Report ML-2019

Antonio Di Mauro, Fabio Murgese.
a.dimauro3@studenti.unipi.it, f.murgese@studenti.unipi.it

ML course 654AA, Academic Year: 2019/2020
Date: 15*/06/2020*
Type of project: **A**

ABSTRACT

We developed a fully-connected Artificial Neural Network in Python programming language with a modular architecture and the possibility to add new activation functions, regularizers, loss functions, optimizers; the models have been selected thanks to the use of a Grid Search method and k-fold cross validation technique.

1. INTRODUCTION

The aim of this project is to implement an Artificial Neural Network that exploits back-propagation to learn and then tweak parameters to find the model that best generalize on new unseen data.

In the next sections we are going to show firstly how our implementation of the network works correctly on MONK's problem, used as a benchmark for the latter experiments.

Then, we are going to show the results obtained on the CUP dataset for the ML-2019 CUP using our neural network with Stochastic Gradient Descent (SGD) as optimizer with momentum, moving average and learning rate decay.

For validation phase we implemented a method that splits the data in a proper way according to k-fold cross validation and another method to perform a Grid Search over a set of parameters.

2. METHOD

We implemented the neural network from scratch in Python using NumPy and Sklearn as support libraries for project type A (no CM).

We engineered the neural network in a modular way, in order to dinamically add layers directly to the network for a personalized topology. All the various components, such as error functions, activation functions, loss functions, optimizers, regularizers are distinct objects that could be dinamically added to the neural network object. In this way is also easy to add or change the behavioural features of the network as preferred.

As defaults, we set Mean Squared Error as default loss function, Mean Euclidean Error as default error function, Sigmoid as activation function for hidden layers, Stochastic Gradient Descent with momentum, learning rate decay and moving average and mini-batch approach as optimizer and a Tikhonov regularization (L2) as regularizer. The only stop condition we impose was the number of epochs, that we change during experiments according to the grid search parameter.

The actual supported components are:

- Activation functions:
  - Sigmoid
  - ReLu
  - TanH
  - Linear
- Error functions:
  - Mean Squared Error (MSE)
  - Mean Euclidean Error (MEE)
- Loss functions:
  - Mean Squared Error (MSE)
  - Sum of Squares Error (SSE)
- Regularizers:
  - Lasso regularization (L1)
  - Tikhonov regularization (L2)
- Optimizers:
  - Stochastic Gradient Descent (SGD)
  - Adam[1]

- Weight initialization:
  - Normal distribution (*mean*: 0, *std*: 1)

For what concernes the MONK's datasets we pre-processed them with One-Hot Encoding.

The CUP dataset has been splitted into Design Set and internal Test Set (TS) in the proportion of 75% and 25% respectively. During the k-fold cross validation phase, performed thanks to Sklearn library, the Design Set has been splitted into Training set (TR) and Validation set (VL), used to make an estimation of the performances of the model on new data. These results are shown in the next sections.

We implemented a learning rate decay strategy: we set a step decay schedule that drops the learning rate linearly until 2/3 of the epochs, and then it keeps the learning rate as the 1% of the original one.

In the preliminary phase, after implementation, we searched for combinations of hyper-parameters that give to our models a certain stability; these combinations have been discovered thanks to the use of different grid searches with a range of parameters from larger to thinner intervals. Once understood the trend of the model with certain parameters, we focused on tuning the configurations to adjust the behaviour of the model to an optimal one, thanks to the use of a final Grid Search.

Once found the best combination of hyper-parameters thanks to model selection phase evaluated considering the lowest error obtained on the Validation set (VL), we performed the model assessment on the inner TS after a re-train of the model using the entire Design Set (75% of the original data). We saved the final model on disk using the *pickle* library.

Then we tried a different learning algorithm, called Adam; we run another Grid Search over a set of hyper-parameters to select the best model that we then compared to the one previously selected using SGD.

