG`, `adam` and `None`) with the aim of building grids for each one. After completing the exploration, the `None` type is set aside, keeping only `NAG` and `adam`

Hyperparameter	Range
Units per layer	[16, 32, 64, 128, 256]
Activation function	[leaky ReLU, tanh]
Batch size	[1, 20, 32, 40, 64, 80, 160]

Hyperparameter	Range
Learning rate (η)	[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]
Alpha (α)	[0, 0.25, 0.5, 0.75, 1]
Lambda (λ)	[0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}]

CUP VALIDATION SCHEMA: MODEL SELECTION

- The second phase is a form of **successive halvings grid search** with coarse granularity to retain the 30 most promising hyperparameters combinations for `adam` and `NAG`. This approach allows to fix the architecture and enables the creation of finer grids to search
- Successive halvings algorithm is composed by the **iteration of three phases** (arm):
  1. performing the training algorithm of the network with all possible configurations of hyperparameters for a number of epochs equal to `min_resources = 5`;
  2. sorting the configurations in ascending order with respect to `val_error`;
  3. both **halving the number of possible configurations**, keeping only the best half in terms of lowest validation error, and **doubling `min_resources`**
- The process was iterated for 3 arms. One successive halvings algorithm took around 8 hours to complete
- Combinations with higher learning rate reached the lowest validation errors at the end of the third arm. It makes sense: lower learning rates were not sufficient epochs to decrease enough, while potentially being as capable of achieving good convergence with more time
- Therefore, 3 different successive halvings were performed both on `adam` and `NAG`, varying the learning rate in decades across three distinct ranges

CUP VALIDATION SCHEMA: MODEL SELECTION

ADAM

Hyperparameter	Range
Units per layer	[32, 64, 128, 256]
Activation function	[leaky ReLU, tanh]
Batch size	[1, 40]
Learning rate (η)	$[5 \cdot 10^{-3}, 10^{-3}, 5 \cdot 10^{-4}]$ $[10^{-4}, 5 \cdot 10^{-5}, 10^{-5}]$ $[5 \cdot 10^{-6}, 10^{-6}]$
Alpha (α)	[0.5]
Lambda (λ)	$[0, 10^{-5}, 10^{-3}]$

NAG

Hyperparameter	Range
Units per layer	[32, 64, 128, 256]
Activation function	[leaky ReLU, tanh]
Batch size	[1, 40, 160]
Learning rate (η)	$[10^{-3}, 5 \cdot 10^{-3}]$ $[10^{-4}, 5 \cdot 10^{-4}]$ $[10^{-5}, 5 \cdot 10^{-5}]$
Alpha (α)	[0, 0.25, 0.5, 0.75, 1]
Lambda (λ)	$[0, 10^{-6}, 10^{-4}, 10^{-2}]$

- At the end of the algorithm, the networks with the most promising hyperparameters have their training completed to `max_resources = 1000`, with `smoothness_check`, `stopping_check` and `overfitting_check = True`
- For any learning rate, `adam` consistently produced the best results for **shallow networks with 256 units per layer**, while NAG performed at its peak for **32 units per layer**

CUP VALIDATION SCHEMA: MODEL SELECTION

- Finally, three **fine-grained grid search** (two for `adam` and one for `NAG`) were implemented with the goal of finding the best hyperparameter combination. Each grid search was conducted with `epochs = 1000`, with `stopping_check`, `overfitting_check`, and `smoothness_check = True`

Optimization	Architecture and parameters	Learning rate (η)	Regularization
Adam	1 hidden layer, 256 neurons, leaky ReLU, online algorithm	$[10^{-5}, 2\cdot10^{-5}, 3\cdot10^{-5}, 4\cdot10^{-5}, 5\cdot10^{-5}, 6\cdot10^{-5}, 7\cdot10^{-5}, 8\cdot10^{-5}, 9\cdot10^{-5}, 10^{-4}]$	Elastic regularization, $\alpha = 0.5$, $\lambda = 10^{-5}$

Optimization	Architecture and parameters	Learning rate (η)	Regularization
Adam	3 hidden layer, 256 neurons, leaky ReLU, batch size = [16, 40, 64, 80]	$[5\cdot10^{-6}, 6\cdot10^{-6}, 7\cdot10^{-6}, 8\cdot10^{-6}, 9\cdot10^{-6}, 10^{-5}, 2\cdot10^{-5}, 3\cdot10^{-5}, 4\cdot10^{-5}, 5\cdot10^{-5}]$	Elastic regularization, $\alpha = 0.5$, $\lambda = 10^{-5}$

