Machine Learning 2023 Project

Authors: (*Gianluca Panzani(1), Emad Chelhi, Corrado Baccheschi*)

EmMelle Team

Master Degree curriculum: Computer Science (AI), Data Science and BI, Informatica Umanistica

emails g.panzani@studenti.unipi.it / e.chelhi@studenti.unipi.it / c.baccheschi@studenti.unipi.it

Date 01/02/2024

Type of Project: B

Our contributions and method

All code is developed using python version 3.10.

- For plotting we used `matplotlib[1]`.
- For numerical operation we used `numpy[2]`.
- For data manipulation we used `pandas[3].`
- We chose to employ Object-Oriented Programming (OOP) to construct an API that could expedite model building and delve into the various tools' functionalities more deeply.
- As mentioned in the introduction, whenever possible, we refrained from mixing the functions and methods of different tools. Consequently, we manually implemented certain functionalities that were not available in specific tools. For instance, we wrote the code for *GridSearch* in Keras`[4]` since it wasn't available. Same choice for *EarlyStopping*, manually implemented in Pytorch`[5]`.
- Moreover, for the GridSearch we chose to use it in two-steps: first *Randomized GridSearch* to capture most interesting intervals and then, after that, an *Exhaustive GridSearch* to obtain final hyperparameters.
- As already mentioned, we built Neural Networks and SVM models. Regarding the NN, next slide we'll list our most important choices, considering that all share same structure.

- For classification task, since we built a *Multi Layer Perceptron* for MONKS, only 1 hidden layer was required so we used *tanh* as hidden activation and *sigmoid* as output function for all the models in Keras, Pytorch and Scikit-learn[6]. For the regression task on Cup, we instead chose to use 2 or 4 layers because was a more difficult task, anyway using *tanh* as hidden activation and one *linear* as output. All layers are *Dense*, that is, each one is *fully* connected with the next one.
- The number of units for the MONK's tasks was very few (2,3,4) since it was an easy task, whereas for the Cup we started from around 60 and we noted better performance with 64-80 up to 100 units for each layer. The right values were the outcomes from our *two-steps GridSearch*.
- We trained our Neural Networks using *Stochastic Gradient Descent (SGD)*[7] as **learning algorithm** with *mini-batch* variant, which is faster to converge. Hence, since this optimizer required a smaller eta, for Keras we applied learning rate decay on the MONK's tasks, and in particular on MONK2 and MONK3 due those looked us more sensitive to diverge with a higher eta value. Even in this case, all the values for batch size and for eta decay, were searched by *GridSearch*.
- We initialized our Neural Networks using *glorot uniform* which extracts the values from a uniform distribution.
- We regularized Neural Networks using *Tikhonov* penalty both in Keras, Pytorch and Scikit-Learn, but exclusively on MONK3 since had some noise. We also added penalties (but lower) on Cup task. Again, all the values were searched through the *GridSearch*.
- For the Neural Networks in Keras, reported in next section, after model selection we tried 5 different inizializations and we reported mean, variance and standard deviation across these 5 models on each MONK.
- As a validation schema, we chose to use first *Randomized GridSearch* to capture most interesting intervals and then, after that, on the outcomes we performed an *Exhaustive GridSearch* to obtaint the best hyperparameters. Then, we used *K-Fold Cross Validation* specifying k=5 and for scoring we computed mean, variance and standard deviation across each folds and the best models were retrained on all the data before testing them. Same methodologies were used both for MONKs and Cup.
- As a stop conditions, we used EarlyStopping provided by Sklearn and based on number of epochs (patience) selected in the grid search for the MONK's and Cup tasks and for both Keras and Sklearn models. The only exception was Pytorch for which we chose to write from scratch a code for stopping conditions based on tolerance on validation loss.

Monk Results - Keras

- We implemented a Keras *MLP* with one hidden layer and 3 hidden units using *tanh* as activation function and *sigmoid* as output with single unit. We noted we converged to 100% on validation set for MONK1, MONK2 in 2/3 trials of *Exhaustive Grid Search* with the below hyperparamers.
- For MONK1, MONK2 we used *nesterov momentum* and for all MONKS an intermediate *patience* for earlystopping set to 10/15. As already discussed, we also chose to decay the learning rate for MONK2 and MONK3 only, each 500 steps with a factor of 0.5. We also computed variance and standard deviation of VL loss across folds, for each trial.

Table 1. Best predictions results obtained on MONK's tasks.

Task	**Parameters**	**MSE(TR/TS)**	**Accuracy(TR/TS)**	**Var/STD**
Monk1	3units, η=0.4, α=0.6, epochs=180, batch=4	7.835229225747753e-05/0.000741630676202476	100%/100%	3.3824e-07/0.0005
Monk2	3units, η=0.9, α=0.6, epochs=180, batch=10	0.0010804852936416864/0.0011045043356716633	100%/100%	3.504e-9/5.91e-05
Monk3(reg)	3units, η=0.1, α=0.3, λ=0.5, epochs=180, batch = 62	0.1849091410636902/0.17679205536842346	94%/97%	0.0004/0.0215
Monk3(noreg)	3 units, η=0.1, α=0.3, batch=62, epochs=270, λ= None	0.05066834166646004/0.04863801226019859	95%/93-94%	0.00092/0.023

Monk results - Keras

- With best hyperparameters, we also tried 5 reinizializations for each MONK and we've computed mean, standard deviation and variance of test loss.

Table 2. Average predictions results obtained on MONK's tasks.

Task	**MSE(TR/TS)**	**Accuracy(TR/TS)**	**Var/STD**
Monk1	0.01502733700326644/0.018227058454067448	97%/96%	0.00048/0.0219
Monk2	0.021094616455957293/0.025959141575731336	97%/96%	0.0009/0.0313
Monk3(reg)	0.20088812708854675/0.20088812708854675	90%/93%	0.0004/0.021
Monk3(noreg)	0.05320663154125214/0.04482369273900986	94%/96%	1.069e-06/0.0010

