ML 2023 Project – Type A

Team Name: All' ultimo momentum

Andriani Paolo
Velardita Michele

Department of Computer Science,
University of Pisa

A/A 23-24

Abstract

In this project, a flexible neural network was implemented in **Python**, allowing the selection of the number of layers, activation functions, and the use of **L2 regularization.**

Gradient descent, stochastic gradient descent (**SGD**), classical moment (**CM**), and accelerated Nesterov gradient (**NAG**) methods were implemented.

Additionally, we developed **grid search** and **k-fold cross validation** for parameter tuning and model validation. Our experiments and hypotheses, as will be better explained later, were based on the results of previous studies [1] [2] [3].

Introduction - reviewed papers

[1] The **importance of weight initialization** together with an increasing **momentum schedule** greatly influences the performance of the model. The authors show how in the transient phase of NN training, momentum-based methods (applied to the stochastic case) maintain better performances than simple SGD.

[2] [3] Although SGD, CM and NAG with mini-batch show the **same order of convergence** in expectance, momentum helps to generalize better. The empirical results in [2] & [3] demonstrates that **NAG** (in stochastic setting) achieves a **good tradeoff** between speed of convergence in TR error and robustness of convergence in Test.

Introduction - hardware specification

All the most time consuming experiments were conducted on a PC with 32 GB RAM Ryzen 7 5800x 8 core/16 thread processor

To expedite the parameter search, we utilized the Joblib[4] library to parallelize the grid search process. This approach significantly reduced the model validation times

Introduction - objectives

Aim

Explore in depth momentum-based methods in neural network training.

Rationale

Studying these methods early saves validation time to find good parameters and provides valuable insights.

Approach

1. Conduct experiments to verify characteristics from literature.
2. Develop and validate models for MONK tasks and the CUP dataset.

Introduction – preliminary experiments

We delved into momentum-based optimization methods by studying and replicating experiments from key research papers. This provided us with several insights:

- Nesterov Accelerated Gradient (NAG) demonstrates **higher tolerance** at elevated momentum **μ** values, especially with larger step sizes **α** . { Appendix 1 }

- Implementing a momentum **schedule** reduces the need for parameter tuning and enhances stability in the learning process. { Appendix 1.1 }

- Overall, momentum-based methods offer superior **generalization** and **stability** compared to standard Stochastic Gradient Descent (SGD). { Appendix 2 }

Introduction - method

In-depth analysis of the method

Having conducted an in-depth study of momentum-based methods, we strategically avoided combining certain parameters values and were able to anticipate the behavior of specific models.

Nested GridSearch

We performed an **initial grid** search with order of magnitude for some parameters and then restricted the search in a smaller interval.

Parallelizing the Cross-Validation

CV with 5 folds and 3 repeats
We took advantage of the joblib library[4] to speed up the trials.

MONK - Preprocessing

A **correlation analysis** was performed in each MONK dataset. This helped us understand how the features relate to the target y and to what extent (Appendix 3).

Furthermore, for MONK3, **misclassified records** were identified in the training set, which affected the generalization performance of the models (Appendix 3.1).

One-hot encoding has been performed on features for all MONK tasks.

MONK - Validation

Initial Grid

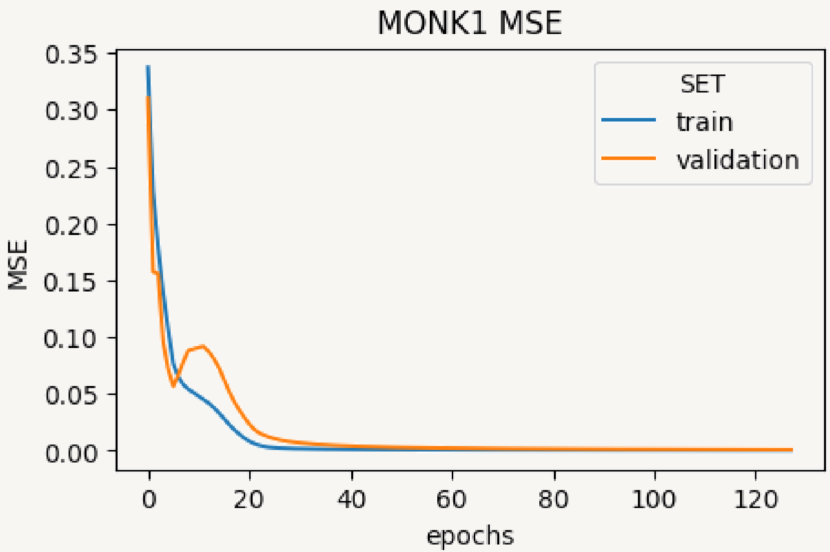
	Batch Size	Momentum μ	Learning Rate α	Activation Function	λ regularizer	Topology (hidden)
Grid	[4, 8, 16]	0.9, 0.95, 0.98 schedule	0.1, 0.01, 0.001	Sigmoid, tanh, relu	0, 0.01, 0.001	[6] [6, 4] [6, 4, 4]

