Machine Learning Project Report

ML Course (654AA), A.Y. 2022/2023

Date: 22/01/2023

Type of Project: B

Authors

Diego Arcelli, 647979, Master Degree in Computer Science,
Curriculum Artificial Intelligence
Dylan Nico Ambrosi, 578586, Master Degree in Computer Science,
Curriculum Artificial Intelligence
Mariami Narchemashvili, 634789, Master Degree in Computer Science,
Curriculum Data Science and Business Infrormatics
Emails: d.arcelli@studenti.unpi.it, d.ambrosi1@studenti.unipi.it, m.narchemashvili@studenti.unpi.it

Abstract

In this work we evaluated the performances of different machine learning models, namely Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors and Neural Networks, using different machine learning libraries, in order to select the best model for a regression task. For the neural networks, we also tested a randomized approach. Finally, we considered an ensemble of the three best models.

1 Introduction

The goal of the project was to compare different Machine Learning models and libraries. The models we considered were SVM, Decision Tree, Random Forest, KNN, and Neural Networks. First, we trained those models to perform the classification task on the three MONK datasets, then we trained them on the 2022 ML CUP dataset to perform a regression task, comparing the performances of the models. For the CUP, we additionally trained an Extreme Learning Machine, and in the end, we considered an ensemble of the three best-performing models for the model assessment.

2 Methods

For all the models we executed a grid search to find the best configuration of hyperparameters, performing model selection using K-fold cross-validation. After the best configuration was found we re-trained the model on the whole training set and we performed a model assessment using hold-out. We performed all the experiments on two of our laptops, one with an Intel i5-7200U CPU and the other with an M1 processor.

Neural Networks

We used two different libraries for the implementation of the neural networks, `Keras` and `Pytorch Lighting` (we refer to the latter as `Pytorch` for the sake of brevity). In both cases, the code to execute the grid searches has been implemented from scratch. For the `Keras` implementation we exploited the default weights initialization, which is the Glorot uniform [1][2], where the initial value of each weight is sampled from the uniform distribution $[-limit, limit]$ where $limit = \sqrt{6/(fan\_in - fan\_out)}$, where fan_in is the number of input units of the layer and fan_out is the number of output units of the layer. Biases are initialized to zero. Also for the `Pytorch` implementation, we used the default weights initialization [3], where weights and biases are initialized sampling uniformly from the distribution $[-limit, limit]$ where $limit = \sqrt{1/(fan\_in)}$. In both cases, we trained the models using gradient descent with momentum, using the MSE as a loss function. Moreover, we used early stopping to halt the training if the validation loss does not improve within 20 epochs, and by setting the maximum number of epochs to 500. When we do the final retraining on the best model, in order to apply early stopping we extract the 20% of the training set to be used as the validation set so that the validation loss can be monitored to apply early stopping. During the final retraining, we fit the model 5 times using different weights initialization, from which we select the median model with respect to the validation error. The hyperparameters considered were the learning rate η, the regularization term λ, the momentum α, the number of units for each hidden layer and the number of hidden layers.

Extreme Learning Machine

The ELM has been implemented using `Keras`. We used only one hidden layer and the first weight matrix is initialized randomly, then the second weight matrix is computed directly using the Least Mean Square (LMS) normal equation. We also used L2 regularization. The hyperparameters considered for the grid searches were the regularization term, the number of units of the hidden layer, and the activation function used for the hidden layer.

Decision Tree and Random Forest

Decision trees are ML models where a tree-like structure is built such that each internal node corresponds to a test on the value of a feature of the training set. Based on the outcome of this test, the internal node will point towards other nodes, and the process is repeated until a leaf node is reached. In classification tasks, leaf nodes correspond to classes, while in regression they correspond to real values. To make a prediction for an input pattern, the pattern traverses the tree from the root to a leaf, and it is assigned to the value of that leaf. Random forests are ensembles of decision trees, that combine the predictions of many different trees. For the implementation of decision trees and random forests, we used the `Scikit-learn` library. In both cases, we used the `GridSearchCV` class to perform the grid search. For decision trees, we performed a grid search to tune the following hyperparameters: *ccp_alpha* used as a regularization parameter for pruning, *min_weight_fraction_leaf* as the minimum weighted fraction of the sum total of weights, *min_samples_split* that is the minimum number of samples required to split an internal node, *max_depth* of the tree, *criterion* which measure the quality of the split

and *splitter* parameter as the strategy used to choose the split at each node. For random forest we proceeded in the same way, tuning an additional hyperparameter, the *number of estimators*.

