ELMvsNN

Machine Learning course a.a. 2020/2021 (654AA)

Luca Strazzera, l.strazzera@studenti.unipi.it
Jacopo Bandoni, j.bandoni@studenti.unipi.it

Date: *25/01/2021*
Type of project: **A with CM**

Abstract

In this document we are going to illustrate the work done for the machine learning course project. We implemented a python library to build neural networks, with different typologies and methods. Indeed, we built techniques to optimize hyperparameters and validate models comparing their score.

1 Introduction

Our aim is to find the neural network model that better generalize on the given datasets: the well know MONK's dataset [1], a classification problem used to settle our implementation, and ML CUP dataset, a regression problem where focus is to achieve the best result in a blind test set, given as private dataset.
We use the library developed by our self, coded from scratch to obtain a full control over the parameters. We implement a free choice Neural Network, called FFNN, starting from Extreme Learning Machine, called ELM, approach to the deeper model, with the possibility to change solver. It is interesting to show how much is the gain to use a refined version of Steepest Gradient Descent to explore the importance of the momentum approaches, as well as investigate if a direct solver as Cholesky can compete versus the standard approach. We fully implement the control of hyperparameters to allows the exploration in the choices made.

2 Method

Considering that we have chosen the project A with CM, no external tools were used to do inner process. The language selected is Python, easier to read and to mantein, libraries are: numpy, joblib and matplotlib. The first to use his matrix representation, the medium to exploit the parallelism in our code and the latter to plot graphs. The following is a summary list of what we implemented from scratch:

- **Architectures**: possibility to build either a *Feed Forward Neural Network* with any number of hidden layer and any number of neurons or an *Extreme Learning Machine* [2] with one hidden layer and any number of neurons, this special version of a general NN is trainable only in the output weights, the first layer is fixed and randomly generated at building time.
- **Activations**: possibility to choose betwen *Sigmoid* and *Tanh*, to not lose differentiability.
- **Training Algorithms**: possibility to chose from the following list, all configures to work with MSE as error function.
  - `Steepest gradient descent`: the well know optimization algorithm, used for Feed forward Neural Network. We add the possibility to tune *momentum* and *penalty l2 term*. Can be used in three version: Batch, mini-batch and online. The stopping condition is based either on the number of epochs or by early stopping.
  - `Adam`[3]: a variation of SGD, used for Feed Forward Neural Network, optimization algorithm that uses *two momentum* in adaptive way. We add the possibility to tune *penalty l2 term*. Can be used in three version:

batch, mini-batch and online. The stopping condition is based either on the number of epochs or by early stopping.

  - `Extreme Adam`: a variation of normal Adam adapted to work with Extreme Learning Machine. It means that instead of learning all parameters, it learns only in the last layer. It uses *different momentum* in adaptive way and we add the possibility to tune *penalty l2 term*. Three version can be used: batch, mini-batch and online. The stopping condition is based either on the number of epochs or by early stopping.
  - `Cholesky`[4]: a direct analytical solver, used for Extreme Learning Machine, exploit the possibility to compute the exact solution of our problem, given a differentiable loss definition. It factorize the input matrix, assuming to be symmetric and positive definite, to a lower triangular matrix L and its transpose L^T. We add the possibility to tune *penalty l2 term*.

- **Weights**: the weights initialization of the network can be either from a random uniform distribution with min value = -0.7 and max value = +0.7 or from a random uniform distribution with *Xavier initialization*. It calculates dinamically the min value and max value considering number of input and number of output for each neuron.
- **Preprocessing**: possibility to apply *one-hot-encoding* to Monk's dataset.
- **Score function**: *Mean Squared Error* to evaluate our classification score, *Mean Euclidean Error* to evaluate our regression score and *accuracy*.

For **validation** we implemented *grid-search*, *k-fold* and *hold-out*. For details please refere to section 3. Those are main steps of our schema:

1. We leave as it is the TR of monk dataset. We apply an *hold-out* with 25% to the TR of CUP.
2. We apply a parallelized *outher grid-search* where for each configuration we apply a *3-fold Cross Validation* on the TR. Produces a subset of values and hyperparameters.
3. We apply a parallelized *inner grid-search*, where for each configuration we apply a *3-fold Cross Validation* on the TR. Produces final values and hyperparameters.
4. We apply a *6-fold Cross Validation* where we use plot of MSE on VL to make final choice. Produces final model.
5. For the model assessment we retrain the final model with whole TR and test performance on TS with mean from 3 trials.

3 Experiments

We followed up the validation schema expressed above, details will be written explicitly.