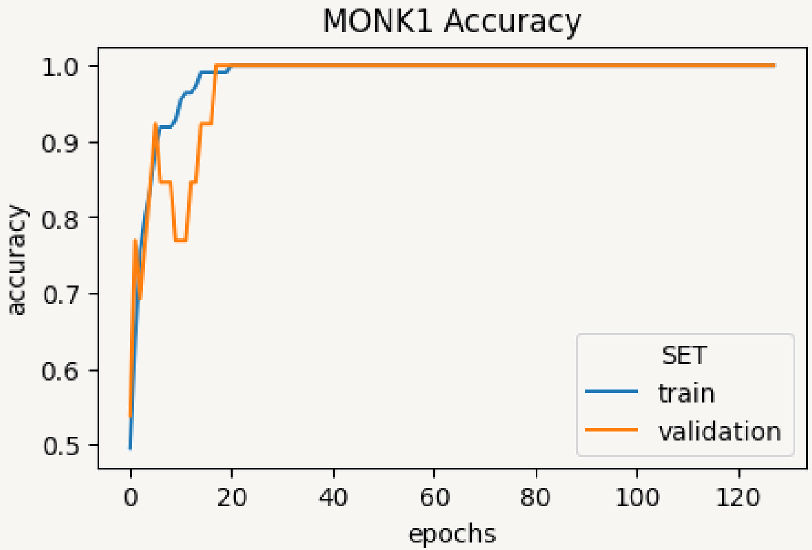
#Grid = 324

MONK result: Summary

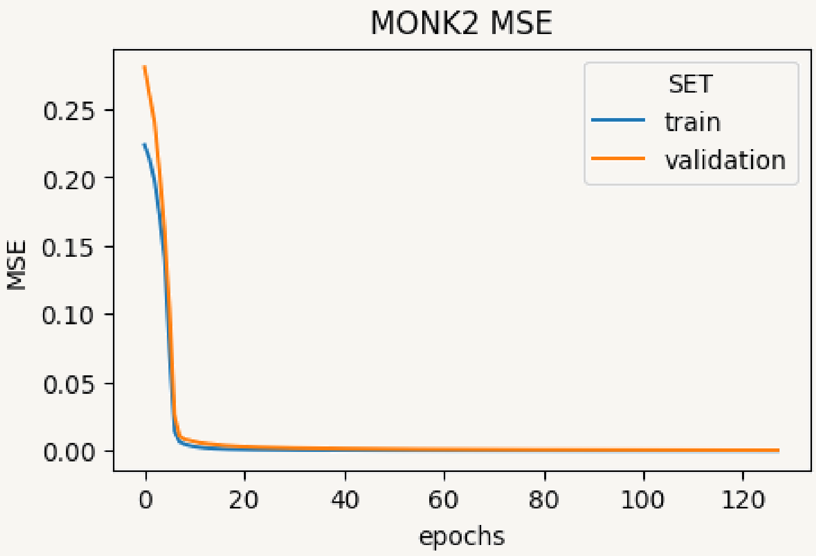
Task	Hyperparameters	MSE (TR / TS)	Accuracy (TR / TS)
MONK 1	**μ**: 0.95, **α**: 0.05, **λ**: 0.001 **Topology**: [n, 6, 4, 1], **f**: tanh **Init**: he, **batch_size**: 8, **epochs**: 128	0.0003±0.001 0.002±0.003	100% / 100%
MONK 2	**μ**: 0.9, **α**: 0.05, **λ**: 0 **Topology**: [n, 6, 4, 1], **f**: tanh **Init**: he, **batch_size**: 4, **epochs**: 128	0.0001±0.0001 0.0002±0.0001	100% / 100%
MONK 3	**μ**: 0.95, **α**: 0.05, **λ**: 0 **Topology**: [n, 6, 4, 1], **f**: sigmoid **Init**: he, **batch_size**: 8, **epochs**: 128	0.0500±0.005 0.0237±0.005	94% / 98%±1%