Support Vector Machine

For the implementation of support vector machines, we used the `Scikit-learn` library, again resorting to `GridSearchCV` to execute the grid search. Since the SVM is a single target regressor and the CUP dataset has two targets, we decided to use the `MultiOutputRegressor` class of `Scikit-learn`, which allows to fit one SVM per target. The drawback of this approach is that when we execute the grid search, we have to use the same hyperparameters for both SVM models. The alternative would have been executing a grid search separately for each one of the two SVMs, but in this case, each gird search would have returned two separate validation errors, which cannot be compared with the validation error of the other models, therefore we decided to use `MultiOutputRegressor`. The kernels considered for the grid search were linear, polynomial and radial basis function. For the polynomial, we considered the degree of the polynomial and for the radial basis function we consider the γ value which is used in the RBF kernel formula.

K Nearest Neighbors

For the implementation of KNN, we used the `Scikit-learn` library. KNN can be extended for regression tasks by assigning to the pattern to predict the average of the K nearest neighbors instead of the majority vote. We performed a grid search to tune the following parameters: *n_neighbors*, *weights*, *algorithm*, using as distance metric the euclidean.

3 Experiments

3.1 Monk Results

For the three monks we applied one hot encoding to the features of the datasets, then for each model we executed a grid to find the best hyperparameters. For the two neural networks, we used gradient descent computing at each epoch the gradient over the whole training set. The output layer has one unit and it uses the sigmoid as activation function. For the MONK3 dataset, we additionally used L1 regularization. Model selection has been done using 4-fold cross-validation. The results and the hyperparameters used for the training are reported in tables 1 and 2, while the learning curves can be seen in figures 1, 2, 3, 4, 5 and 6.

Task	**#Units**	$\boldsymbol{\eta}$	$\boldsymbol{\alpha}$	$\boldsymbol{\lambda}$	**MSE (TR/TS)**	**Acc. (TR/TS)(%)**
MONK1	8	0.8	0.7	0	0.0002/0.004	100%/100%
MONK2	8	0.1	0.8	0	0.0001/0.0002	100%/100%
MONK3	16	0.01	0.7	0.001	0.12/0.10	92.8%/96.5%

Table 1: Median prediction results of `Keras` networks for MONKs

For all the other models we applied 5-fold cross-validation using as a metric the validation accuracy. The performances of decision trees are reported in table 3. Due to space problems,

Task	#Units	η	α	λ	MSE (TR/TS)	Acc. (TR/TS)(%)
MONK1	8	0.2	0.8	0	0.003/0.007	100%/100%
MONK2	8	0.7	0.5	0	0.0013/0.0015	100%/100%
MONK3	16	0.05	0.8	0.001	0.068/0.043	92.7%/97.2%

Table 2: Median prediction results of `Pytorch` networks for MONKs

we refer to min_sample_leaf, min_sample_split, min_weight_fraction_leaf and max_depth as MSL, MSP, MWFL and MD.

Task	Alpha	MD	MSL, MSP, MWFL	Criterion	Splitter	Acc. (TR/TS)(%)
MONK1	0.01	10	1, 2, 0.005	None	None	98%/96%
MONK2	0.0001	41	1, 2, 0.0001	Gini	Random	100%/81%
MONK3	0.1	7	None	Gini	Random	93%/97%

Table 3: Accuracy obtained with decision trees

The performances of random forest after the grid search are referred to in table 4.

Task	Alpha	n_estimators	Criterion	Acc. (TR/TS)(%)
MONK1	0.0005	65	Entropy	100%/95%
MONK2	0.0001	85	Gini	100%/73%
MONK3	0.0001	120	Log_loss	93%/96%

Table 4: Accuracy obtained with random forests for MONKs

The performances of KNN after the grid search are referred to in table 5.

