Machine Learning project report

Dalla Noce Niko, Ristori Alessandro.

Master Degree in Computer science.

n.dallanoce@studenti.unipi.it, a.ristori5@studenti.unipi.it.

Machine Learning (654AA), Academic Year: 2020/2021

Date: 24/01/2021

Type of project: **A**

Abstract

The purpose of this report is to show the results obtained in our project by applying the techniques we learnt during the machine learning course. We developed a python library that builds a fully connected neural network and tunes its hyper-parameters using a K-fold cross validation with grid search.

1 Introduction

The goal of the project was to build a network by implementing its main functions (feed forward, back-propagation, weights update) with the aim to solve classification tasks on the MONKS datasets and regression tasks on the CUP dataset, the former for a preliminary network setup and the latter to take part in the professor's proposed challenge. Our library allows the management of the many hyper-parameters used by network: #layer, #nodes for layer, learning rate, Nestorov's momentum, weight decay, batch size (ranging from a stochastic to a full-batch approach), activation functions for the hidden and output layers and the loss functions for data inference.

To reach our goal we've found the best model for each task by choosing the hyper-parameters trough a grid-search with k-fold cross-validation, keeping a bias towards the configurations that produced smoother plots and with less overall noise.

2 Method

The network was implemented on python 3.8 with the help, mainly, of the following libraries:

- numpy, for linear algebra operations, which we used a lot, and for the multidimensional arrays management;
- matplotlib.pyplot, to build the plots;
- sklearn.modelselection, only the subroutines that manage the k-fold creation.

The library we built consists of classes that allow the construction and management of the network (training, cross-validation on the development set etc...). The *Layer* represents a layer (hidden or output) of the network and implements the node values (those generated by the activation function), weights, deltas and gradients; the *NeuralNetwork* class builds the hidden and output layers given the structure of the network and the activation functions of each layer: sigmoidal *Sigmoid(α)*, hyperbolic tangent *TanH()*, *LeakyReLU()* and *Linear(α)*; then the nodes are created with their weights chosen randomly in the range $[\frac{-0.7*fan-in}{2}, \frac{0.7*fan-in}{2}]$. The α parameter in *Linear* is the angular coefficient of the straight, while $\alpha = 1$ in *Sigmoid* represents the logistic function. The *NeuralNetwork* class defines the *train* method that, as the name says, trains the network given a set of hyper-parameters and a Loss function, once the method ends it returns the average errors on the training and validation set; if we're dealing with a classification task, the *train* method also returns the accuracy on the training and validation sets. The *train* method is, in fact, the heart of our library since it implements the main functions (using many submethods): feed forward, back-propagation (only based on the MSE for time and simplicity reasons, based on what we've seen in the Haykin book[1]) and updates the weights at the end of each batch.

The *GridSearchCV* class takes care of the selection phase of the best hyperparameters by combining grid search with k-fold cross validation. Given a set of hyperparameters, k (# of folds) and Cost Function, it applies k-fold cross validation to each combination of hyperparameters and calculates the average training and validation error in the k-folds, in order to select the best model. The hyperparameters on which the grid search can be applied are: the network structure (number of hidden layers and nodes), learning rate, Nesterov momentum, Tikhonov regularization (weight decay) and batch size. The activation functions have not been included in the hyperparameters because a pre-test was made before the grid search, in order to understand which one was the best, based on the smoothness of the learning curve and the error on the validation set.

2.1 Preprocessing

Inside the library there is a *Utilities* class that does the preprocessing work. In the MONKS dataset, after reading the training and test files, it applies 1-k encoding on the inputs. In the CUP data set, it partitions the ML-CUP20-TR.csv set in such a way as to obtain two independent sets: the development set (on which we apply the cross-validation), that consists of the training and validation set, and the (internal) test set. Further details on partitioning has been described in 3.2. For both MONKS and CUP, the entire development dataset has been divided into inputs and targets.