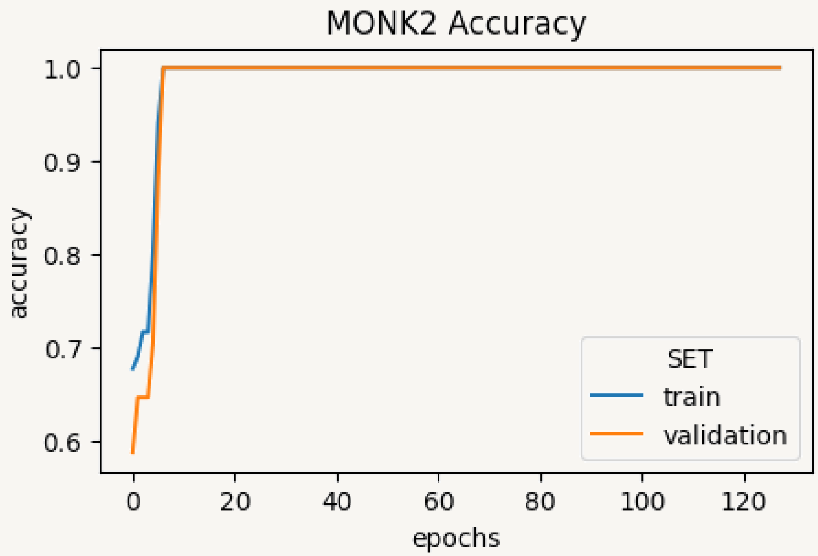
MONK result: Plot



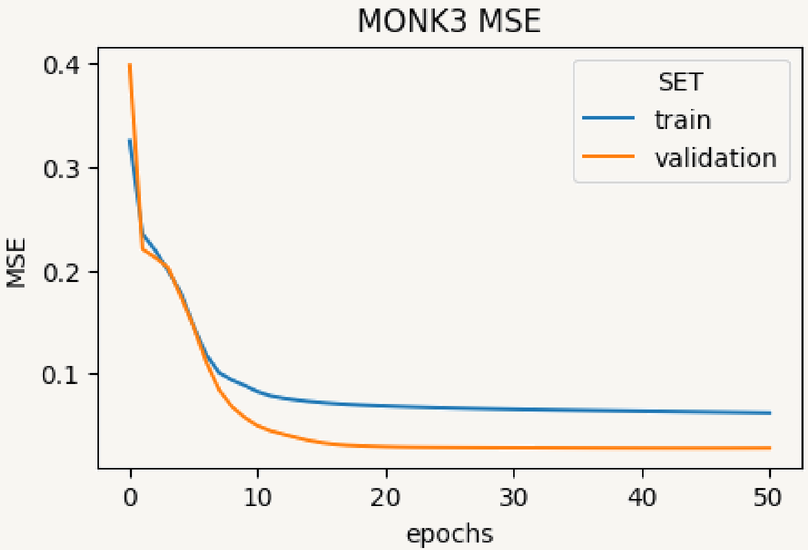


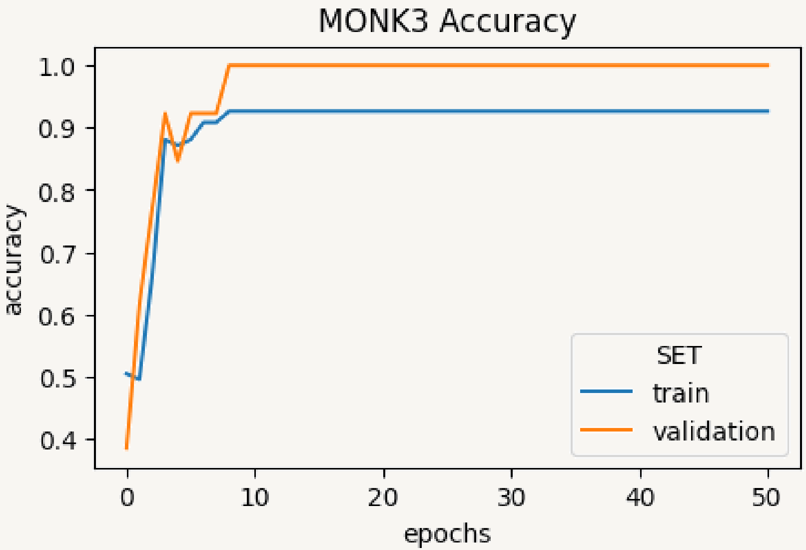
MONK result: Plot





MONK result: Plot





CUP: Validation Schema

0. Train Set / Holdout Test Set Split (80%/20%)

1. Repeated K-Fold CV*
(Split 80%/20%, n_fold=5, n_repeat=3) → Best Hyperparameters

2. Retrain NN with the best hyperprameters on TR Set with early stopping → Trained NN

3. Assessment on Holdout Test Set → Performance measure (MSE/MEE)

* Parallelized with joblib [4]

CUP: Grid Search strategy

	Batch Size	Momentum μ	Learning Rate α	Activation Function	λ regularizer	Topology (hidden)