Task	n_neighbors	Weights	Algorithm	Acc. (TR/TS)(%)
MONK1	0.0005	65	Entropy	100%/95%
MONK2	0.0001	85	Gini	100%/73%
MONK3	0.0001	120	Log_loss	93%/96%

Table 5: Accuracy obtained with KNN for MONKs

The performances of SVMs are referred to in table 6.

Task	Kernel	C	Gamma	Acc. (TR/TS)(%)
MONK1	rbf	100	0.05	100%/100%
MONK2	rbf	10000	0.01	100%/100%
MONK3	linear	1	scale	93%/97%

Table 6: Accuracy obtained with SVM classifier for MONKs

3.2 Cup Results

For the 2022 ML CUP dataset, we split the training set into an internal training set (80% of the data) and an internal test set (20% of the data). For each model we executed a grid search on its hyperparameters, applying K-fold cross-validation for each configuration of hyperparameters. Then we compared the models with respect to the validation MEE. In general for the grid searches, we performed some preliminary trials, by testing large ranges for the values of each hyperparameter at the beginning, and then we repeatedly narrowed the range around the values which where preforming the best. We did this with particular care for the neural networks, since their training for just one configuration was extremely slow, while for models which are much faster to train, like the ELM, we executed the grid searches on larger ranges. For the preprocessing, we applied standard normalization to the attributes of the dataset for all the models, except for decision trees and random forests, since they are not affected by the different scales of the attributes. For all the models besides the neural networks, we used 5-fold cross-validation using as a metric the validation MEE, while for the neural networks we use 4-fold cross-validation using as a metric the validation MSE.

Neural Networks

For neural networks in addition to what we did for the monks, we also used a linear learning rate decay, which decreases linearly the initial value of the learning rate for 100 training epochs and it stops at 0.01. Moreover, we used a mini-batch gradient descent fixing the batch size to 16. This time we also added a required improvement of at least 0.01 for the early stopping, which means that if during the training the validation loss does not improve by at least 0.01 within 20 epochs the training is stopped. We did it since for the preliminary trials we noticed that without putting this requirement, the training process kept going for all the 500 epochs, but with almost no improvement after a few epochs, often resulting in overfitting. The output layer has two units (one for the target) and the activation function we used was the linear. From the preliminary trials, we noticed that no more than 3 hidden layers were needed to achieve good performances, and no more than 50 units per layer were needed. Moreover, we found out that alternating *tanh* and *ReLU* activation functions lead to the best performances. From the preliminary trials, we also noticed that in many cases the value of the loss function was returned as NaN, presumably because of exploding gradient, therefore we applied standard normalization also to the targets and this solved the problem. The preliminary trials gave the same results for both `Keras` and `Pytorch`, therefore we used the same values of the hyperparameters for both implementations. The values of the hyperparameters tested in the final grid search are reported in table 7. For the configuration with 2 hidden layers the activation functions used where *tanh* for the first layer, and *ReLU* for the second, while in the configuration with 3 hidden layers, the third uses the *tanh*. The best `Keras` network had training and validation MEE of respectively **1.354** and **1.426**, while for the best `Pytorch` network the MEE was **1.560** on the training set and **1.569** on the validation set. The time required for the final grid search was around 3 hours for both implementations.

Parameter	Values
Hidden layers	[(10, 10, 10), (25, 25, 25), (50, 50, 50), (30, 30), (50, 50)]
1.426 Learning rate	[0.1, 0.2, 0.3]
Regularization	[0.001, 0.0001, 0.00001]
Momentum	[0.1, 0.2, 0.3, 0.4]

Table 7: Hyperparameters used in the grid search for neural networks

Model	Hidden layers	Eta	Alhpa	Lambda	Traing MEE	Val. MEE
Keras	(50, 50, 50)	0.3	0.1	1e-05	1.354	1.426
Pytroch	(50, 50, 50)	0.3	0.3	1e-04	1.560	1.569