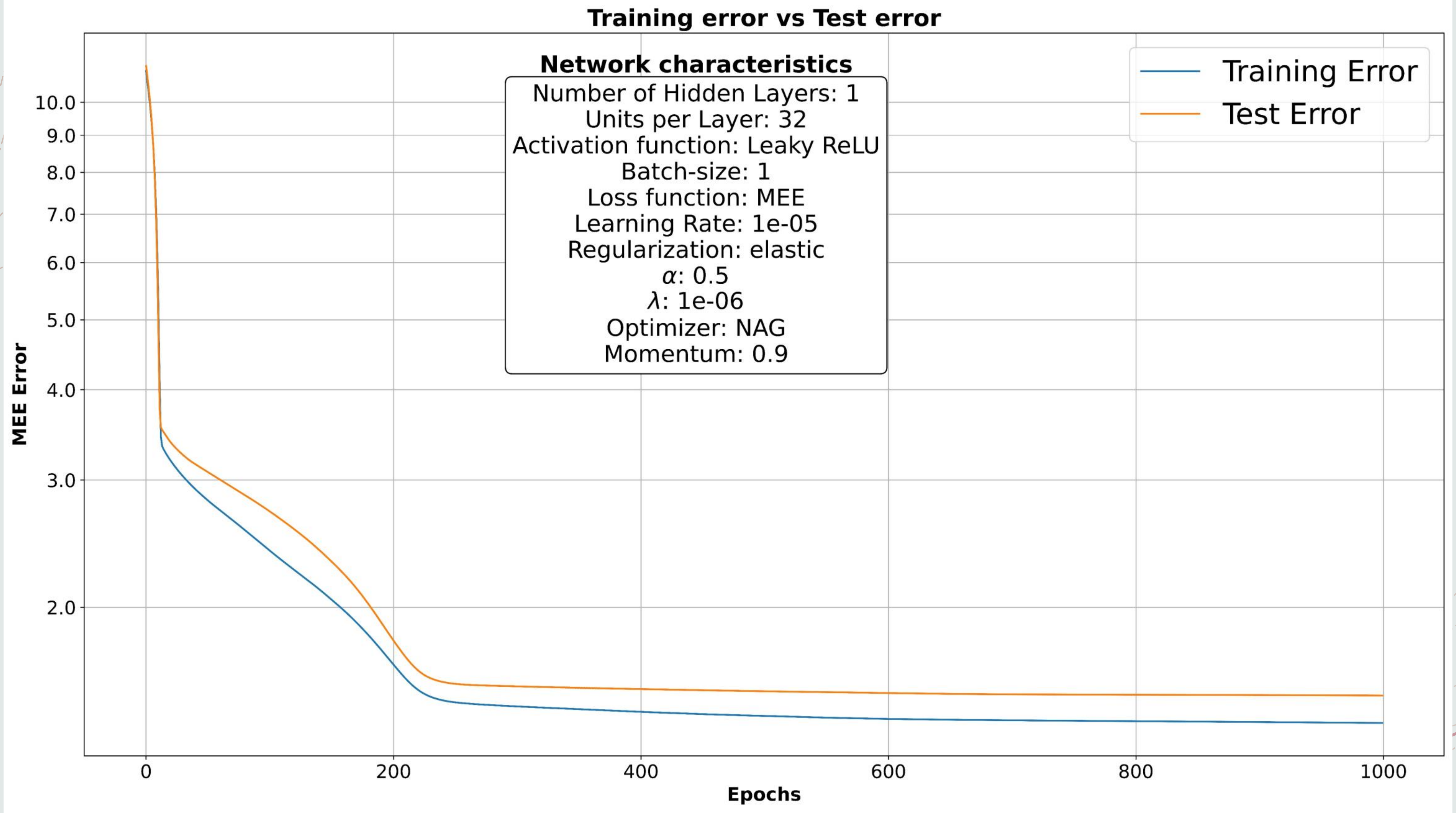
Optimization	Architecture and parameters	Learning rate (η)	Regularization
NAG	1 hidden layer, 32 neurons, leaky ReLU, batch size = [1, 32]	$[10^{-5}, 3\cdot10^{-5}, 5\cdot10^{-5}, 7\cdot10^{-5}, 9\cdot10^{-5}, 10^{-4}, 3\cdot10^{-4}, 5\cdot10^{-4}, 7\cdot10^{-4}, 9\cdot10^{-4}]$	Elastic regularization, $\alpha = 0.5$, $\lambda = 10^{-6}$