3.1 Monk's results

Two common considerations are coming out: using *xavier initialization* or simpler *uniform initialization* doesn't give any gain in performance or variance, then we choose to apply the uniform. Using *tanh* as activation not only allows to achieve a lower MSE on VL, but keeps lower the variance, very high using *sigmoid*. Except from Cholesky where *sigmoid* achieves better results. Some fluctuations are because we are using batch size of 32, in our test the best compromise between accuracy and speedup.
In FFNN without applying one-hot-encoding, MSE and Accuracy on VL, was poor then we chose to apply it to all problems. Extreme Adam, instead, perform the same either applying one-hot-encoding or not, then we leave data as it. As counterpart Cholesky performs better with one-hot-encoding applied. The following scores: Table 1, 2 and 3 are mean from 6 trials.

Solver	**#units**	**eta**	**lambda**	**momentum**	**MSE**(TR/TS)	**ACC**(TR/TS)
Sgd	5	0.9	0	0.2	7.8e-4/8.7e-4	100%/100%
Adam	3	1e-2	5e-4	adaptive	4.1e-3/5.2e-3	100%/100%
Extreme Adam	100	9e-4	0	adaptive	8e-2/1.2e-1	89%/85%
Cholesky	600	-	5e-3	-	3.1e-4/4.9e-2	100%/98,8%

Table 1: Monk 1 results

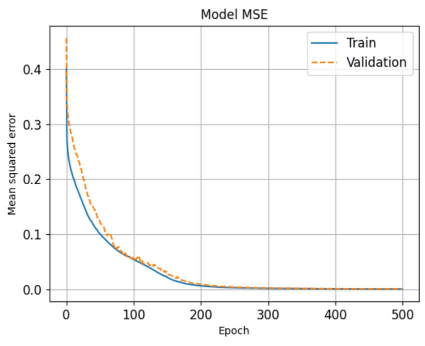
Solver	**#units**	**eta**	**lambda**	**momentum**	**MSE**(TR/TS)	**ACC**(TR/TS)
Sgd	5	0.9	0	0.2	4.5e-3/6.2e-3	100%/100%
Adam	3	1e-2	5e-4	adaptive	7.8e-3/8.7e-3	100%/100%
Extreme Adam	100	9e-4	5e-3	adaptive	2.2e-1/2e-1	63%/68%
Cholesky	1000	-	0.01	-	9.8e-3/0.38	100%/68.9%

Table 2: Monk 2 results

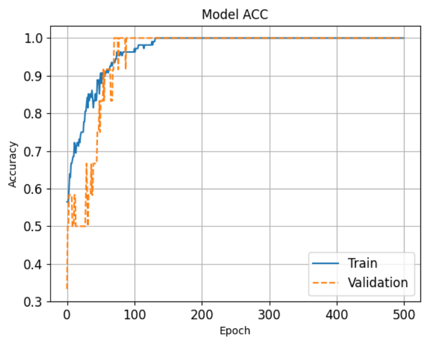
Solver	**#units**	**eta**	**lambda**	**momentum**	**MSE**(TR/TS)	**ACC**(TR/TS)
Sgd	5	0.9	0.002	0.2	4.4e-2/3.6e-2	94%/97%
Adam	3	1e-3	5e-4	adaptive	0.1/9.4e-2	93.4%/97.2%
Extreme Adam	100	9e-4	5e-3	adaptive	1e-1/1.3e-1	87%/82%
Cholesky	3000	-	1	-	2.2e-2/4.3e-2	98.3%/96%

Table 3: Monk 3 results

We plot only standard SGD with FFNN, plots are: Figure 1, 2, 3. For others models plots please check the appendix (Figures: A.8, A.9 and A.10).

