Machine Learning Report

Jacopo Gasparro, Marco Sorrenti, Roberto Cannarella**

CS: Data and Knowledge / Digital Humanities*

j.gasparro1@studenti.unipi.it, m.sorrenti4@studenti.unipi.it, r.cannarella1@studenti.unipi.it.

ML course (654AA), Academic Year: 2021/22

Date: 24/01/2022

Type of project: **A**

Abstract

Our project consisted in the implementation of a MLP with back-propagation, which has been used for both classification (MONK datasets) and regression (CUP dataset). We have employed K-fold CV and Hold-out as validation techniques, and, for the CUP, tested 2000+ configurations of hyperparameters through two grid searches.

1 Introduction

The project we will present in this report has been developed for the Machine Learning course and is an example of a type [A] project: as such, it has consisted in us implementing a Multi-Layer Perceptron (MLP) with back-propagation and then using it for both classification and regression. Our aim was, other than to understand thoroughly how the algorithm works, to see how different hyperparameter values (and different combinations of them) can change the behaviour of the model.

The report is structured as follows. Section 2 briefly describes the classes that have been implemented. With regard to the experiments (Section 3), Section 3.1 focuses on the results on the MONK datasets, which have been used as a benchmark for assessing the correctness of our implementation. Then, two grid searches with K-fold cross-validation have been performed for detecting the best hyperparameters to be used for the CUP dataset (Section 3.2). Once they have been found, the model has been retrained on the whole training set and used for predicting the target values of the "blind" dataset.

2 Method

In this section, we will describe design choices we have made in order to implement the MLP. The network has been created using Python, relying on Numpy for the mathematical computations. The structure of the project is the following:

- The `model` folder is the essential one, and contains:
  - The `Layer` class, which is used to provide the characteristics of each layer which then composes the network, and to compute the output, the gradient and the error of the units. At this level, one can define hyperparameters such as $\#units$, the activation function and the weights initialization.
  - The `NeuralNetwork` class, which is the one where the different layers are stored and the neural network structure is built. The `compile` function associates the network to an `SGD` object, providing variables such as the evaluation metrics and hyperparameters values such as η.
  - The `SGD` class, which is contained in `Optimizer`, implements the Stochastic Gradient Descent. Both the classic momentum and the Nesterov momentum are available for use. Thus, the available hyperparameters are η (learning rate), α (momentum), the batch size (which allows minibatch, batch and online learning), λ (for Tikhonov regularization). Finally, `Optimizer` also contains the `EarlyStopping` class, which permits to set a tolerance and a patience value which controls the validation loss and halts the training when the network does not improve anymore.
- The `utils` folder contain various utility functions: `activation.py` provides the Linear, the Sigmoid, the Tanh, the Relu and the Softmax activation functions; `metrics.py` contains functions for MSE (Mean Square Error), MEE (Mean Euclidian Error) and Accuracy; `weights_initializer.py` provides a function for random weight initialization, Xavier Normal, Xavier Uniform, He Normal and He Uniform initializations. The `lr_decay.py` file contains a class for making the η value decrease linearly until a provided τ iteration. Finally, `regularization.py` provides functions for L1 (Lasso) regression and for L2 (Tikhonov) regularization.
- The `model_selection.py` file contains two main classes: `KFoldCV`, which implements the K-fold cross-validation method, and `GridSearchCVNN`, which is used for performing a grid search based on a starting parameter grid. Additionally, `GridSearchCVNNParallel` allows to perform grid searches in a parallelized way.

For what concerns *preprocessing*, the MONK datasets contain samples defined through 17 binary features which have been encoded in one-hot fashion for allowing classification using Pandas. No preprocessing was needed for the CUP dataset.

Coming to the employed *Validation Schema*, the MONK datasets are already split into a training and a test sets. The first has been used for the training process and the test set, despite its name, has been used for the model selection (thus, a proper model assessment phase was not conducted, which is not a problem from our perspective, since the MONK datasets have been used just as benchmarks). For CUP, instead, we used a more rigorous Validation Schema, which involved deriving an internal Test Set. Details can be found in 3.2.

3 Experiments

In this section, we will report the results of the experiments we have run. Note that they have all been performed on an AMD Ryzen 5 3600 CPU with 6 cores at 3.60 GHz.

