Machine Learning Project

Luca Santarella, Michele Morisco,Niccolò De Paolis
Computer Science (Artificial Intelligence) l.santarella@studenti.unipi.it
Computer Science (Artificial Intelligence) m.morisco@studenti.unipi.it
Computer Science (Artificial Intelligence) n.depaolis1@studenti.unipi.it
ML course (654AA), Academic Year: 2021/2022
Date: 24/01/2022
Type of project: **B**

Abstract

This report presents the implementation and comparison of three different neural network simulators implemented using three libraries (Keras, PyTorch and Scikit-Learn). The models created are tested for both classification and regression tasks. Various techniques are used such as K-fold cross validation for model selection and the ensemble of the chosen models in the final phase.

1 Introduction

This project is focused on the implementation of different neural network simulators built not from scratch but by using some libraries in the Python programming language environment, hence the focus is more on the experimental/comparison phase rather than on the implementation one.
The aim was to compare the different approaches used by various libraries to implement a neural network and the means by which we can perform hyperparameters tuning and model selection. The models were first tested on a classification task on the well known MONK datasets used as a benchmark. The second part of the project consists instead in a regression task on a dataset consisting of 10 dimensions inputs and bidimensional outputs.

2 Method

The programming language used is Python together with Pandas and Numpy for data manipulation and mathematical operations respectively, we used Jupyter Notebook as computing platform. Three different libraries were used: Keras, Pytorch and Scikit-Learn, we explored these frameworks which provide ways to build many different kinds of ANN with specific architectures. Since PyTorch does not provide any tools to perform

hyperparameter search we have used Skorch as a wrapper to use the scikit-learn grid search and other methods such as `.fit()` and `.predict()`.

We opted for a feed-forward fully connected architecture where the number of hidden layers, units and other hyperparameters have been chosen using the model selection tools provided by the libraries.

2.1 Preprocessing

We preprocessed the MONK dataset as it contains categorical attributes. We applied one-hot encoding to all three tasks, obtaining 17 numeric attributes from the original 6.

2.2 Hyperparameters optimization

The model selection consisted in two consecutive randomized search, the first one used to find good values. The frameworks Keras and PyTorch give the user the possibility to wrap the model in order to use Scikit-Learn's simple interface and thus the randomized search algorithm provided by the framework. We decided to do this for PyTorch, whereas for Keras we decided to use its native tool Tuner.
In this way we explored three different settings: Scikit-Lern's implementation with its native random search, PyTorch with a wrapper (Skorch wrapper) to use Scikit's learn random search tool and Keras with its native Tuner search tool which had been modified since it does not inherently give the possibility to perform k-fold cross validation.

2.3 Validation schema

For the validation phase we used the hold-out method to extract the test set (10%), whereas the training set and validation set were partitioned by means of a K-10 folds cross validation. The final model chosen is an ensemble of the 10 best models discovered.

2.4 Evaluation Phase

Once the ten best models have been chosen they are used to as a voting committee. The final ensemble model predicted the data on the internal test set by computing the average of the ten models as our final prediction.

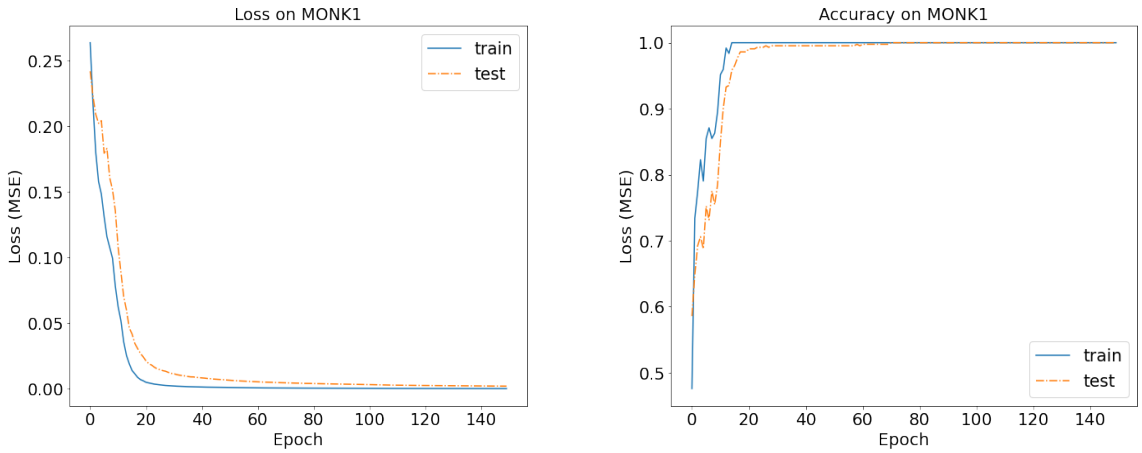
3 Experiments

3.1 Monk Results

The MONKs results are shown in table 1, the learning curves for the models using Keras in figure 1, in figure 2 and figure 3 (see more in the appendix A).

