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: 30*/01/2020*
Type of project: **A with CM**

ABSTRACT

We developed a new Artificial Neural Network simulator in Python programming language with the support of NumPy library and Sklearn library (just for splitting the training dataset into training and validation datasets for k-fold cross validation).

The topology of the network used in the final model is made up by 2 hidden layers with 20 units and Sigmoid activation function, and one output layer with 2 units and Linear activation function. We performed a grid search with k-fold cross validation and 10 combinations of settings and we chose the best one having these parameters: learning rate = 0.002; epochs = 500; momentum = 0.08; regularization L2 with lambda = 1e-04; minibatch = 300; loss function = Mean Squared Error.

1. INTRODUCTION

The aim of this project is to tweak parameters and find the model that best generalize on new unseen data.

In the next sections we are going to show firstly how our implementation of an Artificial Neural Network from scratch 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) with momentum as optimizer.

Finally, we compare the best results obtained on CUP dataset with SGD, with Adam[1], an optimizer that uses Accelerated Conjugate Gradient Descent approach to optimize our Neural Network. The expectations are that with Adam we will have a smaller error with respect to SGD with the same combinations of parameters, due to its speed of convergence.

2. METHOD

We implemented the neural network from scratch in Python programming language using NumPy and Sklearn as support libraries for project type A with CM.

We engineered the neural network in a modular way, in order to dinamically add hidden 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 and mini-batch approach as optimizer and a Tikhonov regularization (L^2) as regularizer. The only stop condition we impose was the number of epochs, that we change during experiments.

For what concernes the MONK's datasets we preprocessed them with One-Hot Encoding to make the problem linearly separable.

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

In the preliminary phase, after implementation, we searched for the best combinations of hyper-parameters that gives our models a certain stability; these combinations allowed us to understand the trend of the model with certain parameters. Then we focused on tooning the configurations to adjust the behaviour of the model to an optimal one, thanks to the use of Grid Search approach.

After choosing our best model, we compared it with a different learning algorithm, called Adam, that exploits an Accelerated Conjugate Gradient Descent technique.

3. EXPERIMENTS

3.1 MONK'S RESULTS

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

Task	#Units, eta, lambda, ..	MSE (TR/TS)	Accuracy (TR/TS) (%)[i]
MONK 1	Units = 4, eta = 0.27, momentum = 0.5, minibatch = 10, activation = Sigmoid , epochs = 500	0.002826940832 / 0.006344124288	100% / 100%
MONK 2	Units = 3, eta = 0.3, momentum = 0.3, minibatch = 20,	0.002770414752 / 0.003240810558	100% / 100%

	activation = Sigmoid , epochs = 500		
MONK 3	Units = 3, eta = 0.27, momentum = 0.3, minibatch = 20, activation = Sigmoid , epochs = 500	0.032236562790 / 0.033359207571	95.08% / 95.83%
MONK3 (L2 regularization)	Units = 4, eta = 0.23, momentum = 0.2, lambda = 1e-05, minibatch = 20, activation = Sigmoid , epochs = 400	0.035959140862 / 0.037319801250	95.08% / 96.30%

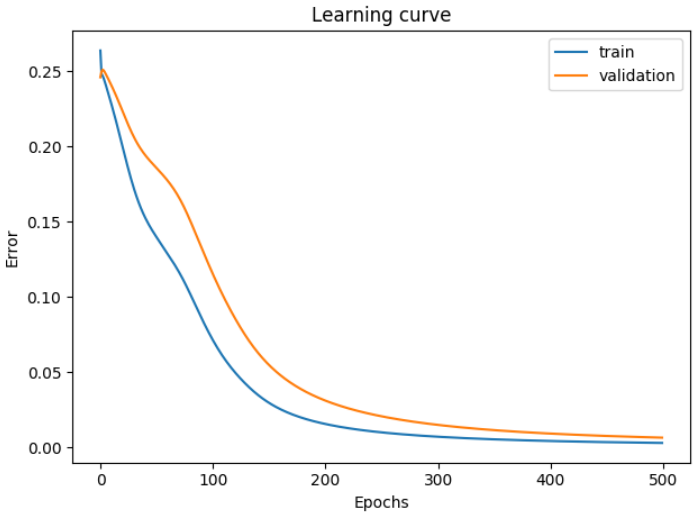

Figure 1: Plot of the MSE on MONK 1 benchmark (without regularization).

