Machine Learning Project Report

Author/Authors (*Marco Natali, William Simoni*).
Master Degree. m.natali10@studenti.unipi.it, w.simoni@studenti.unipi.it .

ML 654AA, Academic Year: 2020/2021
Date: 03/01/2021
Type of project: A

Abstract

This short report describes a personal implementation of an Artificial Neural Network using Python language and we have performed the model selection through a grid search on various hyper-parameters.
We explore a thousand of configuration using k-fold cross validation and to express the effectiveness of our implementation we tested on the monk datasets and then we apply to the ML cup dataset through an ensemble of 10 models.

1 Introduction

The project consists in developing a Artificial Neural Network simulator from scratch using Python language, that is able to solve both classification and regression tasks.
The aim of the project consist to develop a rigorous method to implement models, find the best combination and then validate the proposed solution.
To implement this we develop a grid search on its hyperparameters using a k-fold cross validation, with k usually set to 4 but is possible to set different values, and a more detailed description will be done in section 2.
The model was used to solve the well-known monk problem and to predict the output coordinates given the real data sensors of the ML-cup ([2]); the results of the model selection phase and the experiments are described in section 3. We also choose to normalize input data for ML-CUP using a standard scaler, in order to prevent numerical problem in the model implementation.

2 Method

We develop the Neural Network simulator using the Python language, with numpy library for all matrix operation, multiprocessing library for achieving a parallel execution during grid search, but the important aspect is that no off-the-shelf machine learning library, tool or model were used for this project.
We implemented a fully connected feed forward Neural Network that learns its weights through backpropagation using Gradient Descent technique over multiple epochs: it is possible to use batch, minibatch and stochastic approach but for performance in our implementation is suggested to use batch or minibatch with large l.
We also implemented some variations and improvements such as regularization, momentum, different kind of weights initialization, different activation functions, and different learning rate behavior (constant or linear decay).

Each one of this implemented features can be tuned through a different hyper-parameter and in next paragraph are listed all the hyper-parameters considered with also their description.
For ML-CUP we have decided to perform a preprocessing of data using the standard scaler and in paragraph 2.3 describes how we select the best model for ML-CUP.

2.1 Implementation choices

We develop the simulator using an OOP approach, so we have *NeuralNetwork* class to represent a Neural Network, which contains all hyper parameters and has a list of 1 to n layer (the last one must be an Output layer), implemented with Layer class.
ActivationFunction class implement an activation function with their derivatives (the principal ones implemented by us are TanH, Relu, Linear, LeakyRelu), we have also developed 3 weight initialization (he, xavier and ranged uniform) and some loss and accuracy measures function to evaluate our models.
Classification tasks was evaluated using the classic accuracy function, instead Regression tasks was evaluated using Mean Euclidean Error as accuracy metric, and we have chosen to not implement a subclass used to represent classification and regression tasks since we consider out of scope of this project.
As loss function we have chosen to use always the MSE, since is a common choice for Regression and can be also used to classification tasks, and also Backprogation was only derived for that loss function.

However, we did not implement some parameters that are out of scope for this project(such as Nesterovs Momentum and other solver techniques like Adagram or Adam) and for a software engineering perspective our implementation is interchangeable from Scikit Learn and/or Tensorflow library.

2.2 Hyper-parameters considered

In our implementation we have chosen to consider the following hyper-parameters:

learning_rate: represent the learning rate used in training.

momentum: represent the momentum coefficient.

regularization: represent the regularization value used.

weight_initialization indicate the weight initialization used

activation_hidden: indicate the activation function used for all hidden layers.

type_nn indicate the type of NN (usually set to batch for efficiency).

batch_size indicate the batch size for Gradient Descent (not used if type_nn is set to batch).

topology: indicate the topology used for hidden layers, so $(20, 20)$ indicate two hidden layers with 20 nodes each one.

num_epochs: indicate the num epochs to use to training NN models.

In our experiments we have chosen to use the same activation function for all hidden layers and also that activation function for output layer was chosen from the ones more indicated for the task, so linear for regression and tanh/sigmoid for classification.