3.1 Monk Results

For solving each one of the MONK's tasks, we have tested different combinations of hyperparameters. For all the MONK experiments, we have used an architecture with one hidden layer, containing 4 *#units* and having Tanh (MONK1, MONK2) or Sigmoid (MONK3, MONK3R) as activation function. For all the experiments, the Xavier uniform weight initialization has been used, and the used output function is the sigmoid one (we are performing classification). The models for MONK1 and MONK2 have been trained for 400 epochs, the ones for MONK3 and MONK3R for 500 epochs. The written results can be found in Table 1, the plots of the learning curves in Table 2.

Task	η	λ	α	**MSE (TR)**	**MSE (TS)**	**Acc. (TR)**	**Acc. (TS)**
MONK1	0.8	0	0.9	0.0010±0.001	0.008±0.013	100%±0	99.05%±1.55
MONK2	0.9	0	0.9	0.0002±0.0001	0.004±0.006	100%±0	99.42%±1.01
MONK3	0.8	0	0.7	0.0426±0.001	0.046±0.0009	95.65%±0.37	94.86%±0.41
MONK3R	0.8	0.0001	0.6	0.0605±0.005	0.044±0.001	94.01%±0.37	96.38%±0.54

Table 1: Average prediction results obtained for the MONK's tasks, based on 10 different trainings.

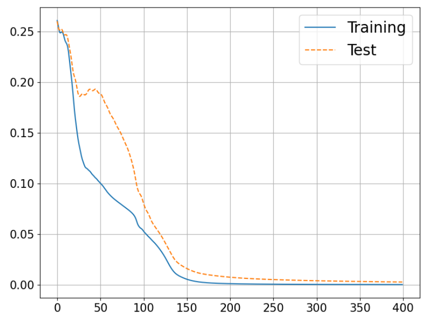
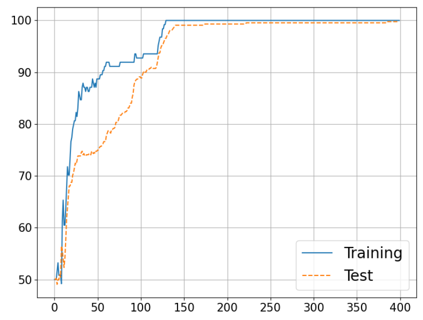
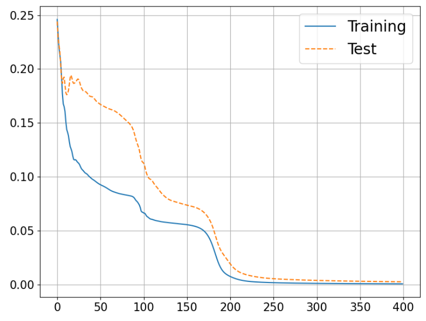
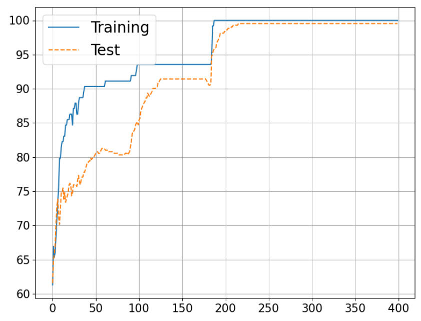
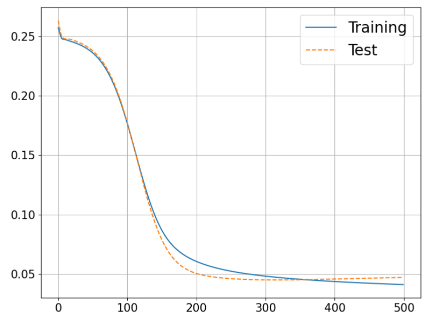
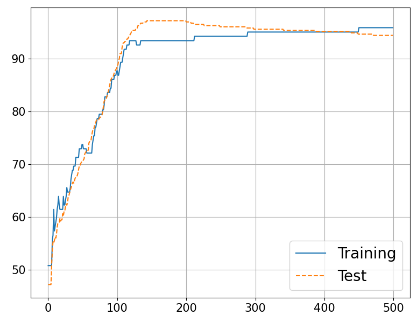
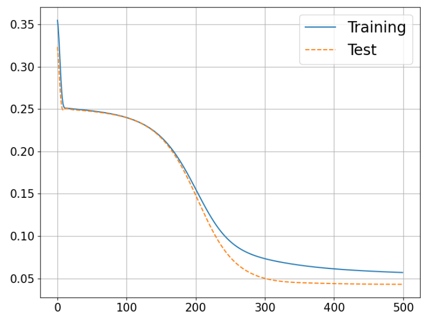
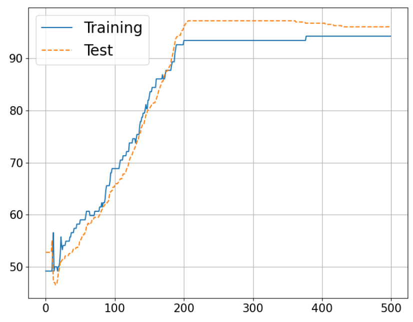
Task	**MSE**	**Accuracy**
MONK1		
MONK2		
MONK3		
MONK3R		