Grid 1	16, 32, 64	0.90, 0.95, 0.98 schedule	0.1, 0.01, 0.001	Sigmoid, tanh, relu	0, 0.001, 0.0001	[10, 20, 10] [32, 64, 32]
Grid 2	32, 64	0.95, schedule	0.001, 0.002, 0.005	Sigmoid, tanh	0, 0.001	[32, 64, 32]

#Grid 1: 648
#Grid 2: 48

CUP: Final NN model

Optimizer: Nesterov

- Topology: [10, 32, 64, 32, 3]
- Hidden function: sigmoid
- Output function: linear
- λ regularizer: 0.001
- Init method: **he**
- Max Epochs: 512
- Batchs Size: 64
- Learning Rate: 0.005
- Momentum: **scheduled**

Momentum schedule

a = (1 + sqrt(4a_prev**2+1)) / 2

μ = (a_prev - 1) / (a)

He initialization

Gaussian distribution with

Mean: 0

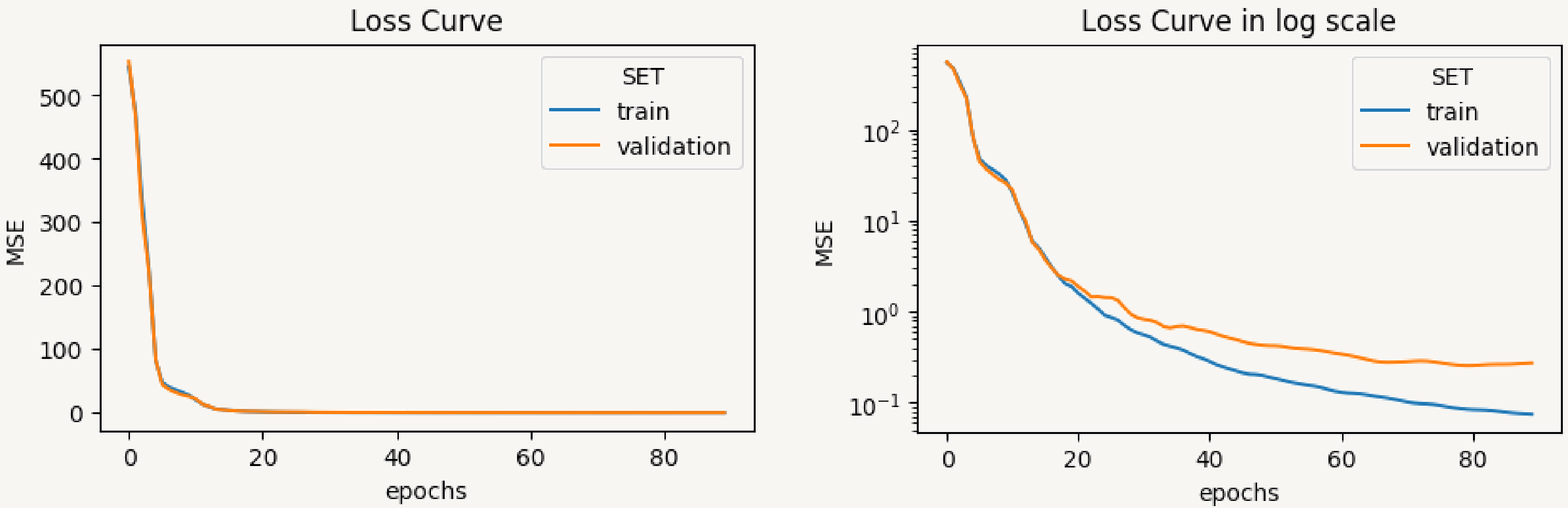
Std: sqrt(2/N_prev)