CUP VALIDATION SCHEMA: MODEL ASSESSMENT

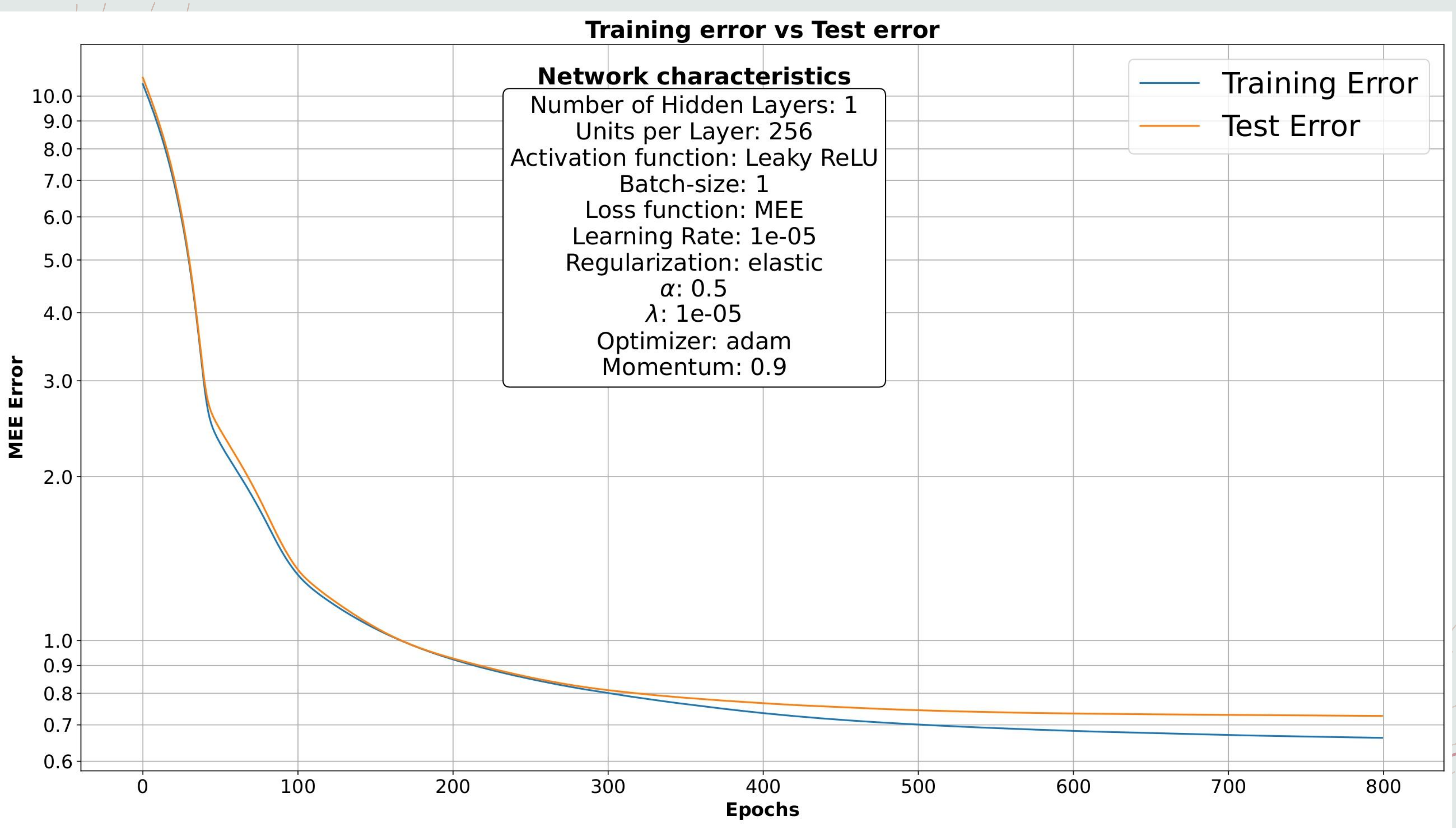
- The learning curves were checked for smoothness, excluding those that were not. Then, for each combination, the mean and variance of the validation error across folds were computed, selecting the combination that minimized these metrics
- Then, model assessment was performed, during which the two best hyperparameter combinations were chosen: one for `adam` and one for `NAG`
- Retraining was then carried out using the entire training + validation set as the new training set, and the internal test set as the test set. The whole process took less than 5 minutes for one configuration. The loss function is the **Mean Euclidean Error**