3. EXPERIMENTS

3.1 MONK'S RESULTS

For Monk Datasets we first pre-processed the data thanks to One-Hot Encoding. Then we tried different Grid Searches with different order of magnitude for each

parameter, from larger to fine-grained ones, with the final objective to run our definitive Grid Search, having these parameters:

- η: 0.17, 0.2, 0.23, 0.27, 0.3
- momentum: 0.2, 0.3, 0.4, 0.5
- λ: 1e-03, 1e-04, 1e-05
- epochs: 300, 400, 500
- hidden units: 3, 4, 5
- mini-batch: 10, 20
- activation functions: Sigmoid, ReLu, Tanh

In Table 1 are shown the results obtained on the MONK's datasets using only 1 hidden layer and training with mini-batch approach. Only for MONK 3 dataset we use an L2 regularization technique, used because of the more complex nature of this dataset (mostly for noise issue).

Table 1. Results obtained for the MONK's tasks.

Task	units	η	α	λ	epochs	mb	activation	MSE (TR/TS)	Accuracy (TR/TS)
MONK 1	4	0.27	0.5	-	500	10	Sigmoid	0.0028 / 0.0063	100% / 100%
MONK 2	3	0.3	0.3	-	500	20	Sigmoid	0.0027 / 0.0032	100% / 100%
MONK 3	3	0.27	0.3	-	500	20	Sigmoid	0.0322 / 0.0333	95.08% / 95.83%
MONK3 (L2 regularization)	4	0.23	0.2	1e-05	400	20	Sigmoid	0.0359 / 0.0373	95.08% / 96.30%

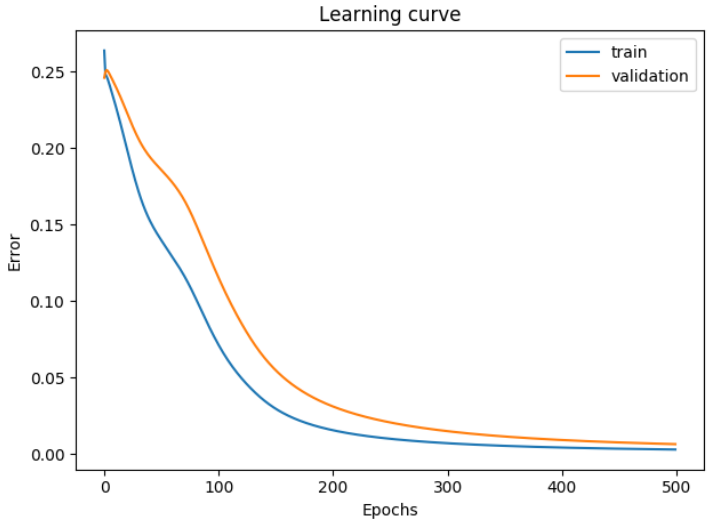

Figure 1: Plot of the MSE on MONK 1 benchmark (without regularization).

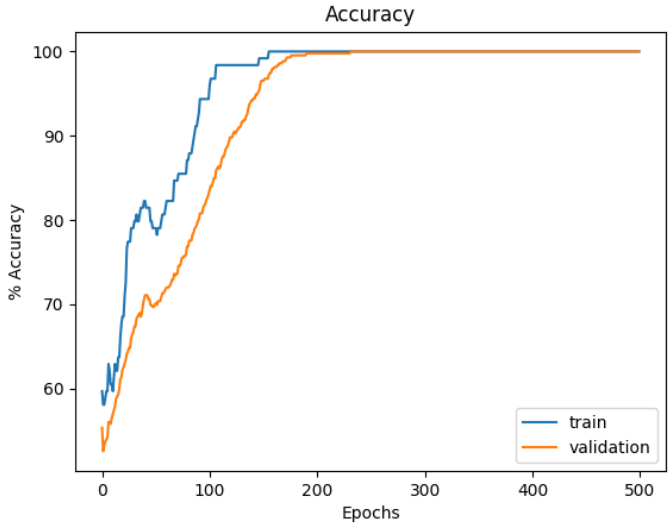

Figure 2: Plot of the accuracy on MONK 1 benchmark (without regularization).

