ML 2023 Project Report

Authors:
Marco Antonio Corallo - Computer science student, Curriculum *SW*
m.corallo2@studenti.unipi.it

Hamza Karoui - Computer science student, Curriculum *ICT*
h.karoui@studenti.unipi.it

Samuele Vezzuto - Computer science student, *Free Choice* Curriculum
s.vezzuto@studenti.unipi.it

Team Name: #3 *Monk(eys)*
Date: 31/01/2024
Type of project: B

Objectives

- Comparison and evaluation of ML models implemented by different Python frameworks
  - Evaluate the usability of these frameworks
  - Evaluate the accuracy and the performances of these models
- Identify the best-performing model to compete in an internal competition among students
  - We evaluated each model performance on the provided development set to choose the best one

Method - Libraries

The entire project was developed onto the Python ecosystem, which gave us high-level benefits. The main core libraries we used are:

- **Pandas** for efficient dataset manipulation;
- **NumPy** for numerical operations and statistical insights;
- **Matplotlib** for creating plots.

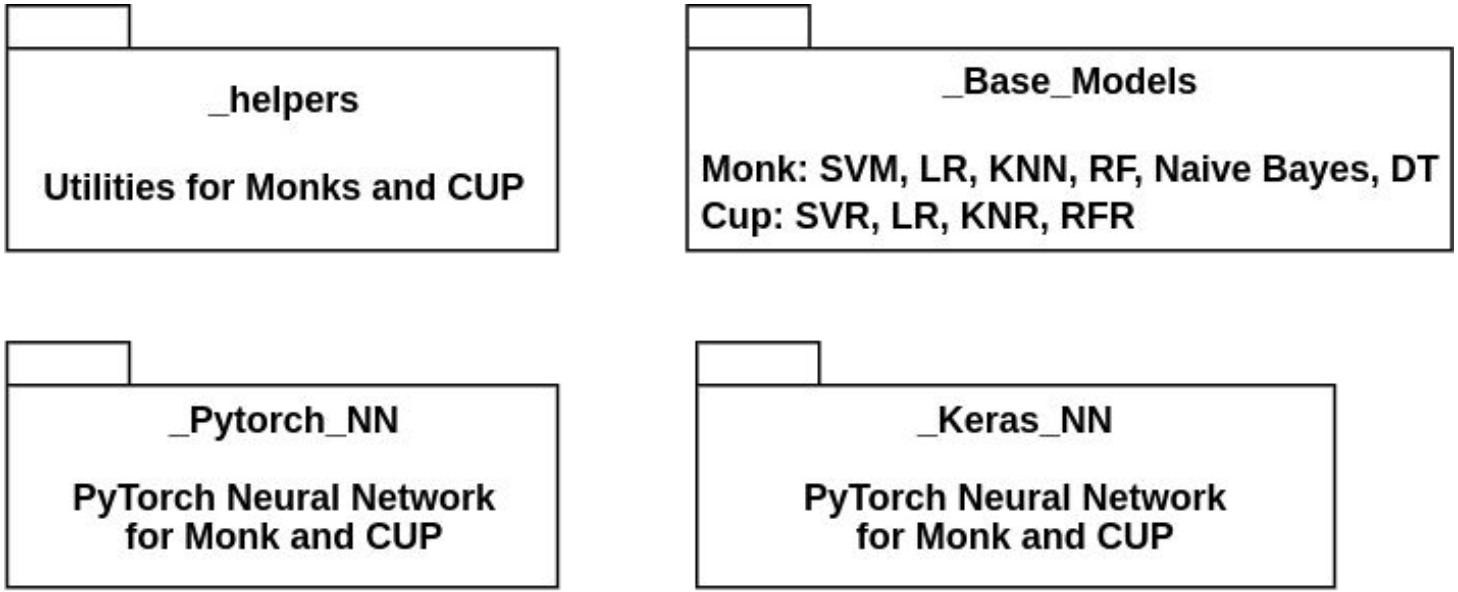
We also used different ML frameworks:

- **PyTorch** for a first implementation of a Neural Network;
- **Keras** on TensorFlow for another implementation of a Neural Network.
- **SciKit-Learn** for the implementation of the other machine learning models and for model selection utilities

Additionally, we used **SciKeras** that provides Keras wrappers for using skl grid search.

Method - Code structure

The structure of the project consist of eight files, divided in half for CUP and Monks.



Method - Software overview

- Base Models make use of Skl's gridsearch, parameters of the search are stored into those files.
- A custom implementation of the gridsearch has been written for PyTorch, due to the incompatibility with Skl and the lack of *bridges*.
- Although SciKeras provides a wrapper for using Skl's gridsearch,
  It presents some problems. In particular, It's not possible to plot *validation* learning curves while using *K-Fold Cross-Validation*, neither to do *early stopping* on validation loss.
  A custom extension has been developed for solving these issues.