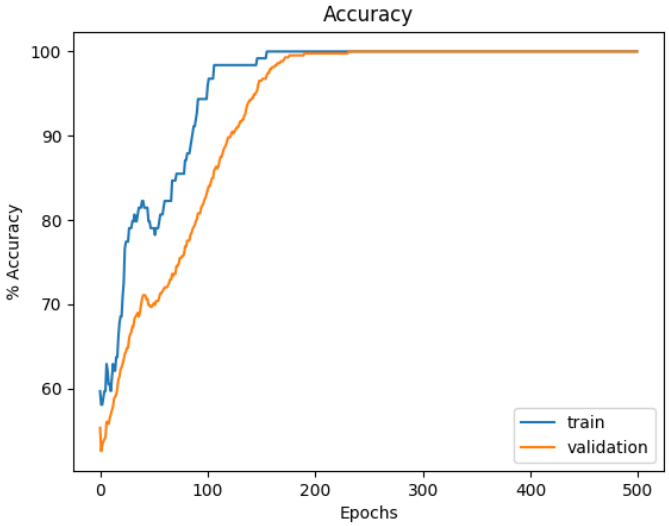

Figure 2: Plot of the accuracy on MONK 1 benchmark (without regularization).

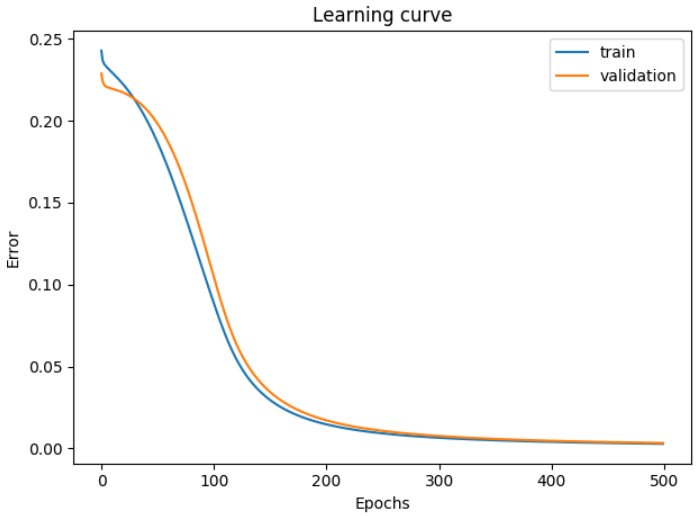

Figure 3: Plot of the MSE on MONK 2 benchmark (without regularization).

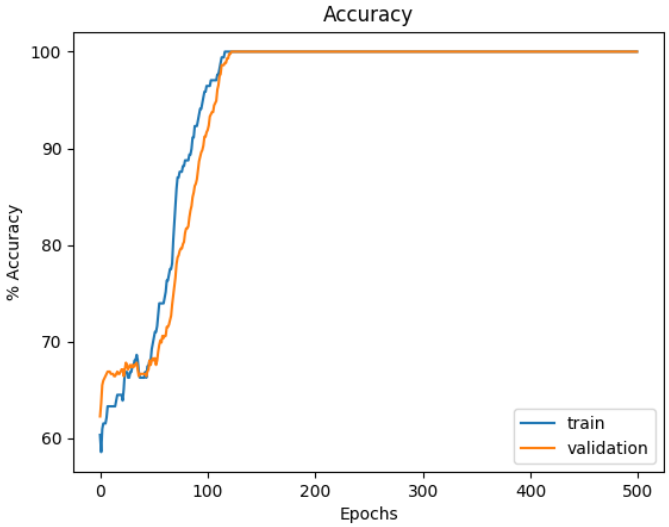

Figure 4: Plot of the accuracy on MONK 2 benchmark (without regularization).