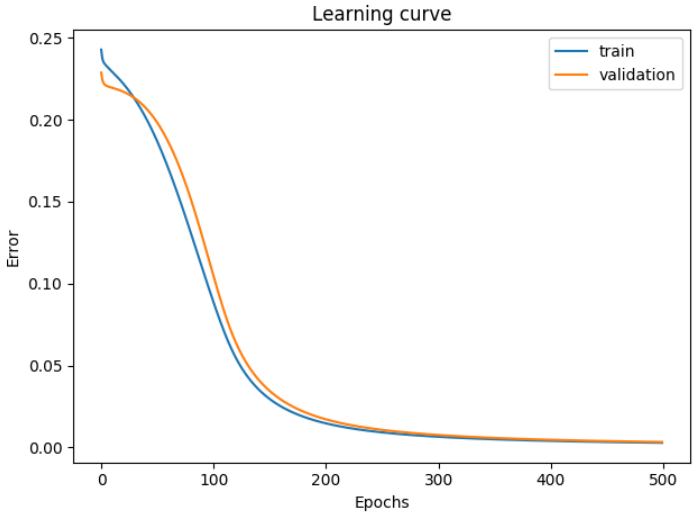

Figure 3: Plot of the MSE on MONK 2 benchmark (without regularization).

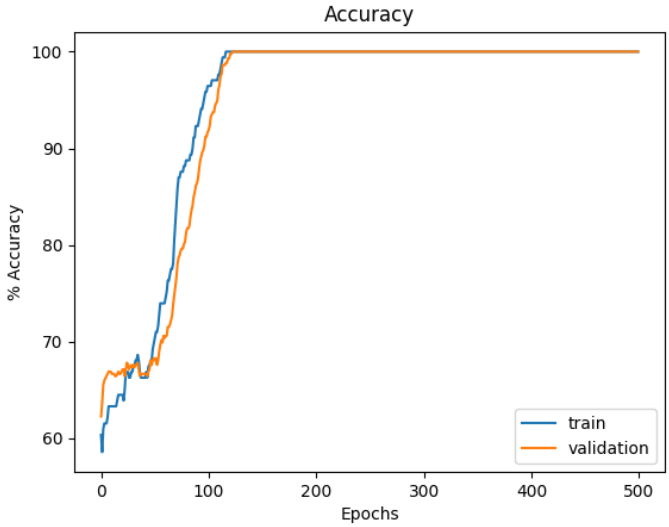

Figure 4: Plot of the accuracy on MONK 2 benchmark (without regularization).

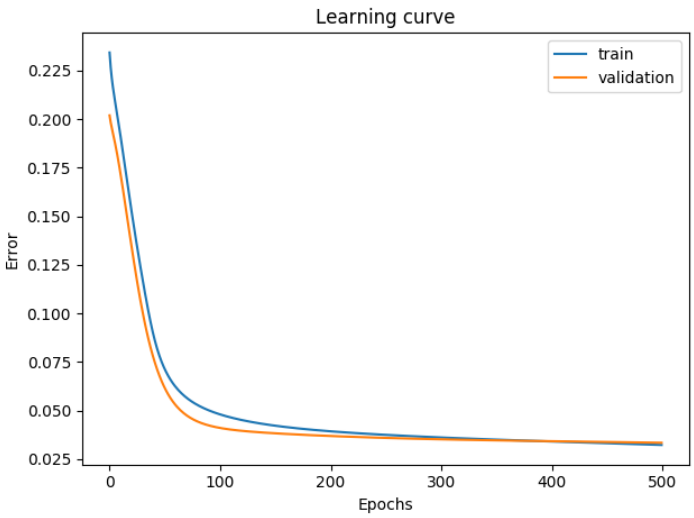

Figure 5: Plot of the MSE on MONK 3 benchmark (without regularization).