Monk Results 1 – Keras

MSE

Accuracy

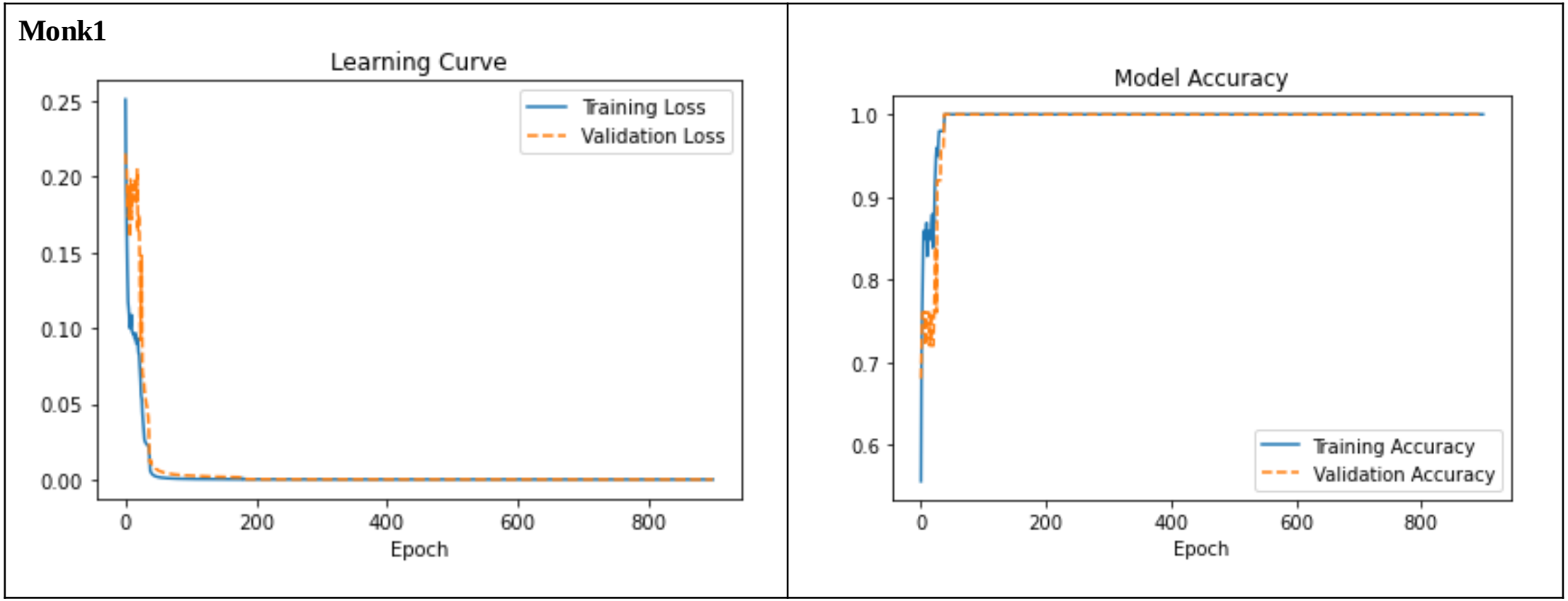

Table 3. Learning Curves and Accuracy on MONK1

Monk Results 1 – Keras

MSE

Accuracy

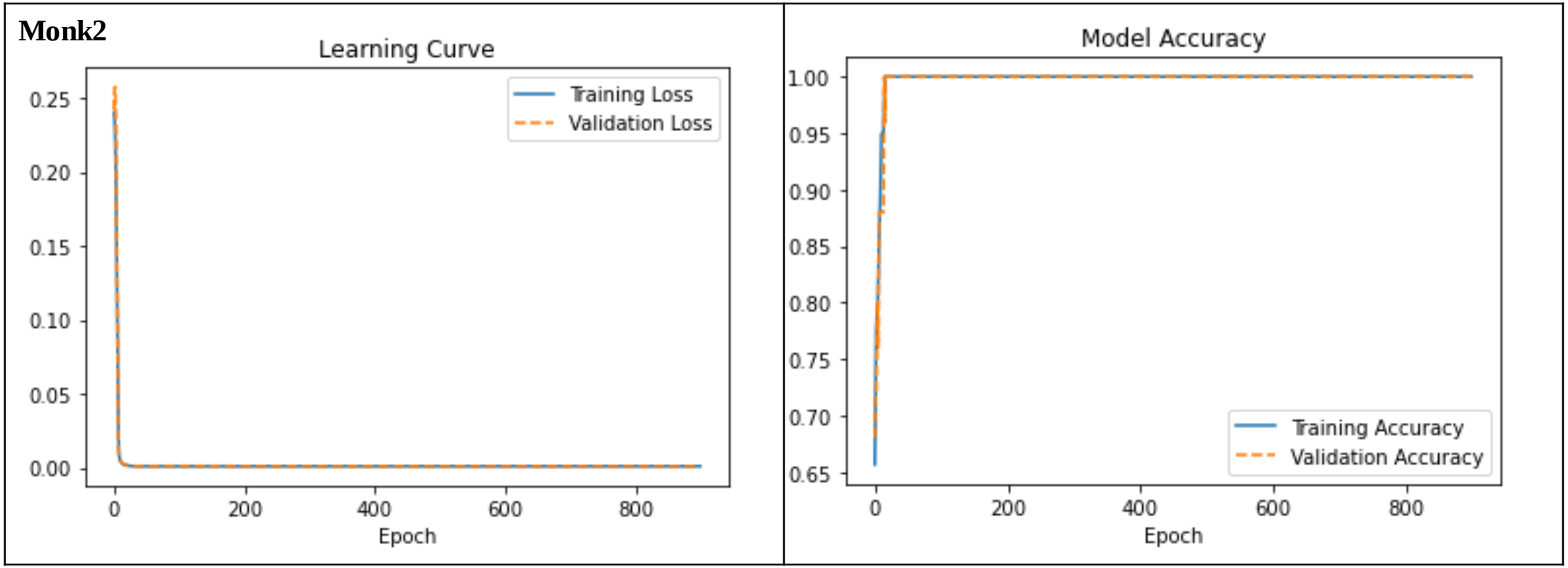

Table 4. Learning Curves and Accuracy on MONK2

Monk Results 1 – Keras

MSE

Accuracy

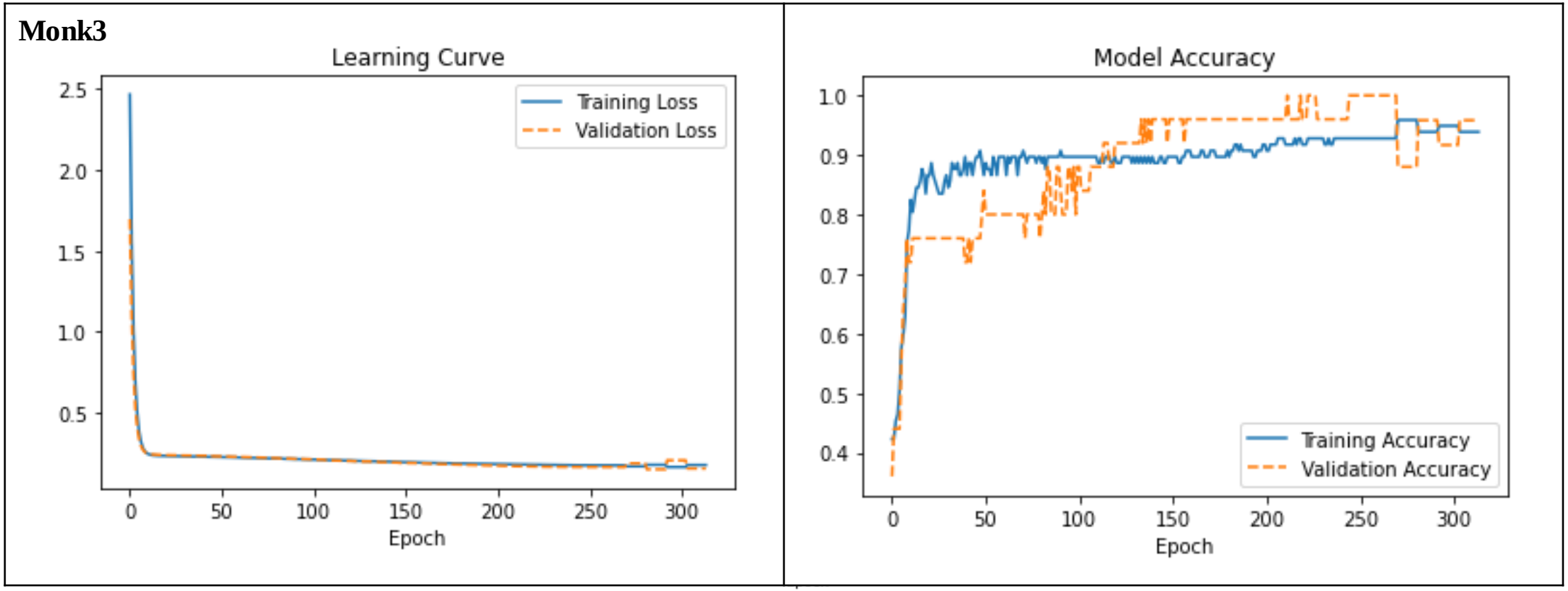

Table 5. Learning Curves and Accuracy on MONK3

Monk Results 1 – Keras

MSE

Accuracy

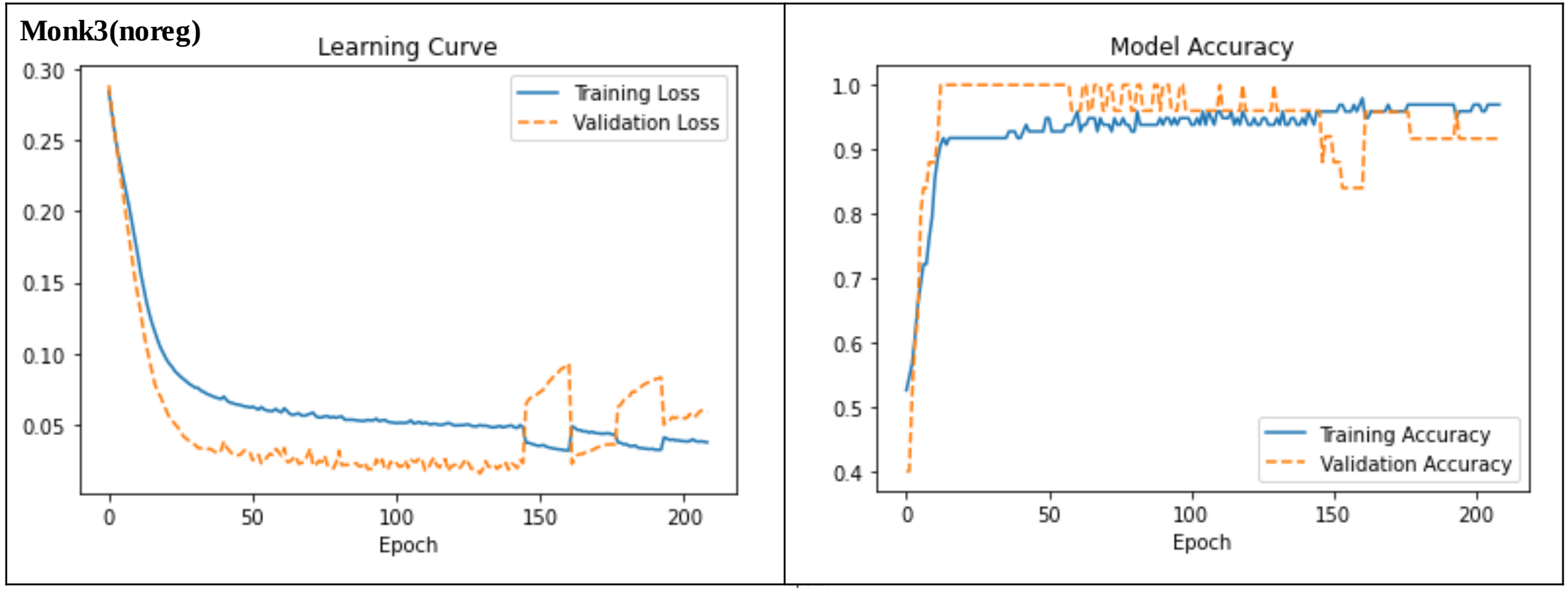

Table 6. Learning Curves and Accuracy on MONK3 without Tikhnov reg. Note the validation loss, orange curve, that goes up after 150 epochs meaning the model overfits the train data.

Monk Results 1 - Scikit-Learn

- We implemented a Scikit-Learn MLPClassifier with one hidden layer and 3-4 hidden units using *tanh* as activation function.
- For each model we used *nesterov momentum*, batch size equal to 16 for monk 1 and 2 while equal to 64 for monks 3, and as solver the best was "sgd" in monk 1, 2 and 3 reg. , while in the last one, monk 3 not reg. "adam" was returned as best.

Table 7. Average predictions results obtained on MONK's tasks.

Task	**Parameters**	**MSE(TR/TS)**	**Accuracy(TR/TS)**
Monk1	3 units, η=0.3, α=0.6, epochs=340	0.013137880790722 83 / 0	99% /100%
Monk2	4 units, η=0.9, α=0.6, epochs=340	0.005998467469776581 / 0	99% / 100%
Monk3(reg)	4 units, η=0.3, α=0.2, epochs=500	0.07518616636343771 / 0.046296296296296294	93% / 97%
Monk3(no reg)	4 units, η=0.8, α=0.6, epochs=500	0.0608396770472895 / 0.037037037037037035	95% / 96%