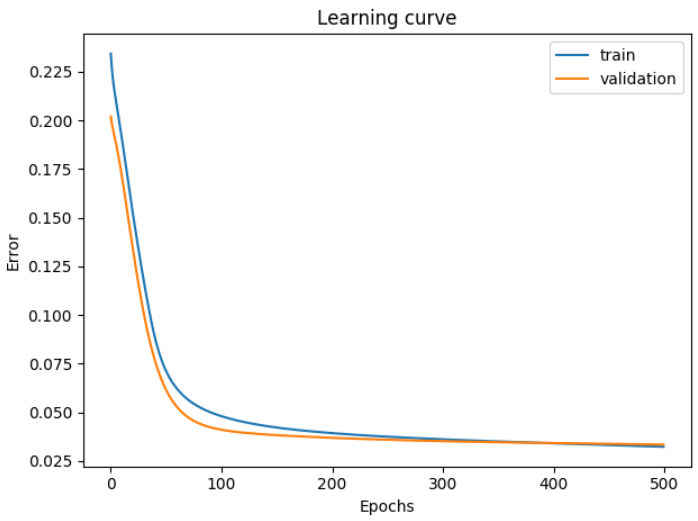

Figure 5: Plot of the MSE on MONK 3 benchmark (without regularization).

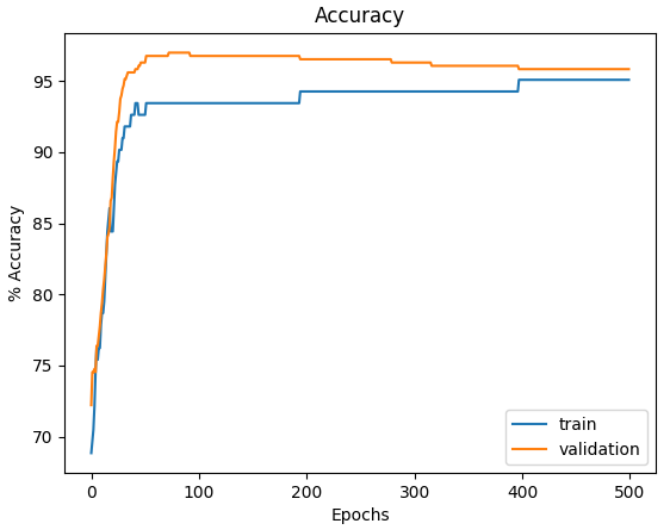

Figure 6: Plot of the accuracy on MONK 3 benchmark (without regularization).

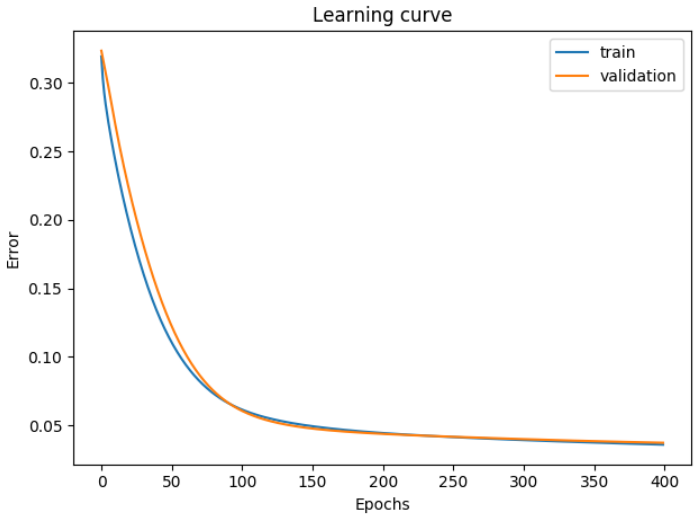

Figure 7: Plot of the MSE on MONK 3 benchmark (with Tikhonov regularization).