Method - Architecture

In both PyTorch and Keras has been used the **SGD** training algorithm, combined with **batch** and **mini-batch** techniques. For the Monk task we employed a single hidden layer neural network for both PyTorch and Keras frameworks. The activation function used in the hidden layer was tanh, while the output layer utilized a sigmoid activation function.

For CUP results, we tested different architectures and the best one resulted on Keras with **three** hidden layers, respectively of size **128**, **256** and **64**, both three using **tanh** as activation function. The NN uses a **mini-batch** of size **32**, **Nesterov** momentum of **0.9**, **L1** regularization and linear **learning rate decay**.

The method of weights initialization has been selected into the model selection phase. PyTorch initializes weight taking the values with uniform probability from the range [-0.7, 0.7], while Keras uses the *Glorot* uniform distribution. *Early stopping* has been adopted with a patience ranging from 15-30 epochs and a tolerance of 0.0001 on the **validation loss**.

Neural Networks Architectures

Monks Task

- **1 Hidden layer** with few units
- ***Tanh*** as activation function
- **SGD** Algorithm
- **Mini-batch** with little batch size
- Weights initialization uniformly from range [-0.7, 0.7] / **Glorot**
- **no reg**. for Monk1, Monk2, **L1/L2** for Monk3
- **Early stopping**: patience 10-15 epochs, delta 1e-4

CUP Task

- **3** or more **hidden layers** with up to **256** units
- ***Tanh/ReLU*** as activation functions
- **SGD** Algorithm
- **Batch** or **mini-batch** with bigger batch size
- Weights initialization uniformly from range [-0.7, 0.7] / **Glorot**
- **L1/L2**
- **Early stopping**: patience 20-30 epochs, delta 1e-5

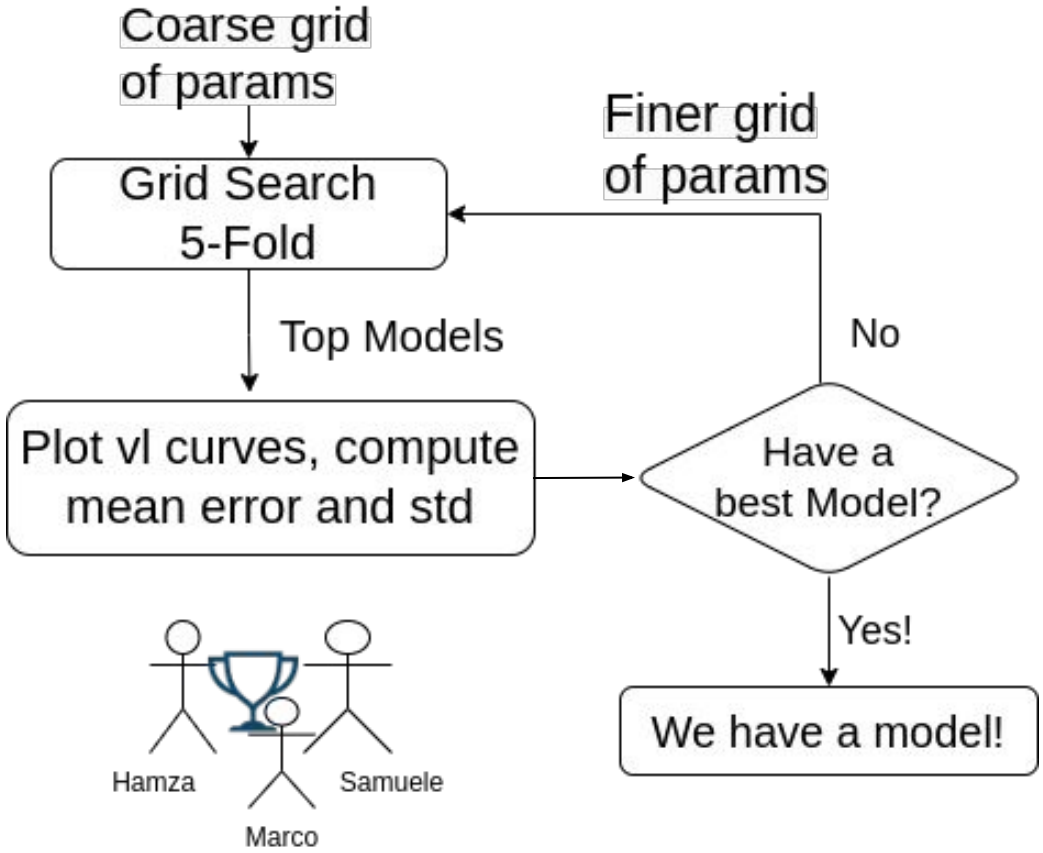
Method - Preprocessing procedure

We employed *1-of-k encodin*g for categorical attributes as part of the preprocessing step for the Monk datasets.
For the CUP data, no preprocessing has been employed.