Table 2: Plots of MSE and Accuracy obtained for the MONK's tasks.

3.2 Cup Results

For the experiments on the CUP datasets, we employed an architecture with 10 input units (one for feature), a *#units* number of hidden units, which has been used as an hyperparameter, and two output units. Note that the number of *#hidden* (i.e. the number of hidden layers) is not fixed and will be used as a hyperparameter. For all the trials, only Tikhonov has been used as a type of regularization.

With respect to the adopted *Validation Schema*, we had a TR dataset (containing 1477 patterns) and a TS dataset with no labels (`ML-CUP21-TS.csv`), i.e. to be used only for the external/"blind" model assessment. For this reason, we have first and foremost extracted 15% of the TR datasets and stored it as an internal TS. This has later been used for model assessment through hold-out. The remaining TR data (which are 1255) have been used for performing grid searches using 4-folds for cross-validation. Once the best model has been found, we retrain it using a part of the TR dataset (85%) and evaluate it through another subpart (15%).

Some preliminary trials have been conducted during the *screening phase* in order to start exploring the behaviour of the network depending on changes on the hyperparameters. These trials led us to consider the hyperparameters and the ranges reported in Table 3 for a first and large grid search, after which another, finer grid search has been performed.

As already stated, in our model *#units* and *#hidden* are two hyperparameters: after fixing the second at 1, we have seen that the behaviour of the network can change quite significantly when *#units* values are in the range 20-200; *viceversa*, fixing *#units* at 50 or 100 has shown that setting more than 3 hidden layers leads the model to underfit. We have thus considered only (1,2) as a range. High η values lead the model to converge faster, whereas high λ values helps at avoiding overfitting: a large range of the two hyperparameters has thus been considered for the Grid Search, so as to see how they interact. For the activation function of the hidden layer(s), we have seen that the Tanh function is the one generally performing better, so we have decided to use it for all the Grid Search trials. The same goes for He Uniform as a weight initialization. During the trials, we have seen that 400-500 epochs are usually enough for the models to converge. Still, we decided to train the models for a maximum number of 1000 epochs, because we have used Early Stopping as a stopping criteria. A difference in the batch size can be quite significant for a model's generalization capabilities, so we have included two values of it in the Grid Search. Finally, we have seen that the Nesterov momentum can help the learning curve to be more stable (see Appendix A for a comparison), so we have included its use as a hyperparameter (True, False). The same goes for linear η decay.

With this table of hyperparameters, we have run a total of 1728 combinations for a fist, large grid search with 4-fold CV. Table 4 reports the 8 models/combinations of hyperparameters which are the best, i.e. which performed better on the VL.

Hyperparameter	**Range**
$\#units$	20, 40, 80, 100
$\#hidden$	1, 2
η	0.0001, 0.001, 0.01
λ	0.0001, 0.0001, 0.01
α	0.2, 0.5, 0.9
ES ($\#pat$, $\#tol$)	(250, 1e-23)
Nesterov	True, False
Linear η decay	True, False
Batch size	128, $\#batch/2$

Table 3: The hyperparameters that have been selected for the first/large Grid Search, along with their value ranges.