2.3 Validation schema used

We decided to divide the ML-CUP dataset in two parts: the development set(80%) and the internal test set(20%); the development set was used to find the best model instead the internal test set was used to estimate its generalization performances on unseen data, so internal test set was never used/ watched until we evaluate at the end our best model in model assessment. The module grid_search implement a grid search on a set of hyper-parameter values and each selected configuration is evaluated using a 4-fold cross validation on the development set, implemented in the cross_validation module.
This procedure consists in splitting four times the development set into training set (75%) and validation set (25%), each time using a different part for the validation set, then training a model with the selected configuration of hyper-parameters on the training set and evaluating it on the validation set.
For each configuration the average Mean Euclidean Loss on the training set and validation set is reported as well as their standard deviations, and also configurations that lead to overflows and other numerical errors due to the instability of the hyper-parameters are discarded in a first model selection phase.
The 10 configurations that achieve the lowest Mean Euclidean Loss on the validation set, that are also stable as learning curves, are selected as best models, so the ensemble of those 10 best models is selected as final model.

3 Experiments

3.1 Monk Results

All the models have an input layer of 17 units because of the one-hot encoding technique application, one hidden layer that consists of 15 units with ReLu as activation function, and one final output layer with one unit that uses the sigmoid activation function. Due to this final output layer, we changed the target values in the monk data set to be within the range of the sigmoid function: from 0 to 0.1 and from 1 to 0.9. We initialized the unit weights by using a random uniform distribution in the range [-0.12, 0.12] for Monk 1 and 2, and the he method for Monk 3. The model has been trained using the MSE and the batch gradient descent.

The charts in figure 1 show the average learning and accuracy curves for each Monk dataset. For Monk 3 data set, we have overfitting after 100/150 epochs (on average), as we can notice from figure 1e. It is also notable the fact that, by adding regularization, we avoid overfitting (at least in the first 500 epochs) and we increase the performance in the test set as shown in the table 1 and in figure 1h.

Task	**eta**	**alpha**	**lambda**	**MSE(TR/TS)**	**Accuracy(TR/TS)**
Monk 1	0.8	0.8	0	0.0031/0.0039	100%/100%
Monk 2	0.7	0.8	0	0.0042/0.0056	100%/100%
Monk 3	0.8	0.8	0	0.0149/0.0242	99%/95.9%
Monk 3 (+reg)	0.8	0.8	0.01	0.0548/0.0472	93.44%/97.22%

Table 1: Average of the loss and the accuracy on ten trials.

3.2 Cup Results

For ML CUP dataset we used 80% of the data for training and model validation, and the remaining 20% for the internal test set. We used a grid search to find the best hypothesis. During the grid search, we used cross-validation (with k = 4) to evaluate the models, which returns the average MEE on the validation set, the variance of the MEE on the validation set in the different trials, and the average training error where the MEE on the validation set was minimum (for early stopping). We actually did not use this last measure because we observed that there is not overfitting in the first 500 epochs.

In table 2 there are the hyperparameters used for the grid search, where some values (namely num_epoch, type_nn, batch_size) were fixed for reduce the number of configurations tried but also because we found from preliminary trials that their value was almost irrelevant for the results. We noticed that a learning rate greater than 0.15 bring to an unstable learning curve. Since we wanted stable learning curves, we decided to don't go over 0.15. We also realized that a neural network with one single layer performs worst than a neural network with more layers and the same number of units.
Our grid search for 900 configurations took around 3 hours and half on Intel i7-7700k using 5 cores and from result obtained by grid search we sort by Mean Euclidean Error on validation set in ascending order.
From this sorted result we select the first 10 models that are stable as our 10 best model and in table 3 we can find hyperparameters and performance on these models.
From our grid search we found also that the choice of weight initialization is irrelevant for our dataset, and also that Relu and TanH are the best activation functions for Cup dataset (this was discover from preliminary trials).

Our final model is an ensemble of these 10 best models retrained using different training set obtained by resampling with repetition. Since the validation MEE is not a good estimate of the real test error we computed the predictions on the internal test set(the unused 20% of the ML-CUP dataset).
The predictions of the ensemble model are just the average of the predictions of the constituent models and to compare the performances of the 10 best models with the performances of the ensemble model we computed the Mean Euclidean Error on the internal test set on the predictions of both the constituent models and the ensemble model.
As shown by the table 4 the Mean Euclidean Error of the ensemble model (3.289297) is significantly lower than the one of its constituents and this result is also theoretically grounded (for