For both the Monk and the CUP problems, we used 5-Fold CV, applying *stratification* for the Monk case, shuffling examples and imposing a random seed for replicability purposes.

In the CUP, the cross validation has been done on the 80% of the training set. Indeed, a preliminary hold-out has been done, reserving 20% of the dataset as internal test set.

Method - Validation schema



Method - Preliminary trials

With regard to the CUP, we initially tried to apply the scaling of the training set by means of the Standard Scaler of **SkI**, but we observed that there were no relevant performance improvements.

We also tried different other supporting libraries, like *Keras Tuner* and hand-written custom code that has been then replaced by more efficient routines.

Monk models: Base Classifiers - 1

- **Logistic regression**, linear model, uses as hyperparameters the solver, different regularizations (no, lasso, ridge, elastic-net), and C regularization parameter
- **SVM** includes as hyperparameters the C regularization parameter, the kernel, the degree in case of polynomial kernel, the gamma kernel coefficient in case of rbf and sigmoid kernel;
- **Decision Tree** includes as hyperparameters on the split criterions: minimum samples for node splitting, minimum samples at leaf nodes, required input fraction at a leaf node, and maximum tree depth.
- **K-nearest neighbors** hyperparameters include the quantity of neighbors, the prediction weight function, the distance computation metric, the neighborhood computation algorithm, and the power parameter (p) for the Minkowski metric.

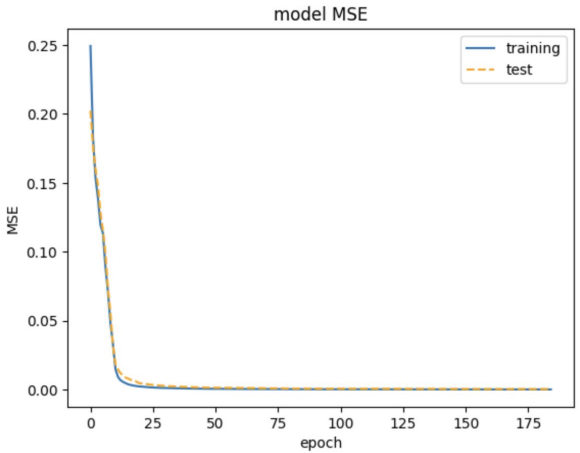
Monk models: Base Classifiers - 2

- **Random Forest** same hyperparameters as the Decision Tree, with the adding of the bootstrapping technique, with tunable tree number.
- **Gaussian** Naive Bayes is a classification technique based on the Gaussian distribution and a probabilistic approach. The only hyperparameter, var_smoothing, is added to feature variances for numerical stability.
- **Bernoulli** naive bayes, hyperparameters include alpha, a smoothing parameter to prevent probabilities for unseen features, and force_alpha.
- **Multinomial** naive bayes, is another type of naive bayes classifier, there are no hyperparameters used.

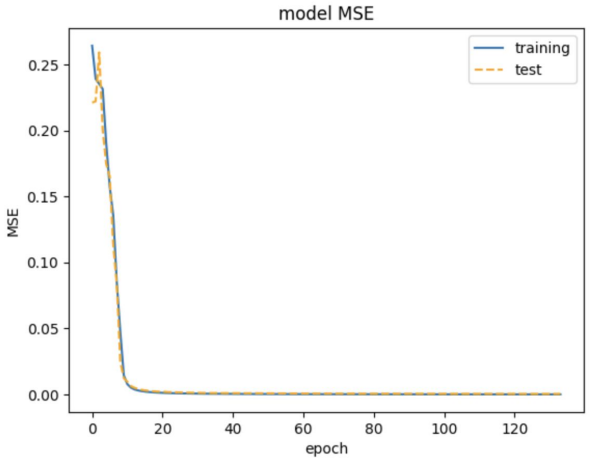
Best Monk Results - Keras

Data Set	#units, eta, lambda	momentum	Batch	MSE(TR/TS)	%Accuracy (TR/TS)
Monk1	4, 0.25, -	0.8	4	1.1e-4 ± (7e-6) / 1e-3 ± (1.5e-3)	100/100
Monk2	4, 0.25, -	0.8	4	1e-4 ± (4e-6) / 2.1e-3 ± (1e-4)	100/100
Monk3 [Reg]	4, 0.02, 0.002	0.69	4	9.8e-2 ± (2.4e-3) / 8.5e-2 ± (2.8e-3)	93.4/97.2