Table 8: Best hyperparameters and MEE of neural networks

Extreme Learning Machine

We considered an ELM with just one hidden layer, where this first it is randomly initialized sampling the values of the weights from a Gaussian distribution of 0 mean and 1 variance, and then its weights are frozen, and only the second layer is trained using the Least Mean Square normal equation. For the output layer, we use the linear activation function, which is needed to compute the weights matrix using the normal equation. The hyperparameters on which we execute the grid search are the activation function of the first layer, the number of units of the hidden layer, and the value of the penalty term for the L2 regularization. The values tested for each parameter are reported in table 9. For the model selection, we apply a 4-fold cross-validation, and in this case, in order to handle the randomicity of the different weights initialization, during the cross-validation, for each train-validation split we fit the model three times and we consider the one with the median MSE on the validation set. We applied this strategy only for ELM since it is much faster to train than the neural networks trained with backpropagation, in fact, a grid search on 192 configurations took around 20 minutes. The best values were 1000 for the number of units, 100 for the regularization term, and *tanh* for the activation function. The training and validation MEE were respectively **1.307** and **1.586**.

Parameter	Values
Units	[10, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
Activation funtion	[relu, tanh]
Rergularization	[0.001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]

Table 9: Hyperparameters used in the grid search of the ELM

K-Nearest Neighbors

With the K-Nearest Regressor (KNR) we performed a grid search, as described in Section 3.2. The range of the hyperparameters is shown in Table 10. The resulting best model and the

MEE computed on TR/VL is a KNR with *algorithm=auto* and *n_neighbors=18*, with a MEE on TR/VL equal to **1.370/ 1.449**.

Hyperparameters	**Values**
n_neighbors	[2...90]
algorithm	[auto, brute, kd_tree, ball_tree]

Table 10: KNR hyperparameters ranges

Decision Tree and Random Forest

For both decision trees and random forests, we proceeded as described in Section 3.2. The acronyms MSL, MSP, MWFL stand for *min_samples_leaf*, *min_samples_split*, *min_weight_fraction_leaf*. The values of the hyperparameters searched are shown in table 11 and 12. The best model chosen by the grid search is a decision tree with *max_depth=7*, *Alpha=0.01*, *min_samples_leaf=1*, *min_samples_split=2* and *min_weight_fraction_leaf=0.005*, with a MEE on TR/VL equal to **1.323/ 1.800**. Then we performed a random search with a wider range of hyperparameters. The best model chosen has hyperparameters: *max_depth=6*, *MLS=4*, *MSP=16*, *Alpha=0.0001*, and *MWFL=0.0005*, with MEE on TR/VL equal to **1.343/ 1.795**. The grid search required 5 minutes approximately, while the random search around 1 minute.

Hyperparameters	**Values**
max_depth	[5,7,10, 20, 30, 40, 50]
alpha	[0.0, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.5, 0.2, 0.0002]
MSL	[1, 2, 3, 4]
MSP	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.5, 0.2]
MWFL	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.5, 0.2]

Table 11: Decision tree grid search hyperparameters ranges

Hyperparameters	**Values**
max_depth	[5 ... 90]
alpha	[0.0, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.5, 0.2, 0.0002]
MSL	[1 ... 20]
MSP	[1 ... 20]
MWFL	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.5, 0.2]

Table 12: Decision tree random search hyperparameters ranges

Regarding random forests, we performed both a standard grid search and a randomized one. Hyperparameters ranges are shown in table 13 and 14. The best model for the grid serch has hyperparameters: *alpha=0.0*, *min_samples_leaf=2*, *min_samples_split=5*, *min_weight_fraction_leaf=0.0*, *n_estimators=40*. The hyperparameters of the best model for the random search are: *alpha=0.0001*, *n_estimators = 95*, *min_samples_leaf=2*, *MSP=8*, *min_weight_fraction_leaf=0.0001*.

MEE on TR/VL sets is, **0.726/ 1.495** (for the grid search) and **0.822/ 1.496** (for the random search). The grid search required more or less 15 minutes, while the random search around 3 minutes.