Task	Archit., eta, alpha, lambda	MSE (TR/TS)	Acc. (TR/TS)(%)
Keras			
MONK1	(17,8,1), 0.81, 0.8, 0	0.00022/0.002	100%/100%
MONK2	(17,8,1), 0.085, 0.85, 0	0.0012/0.0015	100%/100%
MONK3	(17,6,1), 0.27, 0.85	0.004/0.051	100%/92.6%
MONK3+reg.	(17,4,1), 0.01, 0.8, 0.0001	0.081/0.069	93%/97%
PyTorch			
MONK1	(17,4,1), 0.05, 0.45, 0	0/0	100%/100%
MONK2	(17,4,1), 0.1, 0.45, 0	0/0	100%/100%
MONK3	(17,4,1), 0.01, 0.5, 0	0.0492/0.0463	95%/95%
MONK3+reg.	(17,4,1), 0.01, 0.8, 0.0001	0.0410/0.0347	96%/97%
SkLearn			
MONK1	(17,10,1), 0.1, 0.9, 0	0/0	100%/100%
MONK2	(17,10,1), 0.1, 0.9, 0	0/0	100%/100%
MONK3	(17,10,1), 0.2, 0.9, 0	0.016/0.037	98%/96%
MONK3+reg.	(17,10,1), 0.1, 0.9, 0.1	0.0655/0.030	93%/97%

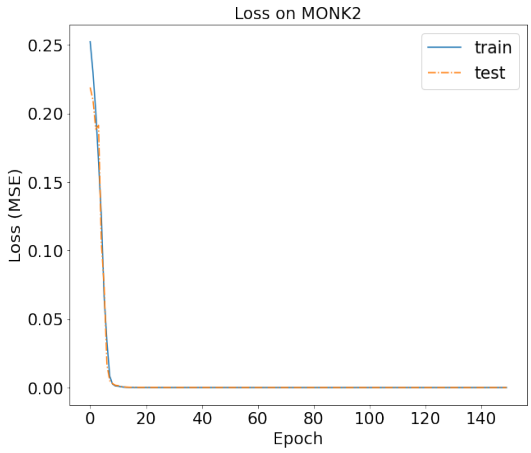
Table 1: Prediction results obtained for the MONK's tasks.