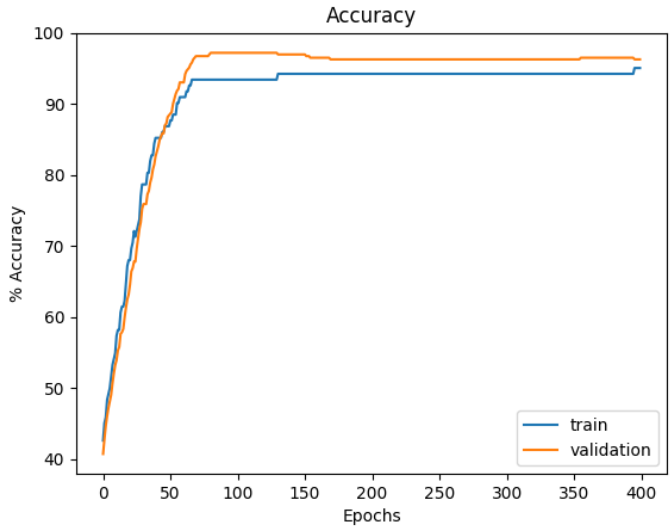

Figure 8: Plot of the accuracy on MONK 3 benchmark (with Tikhonov regularization).

3.2 CUP RESULTS

We have splitted the Cup Dataset (ML-CUP19-TR.csv) into Training 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 Training Set into TR + VL thanks to Sklearn library, that allowed us also to perform reshuffling of the data.

In the preliminary phase we searched for the best combinations of hyper-parameters that gives our models a certain stability; these combinations will be the starting points for our grid search. In the Table 2 we illustrate the configurations of our different models in the Grid Search.

Table 2. Grid Search

#	learning rate	epochs	momentum	lambda	hidden units	mini-batch size	num. folds	activation function
1	0.01	400	0.15	1e-03	10	200	5	Sigmoid
2	0.002	1000	0.3	1e-04	15	300	8	Sigmoid
3	0.002	4000	0.06	1e-04	15	300	8	Sigmoid
4	0.002	4000	0.06	1e-04	25	300	8	Sigmoid
5	0.004	600	0.12	1e-04	20	300	5	Sigmoid
6	0.002	500	0.08	1e-05	20	300	5	Sigmoid
7	0.002	1000	0.08	1e-05	20	300	5	Sigmoid
8	0.005	1000	0.2	1e-05	20	300	5	Sigmoid
9	0.002	1000	0.2	1e-05	20	300	5	Sigmoid
10	0.002	1000	0.2	1e-06	20	300	5	Sigmoid

For each configuration we computed the Mean Euclidean Error on TR, VL and TS (inner), as reported in Table 3:

Table 3. Mean Euclidean Errors of TR, VL and TS

#	MEE TR	MEE VL	MEE TS
1	0.1663608545471103	0.12406449816542889	0.09045044498894607
2	0.1047437109643718	0.14116423265007008	0.07971344965901453
3	0.10507138450528841	0.14227325196363852	0.0809727463669672
4	0.1049401731566297	0.14197695943950445	0.0821730215597762
5	0.10630322073294829	0.10479454229097537	0.07555823085429782
6	0.10272115306724679	0.10302101749223629	0.07030244031861073

7	0.09635966848768121	0.09692891754455209	0.06760202122939038
8	0.08639980590977019	0.09096209799205783	0.06482833712150764
9	0.09483893425651399	0.09636536325990787	0.06984643548646886
10	0.09484769201619367	0.09542510563686304	0.06998183532395494

Among all the configurations we chose the **5th** one because we noticed a better progress of the training and validation curves, even if the MEE isn't the lowest obtained.

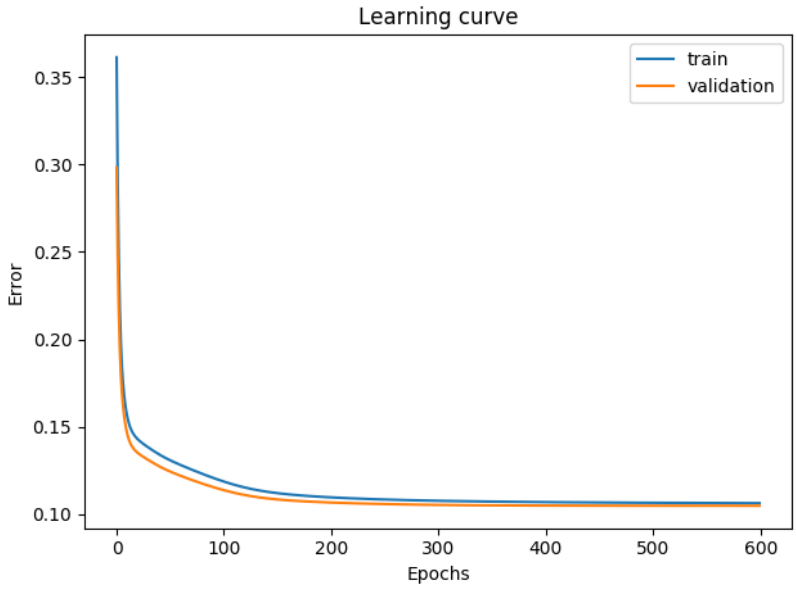

Figure 9: Learning curves of the configuration #5 of the Grid Search.