Hyperparameter	**Possible values**
learning_rate	[0.1, 0.13, 0.15]
momentum	[0.6, 0.8, 1.0]
lambda (regularization coefficient)	[0.0005, 0.00075, 0.001, 0.0025, 0.005]
weight_initialization	['ranged_uniform', 'xavier']
activation_function	['TanH', 'ReLu']
num_epoch	500
type_nn	'batch'
topology	[(30, 20), (20, 20), (10, 5, 5), (15, 15), (20,)]

Table 2: Hyperparameters values choice for Grid Search, whose the description of the rule of each hyperparameter is done in section 2.2. Ranged_uniform is always with range [-0.5,0.5] because we realized during our trials that it was the best range (we tried also with [-0.3,0.3] and [-0.8,0.8])

a detailed explanation see [1]).
Figure 2 shows the learning curve for the models, using Mean Squared Error value at each epoch, on both the development set and the internal test set.

4 Conclusion

We exploited the ensemble technique to get a final model that achieves lower MEE on the internal test set compared to the constituent ones and we found that the learning curve of the ensemble model is also more stable.
We also observed that using normalized data, the learning curve becomes more stable and the final training error becomes smaller.
We used the ensemble model to compute the predictions on the blind test set and we saved this predictions on the file named Scarsenal_ML-CUP20-TS.csv and we are positive that the MEE on the internal test set will be a good approximation of the real performances of our final model on unseen data since the internal test set was not used in the model selection phase.

Acknowledgments

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

References

[1] Richard Maclin. “Popular ensemble methods: An empirical study”. In: (1999). DOI: `10.1613/jair.614`.

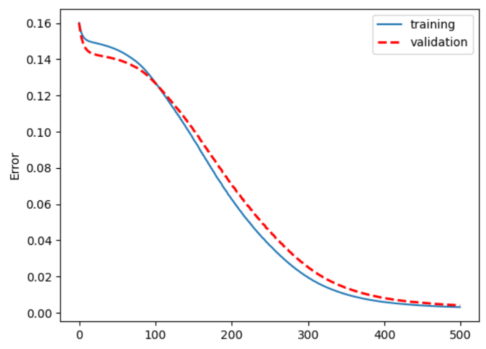
[2] Alessio Micheli. “ML CUP Dataset 2020”. In: (2020).

Accuracy	Variance	learning	lambda	momentum	activation	weight_init	topology
3.670449	0.178306	0.15	0.0005	0.6	TanH	ranged_uniform	(20, 20)
3.716322	0.145311	0.15	0.001	0.8	ReLu	xavier	(15, 15)
3.720840	0.175994	0.15	0.001	0.8	ReLu	ranged_uniform	(15, 15)
3.898049	0.009377	0.15	0.0005	0.8	TanH	ranged_uniform	(10, 5, 5)
3.714822	0.194460	0.15	0.001	1	ReLu	ranged_uniform	(20, 20)
3.768727	0.156904	0.13	0.0005	0.6	TanH	ranged_uniform	(15, 15)
3.825000	0.108781	0.13	0.00075	1.0	ReLu	ranged_uniform	(10, 5, 5)
3.874266	0.123188	0.13	0.0005	1.0	TanH	ranged_uniform	(15, 15)
3.949618	0.066005	0.1	0.00075	1.0	TanH	ranged_uniform	(15, 15)
3.906769	0.125019	0.15	0.0025	0.8	ReLu	xavier	(15, 15)

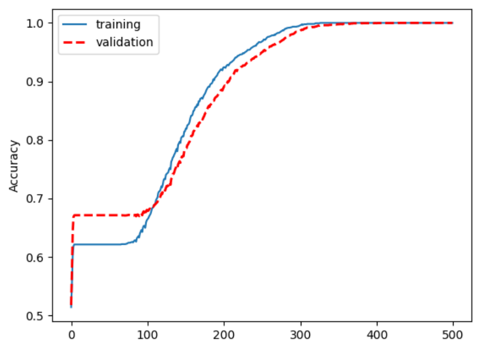
Table 3: Average accuracy (MEE) on the validation set, and variance of the accuracy (MEE) on the validation set in the different trials, obtained on cross-validation (k = 4) of the 10 best models. We ordered models summing accuracy and variance.