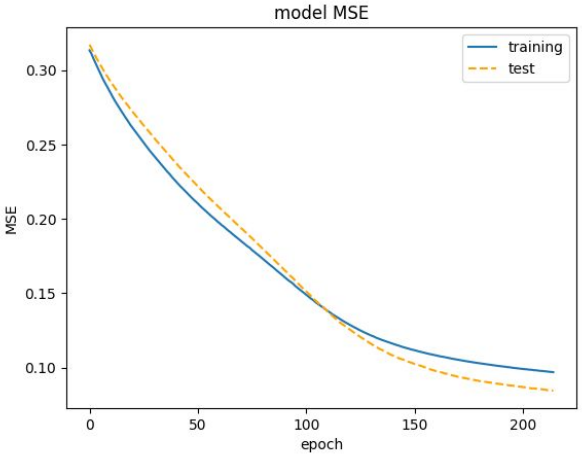
Best Monk Results - Keras MSE



Monk 1

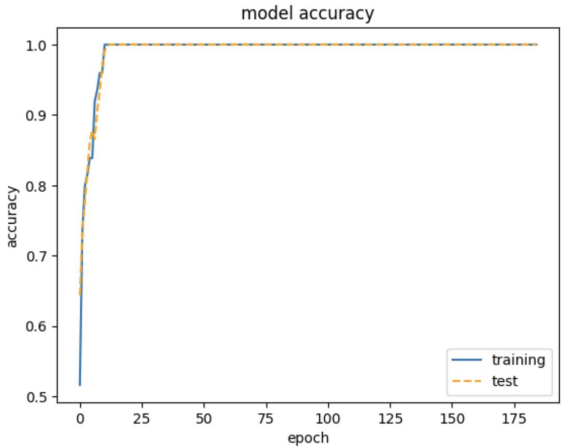


Monk 2

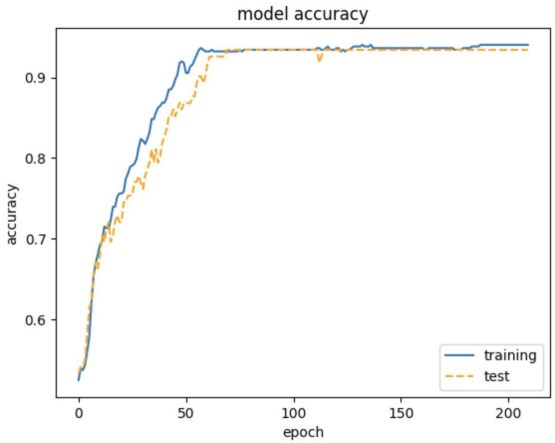


Monk 3

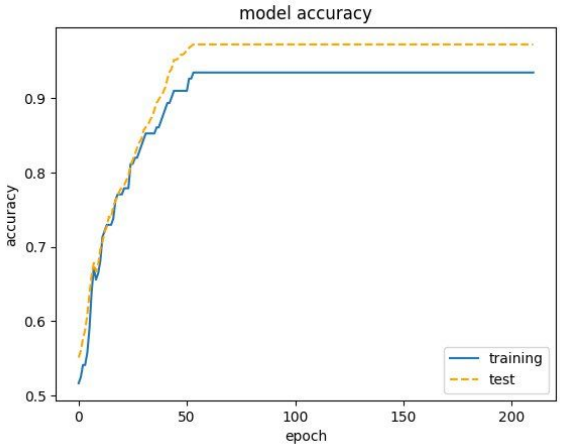
Best Monk Results - Keras Accuracy



Monk 1



Monk 2



Monk 3

Discussion (Monk)

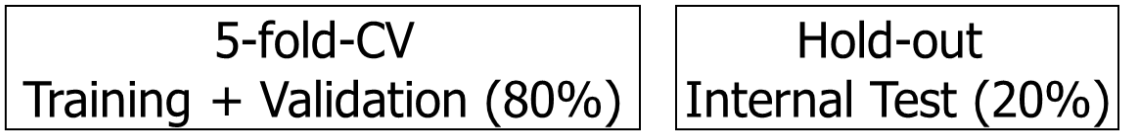
The Monk's experiments have been used to check our understanding of the theory behind machine learning. We started with an easy task to see how to adjust hyperparameters and how to get a good (and smooth) validation curve.

We explored how to tune hyperparameters and which combination of hyperparameters should be tuned together.
These tasks allowed us to make in practice what we studied with simplicity and, furthermore, working on the Monk’s we can gradually learn how to correctly use a ML framework.

We also experimented the benefits of regularization, that became necessary with the Monk3 dataset.

