ML 2023 Project

Noemi Boni (M.Sc. in Digital Humanities)
n.boni1@studenti.unipi.it

Lorenzo Zaffina (M.Sc. in Physics)
l.zaffina@studenti.unipi.it

Team name: **Cantucci_e_Soppressata**

Type of project: **B**
Date: **14/02/2024**

Objectives

- **Aim:** Explore and compare performances of different models.

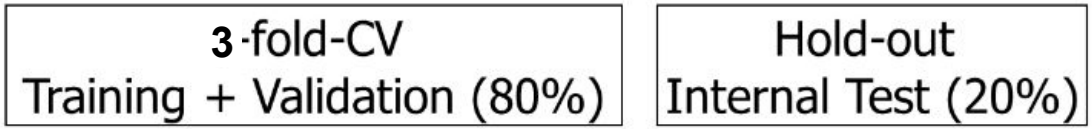
- **Models:**
  - **k-NN**
    - compare different neighbourhood sizes (k)
  - **NNs** trained with backpropagation
    - study the effect of different hyperparameters
    - study the effect of regularization techniques
    - compare different optimizers
  - **SVMs**
    - compare different kernels
    - study the effect of different hyperparameters

- **Software used:**
  - Pythorch (for NNs); Scikit Learn (for SVMs and k-NN)

Validation schema: data splitting

- For the **Monk:**
  - Perform 5-fold CV on the training data (for training and validation)
  - Stratified k-fold
  - Test on the separate test data

- For the **CUP:**
  - Split data in training+validation (80%) and internal test (20%)
  - Perform 3-fold CV on the training+validation data
  - For the selected final model: do a final retraining and then test on the internal test data to estimate performance



Weight Initialization (NN)

- Weights and biases are initialized according to the default initialization of Pythorch (uniform distribution).
- We tried multiple (3) random configurations of initial weights, and took the **mean** and **variance** of errors to evaluate the model.

$$\mathcal{U}(-\sqrt{k}, \sqrt{k})$$

$$k = \frac{1}{\text{in\_features}}$$

Stopping Conditions (NN)

- **Internal Criteria:** Stop when the training loss (Monk) or MEE (CUP) stops decreasing:
  - Decrease smaller than *1e-5* for monk, *1e-3* for the cup (Patience of 20 epochs).
  - Still indicating a max number of epochs (5000) to escape from slow convergence.

Regularization Schema (NN)

- **L2 Regularization:** we explored different values of **λ** during model selection.
- **Dropout:** we tried to implement dropout, but it resulted in noisy training curves dataset, so we chose not to use it. We suppose this behavior is due to the limited dataset and our network architecture.
- **Early Stopping:** we decided not to implement ES, in order not to sacrifice another portion of the already small dataset.

MONK BENCHMARKS

Considerations

- We used a **small** NN with **1 hidden layer** and few units (4/5).
- We used Relu() as activation function, it seemed to perform better with respect to Tanh().
- Mini-batch gradient descent (we tried different batch sizes).

- **Hyperparameters:**
  - Number of hidden units
  - Learning Rate
  - Momentum
  - **λ** (Regularization, only for monk 3)

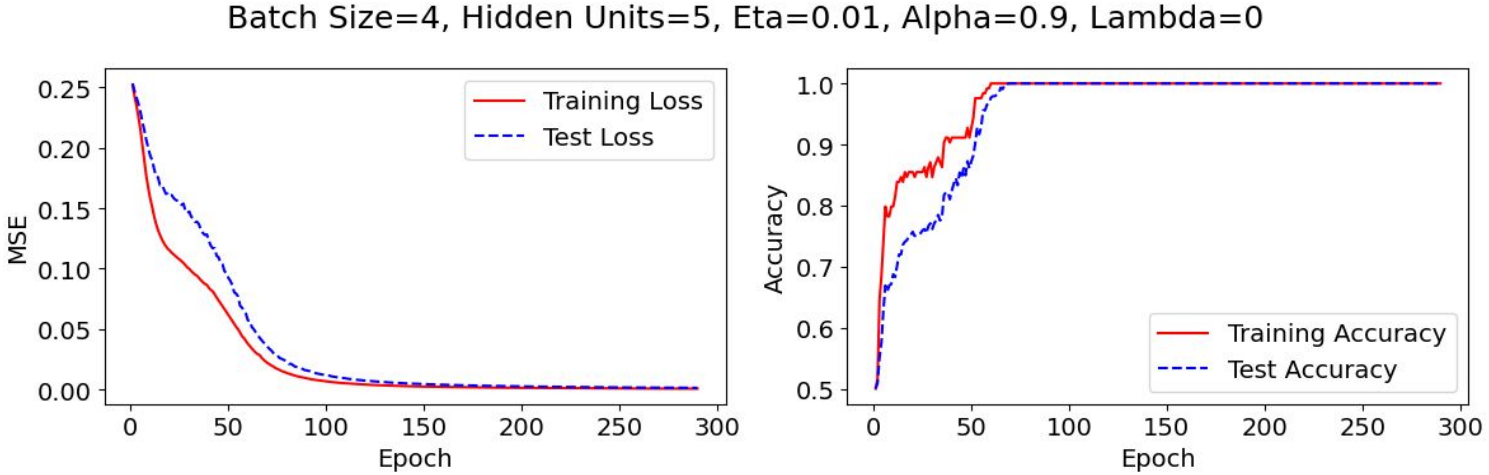
- For **monk 1 and 2** we manually tuned hyperparameters, since we easily found values that performed 100% training and validation accuracy.
- For monk 3 we performed a **small grid search**, and also compared the same model with and without regularization.
- The models were trained using **5** different initializations.

MONK Results - NN

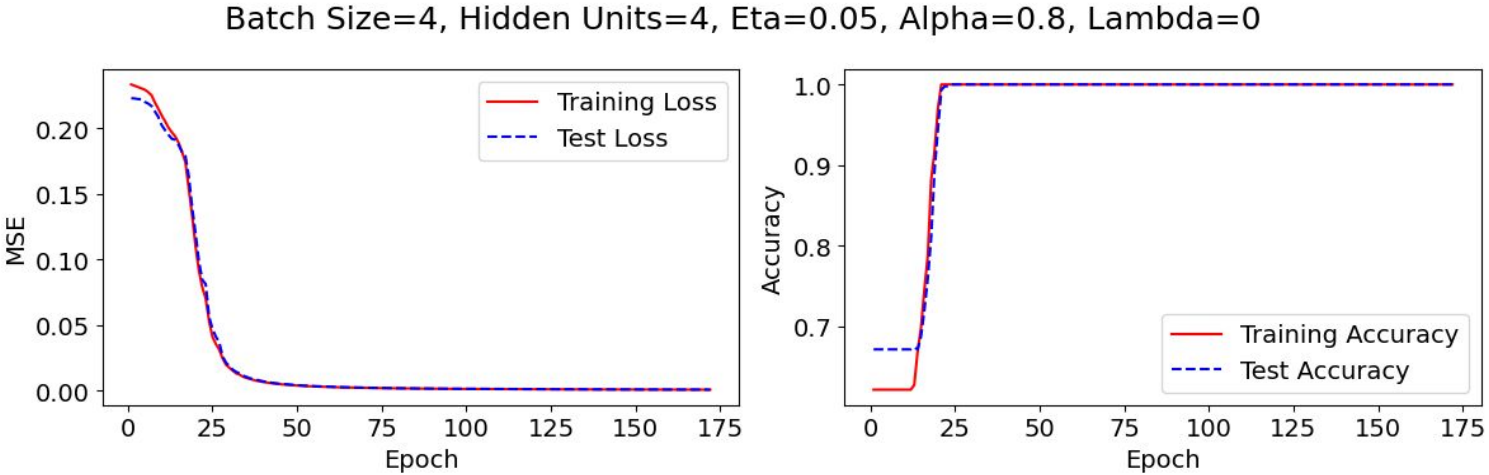
Task	Units	η	α	λ	Batch Size	MSE (TR/TS)	Accuracy (TR/TS)
MONK 1	5	0.01	0.9	-	4	0.00084 +- 0.00010 / 0.00173 +- 0.00031	100 / 100 % *
MONK 2	4	0.05	0.8	-	4	0.000356 +- 0.000095 / 0.00041 +- 0.00014	100 / 100 % *
MONK 3	5	0.003	0.9	-	16	0.0369 +- 0.0053 / 0.05006 +- 0.00076	96.39 +- 0.98 % / 93.89 +- 0.19 %
MONK 3 (with reg.)	5	0.003	0.9	0.01	16	0.0535 +- 0.0023 / 0.0409 +- 0.0028	93.44 % * / 97.22 % *

