Machine Learning Report

Giacomo Fidone* Davide Grande† Bruno Limon‡

Machine Learning Course [654AA]
University of Pisa
A.Y. 2022/2023

Project Type B

Abstract

This report describes a comparative analysis of competitive Machine Learning models for the resolution of the MONK's classification problems and the development of a well-performing model for the CUP multi-regression task.

*Master's Degree in Digital Humanities (Language Technologies) – g.fidone1@studenti.unipi.it
†Master's Degree in Physics (Complex Systems) – d.grande1@studenti.unipi.it
‡Master's Degree in Data Science & Business Informatics – b.limonavila@studenti.unipi.it

Contents

1 Introduction

In this report we will discuss about the development of several Machine Learning models and the analysis of their relative performance. The learning tasks provided are three binary classification tasks and a multi-regression task based, respectively, on the MONK's benchmark (consisting of the three datasets MONK-1, MONK-2, MONK-3) and on the CUP dataset. Since each task requires the consideration of different algorithms, our discussion will be organised into two sections.

The first part considers a comparative evaluation of Neural Networks (NN), Support Vector Machines (SVM), K-Nearest Neighbors (K-NN), Logistic Regression (LR), Decision Tree (DT), Random Forests (RF), Gradient Boosting (GB) and Bernoulli Naïve Bayes (BNB) for solving the MONK's classification tasks.

The second part will be aimed at finding the most performing model among different competitors – NN, SVM, K-NN, RF, Ordinary Linear Regression (OLR) Ridge, Lasso – for the multi-regression CUP task.

2 Method

In both tasks the analysis and development of each model has relied on Python-based tools: *Scikit-learn* for data pre-processing, training, model selection and model evaluation; *Tensorflow* and *Keras*, limitedly to the development of NNs; support libraries (*Pandas*, *Numpy*, *Matplotlib*, *Seaborn*) for data manipulation, numerical computation and data visualization; further user-defined functions for more specific needs.

2.1 MONK

MONK datasets are already provided with a partitioning into training and test data. Pre-processing operations have thus involved the one-hot encoding of each categorical attribute and the partitioning of data into selection and validation data (30%) for hold-out validation. Model selection will also rely on K-Fold cross-validation, which will make use of a stratified 5-Fold with a fixed random state of 42 for reproducibility ends. As for model evaluation, we follow the given partitioning and perform a simple hold-out.

2.1.1 Neural Networks

The procedure we have followed in the study of *Keras* NN starts from a default feed-forward configuration including: random normal as initializer (with 42 as random state), ReLu as hidden activation function, Sigmoid as output activation function, Tikhonov (L2) as regularizer, Stochastic Gradient Descent (SGD) as optimizer, and Mean Square Error (MSE) as loss function. A limit of 50 epochs is set for restricting the search to models with a faster convergence time.

We therefore consider a first grid search over a coarse space of hyper-parameters common to each of the three tasks, shown in Table 1.

Hidden units	2, 3, 4, 5, 6
η	0.01, 0.1, 1
α	0, 0.01, 0.1
Batch size	2, 8, 32
λ	0, 0.01, 0.1

Table 1: Hyper-parameters grid used during model selection on NNs

Since we want to reach optimal results keeping the complexity of the network as low as possible, the number of neurons has been restricted to a small range and refers to a single hidden layer.

If the search results in a unsatisfactory validation score, further searches on finer-grained grids may be considered. For each search, since all scorer objects follow the convention that higher return values are better than lower return values, we have adopted the negative mean squared error.

Given the best hyper-parameter configuration, the model is trained again to visualize the history of the learning curves on training and validation data and assess both the convergence time and the risk of large variance. In case of overfitting, either a direct regularization technique (early stopping) and/or a different hyper-parameter configuration can be considered.

As the model has been optimized on a particular set of initial weights, a further improvement concerns its performance on different random weight initializations. Furthermore, we want to compare the behavior of the different batch modes, initializers, activation functions and optimizers shown in table 2.

Batch size	1, 2, 8, 32, 64, 128
Weight initializer	random_normal, random_uniform, he_normal, he_uniform, glorot_normal
Activation function	ReLu, SeLu, sigmoid, tanh
Optimizer	Adam, SGD, RMSprop

Table 2: Comparison parameters.