CUP: result

Result of 20 trials on the same model. We calculate the MSE and MEE, take the mean and the standard deviation.

	MSE	MEE
Train	0.055 ± 0.023	0.320 ± 0.062
Holdout Test	0.285 ± 0.063	0.633 ± 0.058

CUP: Plot


Loss Curve
SET
train
validation
MSE
0
100
200
300
400
500
0
20
40
60
80
epochs
Loss Curve in log scale
SET
train
validation
MSE
10^{2}
10^{1}
10^{0}
10^{-1}
0
20
40
60
80
epochs

Conclusion

We discovered that although Nesterov Accelerated Gradient (NAG) does not yield significant improvements over classical momentum, it is **more robust and stable**.

Additionally, NAG allows us to experiment with much higher parameters.

Initialization is crucial: a random initialization, tested across multiple trials, results in highly variable outcomes.

Regarding generalization ability, we observed that on more complex datasets (such as CUP), **NAG outperforms classical momentum.**

In conclusion, our theoretical study of these models enabled us to explore hyperparameters more effectively and anticipate the expected results for certain models

Bibliography

[1] Sutskever, I., Martens, J., Dahl, G. & Hinton, G. (2013). *On the importance of initialization and momentum in deep learning*. Proceedings of the 30th International Conference on Machine Learning, 28(3):1139-1147. Available from: https://proceedings.mlr.press/v28/sutskever13.html.

[2] Yang, T., Lin, Q., & Li, Z. (2016). *Unified Convergence analysis of stochastic momentum methods for convex and non-convex optimization*. Available from: https://arxiv.org/abs/1604.03257

[3] Yan, Y., Yang, T., Li, Z., Lin, Q., & Yang, Y. (2018). *A unified analysis of stochastic momentum Methods for Deep learning*. Available from: https://arxiv.org/abs/1808.10396

[4] Joblib: running Python functions as pipeline jobs — *joblib 1.4.2 documentation*. (n.d.). Available from: https://joblib.readthedocs.io/en/stable/

Thanks!

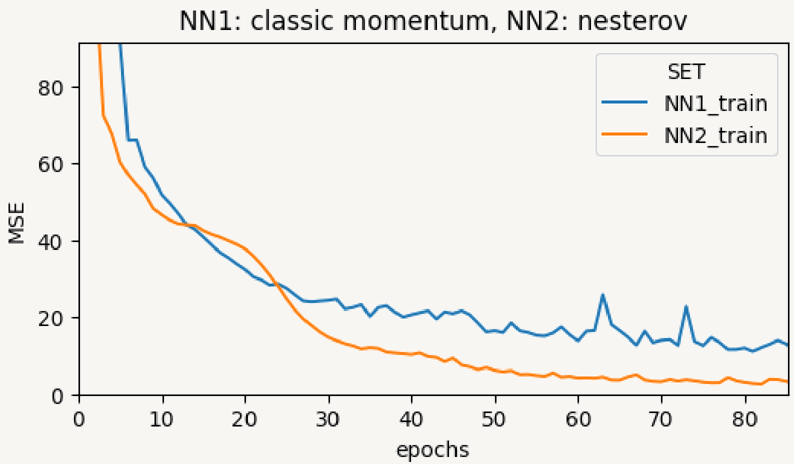
Do you have any question?