CUP Validation schema: Data Splitting

To validate our data, we employed a schema that involved using hold-out validation due to the unavailability of a dedicated test set.



Then, the Cross-Validation has been done on the development set using 5-Fold technique.

At the end of the entire model selection phase, the best-performing model was refitted and compared with the internal test set to assess the model.

CUP Validation schema: model selection - 1 or 2 layers configurations

In our quest for the optimal configuration, we systematically explored all possible combinations involving 1 and 2 layers, utilizing an increasing number of units: 2, 4, 8, 16, 32, 64, 128, 256.

These configurations exhibited inferior performance when compared to combinations involving more than 3 layers.

Consequently, we made the decision to conduct a grid search, focusing on configurations with 3 or more layers.

CUP Validation schema: model selection - 3+ layers

	#layers	**#units**	**Batch Size**	**Momentum**	**eta, lambda**
Pytorch	3, 4, 5	combinations of 2, 4, 8, 16, 32, 64, 128	increasing power of 2 up to 512, No mini batch	0.01, 0.1, 0.5, 0.7, 0.8	eta: [0.0005, 0.001, 0.005, 0.01, 0.1, 0.5, 0.7, 0.8], lambda : [-, 0.001, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5]
Keras	3, 4, 5	combinations of 2, 8, 64, 128, 256, 512	8, 25, 32, 64, 128, 500	0.5, 0.6, 0.7, 0.8, 0.85, 0.9	eta: [0.01, 0.001, 0.005]. lambda: [0.01, 0.001, 0.0001, 0.0005]

Cup models: other models

- We utilized the **MultiOutputRegressor** class from scikit-learn to encapsulate **SVR**, to support multi-output regression problems. The hyperparameters used for tuning, are the same as the classification version, with the inclusion of an extra parameter, epsilon, representing the epsilon tube.
- **LinearRegression,** as linear model from scikit-learn, no relevant hyperparameters.
- For **Lasso class** we explored the feature selection parameter, the alpha regularization parameter, the positive parameter to force coefficients to be positive.
- For **Ridge class**, same hyperparameters as those examined with Lasso, expanding our exploration to include various solvers as additional hyperparameters.
- For **Random Forest, KNR regressors**, same hyperparameters as the MONK.

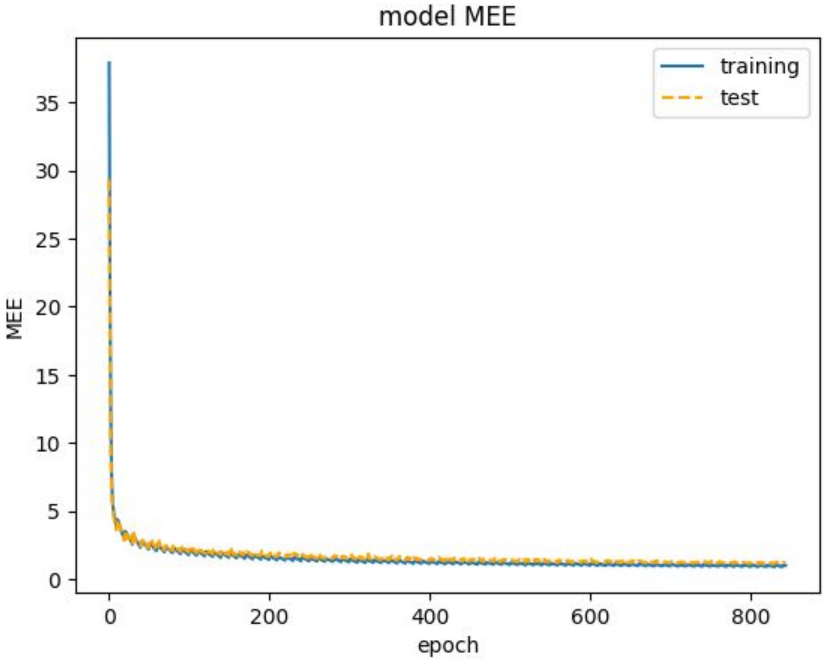
CUP Results

In the next tables we highlighted the most promising results we found conducting a systematic approach to the model selection and a thorough assessment.
The final model has been chosen based on the evaluation of the models performance using the MEE metric, evaluating mean error and standard deviation and analyzing the learning curve.
It's worth noting that during experiments, we tested various combinations, including different architectures and activation functions for the hidden layers. It’s interesting that the best combination of activation functions completely differs from PyTorch to Keras, going from a chain of ReLU to a sequence of tanh.
The selection of the final model was guided by the MEE minimization and the analysis of the learning curve. This process ensured that the model selected and tested on the internal test set is able to learn and generalize.