* No std. is reported because the same accuracy was obtained in all trials

Monk 1

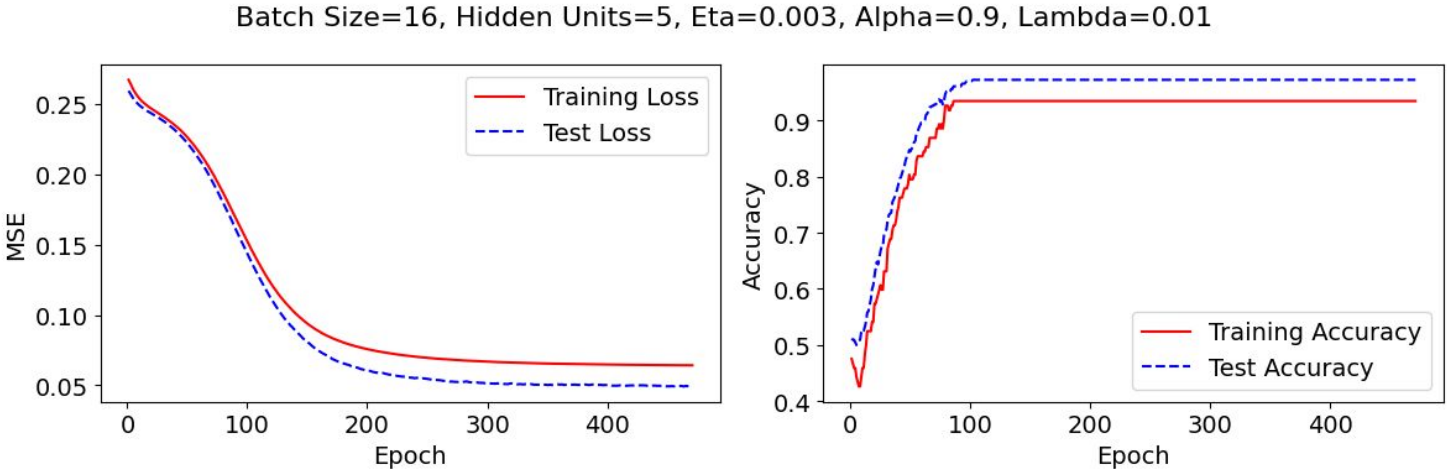

Batch Size=4, Hidden Units=5, Eta=0.01, Alpha=0.9, Lambda=0
MSE
Epoch
Training Loss
Test Loss
Accuracy
Training Accuracy
Test Accuracy

Monk 2


Batch Size=4, Hidden Units=4, Eta=0.05, Alpha=0.8, Lambda=0
MSE
Epoch
Training Loss
Test Loss
Accuracy
Training Accuracy
Test Accuracy

Monk 3

Monk 3
(with regularization)



MONK Results - Other Models

k-NN	k	weights	p	Accuracy (TR/TS)
MONK 1	23	‘distance’	1	100% / 80.8%
MONK 2	24	‘distance’	1	100% / 78.9%
MONK 3	11	‘distance’	1	100% / 91.0%

SVM	Kernel	C	degree	gamma	Accuracy (TR/TS)
MONK 1	polynomial	500	3	-	100% / 100%
MONK 2	rbf	100	-	0.1	100% / 81.5%
MONK 3	polynomial	0.11	4	-	95.9% / 97.7%

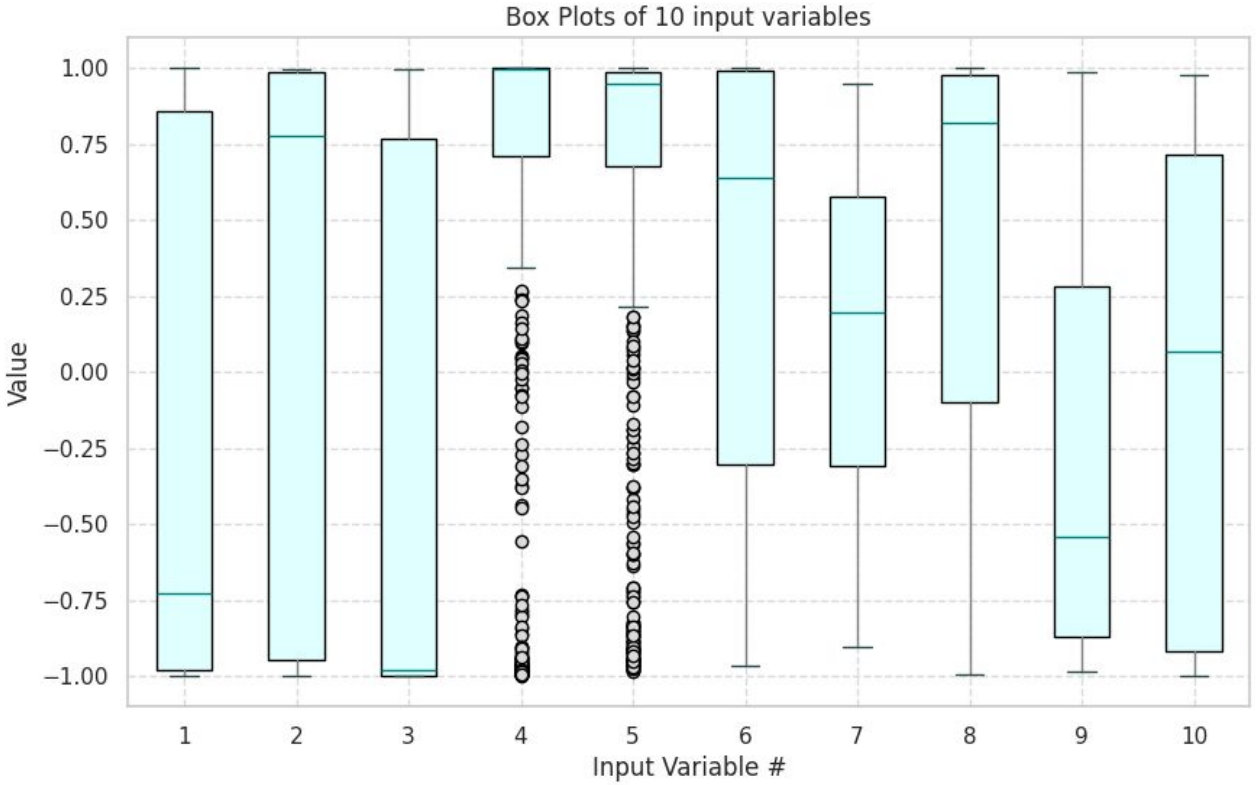
ML CUP

Data Exploration

As a first step we visualized the input data from the **training set**.

All values are in the range [-1,1].

We decided not to rescale them and use them as they were.