Architecture and parameters	Regularization configuration	Optimization configuration	MEE (TR/VL)	MEE (TR+VL/TS)	Epochs
1 hidden layer, 256 neurons, leaky ReLU, online algorithm, $\eta = 10^{-5}$	Elastic, $\alpha = 0.5$, $\lambda = 10^{-5}$	Adam $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 10^{-8}$	TR: 0.68 ± 0.02 VL: 0.77 ± 0.06	TR+VL: 0.66 TS: 0.73	1000 Overfit at 800
1 hidden layer, 32 neurons, leaky ReLU, batch size = 32 $\eta = 10^{-5}$	Elastic, $\alpha = 0.5$, $\lambda = 10^{-6}$	NAG $\mu = 0.9$	TR: 1.39 ± 0.02 VL: 1.5 ± 0.1	TR+VL: 1.38 TS: 1.51	2000 Early stop at 1000

NAG CUP RESULTS

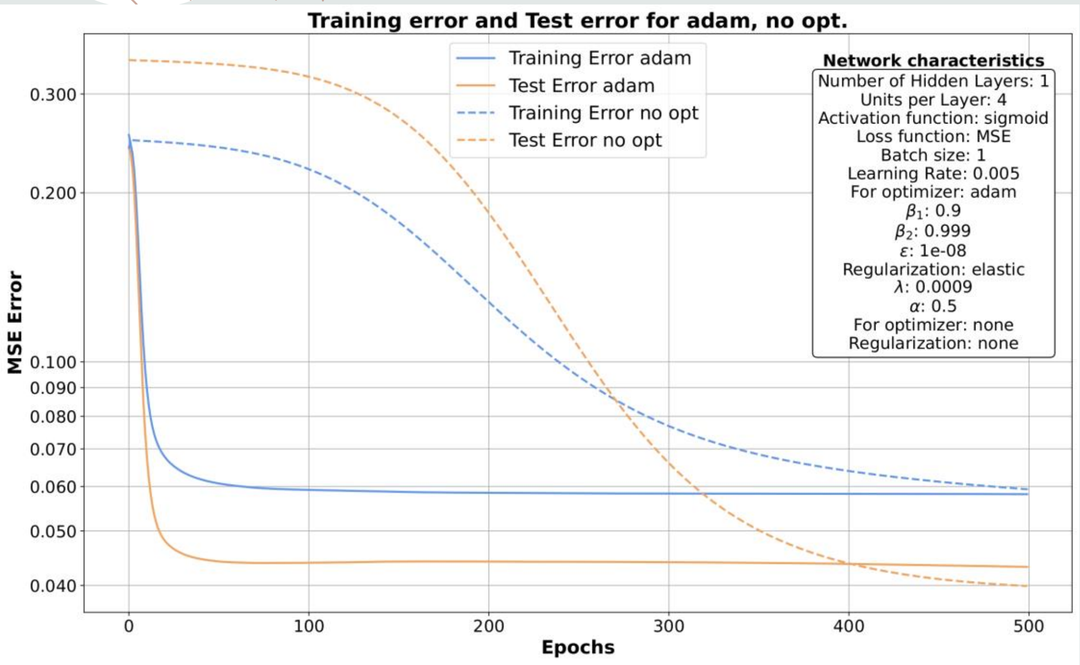


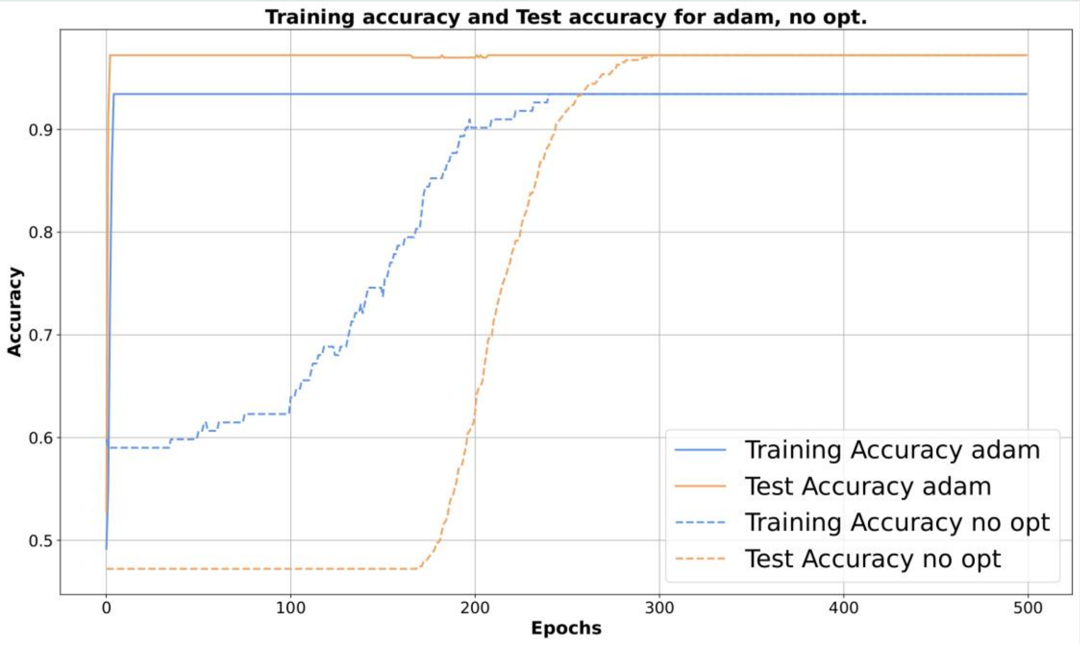
ADAM CUP RESULTS



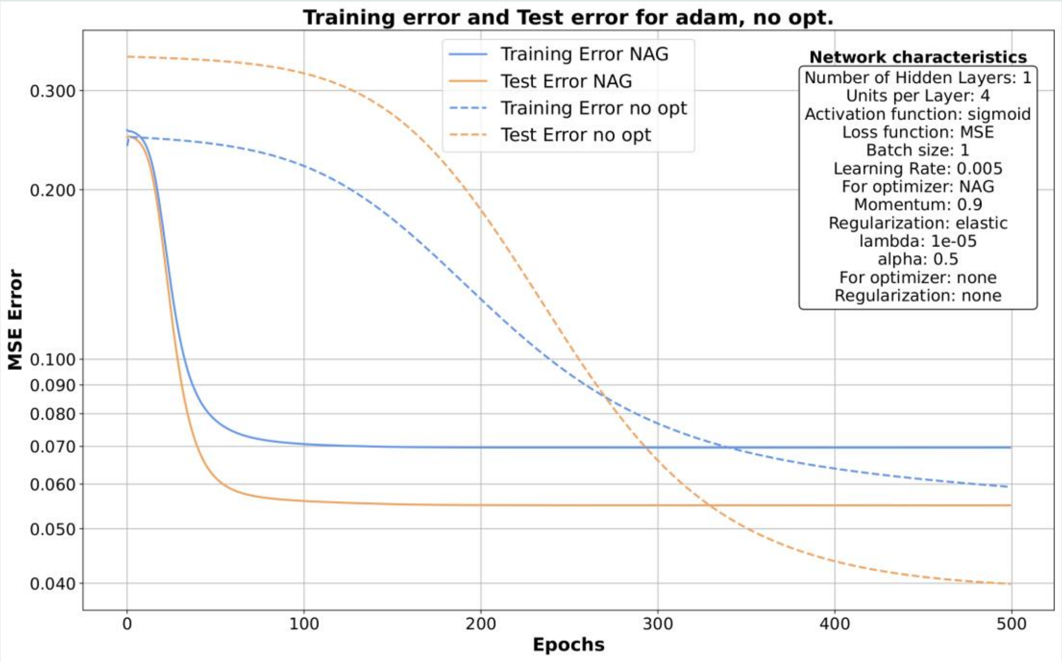
EFFECTS OF OPTIMIZATION ON MONK

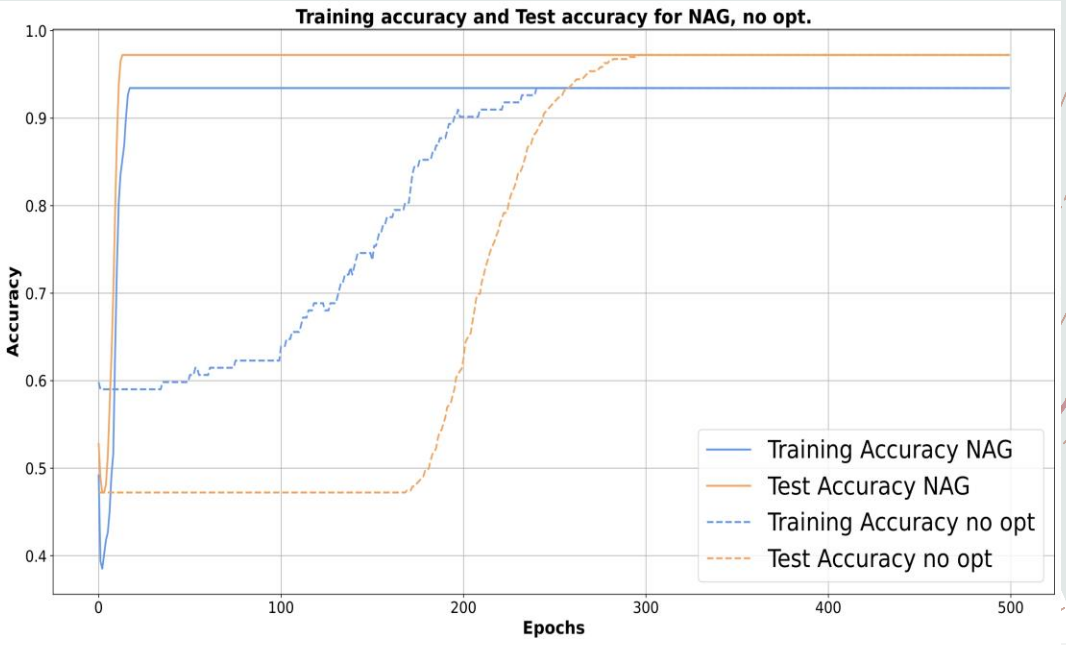
ADAM



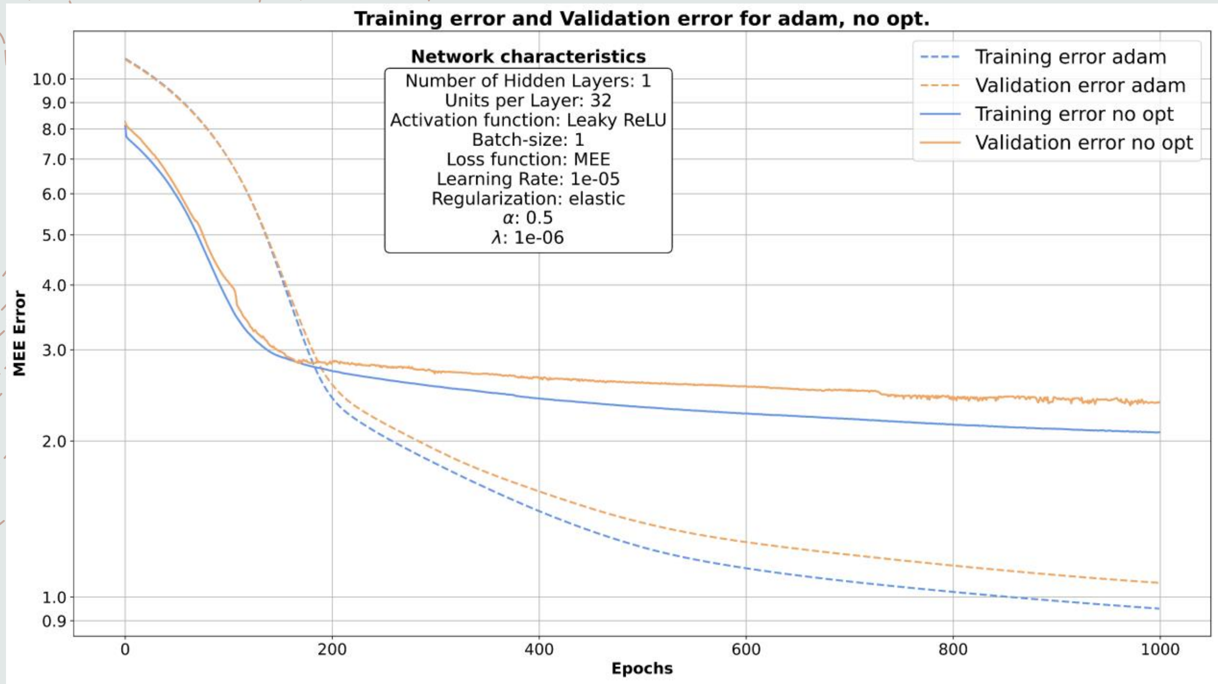