Monk Results 1 – Scikit-Learn

MSE

Accuracy

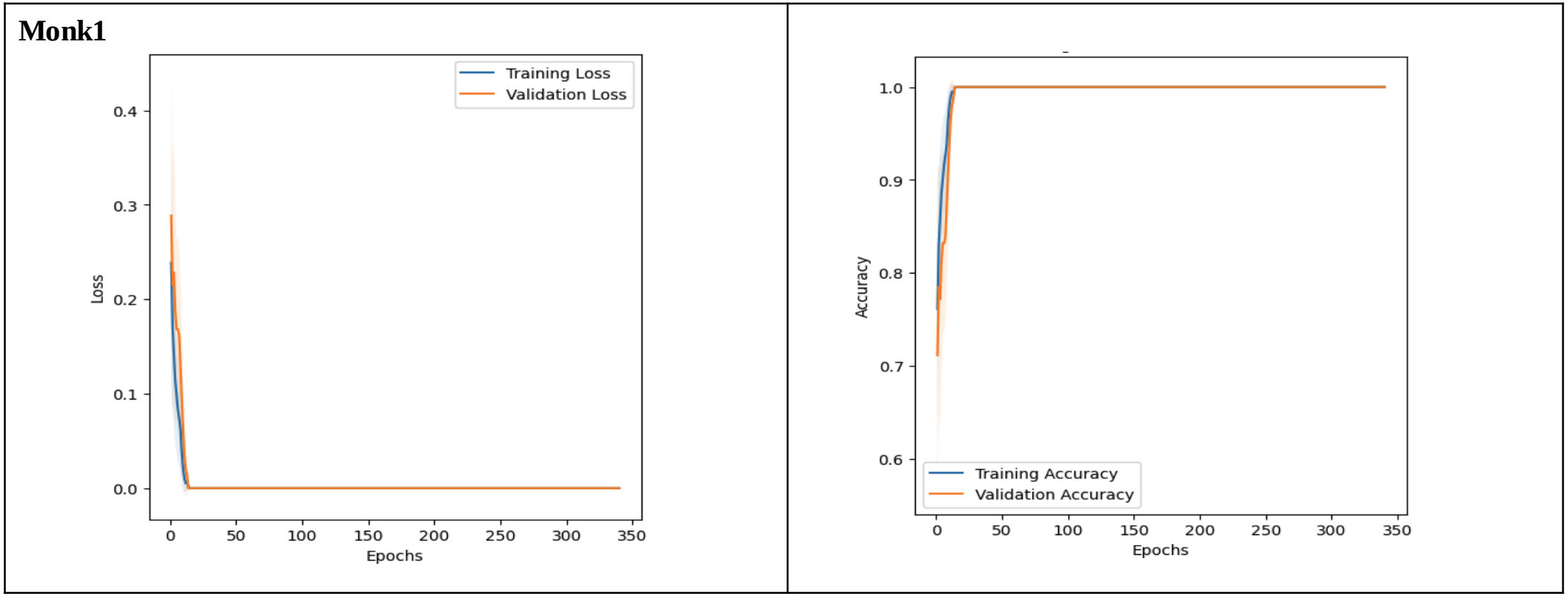

Table 8: Learning Curves and Accuracy on MONK 1

Monk Results 2 – Scikit-Learn

MSE

Accuracy

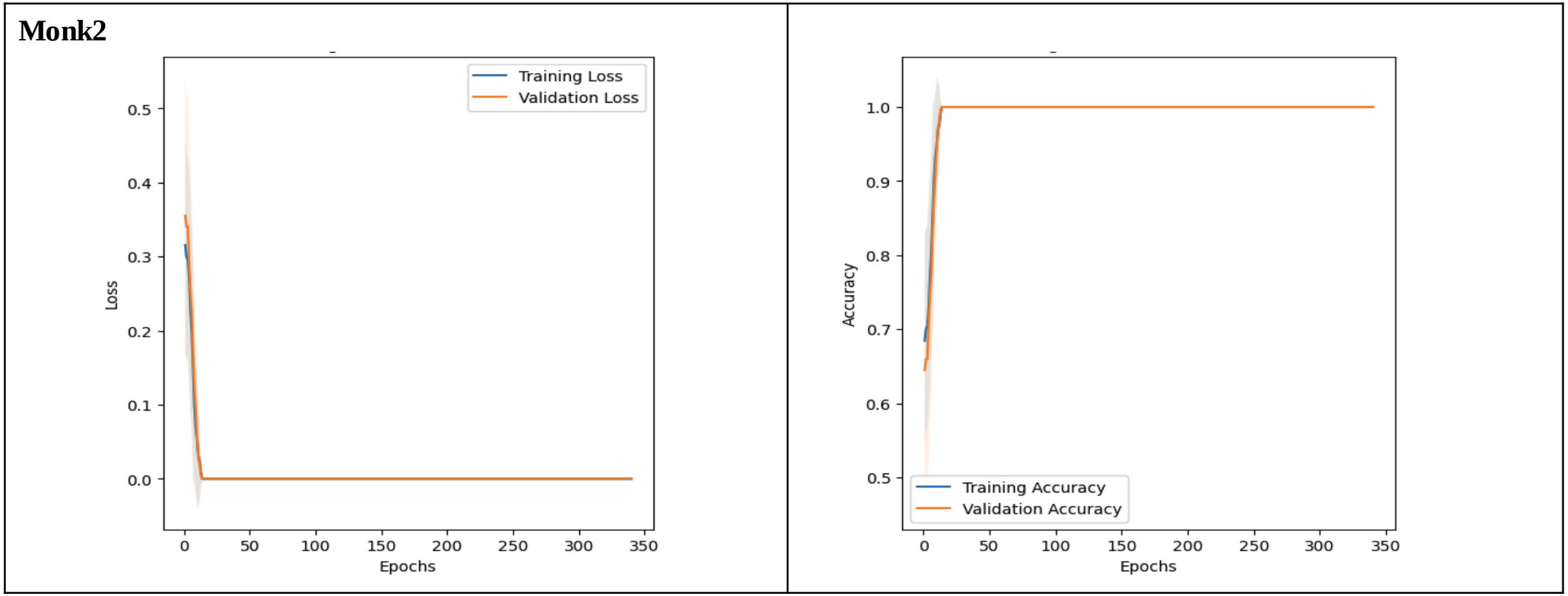

Table 9. Learning Curves and Accuracy on MONK 2

Monk Results 3 – Scikit-Learn

MSE

Accuracy

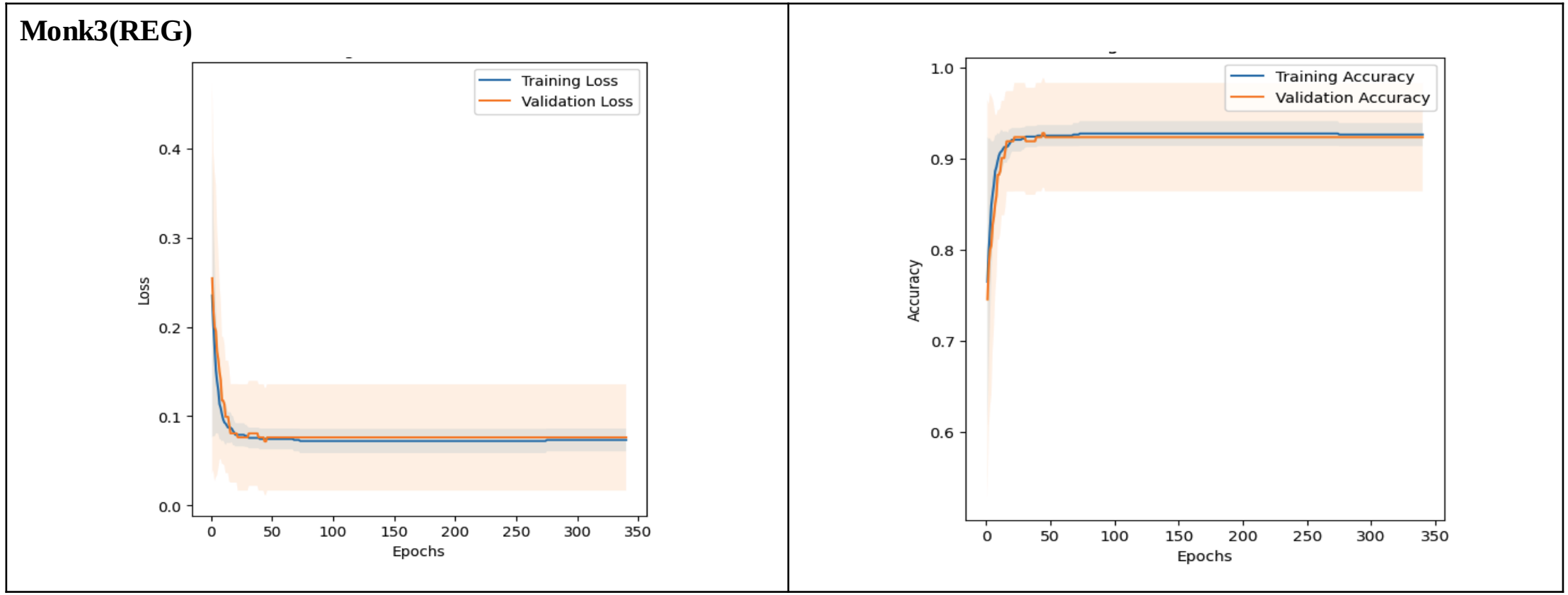

Table 10. Learning Curves and Accuracy on MONK 3 with L2 regularisation

Monk Results 3 – Scikit-Learn

MSE

Accuracy

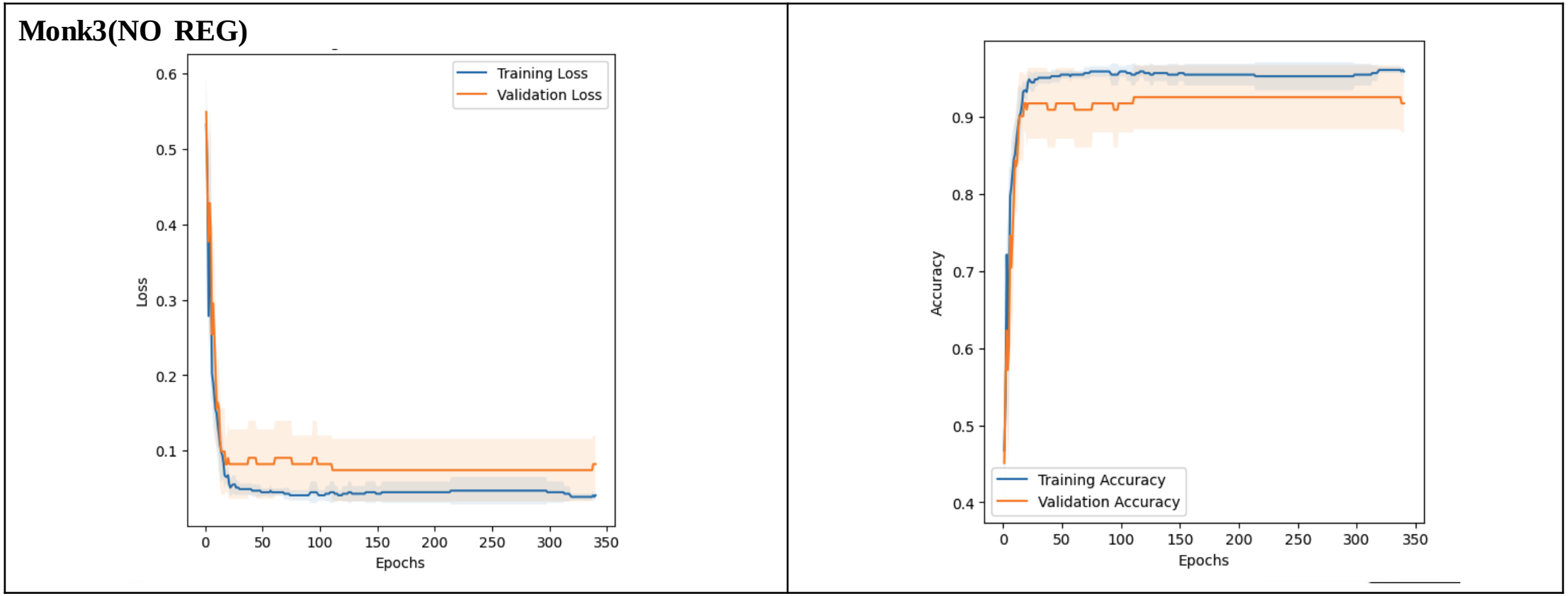

Table 11. Learning Curves and Accuracy on MONK3 not regularised. Evident overfitting starting from the 10th epoch.

Monk Results - Pytorch

- We implemented a Pytorch MLP with one hidden layer and 3-4 hidden units in a single hidden layer.
- We got 0.99 on validation set for MONK1, MONK2 in about 2/3 trials with an *Exhaustive Grid Search*.
- We implemented the early stopping using *patience* and *tolerance* set respectively to 10 and 0.01.
- The Model Selection's strategy consisted in a *K-fold CV* with *Online-Batch Gradient Descend* used to search the best hyperparameters' combination.