Start with a simple approach: K-nn

Hyperparameters:

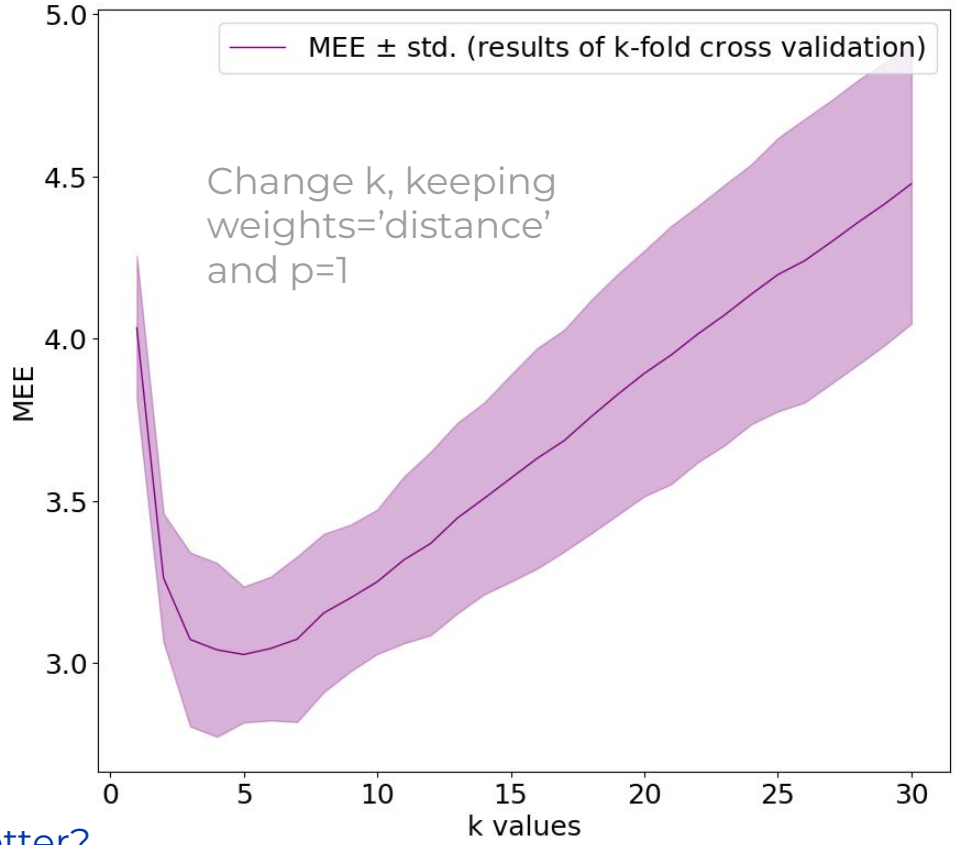
- **k** (number of nearest neighbors)
- **weights** (how to weight points' contribution in each neighborhood)
- **p** (power parameter for the Minkowski metric)

Grid Search Results:

- **k**=5
- **weights** = 'distance'
- **p**=1 (Manhattan)

Validation MEE
3.027 +- 0.2094

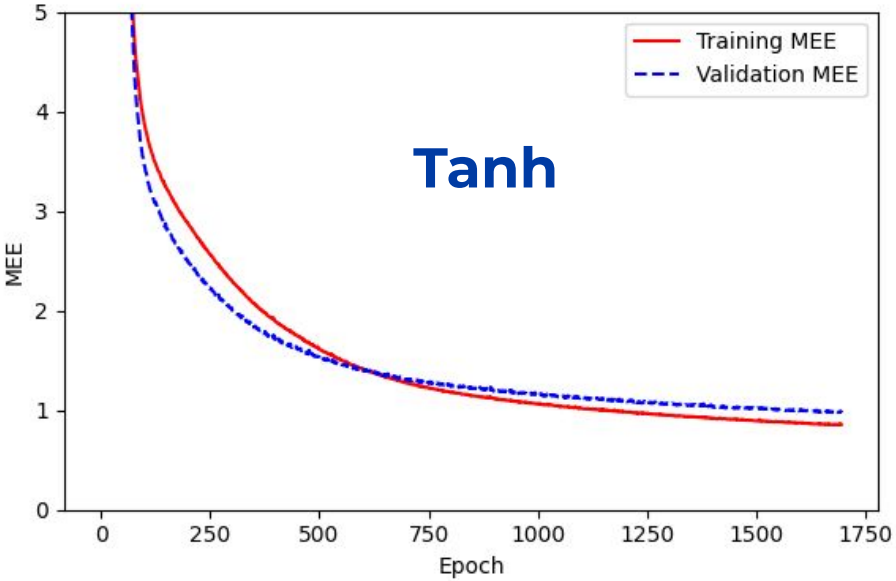
Can we do better?

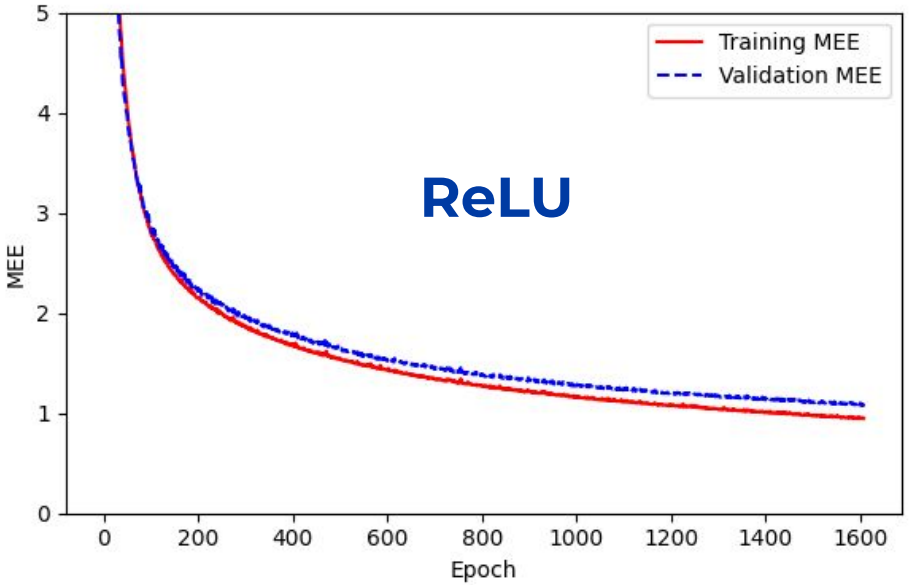


Neural Networks - first Manual Trials (SGD)

Changing other hyperparameters, different activation functions lead to comparable results, but in the case of ReLU the curve is typically less smooth. For this reason we decided to use Tanh in the following. We report here an example of training curves (curves of 1 of the 3 folds):

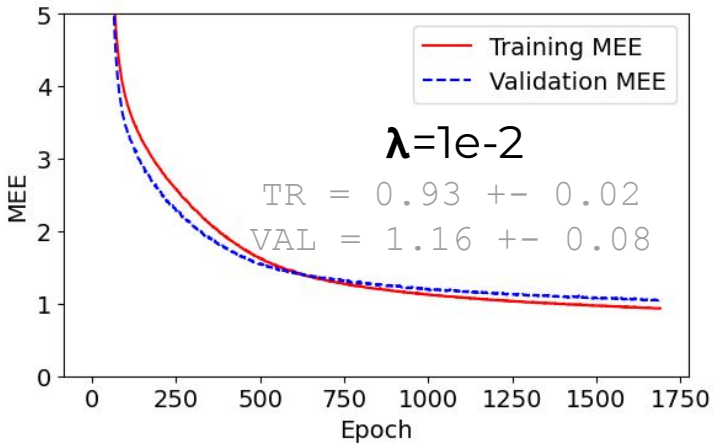
```
layers=3; #neurons=1000; lr=1e-4; alpha = 0.9; batch size = 128; lambda = 1e-3
```