CUP best results

	#layers	#units	Batch size	Momentum	eta, lambda	MEE (TR/VL/TS)
Keras	3	128, 256,64	32	0.9	eta: 0.01 with decay to 0.0005 lambda: 0.0005	9.3e-1(±1.2390e-2)/ 5.9e-1(±1e-2)/ 1.1e0(±7.6e-3)

CUP Best Results Keras MEE



MEE

CUP refit time

Model	**Refit time**
SVR	0.2239 sec
Linear regression	0.0037 sec
Linear regression (L1)	0.0023 sec
Linear regression (L2)	0.0013 sec
KNR	0.0006 sec
RFR	0.2628 sec
PytorchNN	7.68 sec (tot)
KerasNN	208.68 sec (tot)

Discussion (CUP)

The CUP task posed an initial challenge in front of us in understanding the importance of using an internal test set through the hold-out technique and the relevance to choose a right splitting strategy.

Working on the CUP we can transpose what we learned during the Monks, into a Regression task.

Despite an early stopping mechanism was applied to both the Monk and the CUP tasks, we noticed a notable benefit in the latter. Indeed, early stopping allows to efficiently handle the demanding grid searches, limiting the training number of epochs and optimizing our computational resources, ensuring still effective model training.

Conclusions

Throughout this project, we solidified our understanding of machine learning, reaffirming the theoretical concepts learned in lectures.

It allowed us to see in practice ML concepts and techniques we already studied and, furthermore, we learned different ML tools and frameworks and how to compare them.

Unfortunately, although we followed the theory and a solid cross-validation approach, the MEE value on the internal test set is higher than the mean validation error, even if with a low std, that means stability.

Blind Test Results: **3Monkeys_ML-CUP23-TS.csv**

Acknowledgement

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

Bibliography

[1] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Advances in Neural Information Processing Systems 32 [Internet]. Curran Associates, Inc.; 2019. p. 8024–35. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

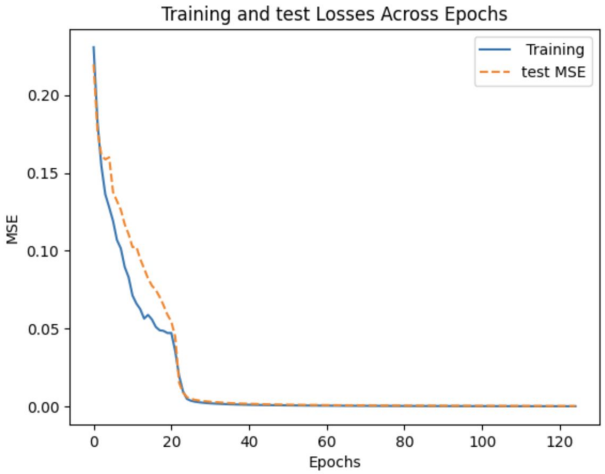
[2] Chollet F, others. Keras [Internet]. GitHub; 2015. Available from: https://github.com/fchollet/keras

[3] Pedregosa F, Varoquaux, Ga"el, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. Journal of machine learning research. 2011;12(Oct):2825–30

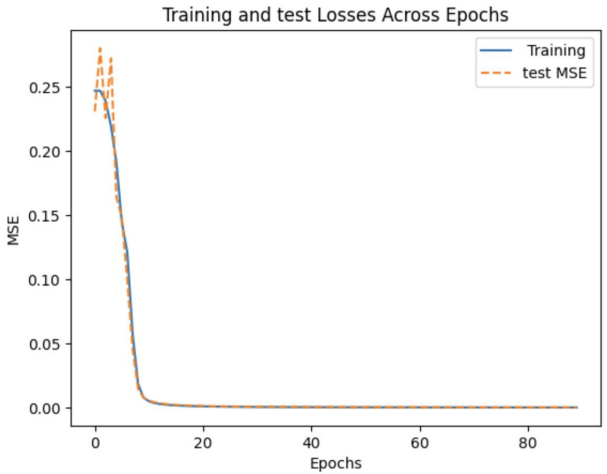
Appendix - Monk Results Pytorch

DataSet	#Units, eta, lambda	Momentum	Batch Size	MSE(TR/TS)	Accuracy (TR/TS) (%)
Monk1	4, 0.8, -	0.5	4	1e-4(±1e-4)/ 1e-3(±1e-3)	100/100
Monk2	4, 0.8, -	0.5	4	1e-3(±2e-3)/ 7e-3(±1e-2)	100/100
Monk3 [REG]	4, 0.1, 0.01	0.4	32	6e-2(±3e-4)/ 5e-2(±1e-3)	93/97