Hyperparameters	**Values**
n_estimators	[10, 20, 30, 40]
alpha	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001]
MSL	[2, 3]
MSP	[3, 4, 5]
MWFL	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001]

Table 13: Random Forest grid search hyperparameters ranges

Hyperparams	**Values**
n_estimators	[2...100]
alpha	[0.0001 ... 0.0009]
MSL	[1 ... 20]
MSP	[1 ... 20]
MWFL	[0.0, 0.1, 0.05, 0.01, 0.005, 0.001,0.2, 0.02, 0.002, 0.0001, 0.0005, 0.0002]

Table 14: Random Forest randomized search hyperparameters ranges

SVM

For the SVM we did three different grid searches, one for the kernels considered, which were linear, polynomial and the radial basis function (RBF), since each one of these kernels has different hyperparameters. Finally, we selected the best configuration of hyperparameters among the three grid searches, which was RBF for the kernel, $C = 10$, $\epsilon = 0.3$, and $\gamma = 0.1$. The values of the hyperparameters tested for each grid search are reported in table 15. The training and validation error were respectively **1.239** and **1.474** .The time required for the grid search was around 20 minutes.

Kenel	**C**	**epsilon**	**param**
Linear	[5,10,15,20,25,30]	[0.1,0.3,0.5,0.7,0.9]	-
Rbf	[0.1,0.5,1,10,20,30,40,50]	[0.1,0.3,0.5,0.7, 0.9]	[0.01,0.05,0.1,0.5,1]
Polynomial	[1,10,25,50,75,100]	[0.1,0.3,0.5,0.7, 0.9]	[2,3,4,5,6]

Table 15: Hyperparameters used for SVM grid seraches

Model	Training MEE	Validation MEE
Keras MLP	1.354	1.426
Pytorch MLP	1.560	1.569
ELM	1.307	1.586
Decision tree	1.343	1.795
Random forest	0.726	1.495
SVM	1.239	1.474
KNN	1.370	1.449

Table 16: Training and validation MEE of all the models

4 Model assessment

For the model assessment, we decided to consider an ensemble of the best models based on their validation MEE, which are summarized in table 16. The three best performing models are the `Keras` neural network, the SVM, and the KNN, which have a validation MEE smaller than 1.5, this is true also for the random forest, but we didn't consider it for the ensemble since given the huge difference between the training and validation MEE we can conclude that the model overfitted. So we computed the average of the prediction of the three considered models, and we use it to measure the MEE on the internal training and test set which were respectively **1.317** and **1.427**. After that, we used the three models to compute the average of the predictions for the blind test set.

5 Conclusion

Table 16 summarize the performances of the models we trained on both training and validation set. For the reasons we explained before, we use for the blind test set, the average of the predictions of SVM, KNN an the `Keras` network. The result is in the file `team-NAA_ML-CUP22-TS.csv`.

Acknowledgments

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

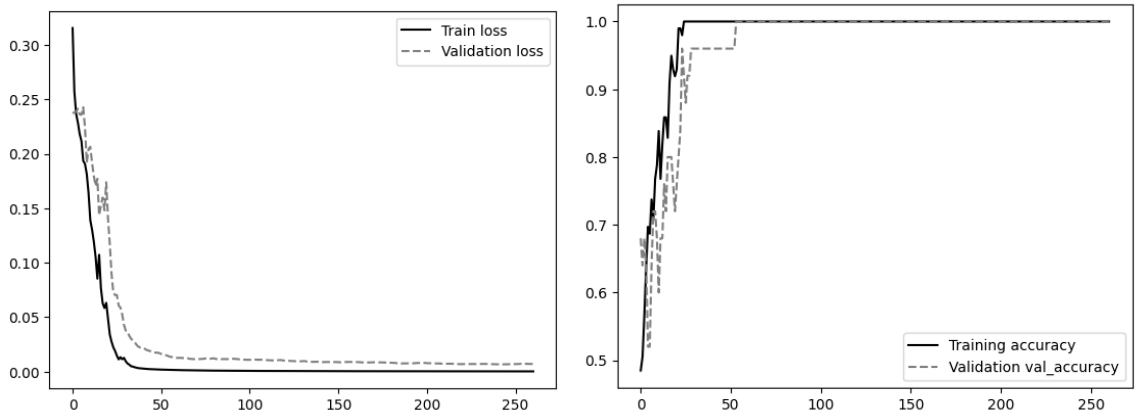
References

[1] Glorot uniform keras. https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotUniform.