For handling both tasks, we use the custom function `begin_comparison()`, which creates (from a given default network) n different models by varying the value of a specified parameter (where n is the number of values for that parameter); and train each model on 50 different weights initializations. Each model is therefore assessed w.r.t. its average accuracy on validation set. Following a greedy approach, each comparison is performed by selecting from time to time the best parameter configuration. A final hold-out evaluation is performed by re-training the best resulting model on all training data with 50

different weights initializations for computing the average accuracy and loss on test data.

Alongside *Keras*, we have considered the NN implementation offered by *Scikit-Learn* MLP. After a brief check of loss curves over 50 iterations with different learning strategies – constant learning rate, adaptive learning rate, inverse-scaling learning rate (with and without momentum and Nesterov), Adam – we have decided to opt for *SGD* with adaptive learning rate and Nesterov's momentum (MONK-1) and for *Adam* (MONK-2, MONK-3), the ones that globally performed better. We have then proceeded with the strategy illustrated above, i.e. with a grid search as model selection and a final evaluation over 50 different weights initializations.

2.1.2 Other Models

For estimating the overall risk of other models we have exploited a nested K-Fold cross validation – $K = 5$ for both outer and inner cycle – by implementing the custom function `nested_cross_validation()`. Table 3 displays the grid used in the inner model selection for each tested model.

	n° neighbors	**weights**	**metric**
K-NN	$[2n]$, $1 \leq n \leq 39$	[uniform, distance]	[euclidean, cityblock, chebyshev]
	criterion	**max depth**	**min samples split**
DT	[gini, entropy]	[1, 2, 4, 6, 8, 10, None]	[2, 3, 4, 5, 6, 7, 8]
	solver	**C**	**max iter**
LR	[saga, lbfgs]	$[10^n]$, $-4 \leq n \leq 3$	[500, 1000]
	criterion	**max depth**	**n° estimators**
RF	[gini, entropy]	[1, 3, 5, 7, 10, None]	[50, 100, 150, 200]
	α	**binarize**	**fit prior**
BNB	$[10^n]$, $-4 \leq n \leq 0$	0	[True, False]
	criterion	**learning rate**	**n° estimators**
GB	[friedman_mse, squared_error]	[.001, .01, .1, 1]	[50, 100, 150, 200]
	C	**kernel**	γ
SVC	$[10^n]$, $-5 \leq n \leq 5$	[poly, rbf, sigmoid]	$[10^n]$, $-5 \leq n \leq 5$

Table 3: Nested cross-validation grids for MONK.

The mean accuracy values are therefore used to choose the best-performing models on each MONK and to optimize them in a separate model selection phase, consisting in

a grid search possibly followed by further finer-grained searches over the neighborhood values of the best previous model.

2.2 CUP

For model evaluation, 10% of the whole CUP dataset has been used as an internal test set. Model selection has been conducted on the remaining 90% by using a stratified K-Fold cross-validation with $K = 5$. Apart from the development of *Keras* and *Scikit-Learn* models (except for the loss), which has followed the same approach proposed for the MONK tasks, other models have been preliminary assessed with a nested cross validation, followed by a grid-search model selection.

	n° neighbors	**weights**	**metric**
K-NN	$[2n]$, $1 \leq n \leq 39$	[uniform, distance]	[euclidean, cityblock, chebyshev]
	fit_intercept	**normalize**	**n_jobs**
OLR	[True, False]	[True, False]	[-1]
	criterion	**max_depth**	**n° estimators**
RF	[gini, entropy]	[1, 3, 5, 7, 10, None]	[50, 100, 150, 200]
	α	**max_iter**	**solver**
Ri	[0.1, 1, 10, 100]	[None, 500, 1000]	[auto, svd, lsqr, saga]
	α	**max_iter**	**tol**
La	[0.001, 0.01, 0.1, 1]	[500, 1000, 1500]	$[5 \cdot 10^{-5}, 10^{-5}, 5 \cdot 10^{-4}]$

Table 4: Nested cross-validation grids for CUP.

3 Experimental Results

3.1 MONK

3.1.1 Neural Networks

Table 5 shows the best models resulting from the model selection procedure described for *Keras* NNs in section 2.2.1.