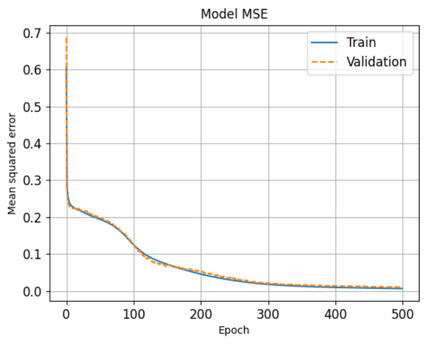


(a) Mean Squared Error

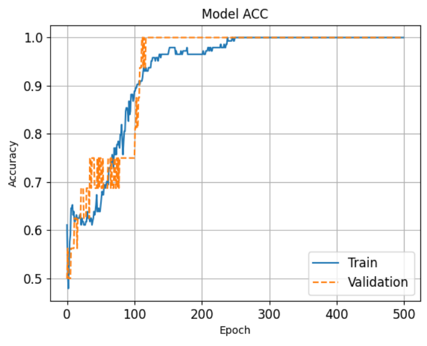


(b) Accuracy

Figure 1: Monk 1 SGD

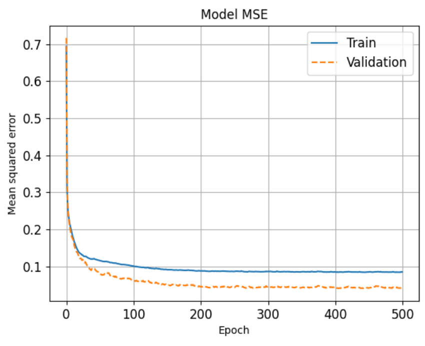


(a) Mean Squared Error

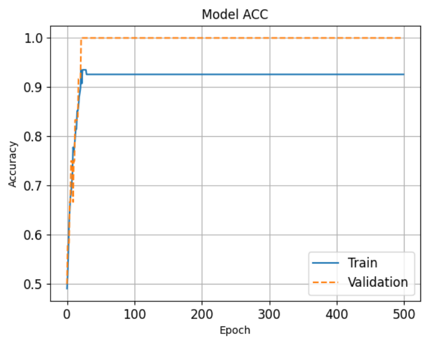


(b) Accuracy

Figure 2: Monk 2 SGD



(a) Mean Squared Error



(b) Accuracy

Figure 3: Monk 3 SGD

Speaking about solver we see as Adam has same performance as SGD. Cholesky achieve a very good MSE on TR, the best in our setup, but clearly a problem when we want generalize.
Speaking about approach the ELM needs to have higher number of neurons. Again, the extreme learning approach has in general higher variance.

3.2 Cup Result

We start trying to implement the Mean Euclidean Error as error function of solvers implemented, but after some test we figure out that the derivative of a square root causes smaller gradients than MSE. Then we choose to leave MSE as error function but show results of MEE in plot and tables to maintain competition rules. The following is the validation schema that we choose, divided into four step.

3.2.1 First step

We apply an hold-out over the previously shuffled TR, the percentage used was 25% to maintain the same proportion of the given TR+TS blind. This allows to have a new TR and a TS to do a correct model assessment.

3.2.2 Second step

We apply an outer grid-search on the new TR. Values of hyperparameters have to be with sufficient gap to have meaningful information, and in each configuration produced we apply a *3-fold Cross Validation.* This allow to have MEE on VL, for models, with corresponding variance. We consider only symmetric number of neurons in each layer, this facilitate the comparison to the ELM approach. We tested different model and solver, table 4 is a summary of test done.

Hyperparam	**SGD**	**Adam**	**Extreme Adam**	**Cholesky**
lambda	[0.0, 5e-5]	[0.0, 5e-4]	[0.0, 5e-5]	[5e-4, 5e-3, 5e-1]
eta	[7e-2, 5e-2]	[1e-3, 1e-2]	[1e-3, 1e-2, 7e-2]	-
momentum	[0.0, 0.2]	adaptive	adaptive	-
batch size	[64, 128]	[64, 128]	[64, 128]	-
layer number	[1, 3]	[1, 3]	1	1
unit	30	[10, 30]	[50, 150, 300]	[100, 1000, 5000]
activation	[tanh, sigmoid]	[tanh, sigmoid]	[tanh, sigmoid]	[tanh, sigmoid]
weights init.	[uniform, xavier]	[uniform, xavier]	[uniform, xavier]	[uniform, xavier]

Table 4: Outer grid search

We noticed overflow with SGD when used a momentum value >= 0.2. Looking to the best results completely shown in appendix (Tables: A.11, A.12, A.13, A.14 and Figures A.11, A.12, A.13), evaluate trough plots and MEE on VL, we choose to move forward different parameters. At this point the learning is very unstable and in some case it is going in overfitting.

3.2.3 Third step

We apply an inner grid-search again with *3-fold Cross Validation*. Values are based on results in Table 4, considering that up to now we have only a point for each param, here we can discover the best values direction. We tested different model and solver, table 5 is a summary of test done.

Hyperparam	**SGD**	**Adam**	**Extreme Adam**	**Cholesky**
lambda	0	5e-5	5e-4	[5e-1, 8e-1]
eta	[9e-4, 9e-3]	[1e-4, 1e-3]	[7e-3, 7e-4]	-
momentum	[0, 1e-3]	adaptive	adaptive	-
batch size	[32, 64]	[32, 64]	[32, 64]	-
layer number	[3, 5]	[3, 5]	1	1
unit	[30, 50]	[30, 50]	[300, 500, 800]	[2000, 6000]
activation hidden	tanh	sigmoid	tanh	sigmoid
weights initialization	xavier	uniform	uniform	linear

Table 5: Inner grid search

We notice an improved stability learning curve, evaluate trough plots and MEE on VL completely shown in appendix (Tables: A.15, A.16, A.17, A.18 and Figures A.14, A.15, A.16). Neural Network approach has lower MEE over VL, as counterpart, up to this point, Extreme Learning approach has higher MEE due to the 1 layer constraint. We have enough data to fine tune by end our models.