[2] Keras dense layer documentation. https://keras.io/api/layers/core_layers/dense/.

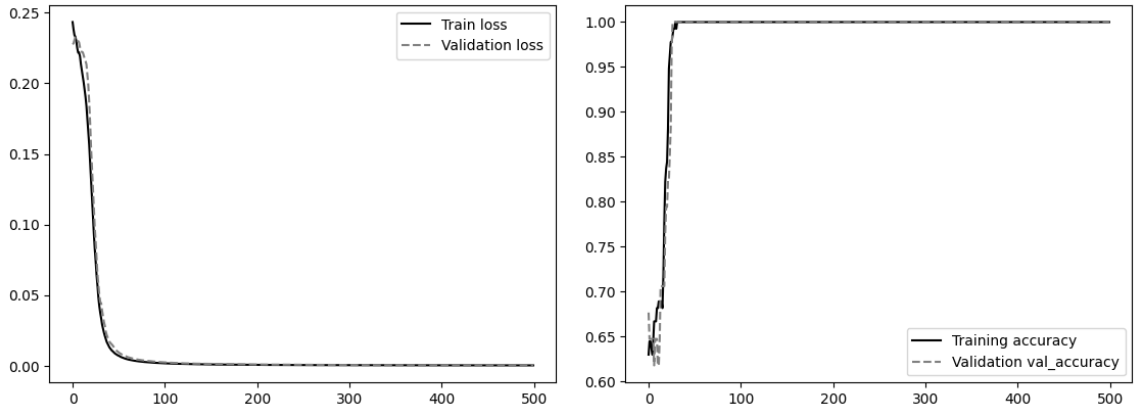
[3] Pytorch documentation. https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear.