	MONK 1	**MONK 2**	**MONK 3**
Hidden units	5	3	4
η	0.2	0.8	0.01
α	-	0.5	-
λ	0	0	0.01
Batch size	1	2	8
Optimizer	Adam	SGD	RMSprop
Hidden Activation	tanh	sigmoid	ReLu
Weights Initializer	he_normal	he_normal	random_normal

Table 5: Best resulting models for MONK

Some brief considerations about each model selection:

1. For MONK-1, despite RMSprop has proved to perform better, the comparative analysis has not improved much its average validation accuracy. We have therefore considered Adam, which has reached good validation results by fine-tuning the value of its learning rate.

2. For MONK-2, the optimal value for the learning rate found in the first grid searches (0.6) decreases the slope of the learning curve as we select sigmoid (which reaches an extensive average accuracy) as activation function. To speed up the time of convergence, the value for η has therefore been increased to 0.8.

3. As MONK-3 has been designed with some noise, the model is more exposed to the risk of large variance and requires the introduction of an indirect regularization approach (L2).

Table 6 reports the average accuracy and loss on training and test data. Figures 1, 2 and 4 display the learning curves respectively for MONK-1, MONK-2 and MONK-3. In Table 6 we also consider the model selected for MONK-3 without regularizer, which correspond to the upper curves visible in Figure 4, where the validation curves reaches a plateau and does not converge.

Keras	**MONK 1**	**MONK 2**	**MONK 3**	**MONK 3 (reg)**
(TR) Accuracy	1.000 ± 0.003	1.00 ± 0.02	0.99 ± 0.01	0.954 ± 0.008
(TR) Loss	0.001 ± 0.003	0.01 ± 0.02	0.011 ± 0.007	0.057 ± 0.006
(TS) Accuracy	1.00 ± 0.01	1.00 ± 0.04	0.96 ± 0.01	0.972 ± 0.002
(TS) Loss	0.003 ± 0.010	0.01 ± 0.03	0.03 ± 0.01	0.052 ± 0.006

Table 6: MONK's Accuracy and Loss for training (TR) and test (TS) with *Keras*.

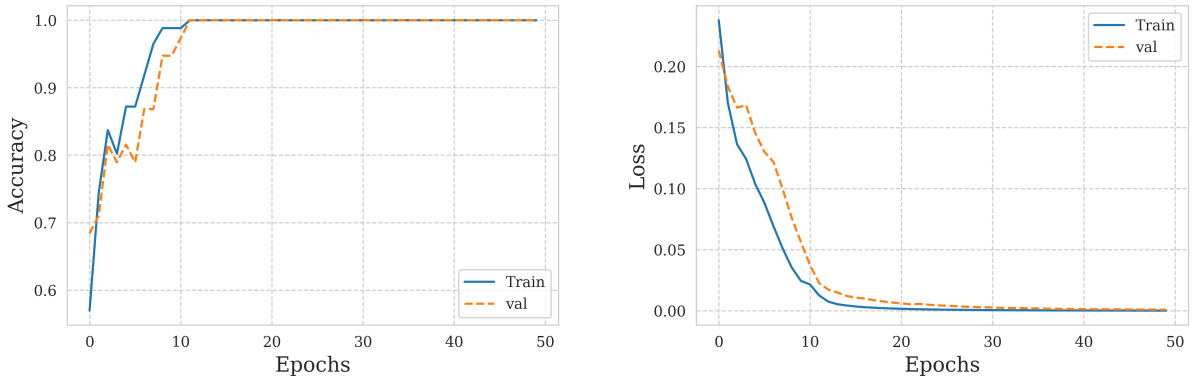

Figure 1: MONK1 training and validation curves.