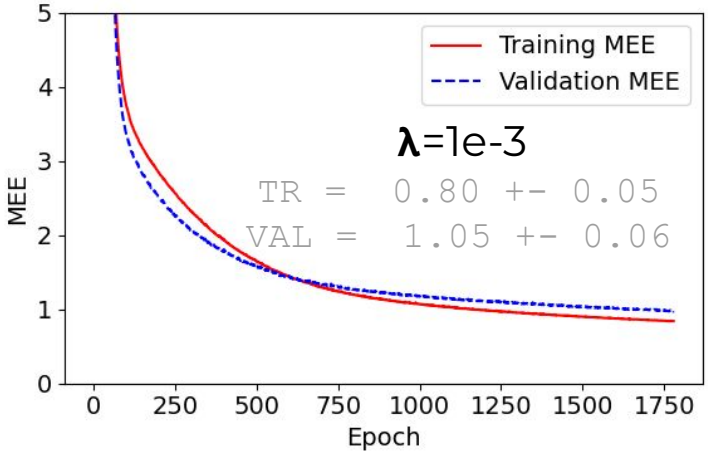


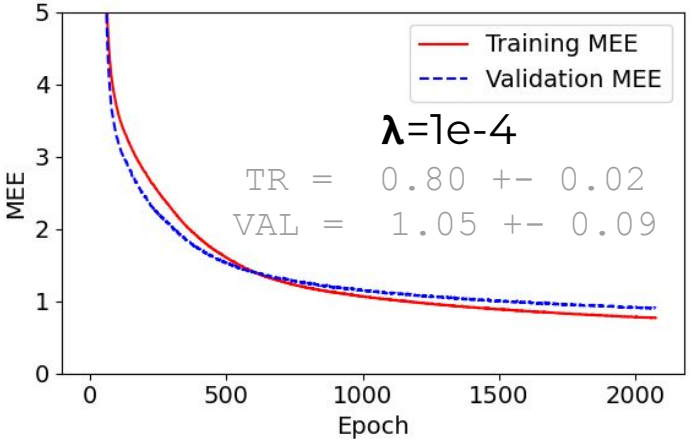


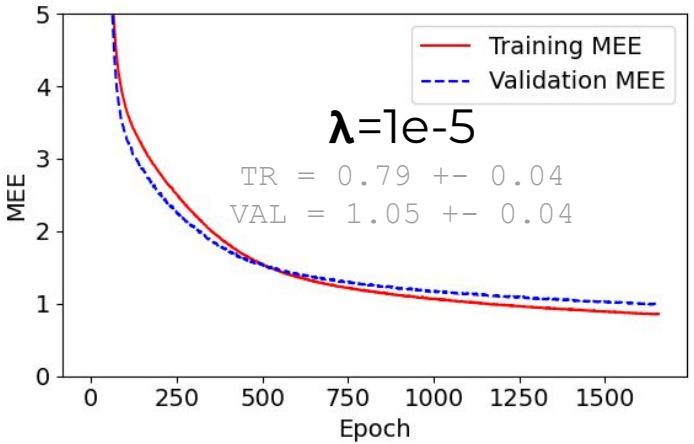
Effect of Regularization (SGD)

Activ: Tanh; layers=3; #neurons=1000; lr=0.0001; alpha = 0.9; batch size = 128;









* We report the training and validation curves over 1 of the 3 k-folds. Reported results are averaged over the 3 folds and 5 different initializations.

The effect of different values of **λ** is not very significant...

We decided to fix its value to **1e-3** in the following searches.

Coarse Grid Search (SGD)

First a **coarse grid:** (218 minutes to run)

```
hidden_neurons = [100,1000]
learning_rates = [1e-3,1e-4,1e-5]
momentums = [0.5,0.8]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [1,2,3]
```

- We try to find the **order of magnitude** of hyperparameters.
  - *hidden_neurons* refers to the **total** number of neurons in the net.
- with **lr**=1e-3 the curve was too noisy: we then excluded this value.
  - **Batch size** was fixed after few manual trials. We tried to avoid noisy curves and to speed up the training also thanks to parallelization.
  - We also fixed **λ** and activation function after initial considerations (see previous slides).

Best Hyperparameters:
activation=Tanh(); layers=3; neuron number=1000;
lr=0.0001; alpha = 0.8; batch size = 128; lambda = 0.001
Training MEE = 0.93 +- 0.042 | Validation MEE = 1.176 +- 0.050

Refined Grid Search (SGD)

Refined **grid:** (180 minutes to run)

```
hidden_neurons = [1000,2000,3000]
learning_rates = [1e-4]
momentums = [0.7,0.8,0.9]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [3]
```

Now we want to go more in depth, especially with the number of total neurons. And with the momentum values.

Best Hyperparameters:
activation=Tanh(); layers=3; neuron number=1000;
lr=0.0001; alpha = 0.9; batch size = 128; lambda = 0.001
Training MEE = 0.7983 +- 0.04923 | Validation MEE = 1.052 +- 0.04534

Try Different Optimizers: Adam

First a **coarse grid:** (108 minutes to run)

```
hidden_neurons = [100,1000]
learning_rates = [1e-4,1e-5]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [1,2,3]
```

activation=Tanh(); layers=3; neuron number=1000; lr=0.0001; batch size = 128; lambda = 0.001

Training MEE = 0.3211 +- 0.01588

Validation MEE = 0.7453 +- 0.1072

Refined **grid:** (90 minutes to run)

```
hidden_neurons = [1000, 2000, 3000]
learning_rates = [1e-4,2e-4,3e-4]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [3]
```

activation=Tanh(); layers=3; neuron number=1000; lr=0.0001; batch size = 128; lambda = 0.001

Training MEE = 0.3345 +- 0.01661

Validation MEE = 0.7594 +- 0.1104

Try Different Optimizers: RMSprop

First a **coarse grid:** (30 minutes to run)

```
hidden_neurons = [100,1000]
learning_rates = [1e-4,1e-5]
momentums = [0.7,0.8,0.9]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [3]
```

BEST: 1000, 1e-05, 0.9, 128, 0.001, Tanh(), 3

Training MEE = 0.297 +- 0.019
Validation MEE = 0.738 +- 0.098

Refined **grid:** (15 minutes to run)

```
hidden_neurons = [1000]
learning_rates = [1e-5,2e-5,3e-5]
momentums = [0.85,0.90,0.95]
batch_sizes = [128]
λ = [1e-3]
activations = [nn.Tanh()]
number of layers = [3]
```

BEST: [1000, 1e-05, 0.9, 128, 0.001, Tanh(), 3]

Training MEE = 0.284 +- 0.0089
Validation MEE = 0.721 +- 0.095

Advanced Techniques

1 - comparison between different optimizers

Optimizer	Batch Size	Hidden Layers	Activ.	total # hidden neurons	lr	momentum	λ	MEE (TR/VAL)	Epochs to converge (average)
SGD	128	3	Tanh	1000	1e-4	0.9	1e-3	0.798 +- 0.049 1.052 +- 0.045	1844
Adam	128	3	Tanh	1000	1e-4	-	1e-3	0.334 +- 0.017 0.76 +- 0.11	1675
RMSprop	128	3	Tanh	1000	1e-5	0.9	1e-3	0.2836 +- 0.0089 0.721 +- 0.095	1024

Advanced Techniques

2 - variable learning rates (and momentums)

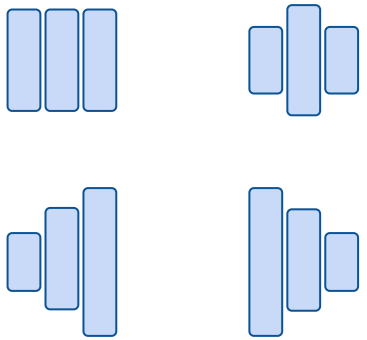
- **Fixed lr:**
  - SGD: Training MEE = 0.798 +- 0.049 | Val MEE = 1.052 +- 0.045
  - Adam: Training MEE = 0.334 +- 0.017 | Val MEE = 0.76 +- 0.11
  - RMS: Training MEE = 0.2836 +- 0.0089 | Val MEE = 0.721 +- 0.095
- **Manually Decaying lr and momentums:**
  - SGD: Training MEE = 0.592 +- 0.022 | Val MEE = 0.87 +- 0.09 *[lr_init=3e-4 factor=0.95, every 200 epochs, min_lr=1e-8]*
  - Adam: Training MEE = 0.31 +- 0.019 | Val MEE = 0.73 +- 0.10 *[lr_init=1e-4 factor=0.9, every 200 epochs, min_lr=5e-8]*
  - RMS: Training MEE = 0.2343 +- 0.0098 | Val MEE = 0.685 +- 0.085 *[lr_init=1e-5 factor=0.85, every 200 epochs, min_lr=5e-10] (on average 1969 epochs to converge)*
- **ReduceLROnPlateau:**
  - SGD: Training MEE = 0.582 +- 0.017 | Val MEE = 0.8551 +- 0.069 *[lr_init=3e-4 factor=0.5, patience=15] (on average 1150 epochs to converge)*
  - ADAM: Training MEE = 0.294 +- 0.013 | Val MEE = 0.74 +- 0.10 *[lr_init=1e-4, factor=0.9, patience=15] (on average 1732 epochs to converge)*
  - RMS: Training MEE = 0.2651 +- 0.0071 | Val MEE = 0.699 +- 0.096 *[lr_init=1e-5, factor=0.5, patience=15] (on average 983 epochs to converge)*

Advanced Techniques

3 -Different proportions of neurons between layers

Until now the total number of neurons was equally distributed between layers. What if we try to change this proportion?