Appendix A



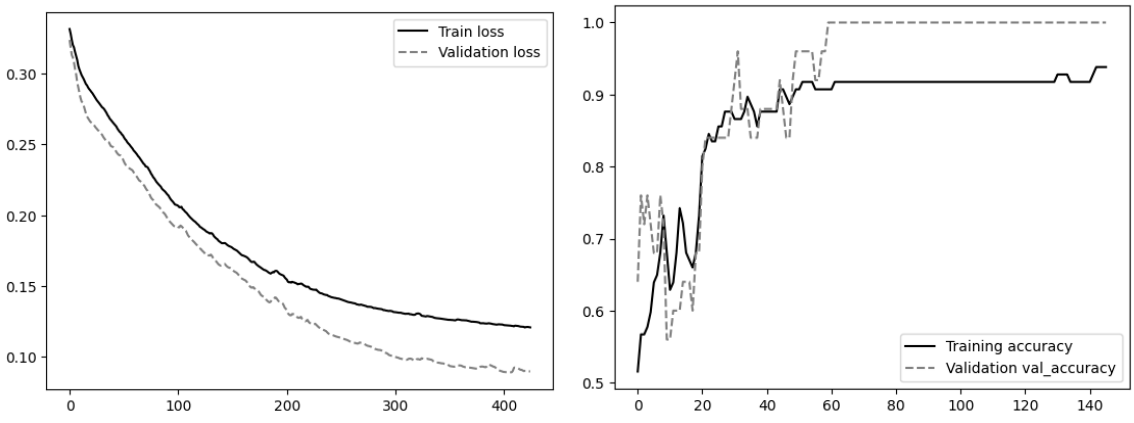
(a) Loss (b) Accuracy

Figure 1: MONK1 learning curves `Keras`



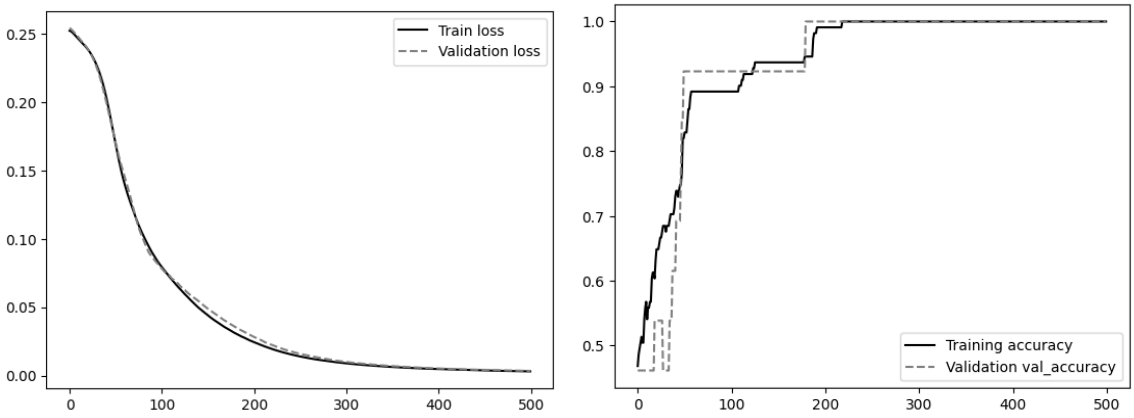
(a) Loss (b) Accuracy

Figure 2: MONK2 learning curves `Keras`



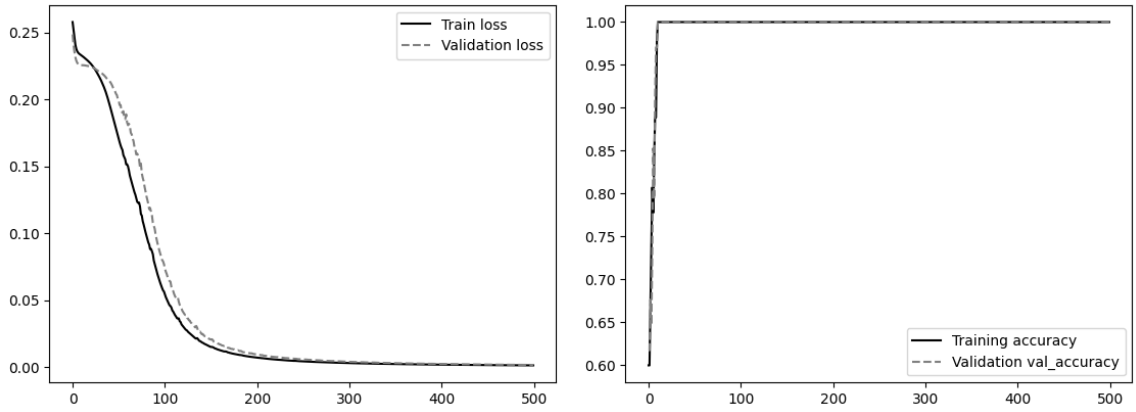
(a) Loss (b) Accuracy

Figure 3: MONK3 learning curves `Keras`



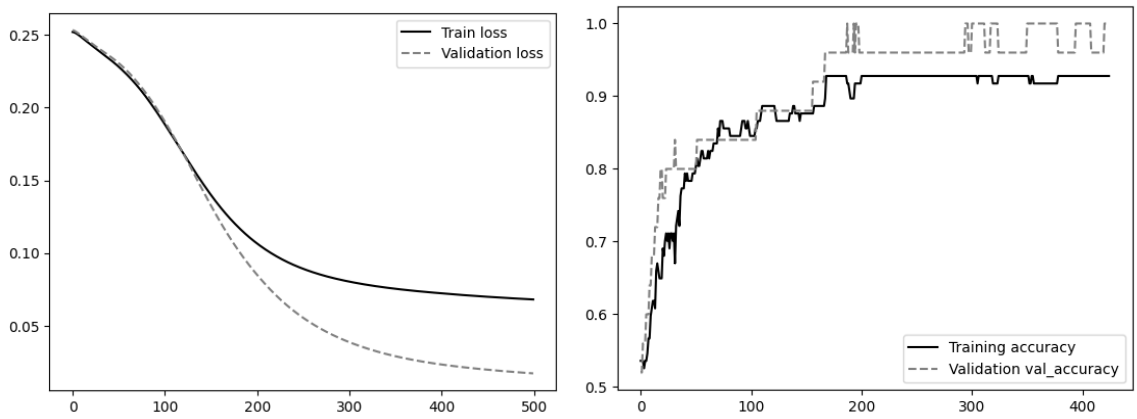
(a) Loss (b) Accuracy