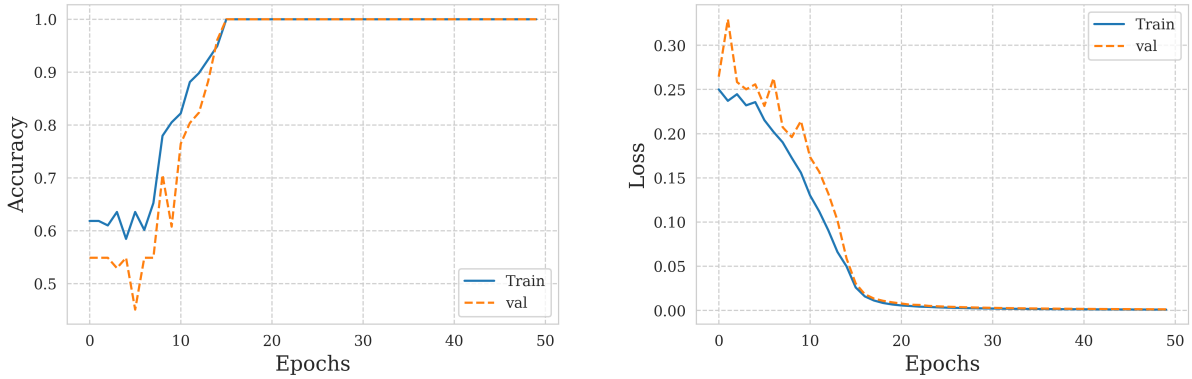

Figure 2: MONK2 training and validation curves (*Keras*)

As for *Scikit-Learn*'s MLP, the best resulting models are described in table 7, while values of accuracy over 50 iterations can be found in Table 8. For checking the respective learning curves, the reader can refer to Figures 6, 7 and 8 in the Appendix.

	MONK 1	**MONK 2**	**MONK 3**
Hidden units	6	6	2
Learning Strategy	Adaptive	-	-
η	1	0.01	0.01
α	0.2	0.1	0
Batch size	8	1	8
Optimizer	SGD	Adam	Adam
Hidden Activation	ReLu	ReLu	ReLu

Table 7: Best resulting models for MONK (*Scikit-Learn*)

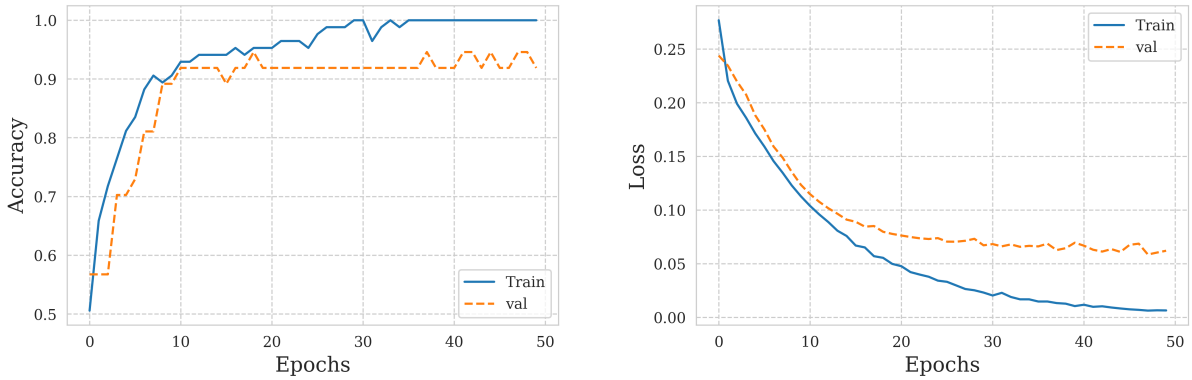

Figure 3: MONK3 training and validation curves without regularization (*Keras*)