As we can see in Figure 9, the two curves maintains a good variance in the first 200 epochs of the algorithm to finally converge correctly. This training is performed in about 6.9 seconds on a machine with a core i7-9750H CPU @ 2,60GHz (6-cores) and 16GB of RAM.

Then we run this model with a different optimizer to see what differences occur in the results. So we used the Adam optimizer instead of Stochastic Gradient Descent and we obtained the learning curve in Figure 10.

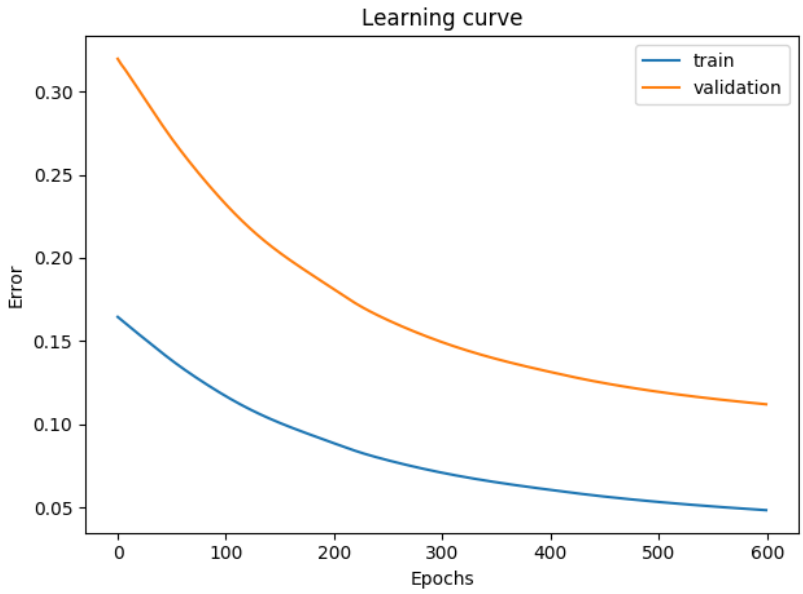

Figure 10: Plot of the MEE on CUP dataset with Adam optimizer.

The training is performed in about 14.4 seconds with the MEEs in comparison with the ones achieved with SGD shown in Table 4.

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

Optimizer	MEE TR	MEE VL	MEE TS
Adam	0.04842679503819429	0.11205728495006721	0.06834736496832484
SGD	0.10630322073294829	0.10479454229097537	0.07555823085429782

We can observe that the MEE of TS reached by Adam optimizer is lower than the one computed by Stochastic Gradient Descent approach; on the other hand, the variance between the training and validation curve of Adam is very high, and this can result in a very unreliable model that could not generalize well on new data.

This results were obtained with the same parameters of the configuration #5 of the Grid Search, reminded in Table 5.

Table 5. Hyper-parameters of the chosen model

learning rate	epochs	momentum	hidden units	mini-batch size	num. folds	activation function
0.004	600	0.12	20	300	5	Sigmoid

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: Stochastic Gradient Descent and Adam. With Adam we expected a lower error during training phase with respect to SGD within the same number of epochs, due to its speed of convergence; moreover we found true the slower but more accurate capability to generalize of SGD, in contrast to the faster but less efficient Adam, with respect to generalize on new unseen data. This was verified in the experiments we shown in the sections above. Nevertheless, we noticed that in experiments made on our inner Test Set Adam reached a better accuracy (MEE) than SGD, but we can't generalize this results saying that Adam is always better than SGD because on other tasks (and other data) its performance can be worse.

Finally, we selected the model we think performs the best 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).