- After retraining the best models on all dataset we rapidly go to the model assesment and evaluation phase.
- The Evaluation's Phase consisted in:
  - the predictions' calculation, using accuracy as a metrics.
  - the print of the ROC curves,
  - the print of the F1-score, the F2-score, the Confusion Matrix and the TS Accuracy.

Monk Results – Pytorch - Hyperparameters

After performed GridSearch, table 11 below shows the best hyperparameters found.

Task	**Monk 1**	**Monk 2**	**Monk 3(noreg)**	**Monk 3 (reg)**
Hyperparameters	3 units, η=0.96, λ=none, epochs=400, α=0.8	4 units, η=0.7, λ=none, epochs=400, α=0.7,	4 units, η=0.96, λ=none, epochs=400	3units, η=0.05, λ=0.001, epochs=400, α=0,
MSE (TR)	0.007691532743501398	0.006273953023095182	0.016183852533058235	0.026557108968263604
MSE (VL)	0.008263618670869707	0.0064739838821840385	0.04369383112269071	0.047550478111345815
MSE (TS)	6.893701120427592e-05	5.707780410880837e-05	0.042948439309942285	0.025029368504981563
TR Accuracy (%)	99%	99%	98%	98%
VL Accuracy (%)	99%	99%	95%	95-96%
TS Accuracy (%)	100%	100%	94%	97%