3.2.4 Fourth step

We apply a 6-fold Cross validation over TR. In this section we manually check the values by little change, looking table and plot from third phase. Tables: 6, 7, 8, 9 show the best configuration for each solver figured out. There is no improving evidence in variance or MEE over VL using batch size lower or greater than 64, therefore, for computational reason we will use mini-batch with 64.

-	**SGD**
lambda	0
eta	7e-3
momentum	0
batch	64
layers	3
units	30
activation	tanh
initialization	xavier
MEE(TR/VL)	**2.80/2.92**

Table 6: SGD best configuration

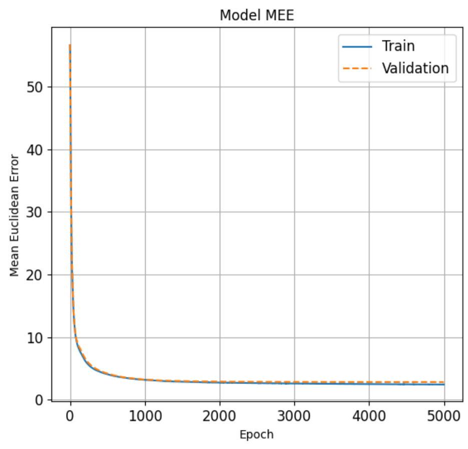


Figure 4: SGD MEE of TR/VL

Plot of SGD is in Figure 4. We notice that using a lambda penalty, even with very low value like 5e-5, it goes in underfitting achieving MEE in VL around 3.40. Indeed, we chose to not insert it. We try to increment number of neurons but doesn't get any improvements.

-	**Adam**
lambda	5e-5
eta	8e-4
momentum	adaptive
batch	64
layers	3
units	50
activation	sigmoid
initialization	uniform
MEE(TR/VL)	**2.86/2.85**

Table 7: Adam best configuration

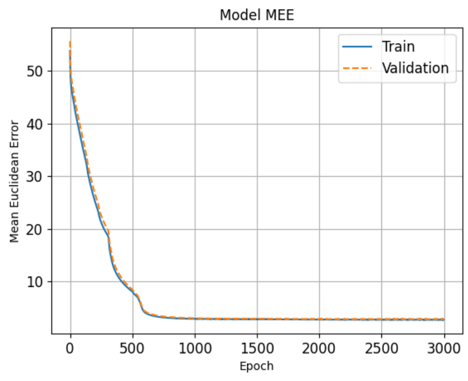


Figure 5: Adam MEE of TR/VL

Adam requires less epochs, but tends to be unstable in learning, cause of adaptive momentum and higher number of neurons compared to SGD. Indeed, is needed a smaller eta.

-	**Extreme Adam**
lambda	5-4
eta	9e-4
momentum	adaptive
batch	64
layers	1
units	800
activation	tanh
initialization	uniform
MEE(TR/VL)	**4.55/4.53**

Table 8: Extreme Adam best configuration

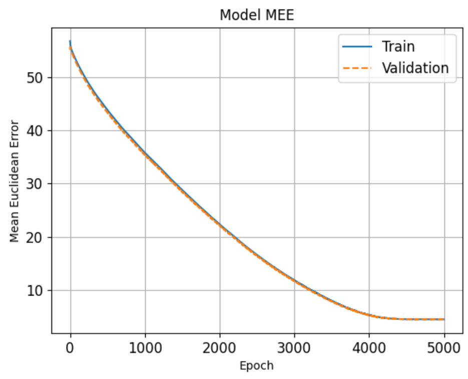


Figure 6: Extreme Adam MEE of TR/VL

Extreme Adam is very high dependent from weight initialization, being trained with one hidden layer. It has a very high variance, up to 5e-2. Once converged there is no needs of going ahead to epochs because the training is blocked, and as consequence we have oscillations in learning.

-	**Cholesky**
lambda	0.99
eta	-
momentum	-
batch	-
layers	1
units	8000
activation	sigmoid
initialization	uniform
MEE(TR/VL)	**3.89/3.90**

Table 9: Cholesky best configuration

Cholesky is very dependent from weights initialization, it could give more or less the same results even with less neurons, depends on good initialization case.

3.2.5 Model Evaluation

We assess solver selected from fourth phase, with corresponding parameters, training it with whole TR and evaluated trough TS holded-out in first phase. The resulting score is a mean from 3 trials.