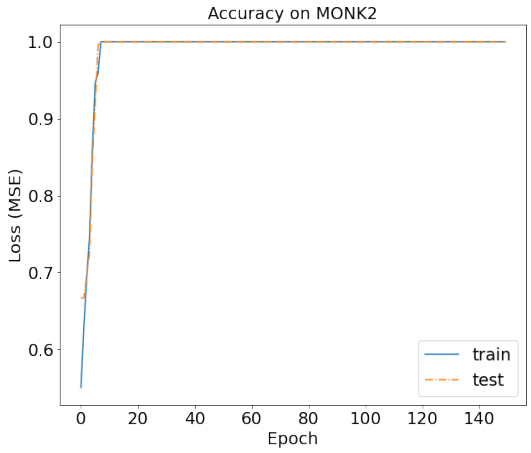
(a) Monk1 train and test loss

(b) Monk1 train and test accuracy

Figure 1: Learning curves for MONK1 using Keras

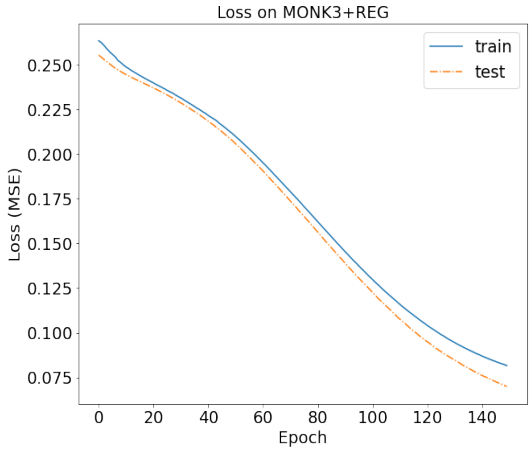


(a) Monk2 train and test loss for Keras

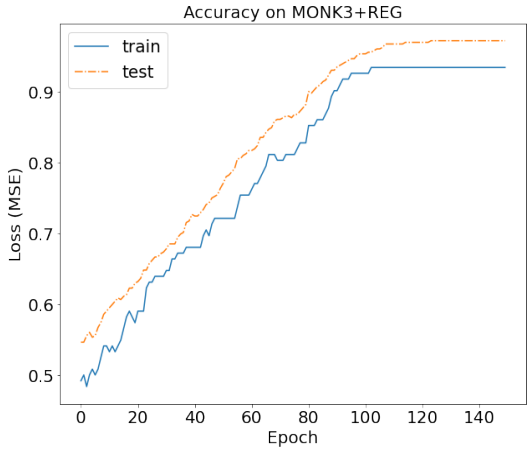


(b) Monk2 train and test accuracy for Keras

Figure 2: Learning curves for MONK2 using Keras



(a) Monk3 train and test loss with regularization for Keras



(b) Monk3 train and test accuracy with regularization for Keras

Figure 3: Learning curves for MONK3 using Keras with regularization

3.2 Cup Results

First we split the dataset into development set (90%) and internal test set (10%) through a simple hold-out, then the development set was used for the model selection using a k-10 cross validation which resulted in the division between training set and validation set. We believe that a number of ten folds is a good trade off which allowed us to waste as less data as possible avoiding a computational heavy grid search, in fact the model selection phase is done in a reasonable short period of time.

For the model selection we first run an initial randomized search which generated 1000 different random models, each one of the models was bench-marked using the Mean Euclidean Error (MEE) metric through a 10-folds cross validation.

Then we selected only the best models (according to the mean score on the 10 folds) and used the interval of the values of these models to perform a second more refined randomized search which generated another 1000 models. We decided to use 1000 iterations for the model selection in combination with the randomized approach to save on time spent on the model selection.
Table 2 shows the numerical intervals used for the two randomized for every library used.

# of search	**# layers**	**# units**	**eta**	**batch**	**lambda**	**alpha**
Keras						
RAND 1	[1,2,3]	[10,..,100]	[0.0001,.., 0.1]	[10,..,1000]	[0.0001,..0.1]	[0,..0.9]
RAND 2	[1,2]	[10,..,100]	[0.0069,..,0.063]	[150,..,890]	[0.00012,..,0.1]	[0.25,..,0.9]
PyTorch						
RAND 1	[1,2]	[10,..,100]	[0.001,..,0.5]	[1,..,1329]	[0.0001,..,0.2]	[0.8,..,0.9]
RAND 2	[1,2]	[10,..,100]	[0.003,..,0.246]	[108, 1297]	[0.0016,..,0.177]	[0.8,..,0.9]
SkLearn						
RAND 1	[1,2]	[10,..,100]	[0.001,..,0.5]	[1,..,1329]	[0.0001,..,0.2]	[0.8,..,0.9]
RAND 2	[1,2]	[10,..,100]	[0.009,..,0.266]	[3,..,1307]	[0.0014,..,0.099]	[0.8,..,0.9]

Table 2: Numerical intervals for the randomized searches

In our model selection we also used the following activation functions: sigmoid, tanh and relu with the standard SGD (Stochastic Gradient Descent) algorithm with minibatch and we also used the early stopping regularization. Regarding the early stopping, for SkLearn we also provided the percentage (20%) of data that would be used to determine when to halt the training phase, whereas for Keras and PyTorch we only provided the number of epochs (patience) and the required value of improving (threshold) on the validation loss.

The two randomized search took one hour each on the SkLearn framework and one hour and fifty minutes each on PyTorch, these searches were conducted on a personal laptop with a Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz. On Keras it took five hours and fifty minutes to complete each randomized search using a laptop with a Intel(R) Core(TM) i7-10750h CPU @2.60GHz.
Finally we decided to choose as final model an ensemble of the top 10 models (see Table 3 for the hyperparameters and scores and see Figure 4 for the learning curves) that were selected, thus the prediction phase of the ensemble consists in an average of the predictions of the committee.