Figure 4: MONK1 learning curves `Pytorch Lighting`



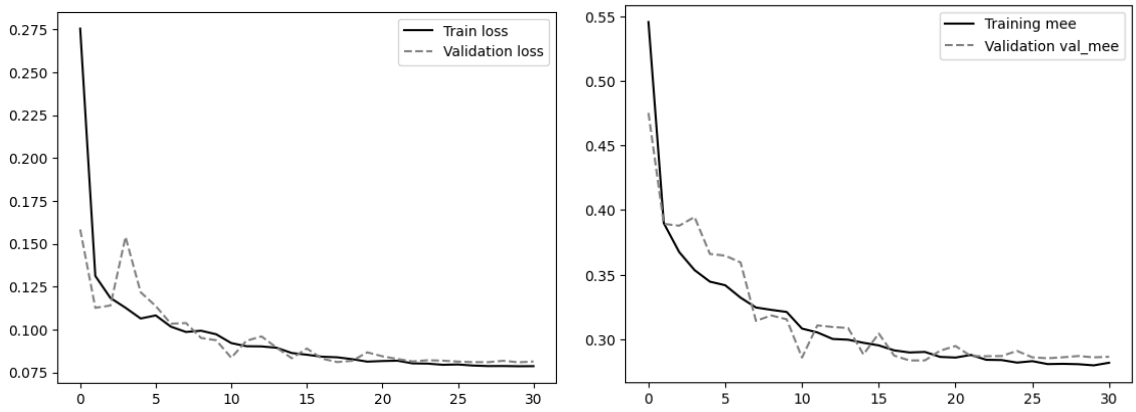
(a) Loss (b) Accuracy

Figure 5: MONK2 learning curves `Pytorch Lighting`



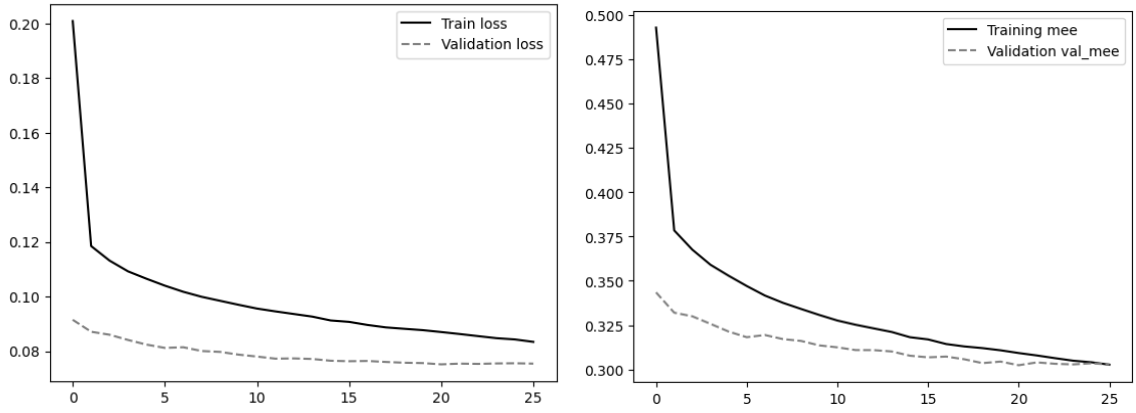
(a) Loss (b) Accuracy

Figure 6: MONK3 learning curves `Pytorch Lighting`



(a) Loss (b) MEE

Figure 7: 2022 ML CUP learning curves `Keras`



(a) Loss (b) MEE

Figure 8: 2022 ML CUP learning curves `Pytorch Lighting`