-	**Adam**
lambda	5e-5
eta	8e-4
momentum	adaptive
batch	64
layers	3
units	50
activation	sigmoid
initialization	uniform
MEE(TR/TS)	**2.83/2.79**

Table 10: Adam best configuration

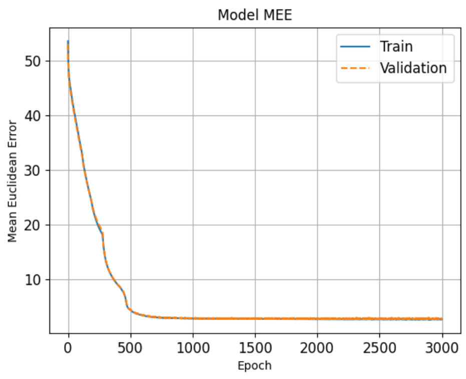


Figure 7: Adam: MEE of TS

We propose Adam as **final model** looking to the lower MEE over VL. As we can demonstrate, with parameters in Table 10 we keep the same convergence speed and shape of the plot in Figure 5, over VL, and in Figure 7, over TS.

3.2.6 Computing Time

All our implementations are slower compared to library found online. The reason is simple, we don't implement any technique to parallelize the code inside solvers. The bottleneck was the backpropagation phase, especially when layer number is greather then 3.
In a i7-9750H @ 2.60GHz the computing time for a SGD or Adam, trained for 5000 epochs, with params stated in Tables 6 and 7, is 14:24 minutes, consequence of 2.8 seconds for 10 epochs.
With the Extreme Learning approach, using Cholesky the computation is very fast even with 10000 neurons, always less then half minute, this is because we don't have the backprop. As counterpart Extreme Adam, with params stated in Tables 8, trained with 5000 epochs, is 6:54 minutes, consequence of 1.3 seconds for 10 epochs.

4 Conclusion

This project allows to us to explore in details how two different approach, NN and ELM, perform. We understand the fashion of deeper model. They are very effective, reducing variance an gives better generalization capabilities. On the other hand Cholesky has less generalization power but effective in training time.
The challenging of implement different solvers and hyperparameters selection figured out to be interesting and one thing that makes we learn a lot. Initially the experience was a little disorienting for the huge amount of point to test, but going ahead the funny part was the same that at the start scare us.

Blind test result nickname file = ELMvsNN_ML-CUP20-TS.csv
Nickname = ELMvsNN

5 Acknowledgements

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking.

References

[1] S. B. Thrun; Jerzy W. Bala; Eric Bloedorn: The MONK's Problems A Performance Comparison of Different Learning Algorithms. *Carneige Mellon University CMU-CS-91-197*. **1992**.

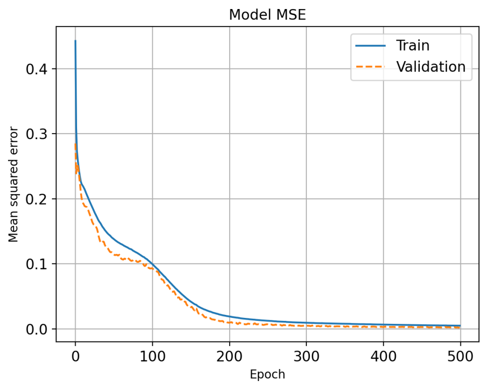
[2] Guang-Bin Huang; Qin-Yu Zhu; Chee-Kheong Siew: Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks. *IEEE International Joint Conference on Neural Networks*. **2004**.

[3] Diederik P. Kingma; Jimmy Ba: Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*. **2014**.

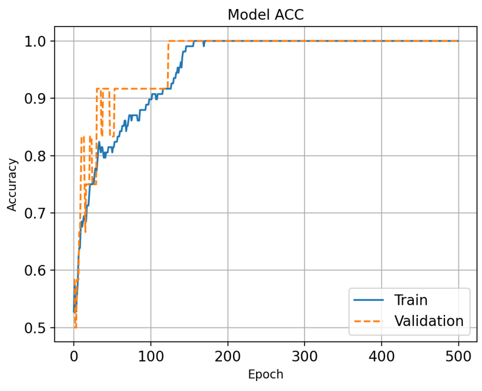
[4] Zuozhi Liu; JinJian Wu; Jianpeng Wang: An Improved Extreme Learning Machine Based on Full Rank Cholesky Factorization. *MATEC Web of Conferences 246(1-3):03018*. **2018**.