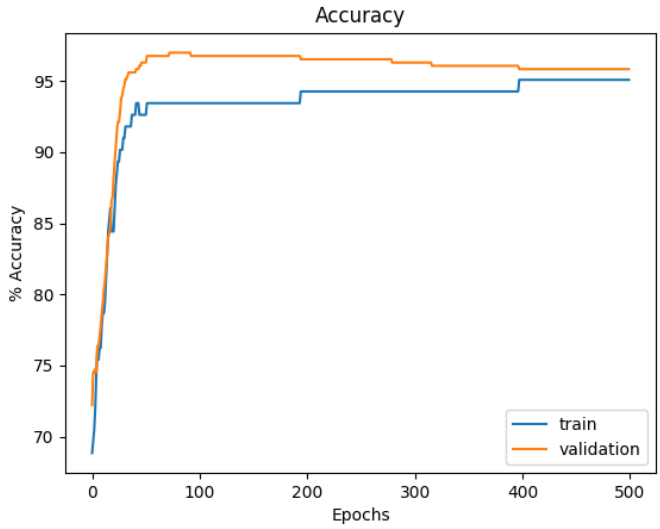

Figure 6: Plot of the accuracy on MONK 3 benchmark (without regularization).

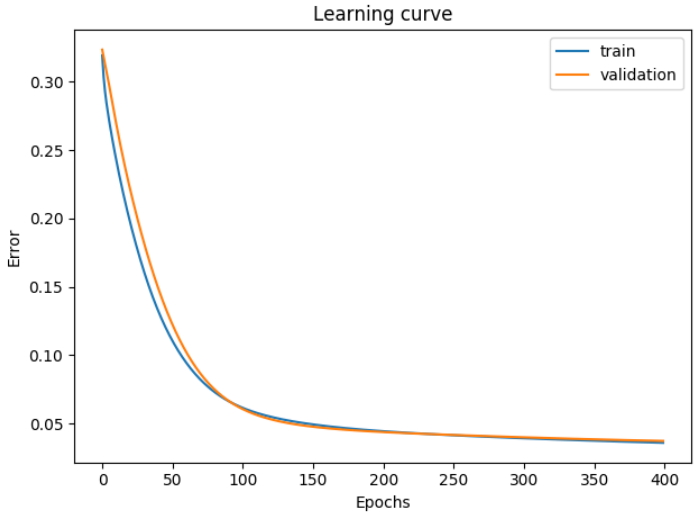

Figure 7: Plot of the MSE on MONK 3 benchmark (with Tikhonov regularization).

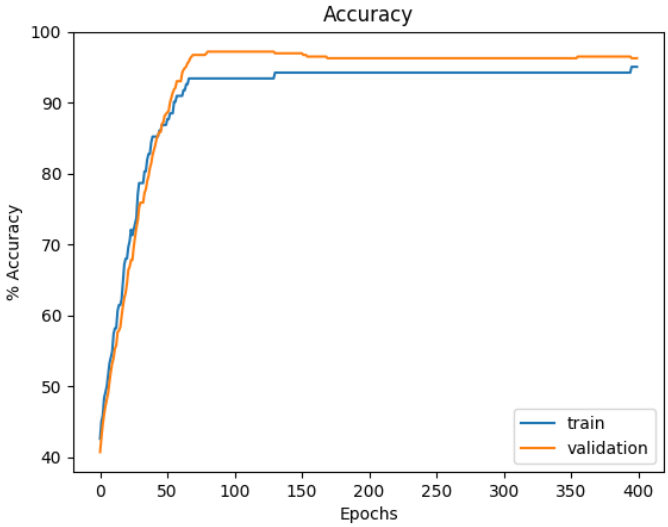

Figure 8: Plot of the accuracy on MONK 3 benchmark (with Tikhonov regularization).

3.2 CUP RESULTS

We have splitted the Cup Dataset (ML-CUP19-TR.csv) into Design Set and inner Test Set. The first corresponds to the 75% of the original dataset, while the latter the remaining 25%.

Then, we performed k-fold cross validation (with k = 5) to split the Design Set into TR + VL thanks to Sklearn library, that allowed us also to perform reshuffling of the data.