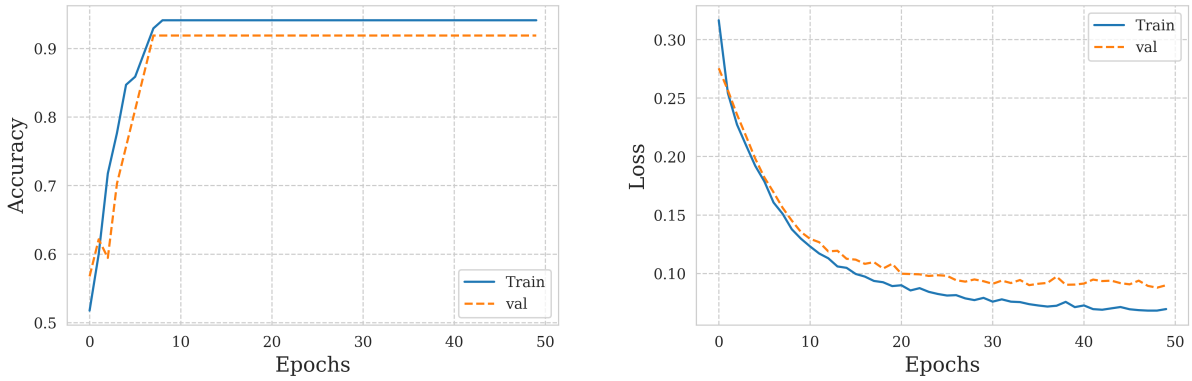

Figure 4: MONK2 training and validation curves with regularization (*Keras*)

Scikit-Learn	**MONK 1**	**MONK 2**	**MONK 3**
(TR) Accuracy	0.99 ± 0.02	0.999 ± 0.007	0.96 ± 0.01
(TR) Loss	0.2 ± 0.8	0.0 ± 0.2	1.3 ± 0.5
(TS) Accuracy	0.95 ± 0.05	0.99 ± 0.04	0.97 ± 0.01
(TS) Loss	1.8 ± 1.8	0 ± 1	1.1 ± 0.4

Table 8: MONK's Accuracy and Loss for training (TR) and test (TS) with *Scikit-Learn*.

3.1.2 Other Models

Table 13 displays the average accuracy computed on the test outer folds for each model and MONK dataset.

Model	**MONK 1**	**MONK 2**	**MONK 3**
SVM	0.97 ± 0.03	0.90 ± 0.08	0.91 ± 0.03
K-NN	0.81 ± 0.06	0.60 ± 0.09	0.89 ± 0.04
LR	0.72 ± 0.04	0.62 ± 0.10	0.92 ± 0.05
DT	0.83 ± 0.11	0.62 ± 0.09	0.89 ± 0.04
RF	0.91 ± 0.05	0.60 ± 0.09	0.93 ± 0.05
GB	1.0 ± 0.0	0.70 ± 0.09	0.93 ± 0.05
BNB	0.74 ± 0.04	0.56 ± 0.13	0.93 ± 0.05

Table 9: Values of accuracy from Nested Cross Validation

Tables 10, 11 and 12 report the best models found for each MONK and the relative accuracy on test set. Accuracy of RFs is averaged over 50 initializations with different random seeds.

Task	C	Kernel	γ	Class Weight	Accuracy
MONK 1	10	RBF	0.1	Balanced	1.0
MONK 2	9600	RBF	0.008	None	1.0
MONK 3	0.065	linear	-	Balanced	0.97

Table 10: Results for SVM classifier on MONK

Task	Criterion	Learning rate	n° estimators	Accuracy
MONK 1	friedman_mse	0.1	200	1.0
MONK 2	friedman_mse	0.5	250	0.78
MONK 3	friedman_mse	0.01	100	1.0

Table 11: Results for GB classifier on MONK

Task	Criterion	Max depth	n° estimators	Accuracy
MONK 1	gini	6	30	0.93 ± 0.03
MONK 2	gini	None	110	0.74 ± 0.01
MONK 3	gini	None	250	0.963 ± 0.001

Table 12: Results for RF classifier on MONK

3.2 CUP

For the CUP task, the model selection approach has been fairly similar to the one previously discussed for the MONK's task: starting from the hyper-parameter space described in Table 2, *Keras* NN and *Scikit-Learn* MLP have been selected with both grid searches and further analyses through the user-defined function `begin_comparison()`. Training and assessment have been conducted by emplying the Mean Euclidean Error (MEE), which is not implemented in the libraries considered and has therefore defined in the `mean_euclidean_error()` custom function. Other models have previously assessed with a nested cross validation and then validated through grid-searches. Table 13 shows the results of nested cross validation, whereas Table 14 provides the performance metrics (MEE, MAE, MSE, R^2) for each assessed model.