Table 12. Model hyperparameters, MSE and Accuracy on Monk's taks

Monk Results – Pytorch

MSE

Accuracy

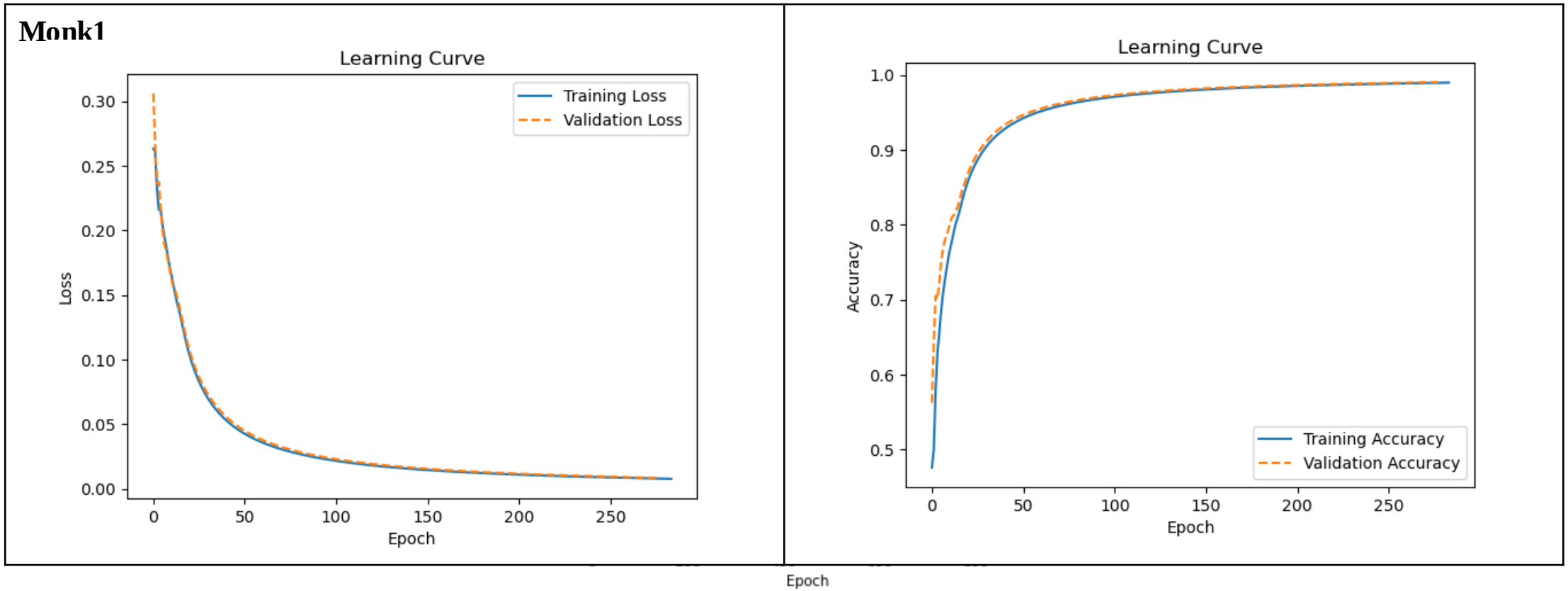

Table 13. Learning Curves and Accuracy on MONK1

Monk Results – Pytorch

MSE

Accuracy

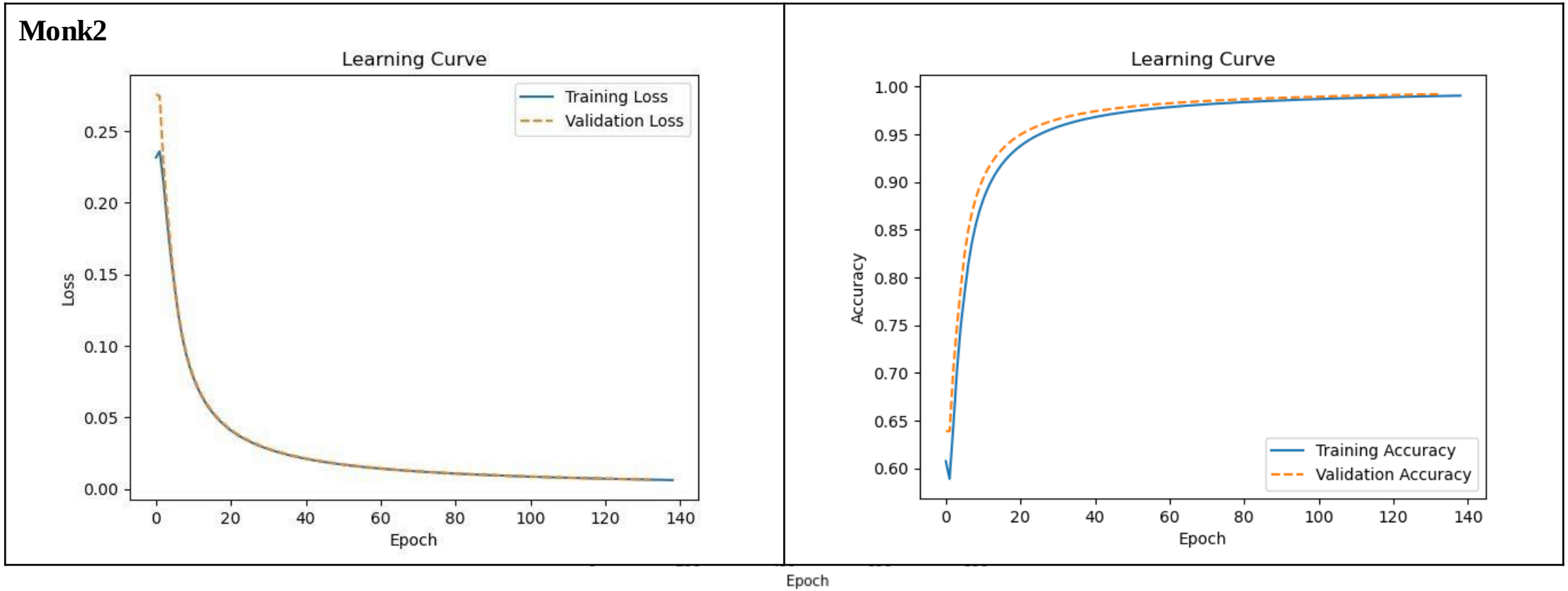

Table 14. Learning Curves and Accuracy on MONK2

Monk Results – Pytorch

MSE

Accuracy

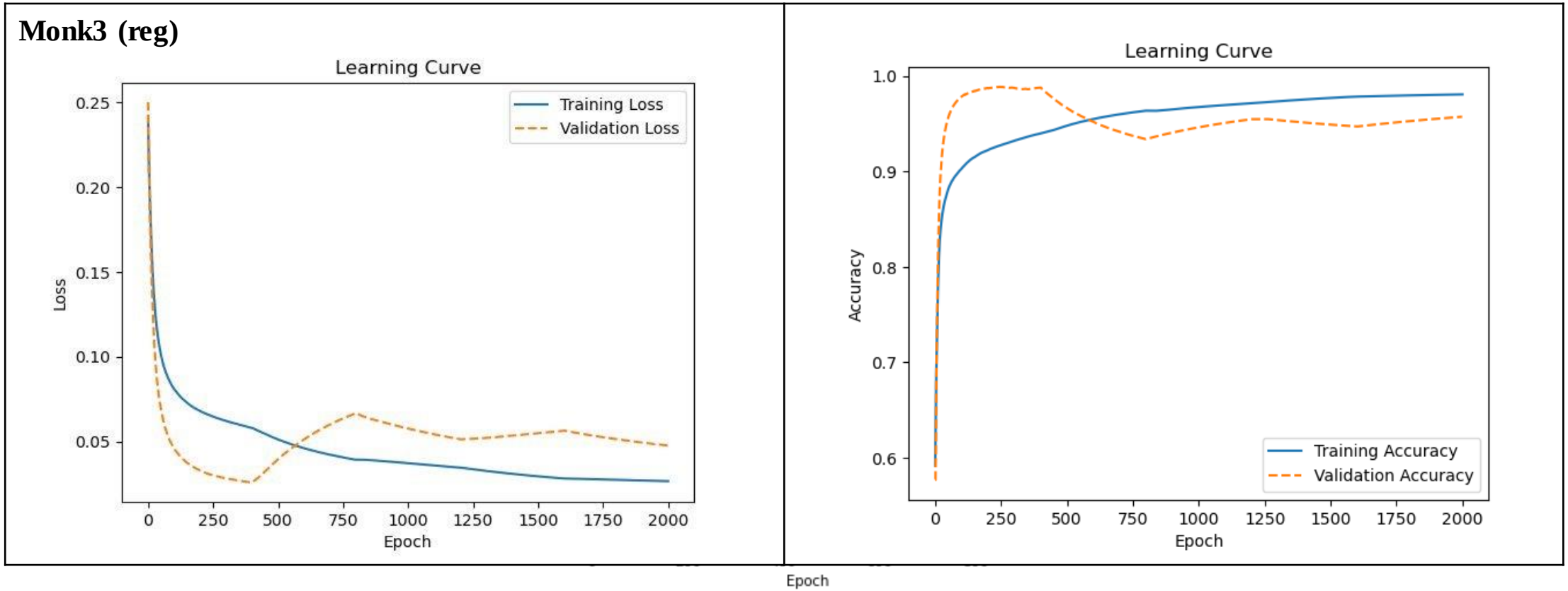

Table 15. Learning Curves and Accuracy on MONK3.

Monk Results – Pytorch

MSE

Accuracy

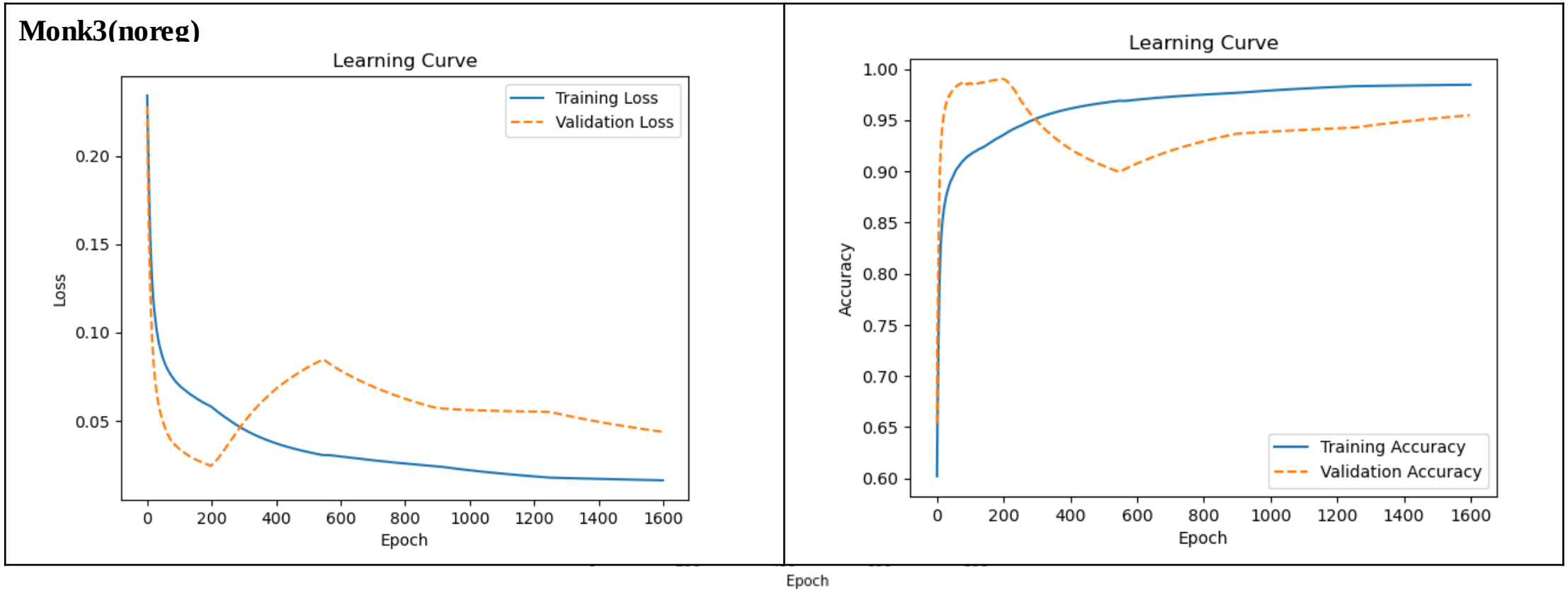

Table 16. Learning Curves and Accuracy on MONK3. Note the higher val loss that is the result of the overfitting