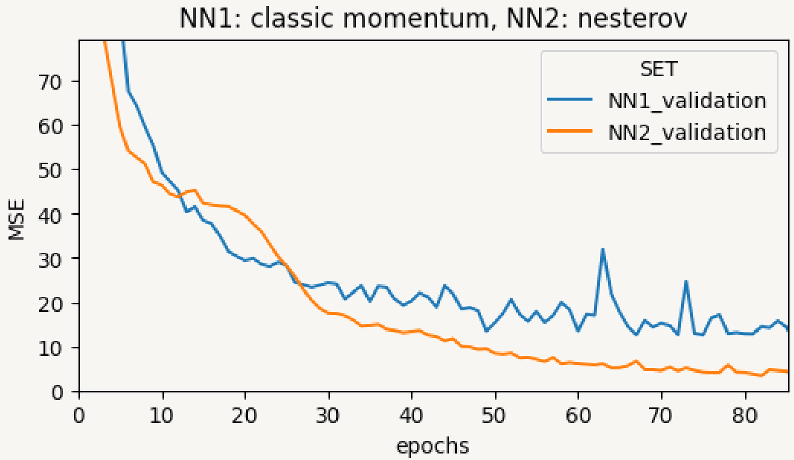
Appendix 1

Based on the results obtained in [1], we conducted experiments on the CUP dataset. (*Experiments–Initialization&momentum.ipynb*)

Nesterov Momentum is much more effective at deceleration, thus making NAG more tolerant to large values of momentum term.

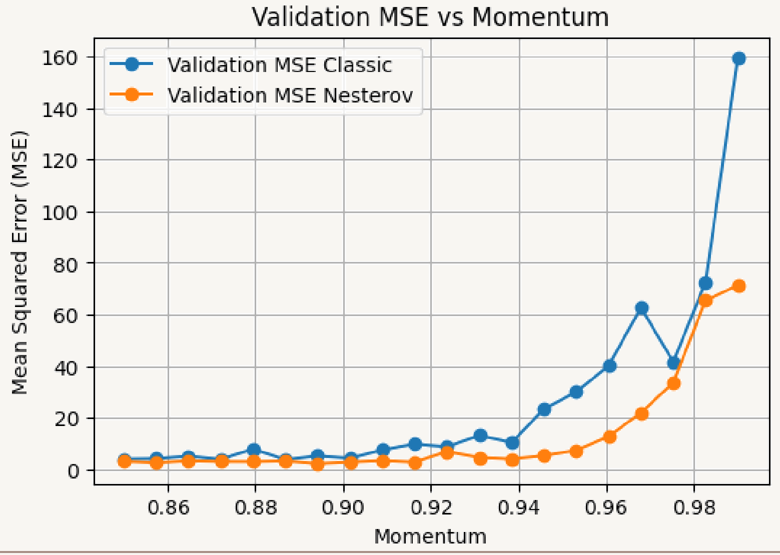
It exihibits less oscillations than classical momentum, making the method more robust especially in a stochastic setting.