Model	**MEE**
K-NN	1.71 ± 0.09
OLR	26.6 ± 0.3
RF	4.1 ± 0.2
Ri	3.17 ± 0.03
La	6.1 ± 0.2

Table 13: Loss from Nested Cross Validation for CUP

Model	MEE (TR)	MEE (TS)	MAE (TR)	MAE (TS)	MSE (TR)	MSE (TS)	R^2
Keras NN	1.98 ± 0.06	2.30 ± 0.07	1.24 ± 0.04	1.44 ± 0.04	2.7 ± 0.1	3.6 ± 0.2	0.84
Sklearn NN	1.6 ± 0.1	1.8 ± 0.1	1.02 ± 0.09	1.14 ± 0.08	2.0 ± 0.2	2.5 ± 0.3	0.88
K-NN	0	1.53	0	0.95	0	1.78	0.91
Ridge	2.13	2.44	1.35	1.57	2.99	3.86	0.83
Random Forest	0.750 ± 0.005	1.65 ± 0.02	0.465 ± 0.003	1.03 ± 0.01	0.478 ± 0.007	2.03 ± 0.04	0.90

Table 14: Performance of selected models on CUP.

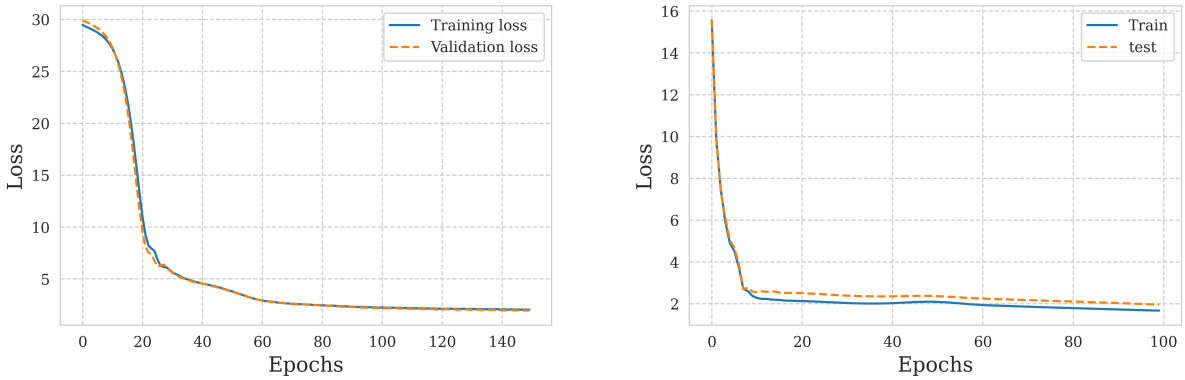

Figure 5: CUP training and validation curves with *Keras* (left) and *Scikit-Learn* (right).

3.3 Final model selection

The final model to be used for the blind test set has been selected not only by looking at the performances observed on Table 14, but also considering the possible advantages of the available solutions. Despite K-NN performing better on all the performance indicators, the performance of a lazy algorithm may be heavily affected by the presence of outliers and noisy data. For this reason we consider a eager algorithm like *Scikit-Learn*'s MLP a more robust solution for handling a blind task. Its hyper-parameters configuration and relative performance metrics are described in Table 15, whereas the correspondent learning curve can be found in Figure 5 (right).

Hid. Layers	Units/layer	η	α	Solver	Batch	MME (TR)	MME (VL)	MME (TS)
3	125	0.1	0.8	Adam	Auto	1.5	1.56	1.65

Table 15: Selected *Scikit-Learn* MLP and Loss functions

4 Conclusions

Between all the models we have analyzed, MLP, GB and SVM outperformed all the others. *Keras* has to be preferred to *Scikit-Learn* due to the difference in performance of the two NNs created, as well as its broader capabilities that allow to produce more flexible models. The results obtained on the blind test are in the file named "SnackOverflow_ML-CUP22-TS.csv".