Comb.	$\#u$	$\#h$	η	λ	α	N	D	Batch	**TR**	**VL**	**Time**
LGS1	80	2	0.01	0.0001	0.5	F	F	$\#batch/2$	1.00±0.003	1.09±0.03	100.79
LGS2	80	2	0.001	0.0001	0.9	T	F	128	0.96±0.02	1.09±0.01	101.96
LGS3	80	2	0.01	0.0001	0.5	T	F	128	0.95±0.05	1.10±0.01	85.42
LGS4	100	2	0.001	0.0001	0.9	T	F	128	0.97±0.01	1.10±0.03	97.24
LGS5	80	2	0.001	0.0001	0.9	F	F	128	0.97±0.01	1.10±0.03	84.07
LGS6	80	2	0.01	0.0001	0.9	F	T	128	0.74±0.04	1.10±0.04	54.43
LGS7	80	2	0.01	0.0001	0.2	T	F	128	1.01±0.01	1.10±0.01	110.35
LGS8	80	2	0.01	0.0001	0.2	F	F	128	0.98±0.01	1.10±0.02	90.41

Table 4: The 8 best configurations after the Large/First Grid Search. **TR** and **VL** values refer to the MEE, **Time** refers to the number of seconds, N stands for Nesterov momentum, D stands for learning rate decay.

Looking at the results of this first Grid Search, we have observed some tendencies which allowed us to focus on some narrower ranges of values. We have observed that a high number of $\#units$ allows the model to perform better, so the new range of this value contains three values over 80. Two $\#hidden$ always appear to be better than 1, so we have included only 2 as a value in the new grid (we have not inserted new, higher values because we are already augmenting the ones of $\#units$). The range that seems the best to be explored for η is the range between 0.01 and 0.001, so we have used it for the finer grid. For λ, we have seen the lowest value to be the most useful, so we have taken values near to that into consideration for the second search. The values of the momentum are quite varying in the best models, but the higher values still result to be the most useful, so we have increased the average value of it in the new range. Learning rate decay is active only once in the best performing models, so we have excluded it from the newer grid. For the Batch hyperparameter, since the lowest of the two values (128) appears to help in the training process, we have decided to insert another value which is near to it.

The so-defined new value ranges are reported in Table 5, and have been used for a second, finer Grid Search. The total number of combinations that have been explored is of 1800. The best 6 models derived from the second Grid Search are reported in Table 6.

Hyperparameter	**Range**
$\#units$	80, 90, 100
$\#hidden$	2
η	0.001, 0.005, 0.007, 0.009, 0.01
λ	0.0001, 0.0003, 0.0005, 0.001
α	0.5, 0.7, 0.9
ES ($\#pat$, $\#tol$)	(250, 1e-23)
Nesterov	True, False
Linear η decay	False
Batch size	128, $\#batch/3$, $\#batch/2$

Table 5: The hyperparameters that have been selected for the second/finer Grid Search, along with their value ranges.

Comb.	$\#u$	$\#h$	η	λ	α	N	Batch	**TR**	**VL**	**Time**
FGS1	80	2	0.009	0.0001	0.8	T	$\#batch/2$	0.94±0.07	1.06±0.02	74.90
FGS2	100	2	0.009	0.0001	0.8	T	$\#batch/2$	1.01±0.06	1.07±0.04	92.14
FGS3	90	2	0.007	0.0001	0.9	T	$\#batch/2$	0.92±0.06	1.08±0.03	77.22
FGS4	90	2	0.007	0.0001	0.7	T	$\#batch/2$	0.98±0.009	1.08±0.04	80.47
FGS5	90	2	0.005	0.0001	0.9	T	$\#batch/2$	0.91±0.04	1.08±0.03	88.25
FGS6	90	2	0.01	0.0001	0.6	T	$\#batch/3$	1.08±0.08	1.08±0.03	81.77

Table 6: The 6 best configurations after the Finer/Second Grid Search. **TR** and **VL** values refer to the MEE, **Time** refers to the number of seconds, N stands for Nesterov momentum.

At this point, we selected the *final model* on the basis of the MEE on the VL. Thus, we selected FGS1. As we already anticipated when discussing the employed Validation Schema, we retrained it on 85% of the TR and used the rest of the TR as VL. We retrained the model for 5 trials so as to choose the best one and partly avoid problematic initializations (see Figure 1 for the learning curves of the best model). After that, we used the trained model on the internal TS. The results are in Table 7 (where we refer to the internal TS simply as TS). Finally, we used the trained model to generate the predicted values on the "blind" TS. The predictions are stored in the `IMontanari_ML-CUP21-TS.csv` file.

Comb.	$\#u$	$\#h$	η	λ	α	N	Batch	**TR**	**VL**	**TS**	**Time**
FGS1	80	2	0.009	0.0001	0.8	T	$\#batch/2$	0.94	1.02	1.06	16.41

Table 7: Results of FGS1 trained on the TR. **TR**, **VL** and **TS** values refer to the MEE, **Time** refers to the number of seconds needed for the training, N stands for Nesterov momentum.