- We tried different distributions of neurons among three layers.
- Keeping fixed other parameters (to the best one so far), we performed a grid search changing the proportion of neurons.



The best configuration was: `20%, 30% and 50% of neurons in each layer , with`

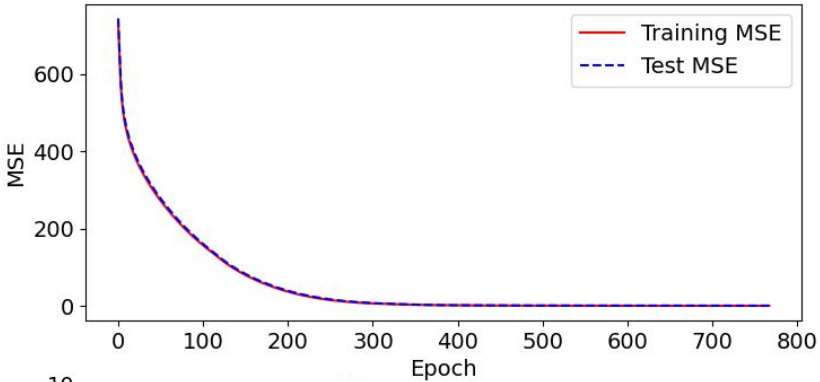
`Training MEE = 0.323 +- 0.010 | Validation MEE = 0.696 +- 0.091`

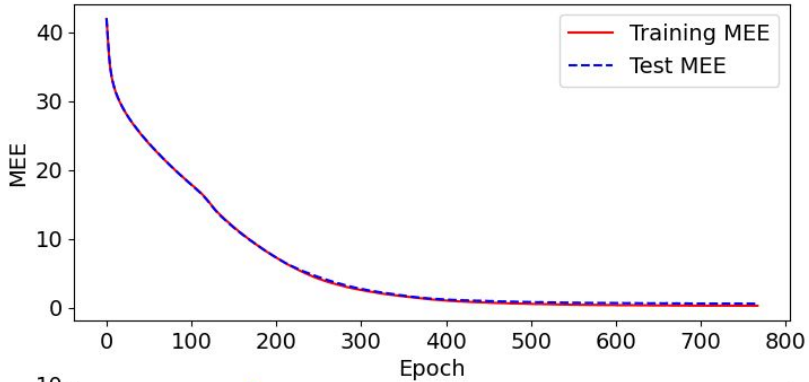
`The performance is comparable with the case of equally distributed neurons, so we keep that as our best model for NNs.`

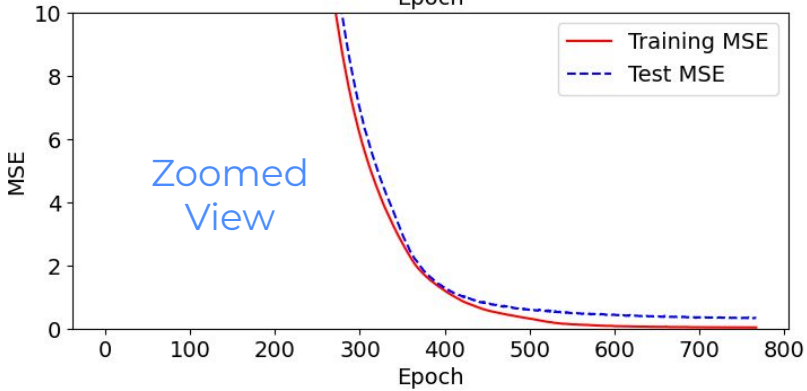
Training Curves (of the best NN model)

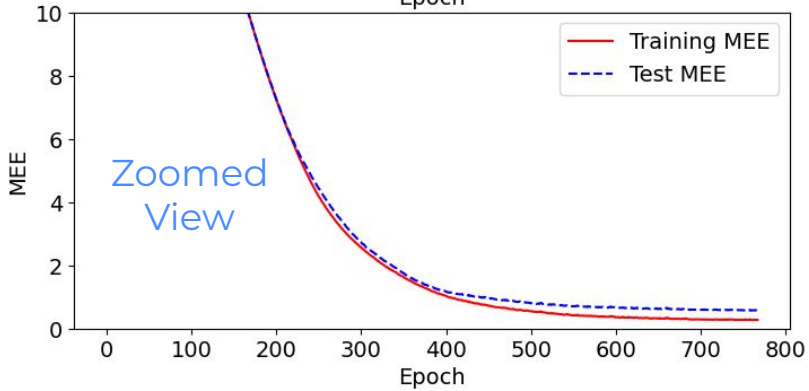
Retraining on all the TR set

Optim: RMS, **Batch Size:** 128, **Layers:** 3, **Activ:** Tanh, **Hidden Neurons:** 1000 (equally distributed), **λ**: 1e-3, **momentum:** 0.9, **ReduceLROnPlateau** (Initial LR: 1e-5, LR Decay Factor=0.5, Patience=15).







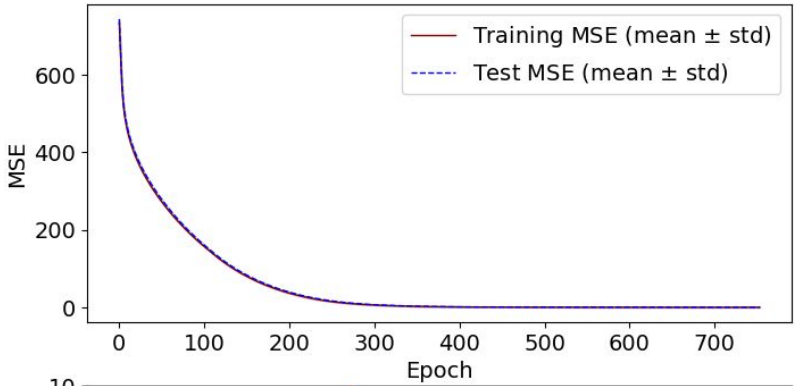


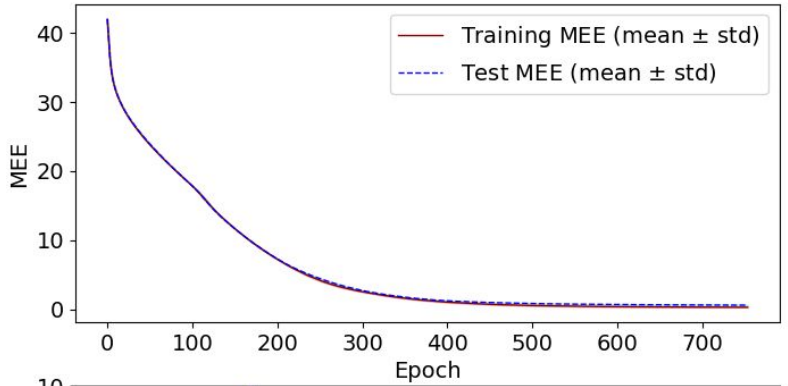
Training Curves (of the best NN model)