Acknowledgments

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

A Appendix

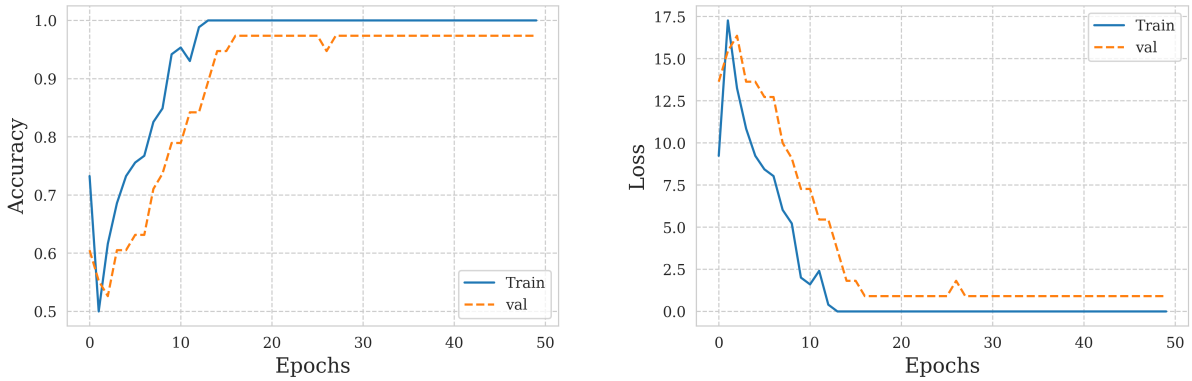

Figure 6: MONK1 training and validation curves (*Scikit-Learn*).

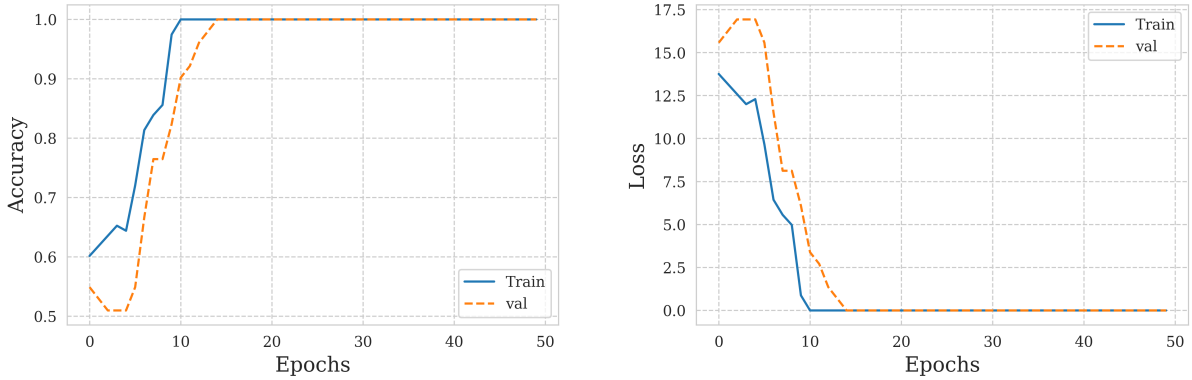

Figure 7: MONK2 training and validation curves (*Scikit-Learn*).

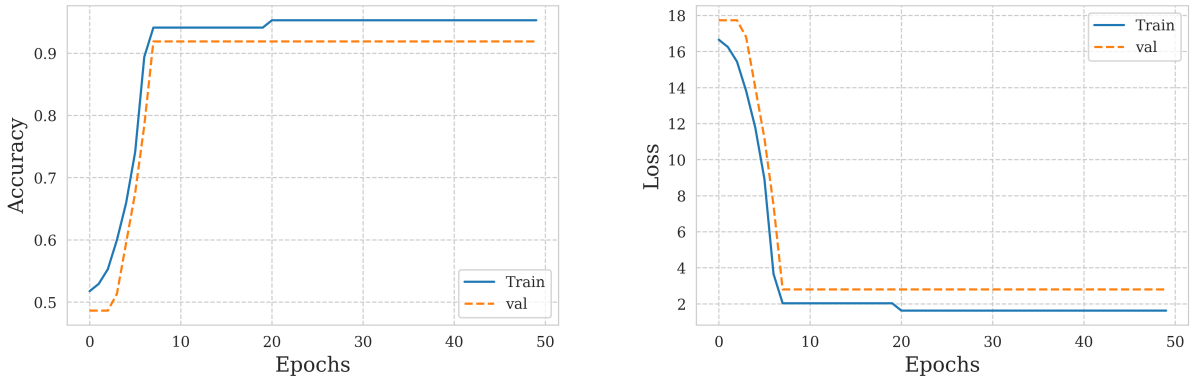

Figure 8: MONK3 training and validation curves (*Scikit-Learn*).