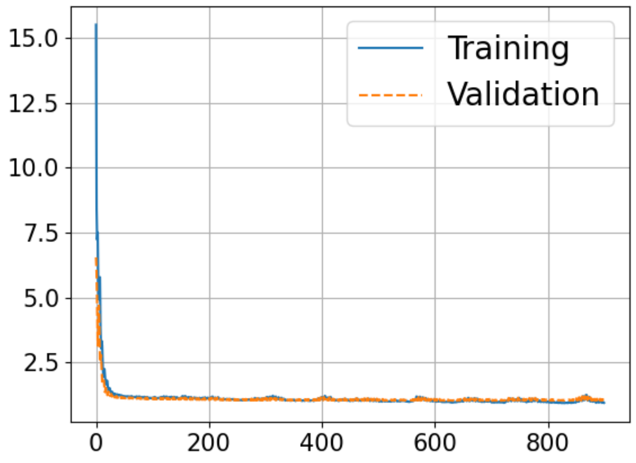


Figure 1: Plot of the learning curves of the final model on the TR and VL sets.

4 Conclusion

Working on this project has allowed us to understand more deeply the theory and the practices at the core of Machine Learning. It has made it clear how different hyperparameters can alter the behaviour of a network, and has also shown us how easily techniques such as grid searches can grow exponentially in the time they require to be performed. This latter aspect, in particular, has forced us to carefully select the most adequate value ranges, which was quite engaging.

Please note that our "blind" test results are stored in the `IMontanari_ML-CUP21-TS.csv` file (hence, our team name is *I Montanari*).

Acknowledgments

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

Appendix

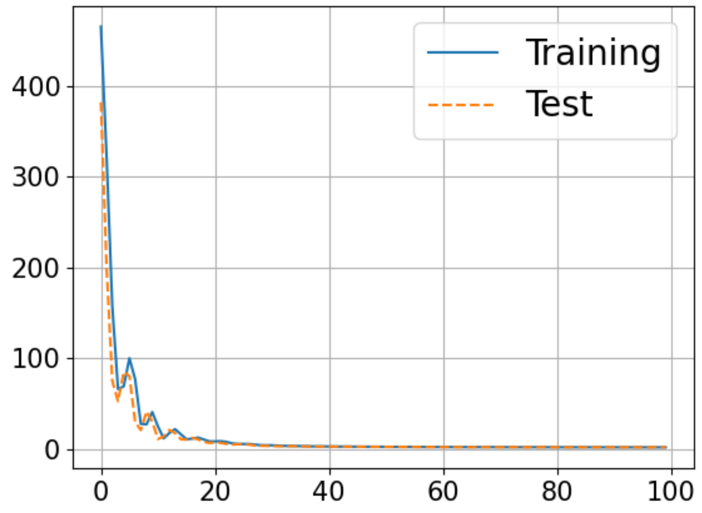


Figure 2: Learning curve of a trial with no Nesterov Momentum.

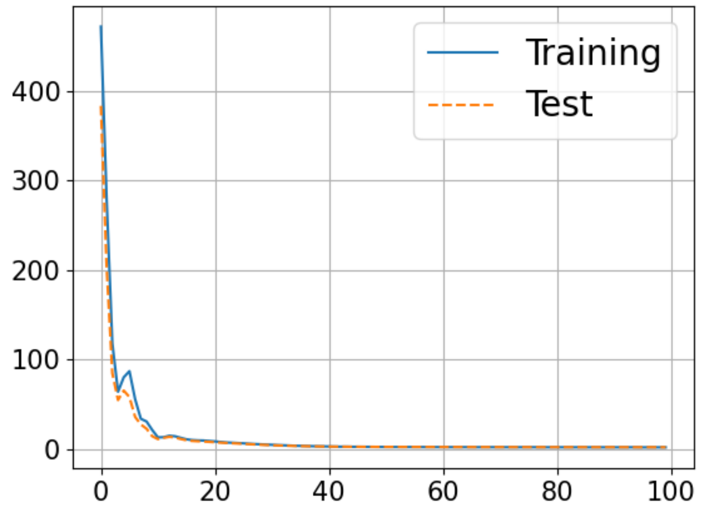


Figure 3: Learning curve of a trial with Nesterov Momentum. Note that the learning is more stable in the first 20 epochs.