# model	act. fun.	archit.	eta	batch	lambda	alpha	Train score	Val score
1	tanh	(75,75)	0.0149	123	0.054	0.85	0.982	1.1340
2	logistic	(25)	0.069	67	0.071	0.85	1.0972	1.1617
3	logistic	(10,10)	0.081	88	0.081	0.8	1.1169	1.1621
4	logistic	(10,10)	0.082	386	0.059	0.9	1.1034	1.1635
5	logistic	(75)	0.065	63	0.061	0.85	1.1017	1.1653
6	tanh	(75,75)	0.016	547	0.092	0.85	1.069	1.1654
7	logistic	(10,10)	0.111	216	0.092	0.8	1.1289	1.1655
8	logistic	(10,10)	0.119	176	0.0191	0.85	1.0967	1.1673
9	logistic	(50,50)	0.0463	60	0.054	0.9	1.0739	1.1685
10	logistic	(25,25)	0.056	300	0.032	0.9	1.1427	1.1704

Table 3: Hyperparameters and scores of the committee of the ten models

The assessment phase was done using the MEE metric on the internal test set (10% of the total dataset) which is data that the model has never seen before.

As shown in table 4 we compared the results of the three libraries on the internal test

model	MEE (internal ts)	MEE (validation set)	MEE (train set)
Committee (SkLearn)	**1.092**	**1.1623**	**1.097**
Committee (PyTorch)	1.108	1.2306	1.077
Committee (Keras)	1.27	1.2971	1.22

Table 4: Final results of the committee of the ten models on different libraries

set and we found out that the library SkLearn has an average of **1.092** MEE value (mean over ten different runs) which was the best performance, PyTorch and Keras instead had a MEE value of **1.108** and **1.27** respectively. The fact the our best performance was with the Scikit-Learn framework and also its simplicity made us choose as our final model the average voting committee of the ten best models which were found with SkLearn. The reason behind our choice of the ensemble technique is that it allows us to further control the possible overfitting and obtain more accurate results.

4 Conclusion

We experimented with randomized searches to keep the model selection light and we believe to have found a good result by using the ensemble approach. Furthermore, Keras and PyTorch turned out to be more flexible and powerful libraries with respect to Scikit-Learn but at the same time we appreciated the simpler approach of the Scikit-Learn which was more suited for a simple regression task as this one.