Model	Training Set MEE	Internal Test Set MEE
Model1	3.655512	3.572540
Model2	3.762747	3.719168
Model3	3.797938	3.612622
Model4	3.944433	3.644124
Model5	3.427667	3.361166
Model6	3.908488	3.684465
Model7	4.091177	4.051030
Model8	3.716116	3.665238
Model9	3.736212	3.716879
Model10	3.728312	3.643450
Ensemble	3.437236	3.289297

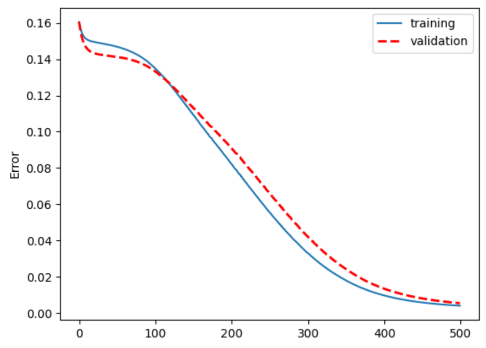
Table 4: Training set MEE and internal test set MEE on the best 10 models.



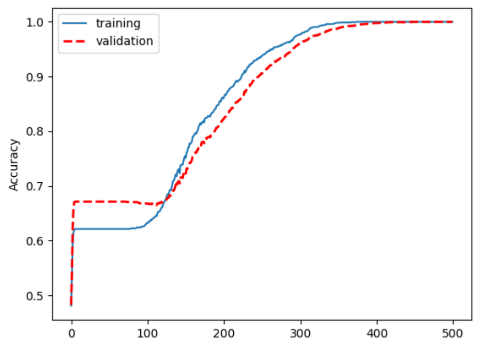
(a) Loss MSE Monk1



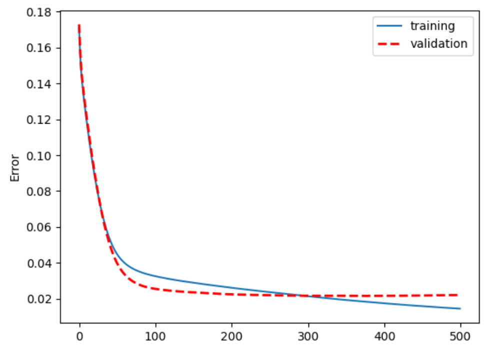
(b) Accuracy Monk1



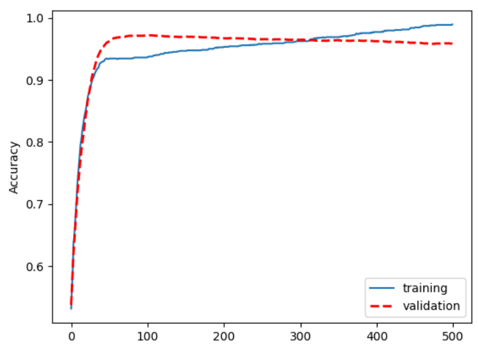
(c) Loss MSE Monk2



(d) Accuracy Monk2

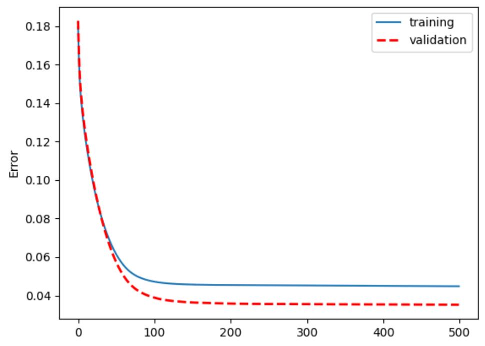


(e) Loss MSE Monk3 no regularization