6 Appendix. A.

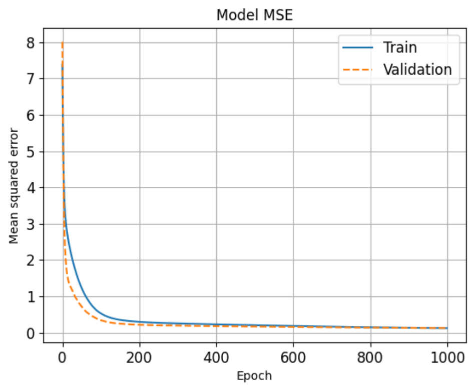
6.1 Monk's plot



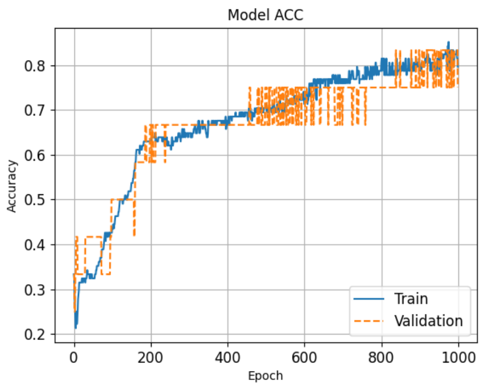
(a) Adam: Mean Squared Error



(b) Adam: Accuracy

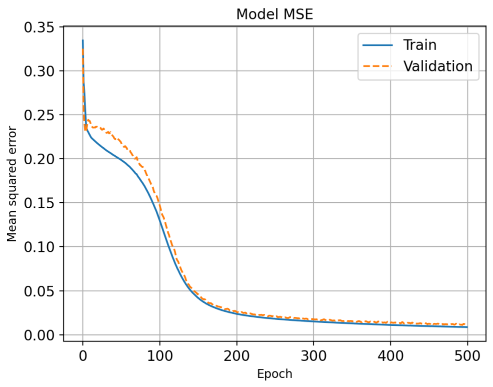


(c) Extreme Adam: Mean Squared Error

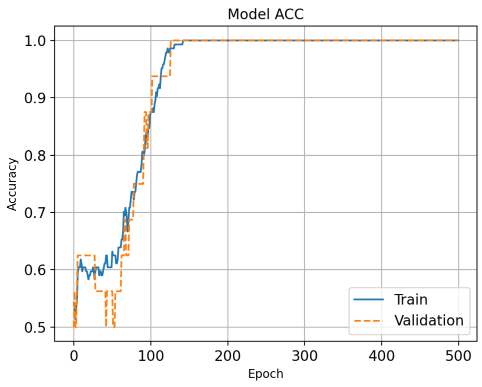


(d) Extreme Adam: Accuracy

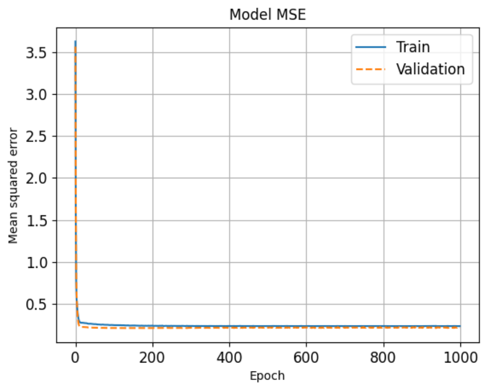
Figure 8: MONK 1 plots



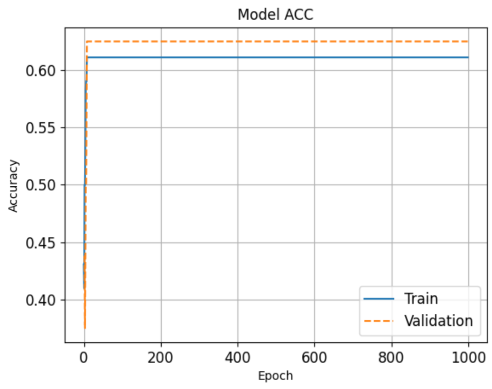
(a) Adam: Mean Squared Error



(b) Adam: Accuracy

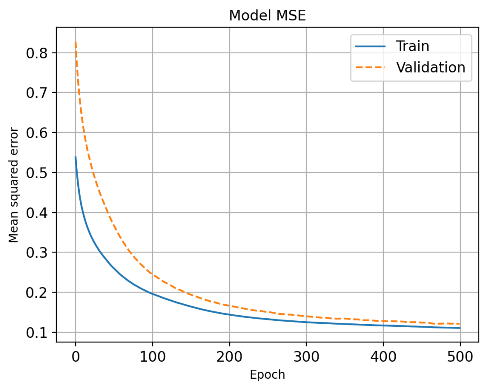


(c) Extreme Adam: Mean Squared Error

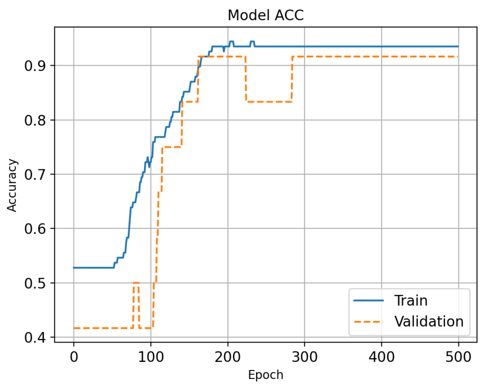


(d) Extreme Adam: Accuracy

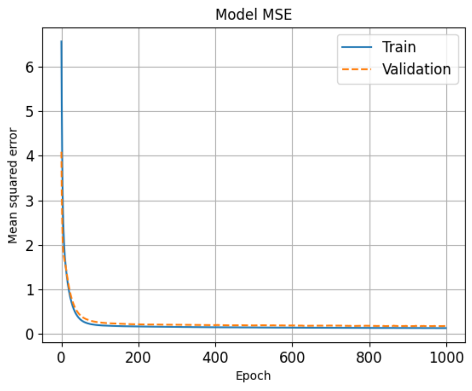
Figure 9: MONK 2 plots



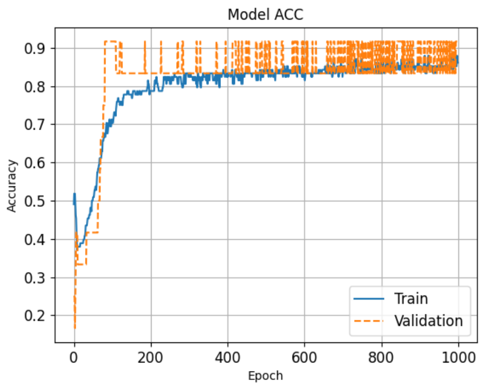
(a) Adam: Mean Squared Error



(b) Adam: Accuracy



(c) Extreme Adam: Mean Squared Error



(d) Extreme Adam: Accuracy

Figure 10: MONK 3 plots

6.2 CUP outer tests

SGD top	**first**	**second**	**third**
lambda	0	0	0