CUP Validation schema: model selection NN

- For the CUP, for the Neural Networks we decided to split dataset in 80% TR and 20% as our (internal) TS. We then performed *Randomized* and *Exhaustive Grid Search* in model selection phase, using *K-Fold Validation* (k=5) to search for the best hyperparameters. Hence, we retrained the found model on all data (without any folds and validation set) and we evaluated it on our internal test. The tried values for the *GridSearch* and the time required are reported below in Table 8.
- We used 2 hidden layers for Keras and 4 hidden layers for Pytorch, and as activation function *tanh* with a linear output with 3 units.
- As loss function we used MSE, whereas as a metrics we used MEE, as requested. In the Table 16, GridSearch values for models in Keras and Pytorch and tools are depicted.

Table 17. GridSearch values for research best hyperparameters in Keras and Pytorch

Models	**η**	**α**	**λ**	**epochs**	**batch**	**patience**	**Num units**	**GridSearch Time**
Pytorch NN	[0.00345, 0.0032]	[0.9, 0.8]	[0.0001]	[700, 800]	[64]	[10]	[80, 100]	~100
Keras NN	[0.03, 0.02]	[0.7, 0.8]	[0.0002, 0.0001]	[650,700]	[64, 128]	[15, 30]	[65, 80]	~ 60

CUP Validation schema: model selection SVM

For the CUP's regression we also computed a support vector machine regression exploiting Sklearn. These are the steps followed: normalization of our data, splitting the data (holdout method) reserving 20% of them to an internal test set to use for the model assessment, training of three different univariate SVR, having 3 class to predict, and GridSearchCV to find the best hyper-parameters. Here below the ones that suited our problem at best.

In the next slide, combined metric goodness is reported.

CLASS	C	degree	epsilon	gamma	kernel	max_iter	GridSearch Time
Class X Class Y Class Z	10	2	0.0001 0.01 0.001	scale	rbf	340	~30

Table 18. GridSearch values for research best hyperparameters in Support Vector Regression

Cup results

The table 19 below shows the "competitors" for the CUP, with *MEE on* TR, VL and Internal TS. Based on internal test results, we consequently choose the Pytorch model.

Model	**Training MEE**	**Validation MEE**	**Internal Test MEE**
Keras NN	0.21802478909492494	0.46461278200149536	0.6990148336991854
Pytorch NN	0.2759609964769802	0.42374866240365067	0.49628444015979767
Sklearn SVR	1.8709765363243005	1.6883674297493021	1.3154145175171537

Table 19. Goodness metrics of the models performed on the CUP regression task

CUP Result - Best Model in Pytorch

- According to previous slide, our best model was in Pytorch, which we trained for 700 epochs with a batch size of 64, as suggested by the *GridSearch*. Below in the Table 9 best values found and *MSE* and *MEE* values are depicted on both TR, VL and (internal)TS.

Task	**CUP Regression**
Hyperparameters	100 units, η=0.00345, λ=0.0001, epochs=700
MSE (TR)	0.2580992881336508
MSE (VL)	0.2884308021994578
MSE (TS)	0.1482980940490961
MEE (TR)	0.27599806940704563
MEE (VL)	0.42374866240365067
MEE (TS)	0.49628444015979767

Table 20. Best model hyp. And MEE, MSE

CUP result – Best model's Plots

MSE

MEE

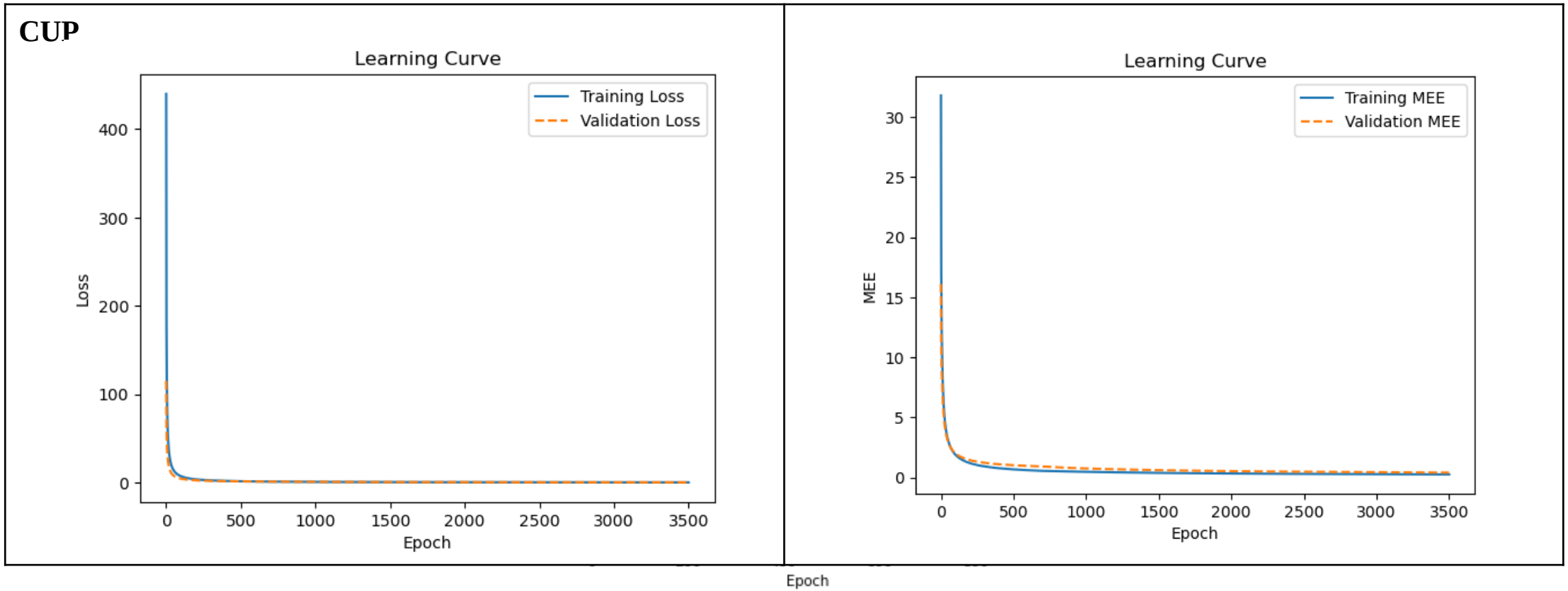

Table 21. Best model' s learning curves on CUP regression task

Discussion

- We found our experiments very usefult to remark the fact that doesn't exist *the* best model, but only the *better* model (among others) for a *particular* task. This is more evident when we compare different models on different tasks and even using different software.
- By implementing the GridSearch approach, we strongly use the empirical factor which allow us to conduct different experiments and trials until we find the more relevant. Although the ML theory it's based on true evidences, it can't be separated from the experiments. For instance, in the MONKS cases we concretely demonstrated that even if SGD requires a low learning rate, this is not a must in every cases, because probably it depends *also* on the tasks and the dataset. Indeed, on MONK3 and MONK2 we used the mentioned decay but not in MONK1, leaving an higher eta without seeing the expected diverging. The theory is surely correct to follow, but only the experiments can show us the right way.