BLIND TEST RESULTS: MLEnjoyer_ML-CUP21-TS.csv

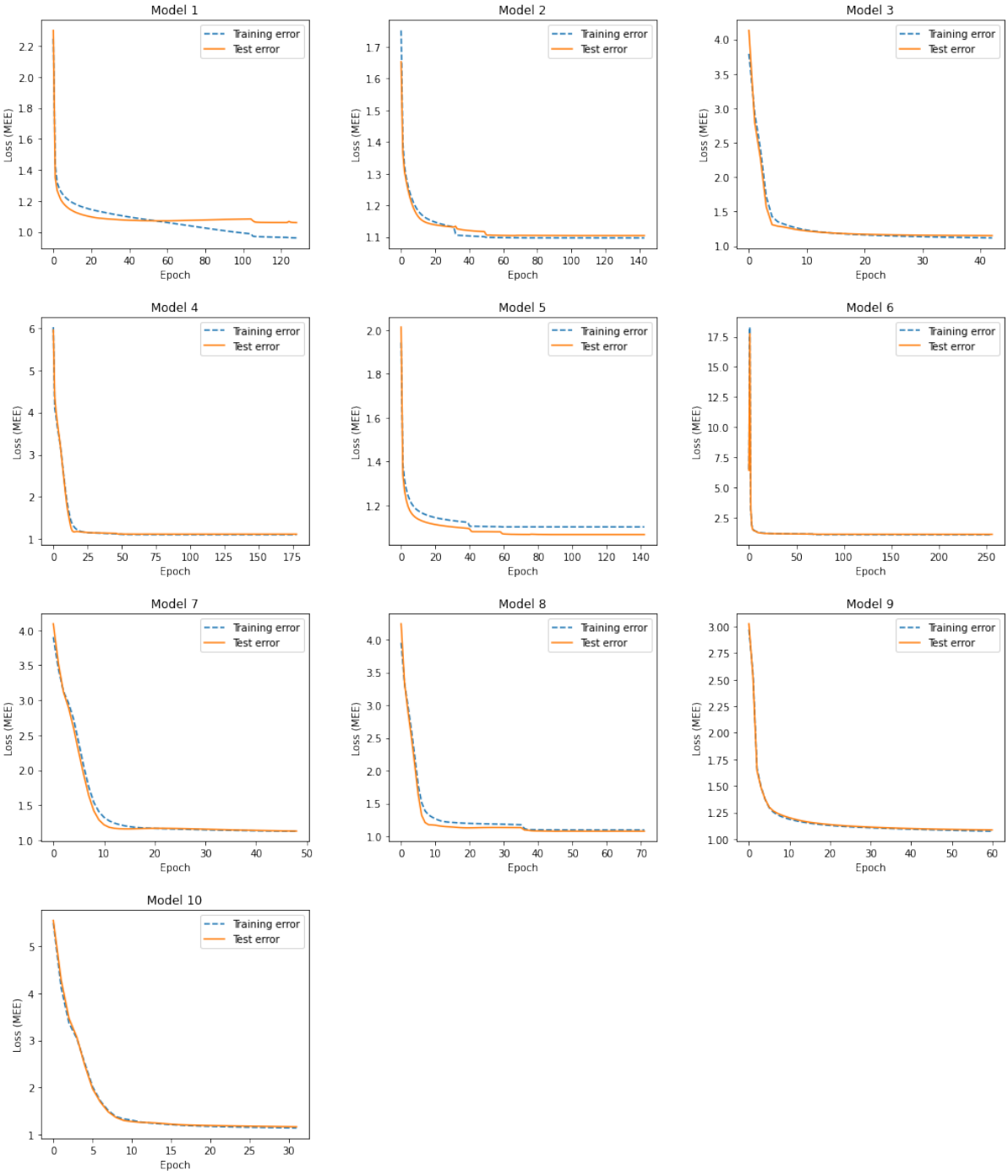
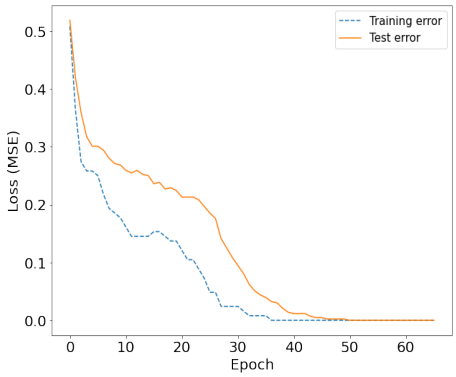


Figure 4: Learning curves of the ten best models (SkLearn)

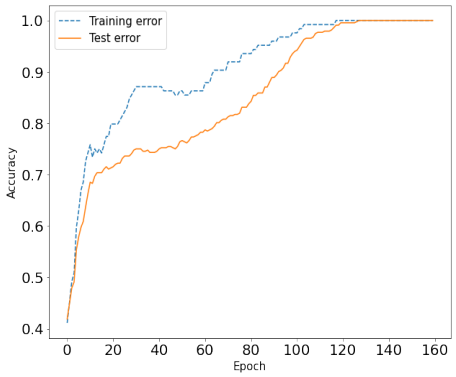
Acknowledgments

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

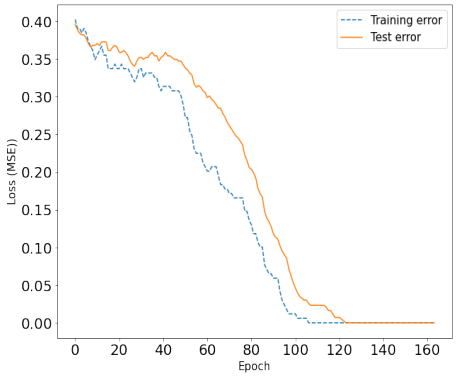
Appendix A



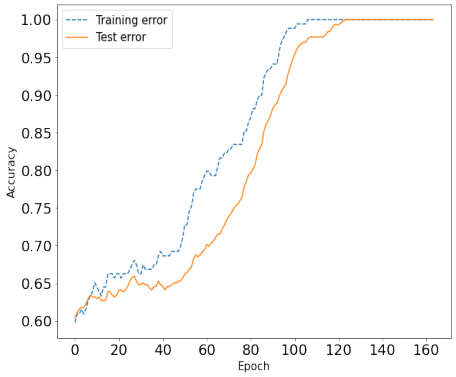
(a) Monk1 train and test loss for SkLearn