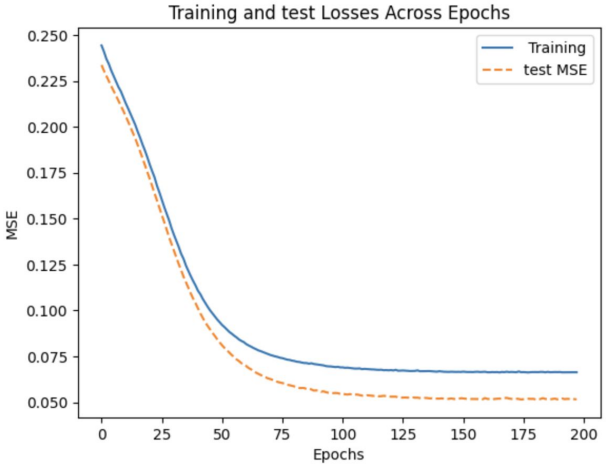
Appendix - Monk Results pytorch mse on test set



Monk 1

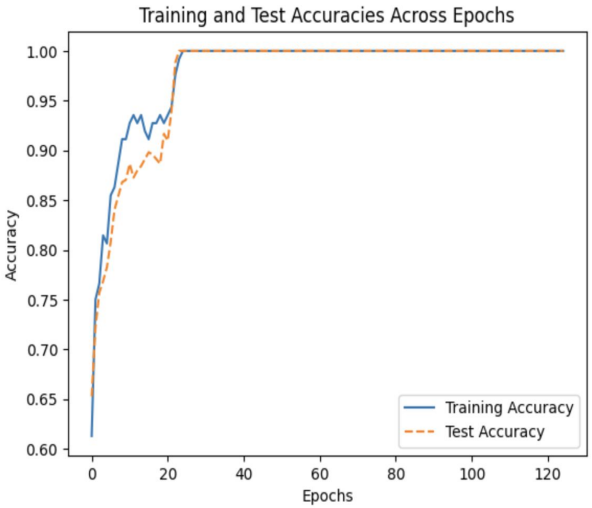


Monk 2

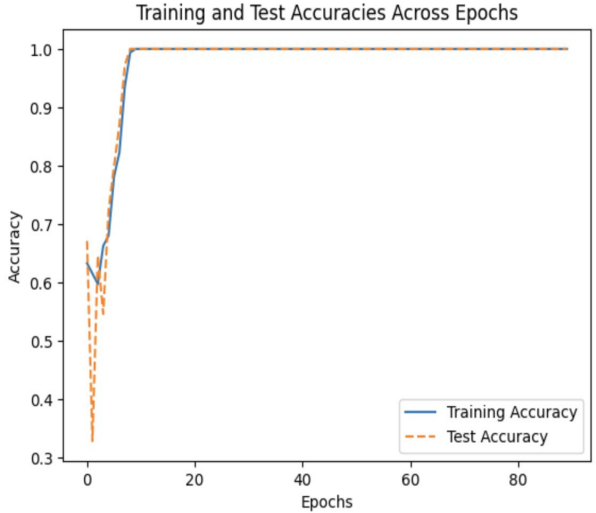


Monk 3

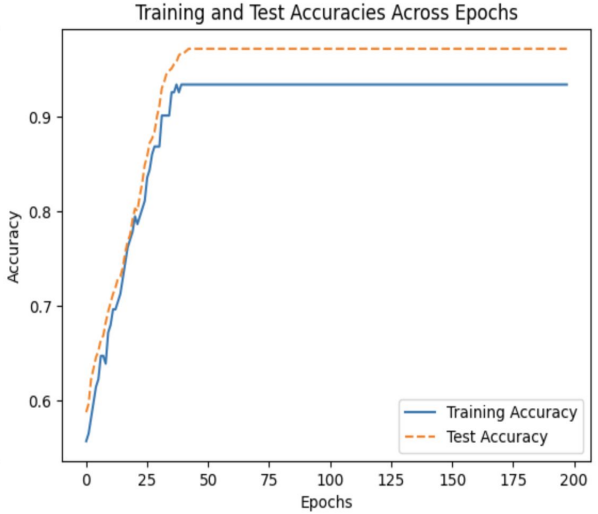
Appendix - Monk Results pytorch accuracy on test set



Monk 1



Monk 2



Monk 3

Appendix - Monk Results other Models

Solver	**DataSet**	**Penalty**	**Accuracy (TR/TS) (%)**
lbfgs	Monk1	None	76/71
liblinear	Monk2	L1	62/67
liblinear	Monk3	L1	93/97

Logistic regression

kernel	DataSet	C	degree	gamma	Accuracy (TR/TS) (%)
poly	Monk1	10	2	scale	100/100
poly	Monk2	100	2	scale	100/100
linear	Monk3	10	-	scale	93/97