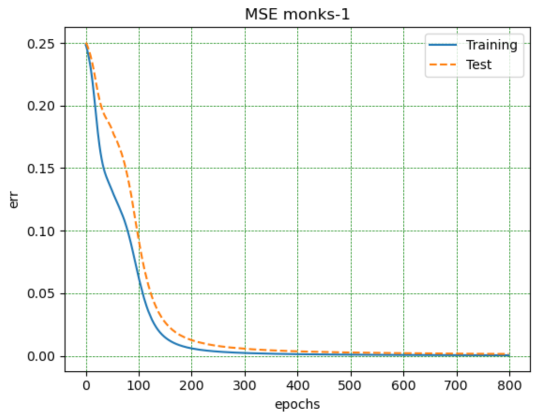
3 Experiments

3.1 Monk Results

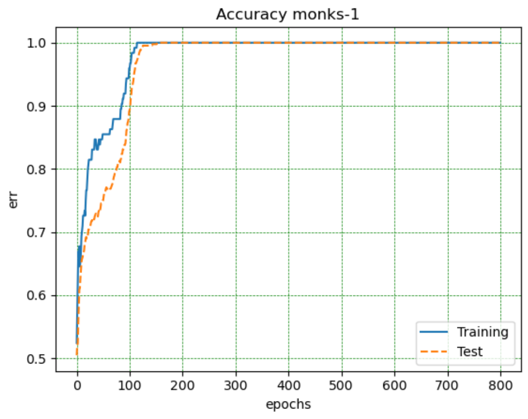
For the MONKS classification task we used a neural network with 17 input units, since we adopted the one-hot encoding, 5 nodes in the hidden layer and one for the output layer. The activation function we used in the layer was *LeakyReLu*, meanwhile for the output layer we used the *Sigmoidal* with its parameter $\alpha = 1$. After many trials, with both mini-batch and batch gradient descent, we choose the latter since it made the loss functions curve smoother. The Tikhonov regularization was only used for *MONK3* dataset because there was clearly a case of overfitting.

Task	**#units**	η	λ	α	**MSE(TR/TS)**	**Acc(TR/TS)**
MONK1	5	0.7	0	0.75	1.5×10^{-3} / 2.5×10^{-3}	100% / 100%
MONK2	5	0.9	0	0.75	1.44×10^{-4} / 1.77×10^{-4}	100% / 100%
MONK3	5	0.8	0	0.85	1.45×10^{-2} / 5.5×10^{-2}	97.21% / 93.94%
MONK3+reg.	5	0.8	0.01	0.85	4.1×10^{-2} / 3.5×10^{-2}	95.41% / 97.03%