- Although we didn't show it, we implemented also an other regressor model, KNN regressor which is called a *lazy, memory-based* model. We obtained poor results and decided to not report it. This small comparison between NN, SVR and KNN showed us the versatility and better capacities of NN over other machine learning models, especially over those considered lazyes as KNN is.

Conclusions

As already discussed in first slides, Pytorch is a low-level framework. While this often means more accurate results and performance, it could really make the development slower. As relevant strengths, with Pythorch it's possible to inspect deeper "what happens" and this really helps for debugging and didactic aims. Furthermore, Pytorch offers methods and classes to handle Tensors and layers in a very simple way.

We found Keras, instead, faster than Pytorch even if this often means worse performances. Keras is simpler since it's a high-level framework and this is the reason why it's widely used nowdaays.

Finally, Scikit-Learn resulted very user-friendly thanks to its simplicity and extensive documentation. Despite lacking of some implementations proper of deep learning, we found it a complete and powerful tool to perform classification and regression thanks to its numerous techniques for hyperparameter tuning and different cross validation options.

Blind Test Results: name of the result files and your nickname

ACKNOWLEDGEMENTS

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

Bibliography

- [1] Matplotlib site: https://matplotlib.org/
- [2] Numpy site: https://numpy.org/
- [3] Pandas site: https://pandas.pydata.org/
- [4] keras site: https://keras.io/api/
- [5] Pytorch site: https://pytorch.org/
- [6] Scikit-learn site: https://scikit-learn.org/
- [7] Refinetti, M., Ingrosso, A. & Goldt, S.. (2023). Neural networks trained with SGD learn distributions of increasing complexity. Proceedings of the 40th International Conference on Machine Learning, in Proceedings of Machine Learning Research 202:28843-28863 Available from https://proceedings.mlr.press/v202/refinetti23a.html.
- [8] Berrar, Daniel. (2018). Cross-Validation. 10.1016/B978-0-12-809633-8.20349-X.

Appendix A

MSE *MEE*

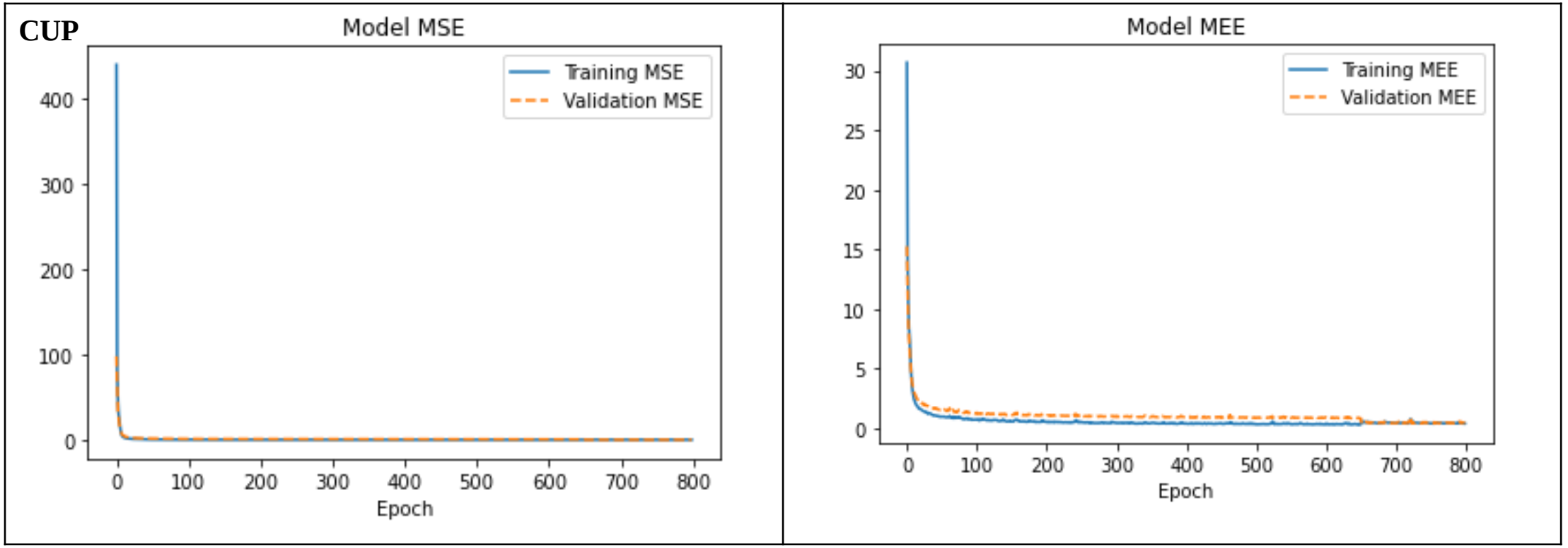

Table 10. Learning Curve (MSE) and model MEE on Keras model CUP

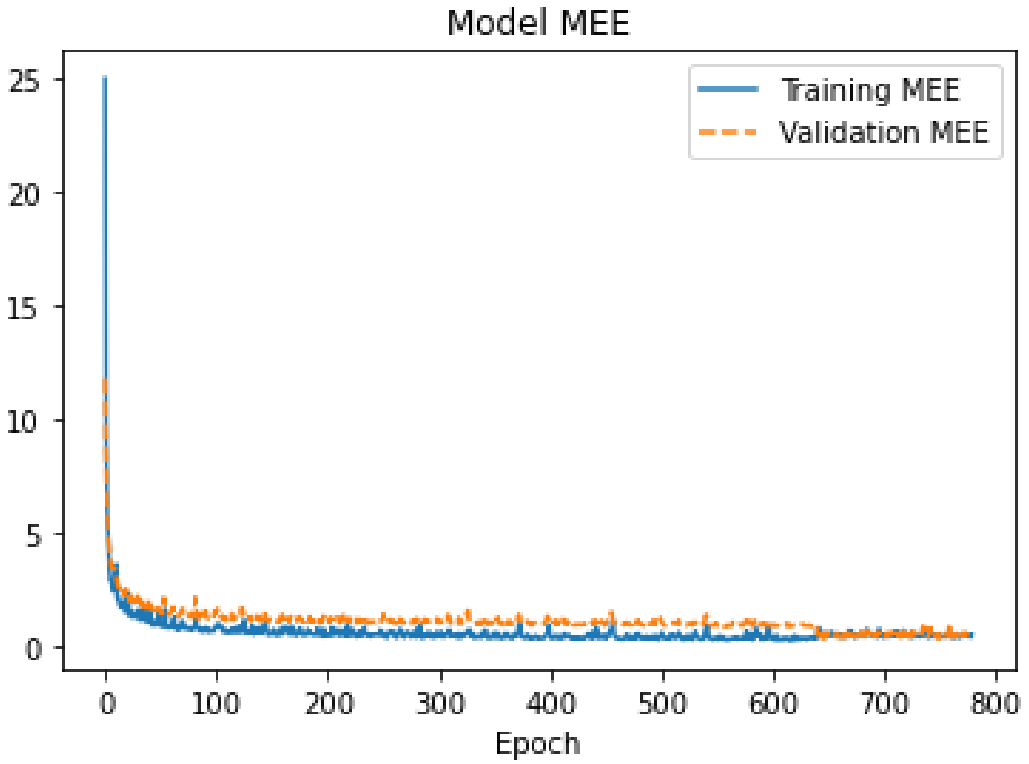

Figure . MEE using Nesterov, note the small spikes and the reduced smoothness.