SVM

Appendix - Monk Results other Models

DataSet	Accuracy (TR/TS) (%)
Monk1	70/71
Monk2	58.5/61.6
Monk3	92.5/97

Multinomial Naive

DataSet	alpha, force_alpha	Accuracy (TR/TS) (%)
Monk1	0, true	73/75
Monk2	0, true	52.6/60
Monk3	0.01, true	93/97

Bernulli Naive

Appendix - Monk Results other Models

DataSet	algorithm	metric	# neighbors	weights	Accuracy (TR/TS) (%)
Monk1	ball tree	manhattan	7	distance	83/88
Monk2	ball tree	euclidean	23	uniform	67/65
Monk3	ball tree	manhattan	23	distance	92/93

Knn

DataSet	var smoothing	Accuracy (TR/TS) (%)
Monk1	0.023	73/75
Monk2	1.0	58.5/63
Monk3	10	93/97

Gaussian Naive

Appendix - Monk Results other Models

criterion	DataSet	Max Depth	Max features	min sample leaf	# of estimators	gamma	Accuracy (TR/TS) (%)
gini	Monk1	15	None	1	128	scale	98/99.5
entropy	Monk2	15	None	1	32	scale	69.8/75
gini	Monk3	None	log2	2	32	scale	94/97.5

Random forest

Appendix - Monk Results other Models

DataSet	criterion	Max Depth	Max features	min sample leaf	min samples split	Accuracy (TR/TS) (%)
Monk1	entropy	10	None	1	2	92/90
Monk2	log loss	10	None	1	2	69/87
Monk3	gini	5	sqrt	1	8	91/80

Decision tree

Appendix - CUP Results other Models

Regularization	alpha	MEE (TR/TS)
-	-	6.07(±5e-2)/ 6.25(±5e-2)
L1	1e-4	6.27(±1e-1)/ 6.26(±5e-2)
L2	0.1	6.276(±2e-1)/ 6.259(±5e-2)

Linear regression

kernel	C	degree	epsilon	tolerance	gamma	MEE(TR/TS)
poly	10000	7	0.1	1e-4	auto	0.89(±0.1)/ 0.779(±0.1)

SVR

Appendix - CUP Results other Models

algorithm	metric	# neighbors	p	weights	MEE(TR/TS)
auto	minkowski	5	1	distance	2.8(±3e-1)/ 2.447(±0)

Knr

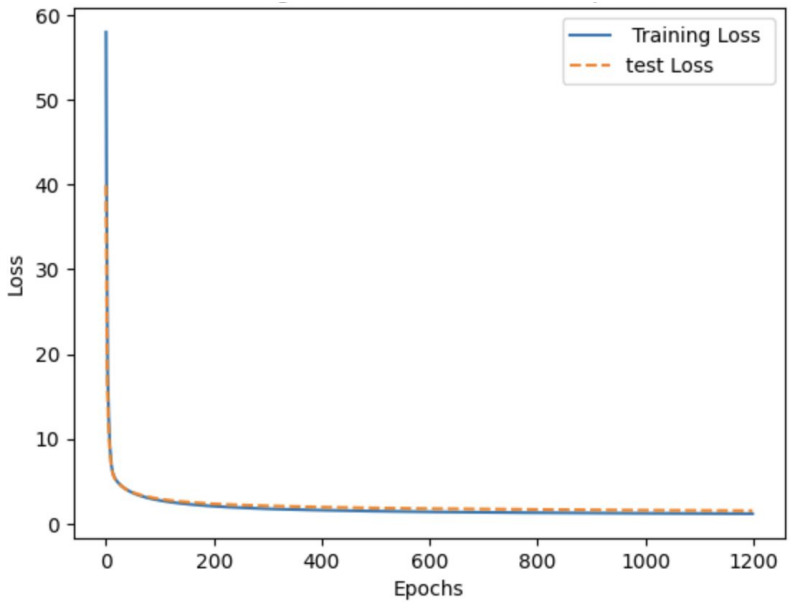
criterion	min weight fraction leaf	ccp alpha	min sample leaf, max depth, min sample split	# of estimators	gamma	MEE(TR/TS)
squared error	15	0.002	1, 9, 4	95	scale	3.55(±4e-1)/ 3.235(±2e-2)

Random forest regression

Appendix - CUP NN Pytorch

	#layers	Hidden size	Batch / Mini Batch size	Momentum	eta, lambda	MEE (TR/TS)
Pytorch	3	[128,64,32]	Batch	0.05	0.001, 0.1	1.148(±3e-2) / 1.428(±5e-2)

Appendix - CUP Best Results Pytorch MEE plot



Pytorch