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

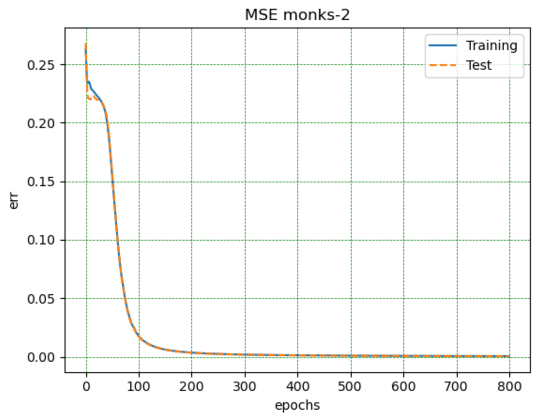


(a) MSE MONK1

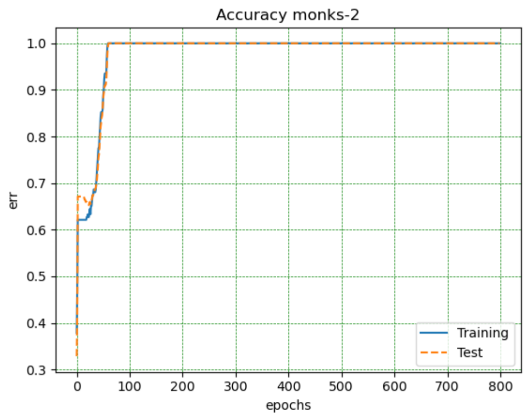


(b) Accuracy MONK1

Figure 1: MSE and accuracy for MONK1

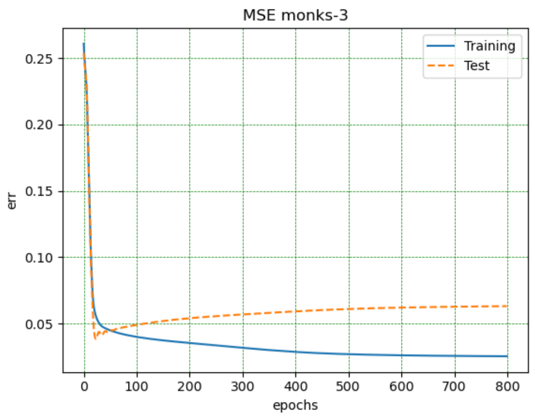


(a) MSE MONK2

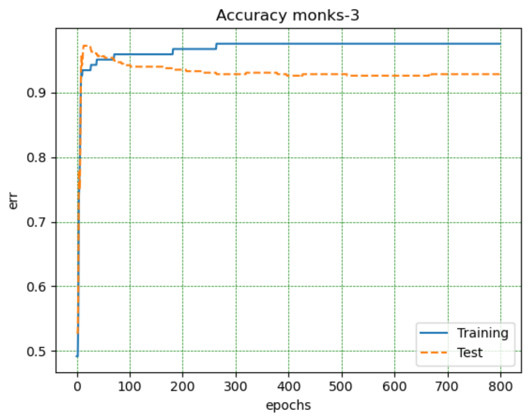


(b) Accuracy MONK2

Figure 2: MSE and accuracy for MONK2

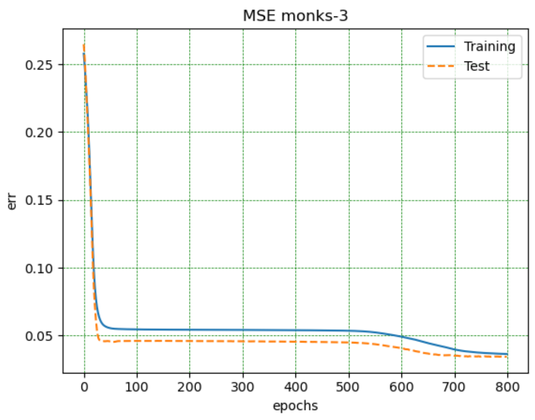


(a) MSE MONK3

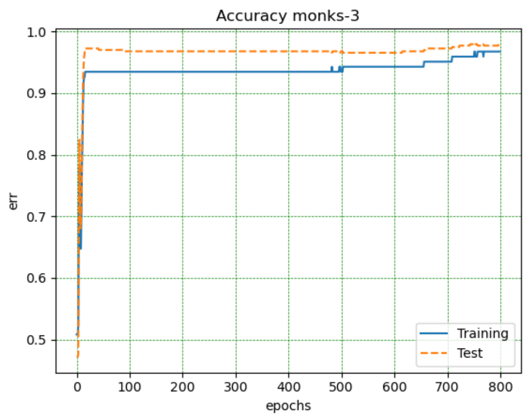


(b) Accuracy MONK3

Figure 3: MSE and accuracy for MONK3



(a) MSE MONK3+reg



(b) Accuracy MONK3+reg

Figure 4: MSE and accuracy for MONK3+reg

For each MONK dataset we showed only one plot for the MSE and the accuracy since all the plot were very similar between each other: the curves on MONK1 and MONK2 converged a little before the 300 epochs, meanwhile for the MONK3 we needed about 500 epochs in its no regularized form and more than 750 for the regularized one. Anyway, we decided to show only the first 800 epochs for each dataset since there wasn't any major improvement over the learning curves.

3.2 Cup Results

3.2.1 Validation

Talking about the validation phase, we have chosen to partition the ML-CUP20-TR dataset in order to obtain an 80% for development and the remaining 20% for internal testing. In order to select the best model, we have chosen to apply the K-Fold cross validation, with 4 folds, for each combination of hyperparameters of the grid search. The choice of k was made taking into account the execution time of the validation phase and preserving a correct balance of data in the training set and in the validation set.

3.2.2 Screening phase

The initial approach was to create two neural networks, which differ only in the activation function of the hidden layer: the first with *TanH*, while the second with *LeakyReLU*, both with only one hidden layer with 30 units and two nodes in the out layer with *Linear* activation function. Then we applied k-fold cross validation with grid search and we observed that the first network was characterized by a smoother learning curve and a lower validation error than the second one. Adding nodes or a hidden layer always produced better behavior for the network with *TanH*. We then repeated the procedure comparing *TanH* and *Sigmoid*. We noticed that the behaviour of the learning curves was very similar, but the *TanH*, in the same configuration, produced a lower error in the validation set. Understood that, we continued the hyperparameters tuning phase using this activation function (*TanH*) and with the batch approach since it didn't add noise to our learning curves.

3.2.3 Hyperparameters

During the screening phase described in 3.2.2 we tried, for each hyperparameter, very large ranges: in particular, the learning rate $\eta \in [0.001, 0.03$ with step 0.002, momentum $\alpha \in [0.3, 0.9]$ with step 0.1, one hidden layer with #nodes $in[15, 35, 60]$ and two hidden layers, respectively with [30, 40, 50] nodes. After this phase, we squeezed the ranges for each hyperparameter and we applied the grid-search with the aim to retrieve the best combinations. The table 2 shows the subsets on wich we applied the grid-search, for a total of 375 configurations; we then applied the internal test on the 10 best ones.

Hyperparameter	**Possible values**
structure	$\{[30,30],[40,30],[40,40],[50,40],[50,50]\}$
α	[0.5, 0.6, 0.7, 0.8, 0.9]
η	[0.001, 0.003, 0.005, 0.007, 0.009]
λ	[0.00001, 0.00003, 0.00005]
activation function (hidden)	TanH
activation function (output)	Linear(1)
batch size	full-batch

Table 2: The hyperparameters subranges on which we executed the grid-search.