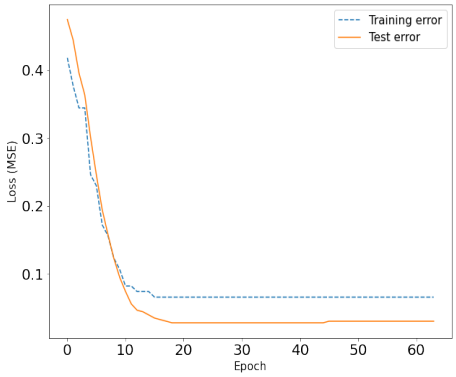
(b) Monk1 train and test accuracy for SkLearn



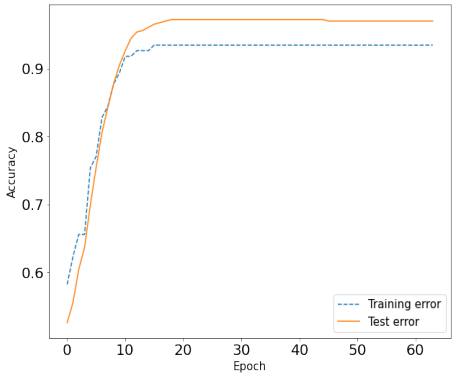
(a) Monk2 train and test loss for SkLearn



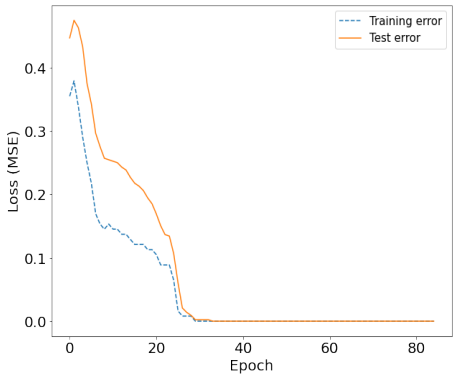
(b) Monk2 train and test accuracy for SkLearn



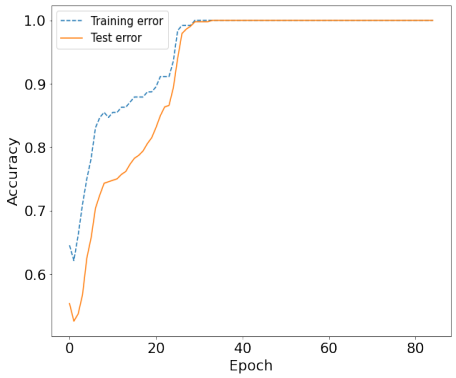
(a) Monk3 train and test loss for SkLearn



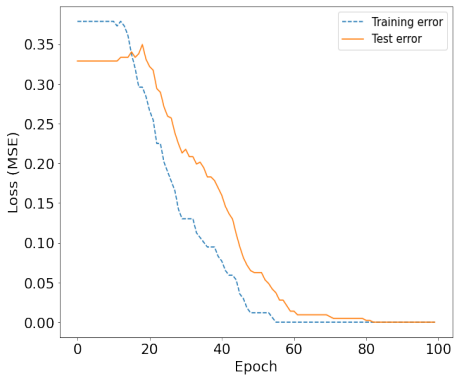
(b) Monk3 train and test accuracy for SkLearn



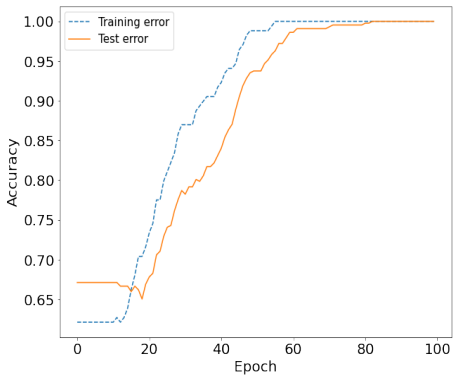
(a) Monk1 train and test loss for PyTorch



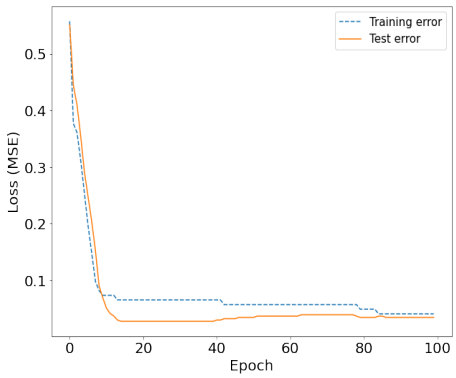
(b) Monk1 train and test accuracy for PyTorch