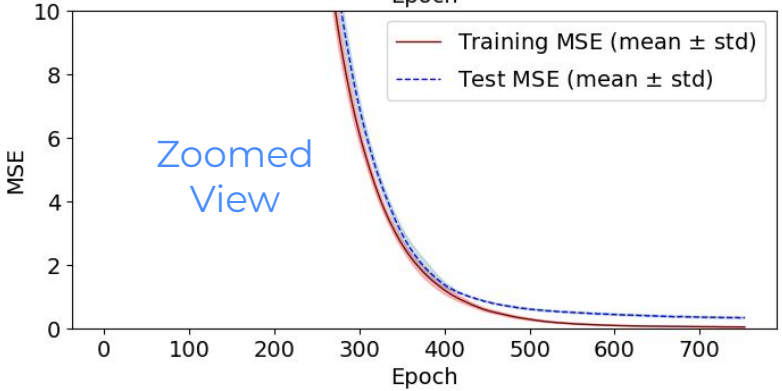
Mean and Variance over 5 random initializations

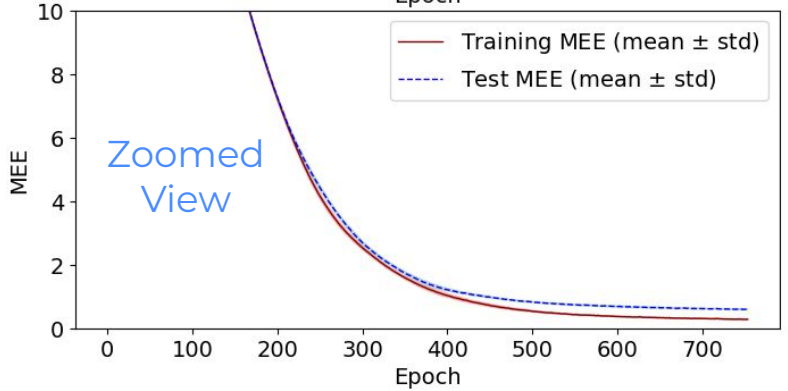
`Training MEE = 0.274 +- 0.022 | Internal Test MEE = 0.592 +- 0.022`

(These values are reported for didactic aim, but weren't considered for model selection purposes)









SVMs

$\gamma = 1/(2\sigma^2)$

1. **Find best kernel (RBF vs Poly):**

kernel = rbf (2 min)

```
Cs = [0.01,0.1,1,10,100,1000]
epsilons = [0.01,0.1,1,10]
gammas = ['scale',0.01,0.1,1,10]
Best Hp: ['rbf', 1000, 0.1, 'scale']
with Val. MEE = 0.640 +- 0.050
```

kernel = poly (34 min)

```
Cs = [1000,2000,3000,4000,5000]
degrees = np.arange(3,30,1)
epsilons = [0.1,0.2,0.3,0.4,0.5]
Best Hp: ['poly', 1000, 0.01, 5]
with Val. MEE = 0.82 +- 0.13
```

Best kernel is rbf

*if `gamma='scale'` then it uses 1 / (n_features * X.var()) as value of gamma

2. Focus on other hyperparameters (21 min)

```
kernel = rbf
Cs = [1000,2000,3000,4000,5000]
epsilons = [0.1,0.2,0.3,0.4,0.5]
gammas = ['scale',0.1,0.2,0.3,0.4,0.5]
Best Hp: ['rbf', 3000, 0.1, 'scale']
with Val. MEE = 0.617 +- 0.066
```

Effect of the regularization hyperparameter C

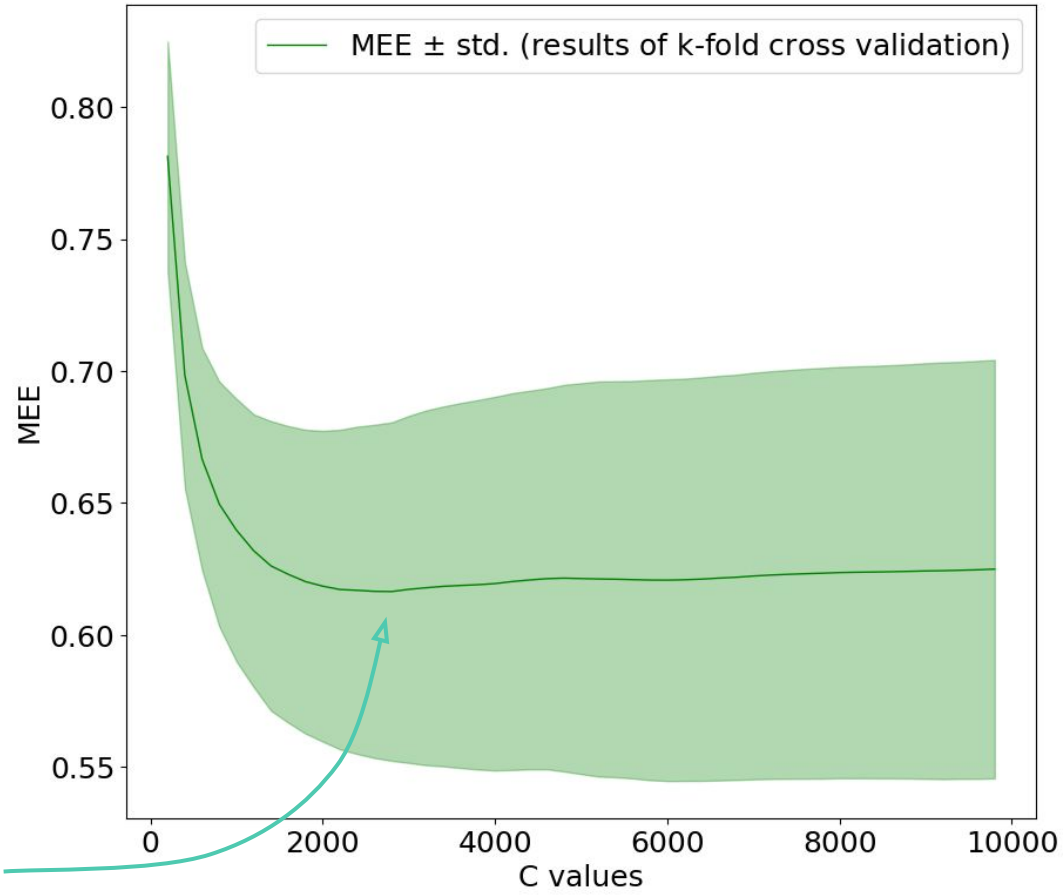
- We keep ε fixed at 0.1
- We explore C values in the range [200,10000] at steps of 200
- (40 mins to run)

Results:

`Best value of C: 2800;`

`with` **`Val. MEE = 0.616 +- 0.064`**

`This is our` **`best model`** `so far!`



Effect of the ε hyperparameter

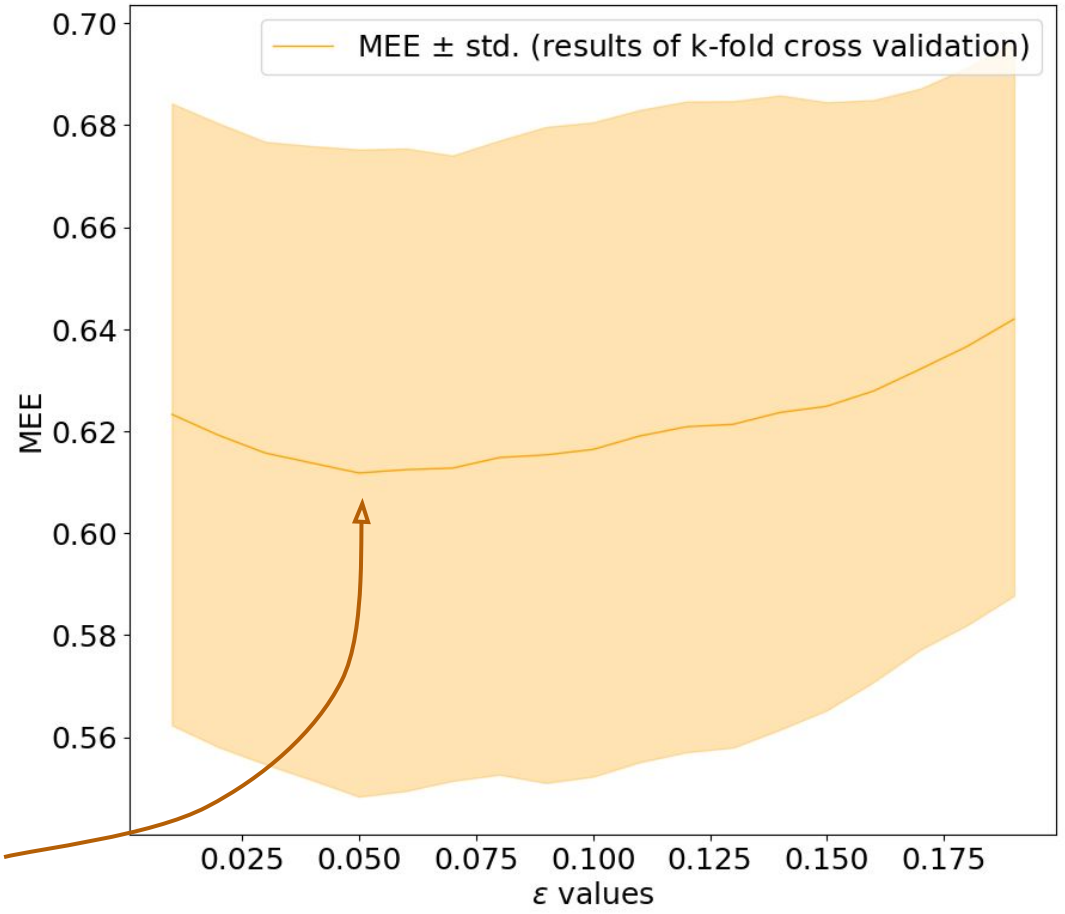
- We keep C fixed at 2800
- We explore **ε** values in the range [0.01,0.20] at steps of 0.01
- (9 mins to run)

Results:

Best value of **ε**: 0.05;

with **Val. MEE = 0.612 +- 0.063**

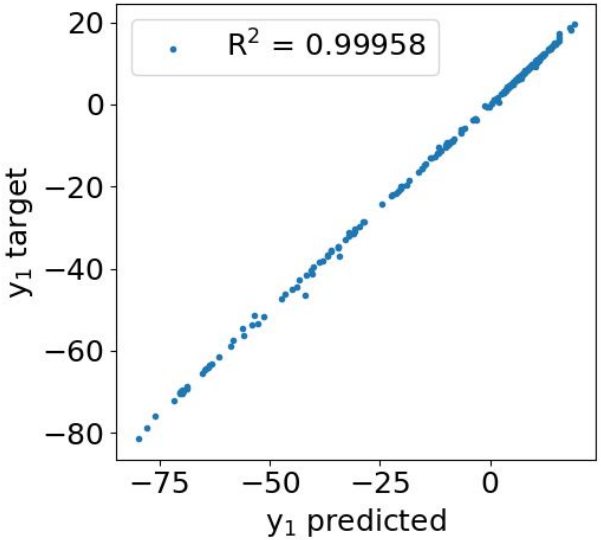
This is our **best model** so far!

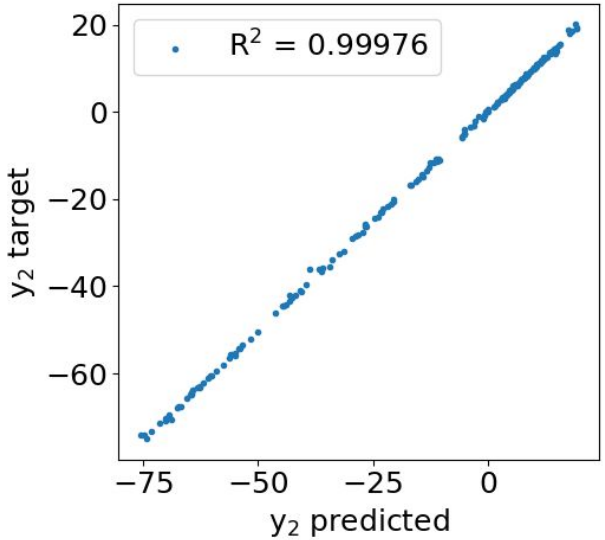


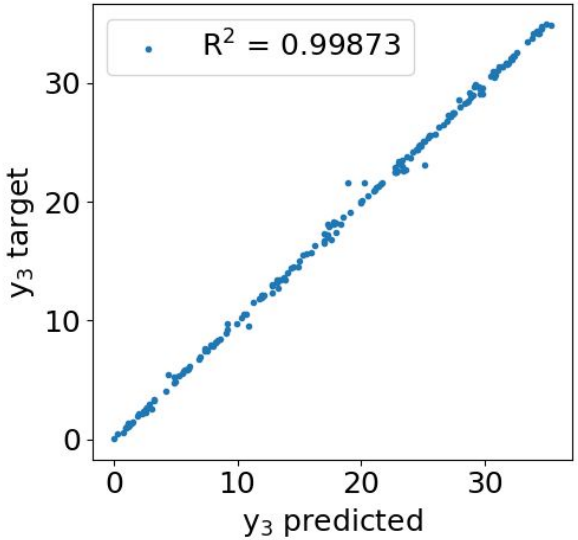
Our Final Model

(i.e. the one with lower validation MEE)

- 3 SVMs each with 639, 651, 603, support vectors respectively
- SVMs with **rbf** kernel, **ε** = 0.05, **C** = 2800;
- We retrained it on all the training data
- We studied the performance using the **internal test set:**







Our Final Model

- 3 SVMs each with 639, 651, 603, support vectors respectively
- SVMs with **rbf** kernel, **ε** = 0.05, **C** = 2800;
- We retrained it on all the training data
- We studied the performance using the **internal test set:**

Training MEE = 0.1696 +- 0.0039

Validation MEE = 0.612 +- 0.063

Internal Test MEE = 0.5097

Final Retraining: before using our model on the blind test set, we **retrained** it on **all the data**, including the internal test set.

Blind Test Results: *Cantucci_e_Soppressata_ML-CUP-23-TS.csv*
Team Name: *Cantucci_e_Soppressata*

Conclusions

- **Knn**
  - We found that this approach was not suitable for our data distribution.
  - Still it was worth to try and study the effect of k.
- **NN**
  - Using **RMSprop** as optimizer, we obtained better results (both in terms of MEE and in terms of efficiency).
  - Adding more neurons is useful, but up to a certain point.
  - Changing the proportions of neurons between layers doesn't improve our results in a significant way.
- **SVM**
  - We obtained better results w.r.t. the other models we tried (comparing validation results).
  - Improvements also in terms of computational time.
  - We keep in mind that SVM is a special case of a NN...

What did we learn?

- The combination of strategies to try are endless.
  - Grid searches are time consuming!
  - We had to make some choices, sacrificing the exploration of some possibilities :(
  - We tried our best to follow a principled approach in making our decisions.
  - It is important to always keep an eye on the training curve, sometimes better MEEs don’t guarantee a smooth curve.
- SVMs are powerful models.
  - They showed to be more suitable than NNs (at least better than the NNs we were able to explore) for our task.
  - They are efficient models, and easier to code :)
  - More than a half of the data revealed to be support vectors, less sparse than we thought.