NAG



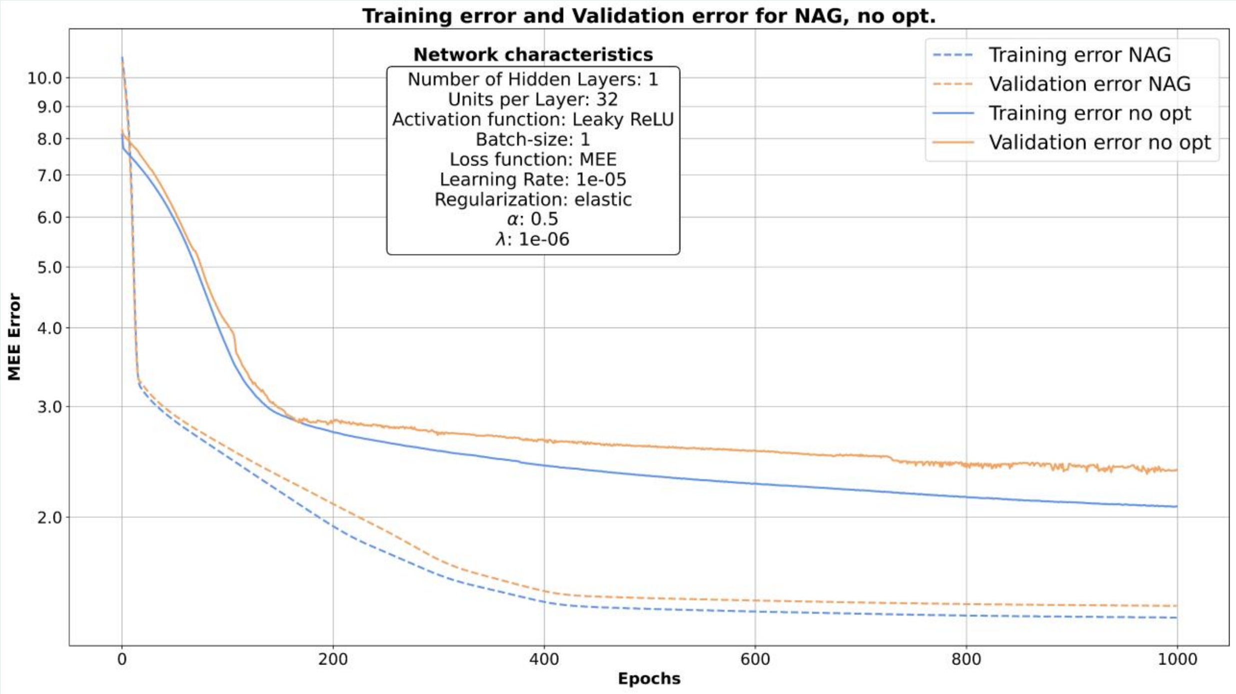


EFFECTS OF OPTIMIZATION ON CUP



ADAM

NAG



CONCLUSIONS

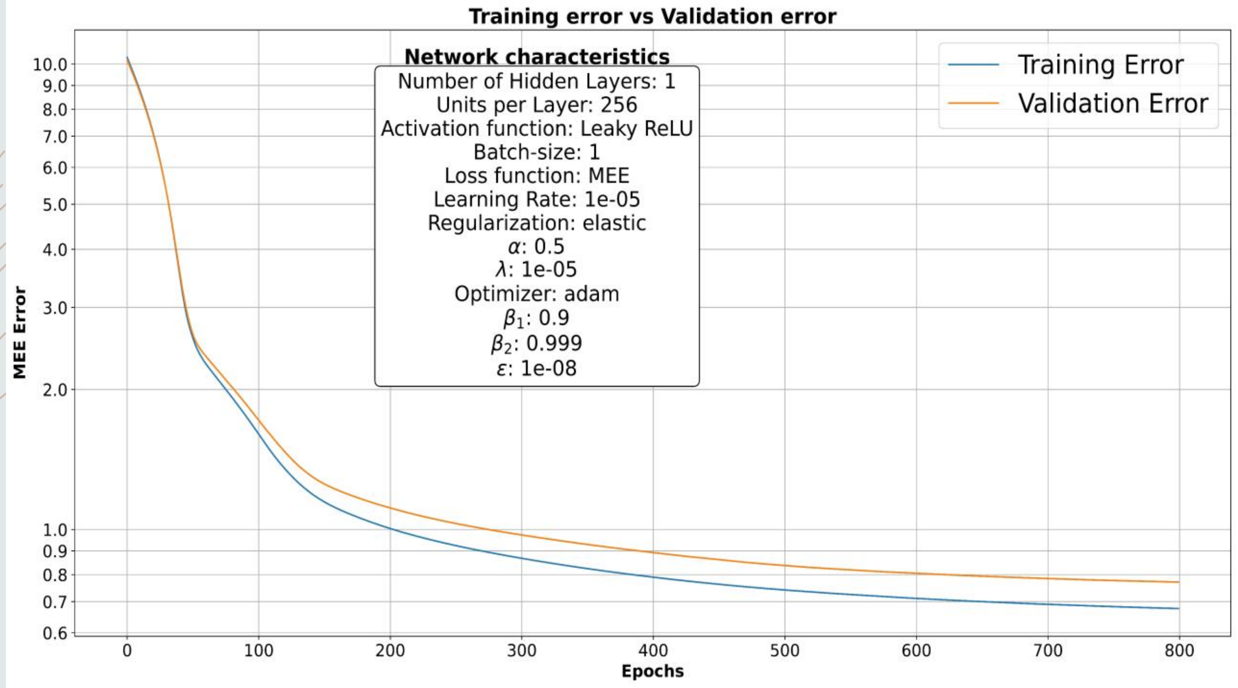
- Learning from theory to practice how to build a neural network from scratch has been extraordinarily academically formative
- Understanding what snowball effect means: how little changes in the initial conditions bring about massive differences in the end
- Equilibrium between digital, time and human resources is key
- Complexity and randomization aren't always synonyms of problems: it is often thanks to them that growth and learning are possible
- Time management and coordination are inestimable skills we hope to have polished during the project
- We would have liked to see the effect of the combination of NAG and Adam optimizers
- Blind Test Result: BG_peppers_ML-CUP24-TS.csv