eta	7e-2	5e-2	7e-2
momentum	0	0	0
batch	64	64	64
layers	3	3	3
units	30	30	30
activation	tanh	tanh	sigmoid
weights init.	xavier	xavier	uniform
MEE(TR/VL)	**2.25/2.93**	**2.35/3.02**	**2.83/3.03**

Table 11: Outer best SGD results

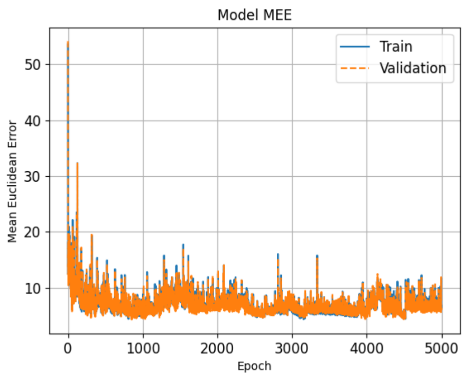


Figure 11: MEE of first

Adam	**first**	**second**	**third**
lambda	0	5e-4	0
eta	1e-3	1e-2	1e-3
momentum	adaptive	adaptive	adaptive
batch	64	64	64
layers	3	3	3
units	30	30	30
activation	sigmoid	sigmoid	sigmoid
weights init.	uniform	uniform	xavier
MEE(TR/VL)	**2.70/3.00**	**2.88/3.07**	**2.75/3.08**

Table 12: Outer best Adam results

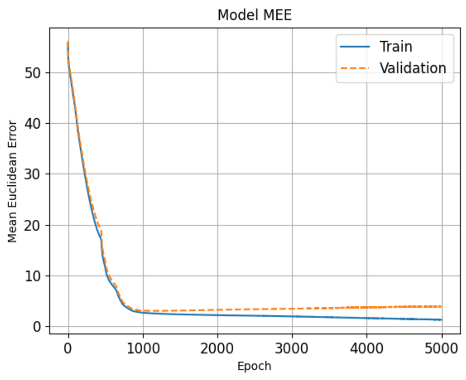


Figure 12: MEE of first

Ext. Adam	**first**	**second**	**third**
lambda	5e-4	5e-4	5e-4
eta	7e-2	7e-2	1e-2
momentum	adaptive	adaptive	adaptive
batch	128	64	64
layers	1	1	1
units	300	300	300
activation	tanh	tanh	tanh
weights init.	uniform	uniform	uniform
MEE(TR/VL)	**3.88/4.58**	**3.91/4.63**	**4.28/4.67**