Then we tried different Grid Searches with different order of magnitude for each parameter, from larger to fine-grained ones, with the final objective to run our definitive Grid Search, having these parameters:

- η: 0.08, 0.09, 0.1, 0.12
- momentum α: 0.7, 0.8, 0.9
- momentum β: 0.7, 0.8, 0.9
- λ (L2): 1e-05, 1e-06, 1e-07, 1e-08
- epochs: 100, 300, 500
- hidden units: 10, 20, 40
- mini-batch: 10, 25, 50, 100
- activation functions: *Sigmoid*, *ReLu*, *Tanh*

For each configuration we computed the Mean Euclidean Error on TR, VL and TS (inner), as reported in Table 2.

Table 2. Mean Euclidean Errors of TR, VL and TS for some of the best configurations obtained

#	Layers	Units	η	λ	α	β	epochs	mb	Activation	MEE (TR/VL/TS)
1	3	20	0.09	1e-07	0.9	0.9	500	25	Sigmoid	0.8805 / 0.8925 / 1.0339
2	3	40	0.1	1e-08	0.7	0.8	300	10	Sigmoid	0.8992 / 0.9073 / 1.0794
3	3	40	0.09	1e-08	0.9	0.8	500	50	Sigmoid	0.9126 / 0.9186 / 1.1061
4	3	20	0.09	1e-07	0.8	0.9	500	25	Tanh	0.8626 / 0.9029 / 1.4008

5	3	20	0.1	1e-07	0.8	0.8	500	25	Tanh	0.9258 / 0.9647 / 1.2353
6	3	40	0.1	1e-06	0.9	0.8	300	50	Sigmoid	0.9763 / 0.9808 / 1.1477
7	3	20	0.1	1e-07	0.8	0.9	300	25	ReLu	0.9471 / 0.9825 / 1.2603
8	5	10	0.08	1e-07	0.9	0.8	500	100	Sigmoid	1.0699 / 1.0828 / 1.1046
9	5	40	0.12	1e-06	0.7	0.8	300	100	Tanh	1.0131 / 1.1351 / 2.0953
10	3	10	0.08	1e-07	0.8	0.9	300	50	ReLu	1.1905 / 1.1960 / 1.2628

The first 3 configurations are the best in terms of trade-off between MEE and smoothness of the learning curves. Among all the configurations we chose the **#1** as final because we noticed a better progress of the training and validation curves (smoothness), having the lowest MEE obtained, evaluated on VL. The other records show other good configurations obtained, ordered by MEE on VL.

Generally, we noticed that the best results were obtained with models using 3 layers and Sigmoid as activation function. The worst results were obtained with ReLu where the learning curves were always unstable. Usually, the best number of hidden units was 20; also, introducing learning rate decay lead to little improvements of the performances of the trained models.

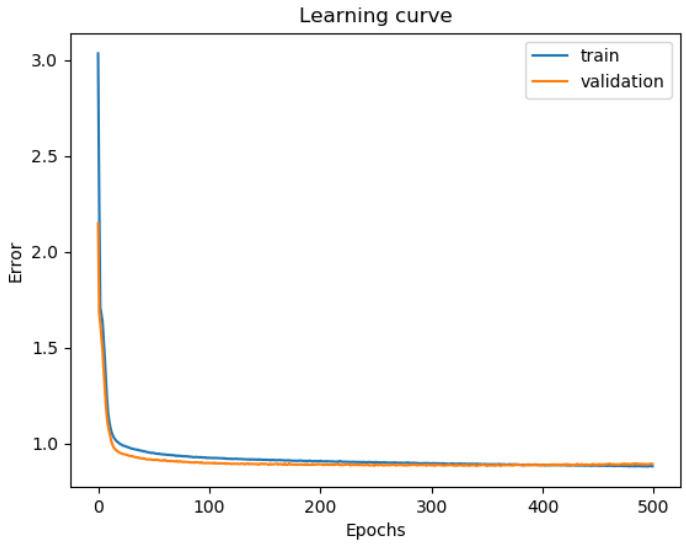

Figure 9: Learning curves of the best configuration obtained from the Grid Search (#1 of Table 2).