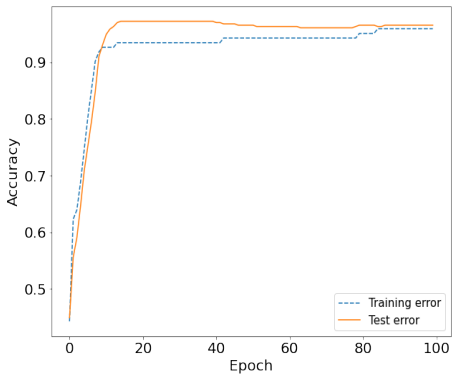
(a) Monk2 train and test loss for PyTorch



(b) Monk2 train and test accuracy for PyTorch



(a) Monk3 train and test loss for PyTorch



(b) Monk3 train and test accuracy for PyTorch

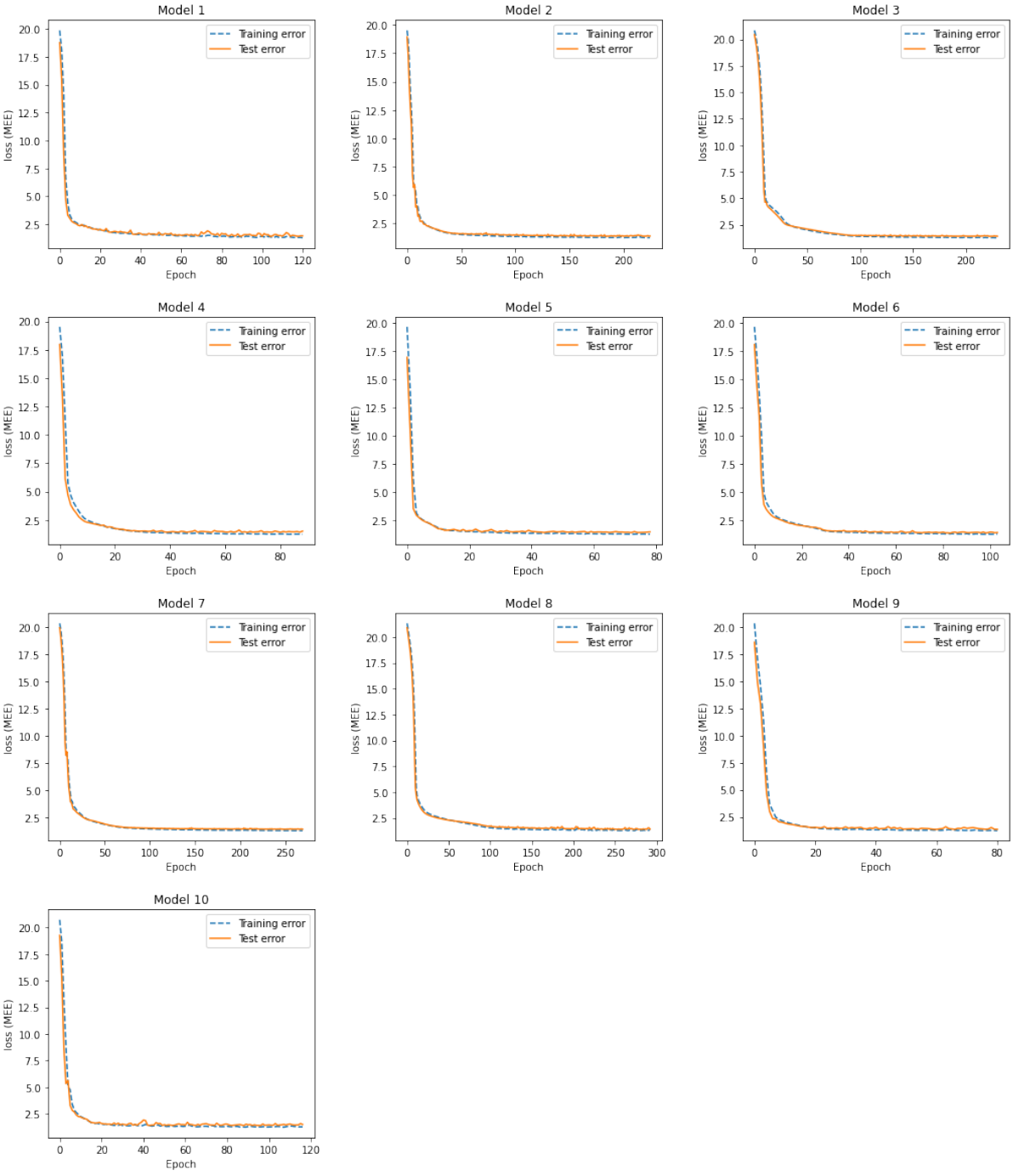


Figure 11: Learning curves on CUP (Keras)