Bibliography

- **Pythorch Documentation:** https://pytorch.org/docs/stable/index.html
- **Scikit Learn Documentation:** https://scikit-learn.org/stable/index.html
- **Adam:** https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
- **RMSprop:** https://pytorch.org/docs/stable/generated/torch.optim.RMSprop.html
- **MONK:** Thrun, S. & Bala, Jerzy & Bloedorn, Eric & Bratko, Ivan & Cheng, J. & De Jong, Kenneth & Džeroski, Sašo & Fahlman, Scott & Fisher, Doug & Hamann, R. & Kaufman, Kenneth & Keller, S. & Kononenko, I. & Kreuziger, J. & Mitchell, T. & Pachowicz, P. & Reich, Yoram & Vafaie, Haleh & Wnek, J.. (1992). The MONK's Problems A Performance Comparison of Different Learning Algorithms.

Appendix

Monk Grid Searches

k-NN:

```
ks = range(1, 31)
weightss=['uniform','distance']
ps = range(1,10)
```

Svm:

```
#first 2 coarse grid search (1 for rbf
and 1 for poly)

kernels = ['rbf']
Cs = [0.01,0.1,1,10,100,1000]
gammas = ['scale',0.01,0.1,1,10]

kernels = ['poly']
Cs = [0.01,0.1,1,10,100,1000]
degrees = np.arange(3,30,1)

#then a finer grid search, depending on
the best kernel
```

Monk Grid Searches

NN (Monk 3):

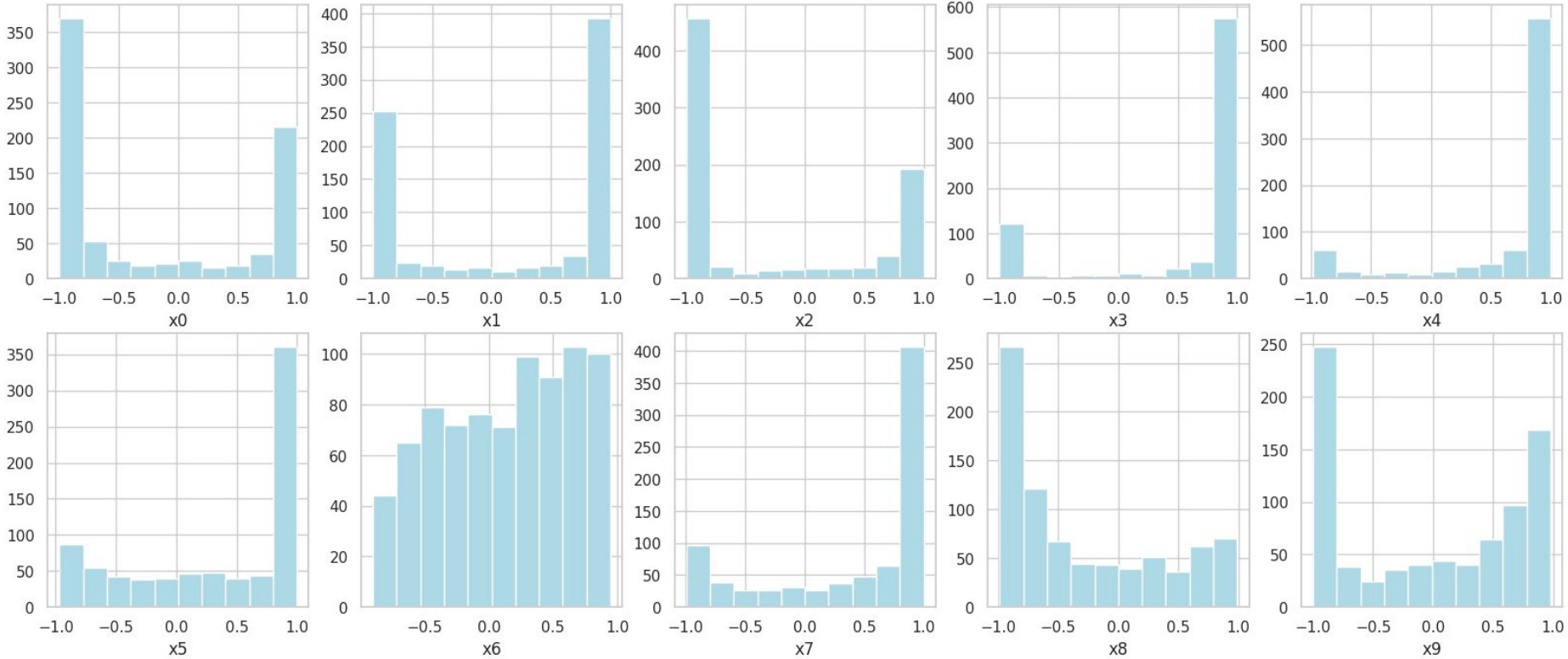
```
hidden_sizes = [5]
learning_rates = [0.1,0.01,0.001]
momentums = [0.7,0.8,0.9]
batch_sizes = [16]
reg_coeffs = [0]
# Best [5, 0.001, 0.9, 16, 0]

hidden_sizes = [5]
learning_rates = [0.001,0.002,0.003]
momentums = [0.7,0.8,0.9]
batch_sizes = [16]
```

NN (Monk 3 with regularization):

```
hidden_sizes = [5]
learning_rates = [0.003]
momentums = [0.9]
batch_sizes = [16]
reg_coeffs = [1e-2,1e-3,1e-4]
```

Histograms of the input data



Grid search for knn

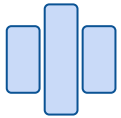
```
ks = range(1, 31)
weightss=['uniform','distance']
ps = range(1,10)
```

Grid Searches for ReduceLROnPlateau

- 2 additional hyperparameters:
  - the factor of reduction of the lr
  - the patience (#epochs)

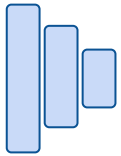
- Keeping the other model hyperparameters fixed, we explored with a grid search:
  - initial lr = [1e-4,2e-4,3e-4] for sgd and adam, [1e-5,2e-5,3e-5] for rsmprop
  - factors = [0.5,0.6,0.7,0.8,0.9]
  - lr_patiences = [5,10,15]

Grid Searches for layers proportions



[[0.1,0.8,0.1],[0.2,0.6,0.2],[0.25,0.50,0.25],
[0.2,0.7,0.1],[0.1,0.7,0.2],[0.3,0.6,0.1]]

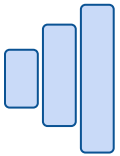
Best = [0.25, 0.5, 0.25]
→ training_mee = 0.2612 +- 0.0024 | val_mee = 0.75 +- 0.10



[[0.8,0.1,0.1],[0.7,0.2,0.1],[0.6,0.3,0.1],[0.6,0.2,0.2],
[0.8,0.15,0.05],[0.5,0.4,0.1],[0.5,0.25,0.25], [0.5,0.3,0.2]]

Best = [0.5, 0.25, 0.25] →

training_mee = 0.231 +- 0.010 | val_mee = 0.73 +- 0.10



[[0.1,0.1,0.8],[0.1,0.2,0.7],[0.1,0.3,0.6],[0.2,0.2,0.6],[0.05,0.15,0.8],[0.1,0
.4,0.5],[0.25,0.25,0.5], [0.2,0.3,0.5]]
[[0.2,0.3,0.5],[0.25,0.25,0.5],[0.15,0.35,0.5],[0.15,0.3,0.55],[0.1,0.25,0.55],

[0.05,0.40,0.55],[0.25,0.35,0.4]]

[0.25, 0.25, 0.5] → training_mee = 0.323 +- 0.01523 | val_mee = 0.6973 +-
0.1045
[0.2, 0.3, 0.5], with training_mee = 0.323 +- 0.010 | val_mee = 0.696 +- 0.091

Our Computers:

Noemi:

- **CPU:** Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
- **GPU:** NVIDIA RTX 3050 Ti Laptop
- **RAM:** 16,0 GB (15,9 GB utilizzabile)

Lorenzo:

- **CPU:** Intel® Core™ i5-8265U CPU @ 1.60GHz × 8
- **GPU:** NVIDIA Corporation GP108M [GeForce MX150]
- **RAM:** 8 GB