BIBLIOGRAPHY

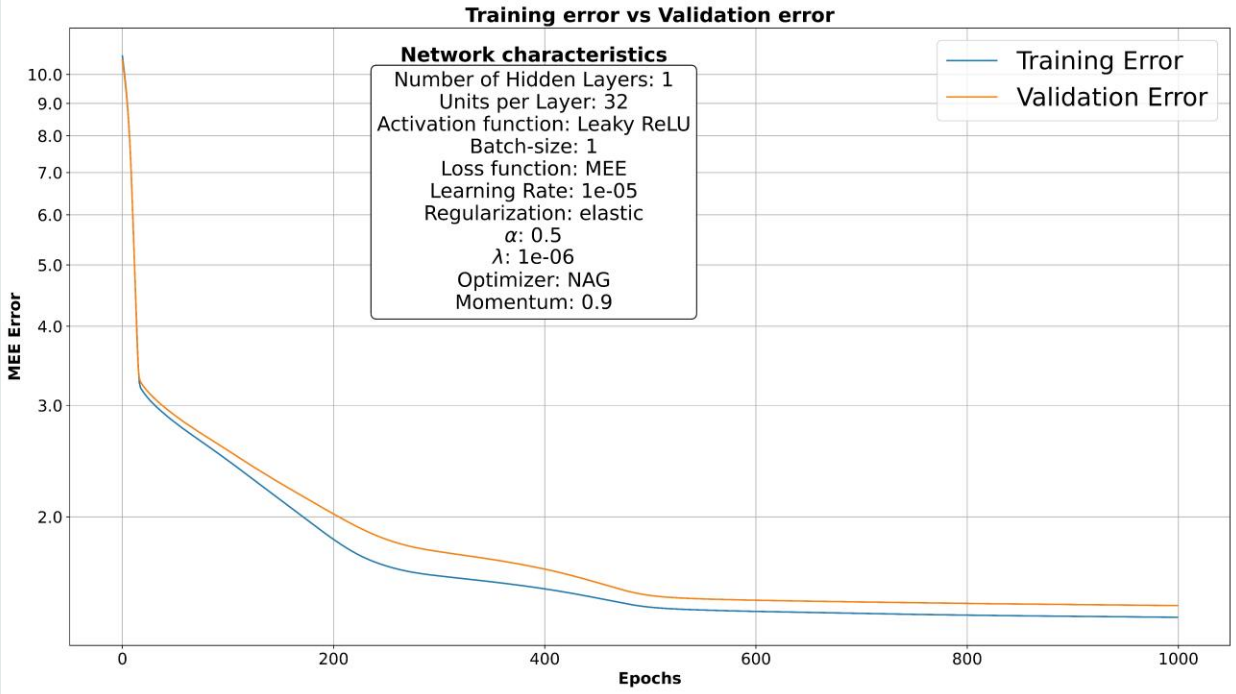
- [1] D. P. Kingma, J. L. Ba: Adam: a method for stochastic optimization. *arXiv*, **2015**, https://arxiv.org/pdf/1412.6980
- [2] T. Dozat: Incorporating Nesterov Momentum into Adam. *OpenReview*, **2016**, https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ
- [3] K. Jamieson, A. Talwalkar: Non-stochastic Best Arm Identification and Hyperparameter Optimization. *PMLR*, **2016**, https://proceedings.mlr.press/v51/jamieson16.html

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

APPENDIX: LEARNING CURVE BEST CONFIGURATIONS



ADAM



NAG