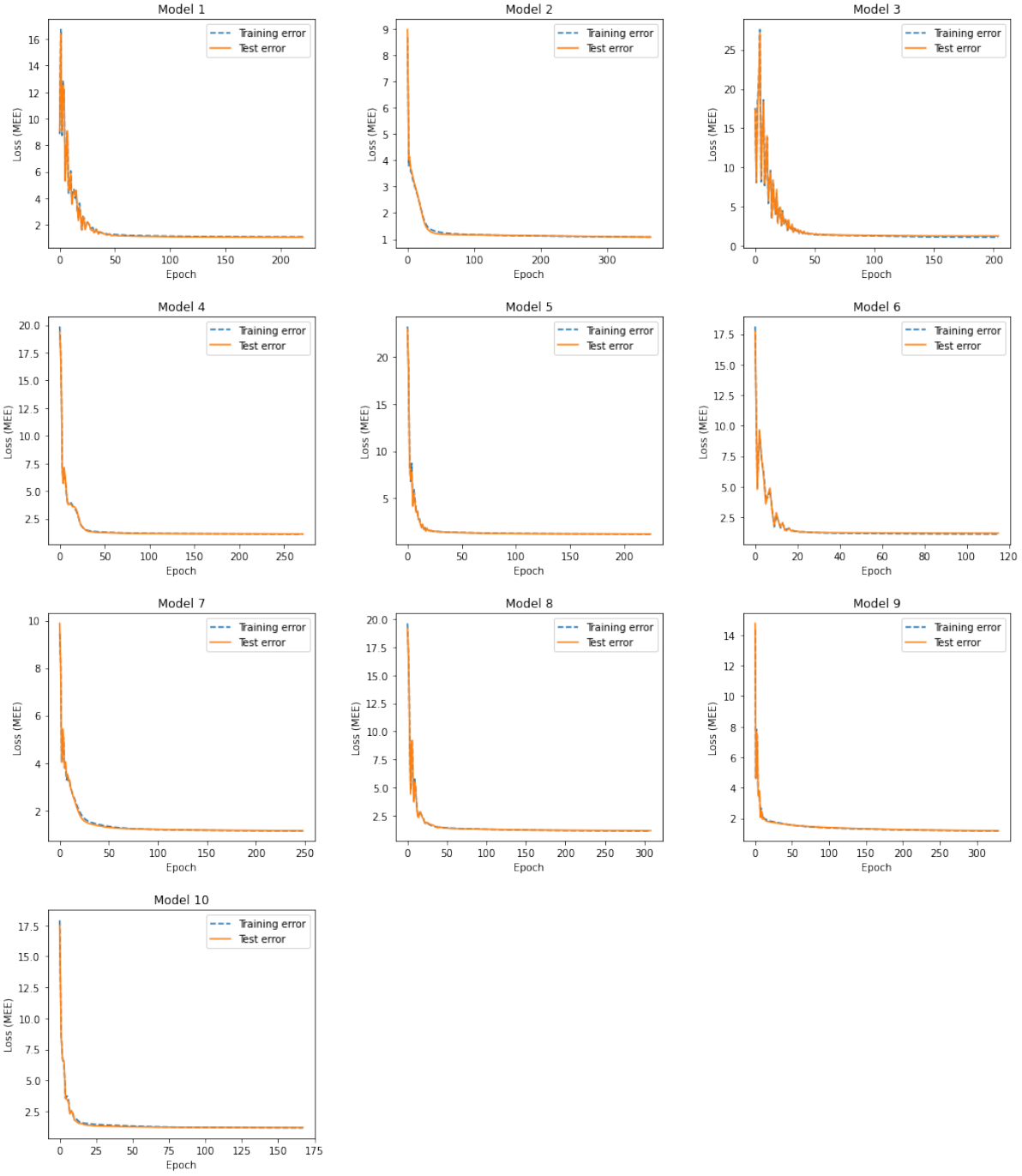


Figure 12: Learning curves on CUP (PyTorch)