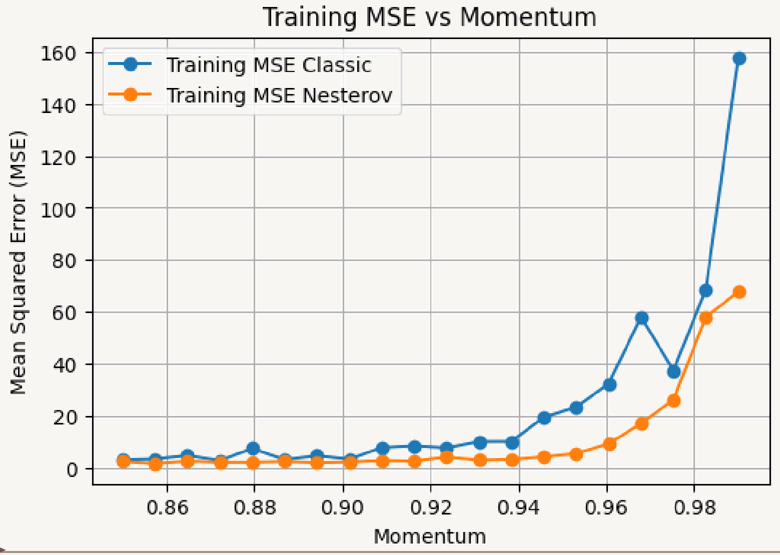




Appendix 1

Using a relatively **high stepsize** (0.02), the difference between CM and NAG is much more visible





Appendix 1.1

We analyzed the effect of the schedule and compared it with different fixed momentum values **μ**

μ	MSE (TR)	MSE (TS)	Std(TR)	Std(TS)
0.9	0.180	0.320	0.042	0.063
0.95	0.129	0.263	0.019	0.039
0.98	17.410	18.220	33.649	33.043
schedule	0.063	0.180	0.025	0.034

Appendix 2

The convergence results in [2, Theorem 3] indicate that stochastic GD, classic momentum (CM), and Nesterov Accelerated Gradient (NAG) have similar convergence rate.

However, **NAG** achieves the best training error and testing error robustness among the three methods.

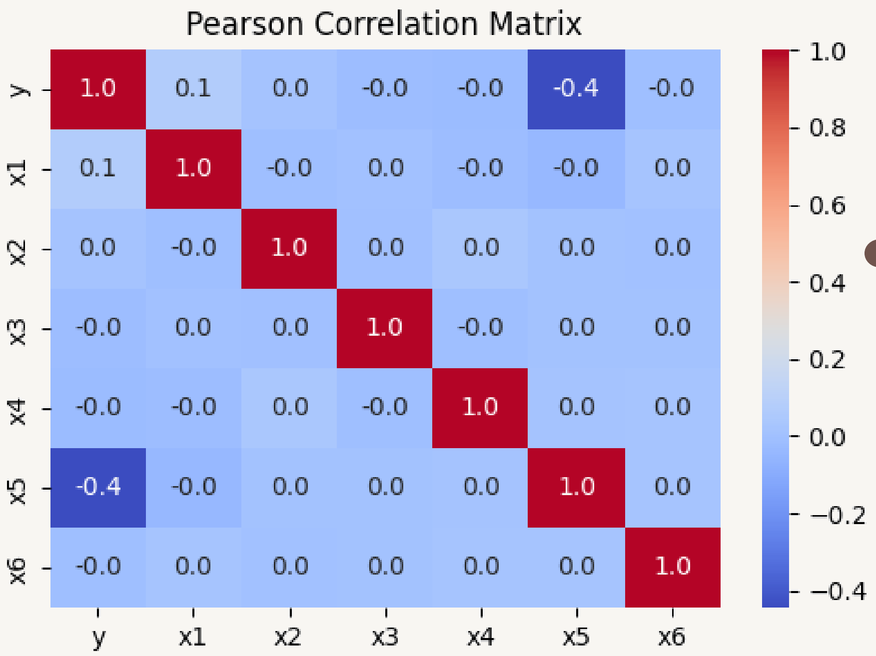
Method	MSE (TR)	MSE (TS)	Std(TR)	Std(TS)
GD	6.919	8.275	1.4299	1.598
CM	0.500	0.748	0.294	0.339
NAG	0.306	0.510	0.036	0.047

Appendix 3 – data exploration

The outcome of y in MONK 1 depends on only 3 variables, thus making the remaining features useless for the model.

In the case of MONK 2, each variable appears to influence the prediction

MONK 1 – linear correlation



Appendix 3.1 – data exploration

In MONK3 we notice a discrepancy between training and testing.

We hypothesized that there was some **misclassified records** in the TR set.

Only x2, x4, x5 contribute to the outcome of y.