As we can see in Figure 9, the train and validation curves converge correctly. This training is performed in about 29 seconds on a machine with a core i7-9750H CPU @ 2,60GHz (6-cores).

The error achieved on the inner TS after the model assessment is '0.8506'.

Then we performed model selection using Adam optimizer to see what differences occur in the results in comparison with the model using SGD. In Figure 10 are shown the learning curves of the selected model having the parameters of Table 3 (after specific Grid Search – see Appendix A). The error achieved on the inner TS after the model assessment is '0.5730'.

Table 3. Final configuration of the final model with Adam

#	layers	units	η	λ	epochs	mb	activation	MEE (TR/VL/TS)
1	3	20	0.8	1e-07	800	50	Sigmoid	0.6292 / 0.6641 / 1.0691

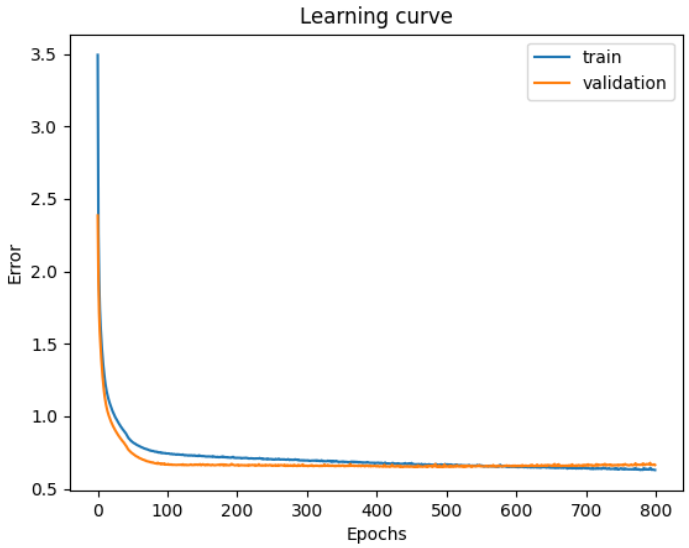

Figure 10. Learning curve of the selected model using Adam optimizer.

The training is performed in about 35 seconds; in Table 4 we can see the MEEs after model assessment in comparison with the ones achieved with SGD.

Table 4. Comparison of Mean Euclidean Errors obtained with Adam and SGD optimizers.

Optimizer	MEE (TR/VL)	MEE (TS)
SGD	0.8805 / 0.8925	0.8506
Adam	0.6292 / 0.6641	0.5730

We can observe that the errors of the model using Adam is always lower than the errors achieved using the model with SGD, so we can conclude that Adam performs better than SGD on this specific case.

4. CONCLUSIONS

This work has been representative of a real case of model selection thanks to a neural network simulator on unseen data. We learned the basics of model selection and evaluating learning curves produced by the model during the experiments. Also, we are more confident about the components of an artificial neural network and the interconnections among them.

We exploited two different learning techniques to make comparisons: SGD and Adam. With Adam we expected a lower error during training phase with respect to SGD.

Finally, we used the saved model with SGD and run the predictions on the blind test set, which results are stored in the .csv file:

BLIND TEST RESULTS: deemoore_ML-CUP19-TS.csv

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Diederik P. Kingma; Jimmy Lei Ba: Adam: a method for stochastic optimization. *Published as a conference paper at ICLR* ***2015*** (https://arxiv.org/pdf/1412.6980v8.pdf).

APPENDIX A

Grid Search for Adam models:

- η: 0.6, 0.7, 0.8
- λ (L2): 1e-06, 1e-07, 1e-08
- epochs: 500, 800
- hidden units: 10, 20, 40
- mini-batch: 10, 25, 50
- activation functions: *Sigmoid*, *ReLu*, *Tanh*