3.2.4 Best models

As we said in 3.2.3, we've taken the best 10 models from the 375 configurations and we applied the internal test on them, the table 3 shows the MEE on the training, validation and the internal test set.

Model	**Structure**	α	η	λ	**MEE Tr**	**MEE Val**	**MEE Ts**
0	[50, 40]	0.6	0.003	0.00003	2.7211	2.9747	2.8590
1	[50, 40]	0.6	0.003	0.00001	2.7544	2.9846	2.8679
2	[50, 50]	0.5	0.005	0.00001	2.6943	2.9915	2.8822
3	[50, 50]	0.6	0.003	0.00001	2.7381	2.9772	2.8609
4	[30, 30]	0.5	0.003	0.00003	2.7858	2.9845	2.9771
5	[40, 40]	0.5	0.003	0.00005	2.7937	2.9942	2.7577
6	[40, 40]	0.6	0.003	0.00003	2.7667	2.9936	2.9444
7	[40, 40]	0.6	0.005	0.00001	2.6731	2.9769	2.8504
8	[50, 40]	0.5	0.003	0.00001	2.7799	2.9891	2.9050
9	[40, 30]	0.5	0.003	0.00001	2.7611	2.9998	2.6899

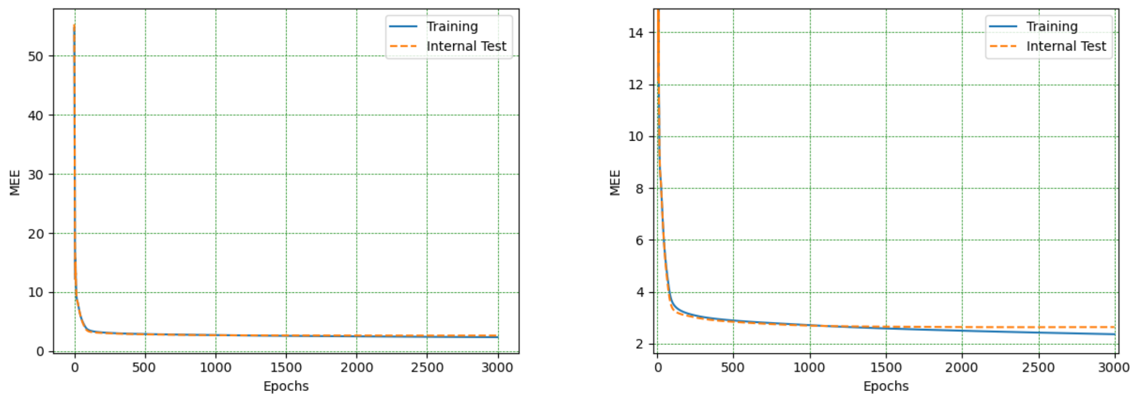
Table 3: MEE for the 10 best models obtained through the grid-search

We want to underline the fact that we obtained models with a training error smaller than the ones shown on table 3 (the range was [1.9, 2.3]), but their validation error was significantly higher and their plots showed a very clear case of overfitting, so we opted to taking in account only those that had the smallest validation error among the 375 configurations; we also noted that for $\alpha > 0.6$ the validation error was never under the 3 threshold. We computed the training and internal test with 3000 epochs for each best model and it took, on average, 390.859 seconds (6.51 minutes) on the following machines:

- Intel i7-1065G7, 1.3GHz, 16GB RAM DDR4;
- AMD Ryzen 7 4800H 2.9GHz, 16GB RAM DDR4.

3.2.5 Chosen model

We chose the best model based, first of all, on the average results produced by each model on the validation set, in reference to the table 3 and as a second yardstick on the error in the internal test. The model that performed better was number 7, the one composed of two hidden layers with 40 nodes each, $\alpha = 0.6$, $\eta = 0.005$ and $\lambda = 0.00001$. As a result, we ran the CUP blind test using this model.



(a) MEE CUP (b) MEE CUP Zoom In

Figure 5: MEE CUP internal test with chosen model.

Figure 5 shows the plot of the training and internal test MEE by using model 7.

4 Conclusion

The project gave us the opportunity to prove what we've learnt during the course, the MONK datasets were extremely useful to understand how the main hyperparameters work and their effects on the learning curves and with the CUP dataset we understood how a grid-search works and how time consuming it is. It was, overall, an important experience for our career and teamwork, in our opinion, was fundamental to overcome any problem that arised during the project.

The blind test results are on the SushiPizza_ML-CUP20-TS.csv file, our group name is SushiPizza.

Acknowledgments

We agree to the disclosure and publication of my name, and of the results with preliminary and final ranking.

References

[1] Simon S Haykin et al. *Neural networks and learning machines/Simon Haykin.* New York: Prentice Hall,, 3 edition, 2009.