Table 13: Outer best Extreme Adam results

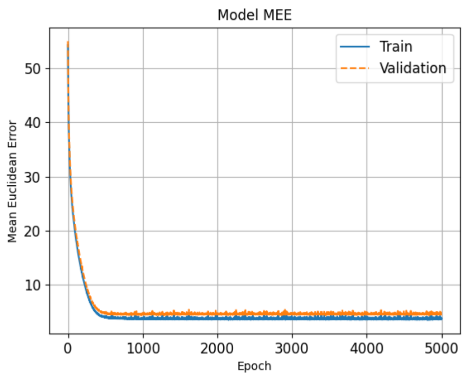


Figure 13: MEE of first

Cholesky	**first**	**second**	**third**
lambda	0.5	0.5	5e-3
eta	-	-	-
momentum	-	-	-
batch	-	-	-
layers	1	1	1
units	1000	5000	100
activation	sigmoid	sigmoid	sigmoid
weights init.	uniform	uniform	xavier
MEE(TR/VL)	**3.94/4.40**	**3.36/4.61**	**4.81/4.63**

Table 14: Outer best Cholesky results

6.3 CUP inner tests

SGD	**first**	**second**	**third**
lambda	0	0	0
eta	9e-3	9e-4	9e-3
momentum	0	0	0
batch	64	32	64
layers	3	3	3
units	30	50	50
activation	tanh	tanh	tanh
weights init.	xavier	xavier	xavier
MEE(TR/VL)	**2.40/3.01**	**2.70/3.08**	**2.34/3.10**

Table 15: Inner best SGD results

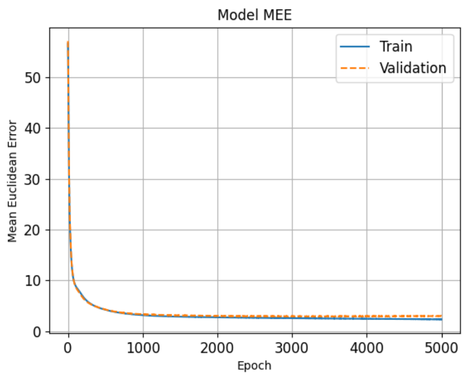


Figure 14: MEE of first

Adam	**first**	**second**	**third**
lambda	5e-5	5e-5	5e-5
eta	1e-3	1e-3	1e-3
momentum	adaptive	adaptive	adaptive
batch	64	64	32
layers	3	5	5
units	50	50	50
activation	sigmoid	sigmoid	sigmoid
weights init.	uniform	uniform	uniform
MEE(TR/VL)	**2.56/2.90**	**2.63/2.90**	**2.73/2.91**

Table 16: Inner best Adam results

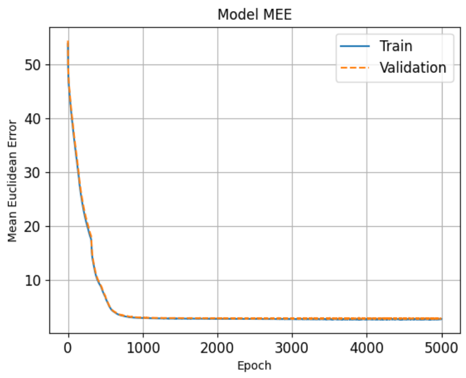


Figure 15: MEE of first

Ext. Adam	**first**	**second**	**third**
lambda	5e-4	5e-4	5e-4
eta	7e-3	7e-3	7e-3
momentum	adaptive	adaptive	adaptive
batch	64	64	32
layers	1	1	1
units	800	500	500
activation	tanh	tanh	tanh
weights init.	uniform	uniform	uniform
MEE(TR/VL)	**3.84/4.49**	**3.96/4.50**	**4.10/4.55**

Table 17: Inner best Extreme Adam results

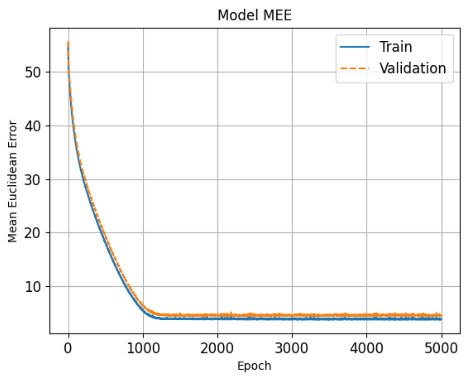


Figure 16: MEE of first

Cholesky	**first**	**second**	**third**
lambda	8e-1	8e-1	5e-1
eta	-	-	-
momentum	-	-	-
batch	-	-	-
layers	1	1	1
units	6000	2000	6000
activation	sigmoid	sigmoid	sigmoid
weights init.	uniform	uniform	uniform
MEE(TR/VL)	**3.54/4.44**	**3.78/4.49**	**3.31/4.54**

Table 18: Inner best Cholesky results