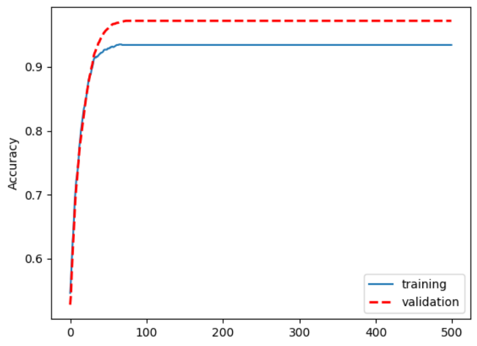


(f) Accuracy Monk3 no regularization

Figure 1: Average learning curves for Loss and Accuracy for Monk

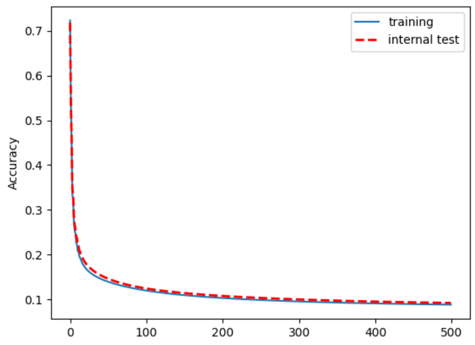


(g) Loss MSE Monk3 with regularization

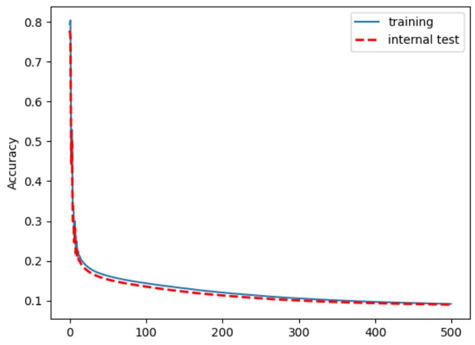


(h) Accuracy Monk3 with regularization

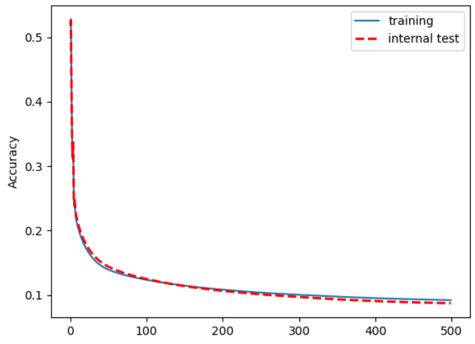
Figure 1: Learning curves for Loss and Accuracy for Monk



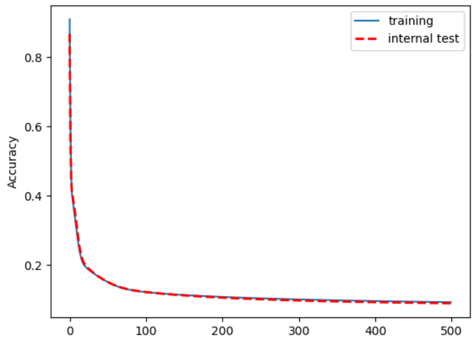
(a) Accuracy of 1° model



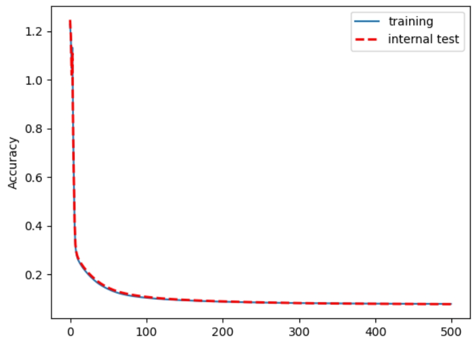
(b) Accuracy of 2° model



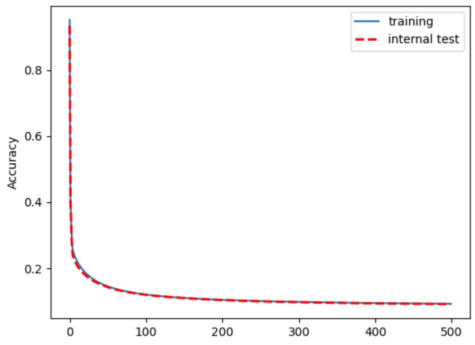
(c) Accuracy of 3° model



(d) Accuracy of 4° model

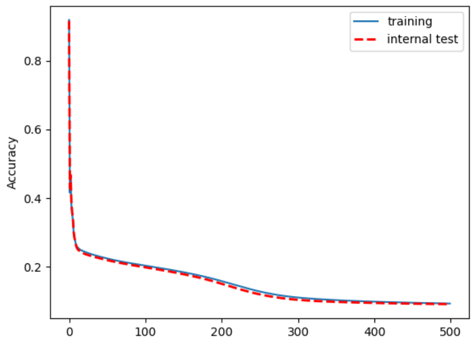


(e) Accuracy of 5° model

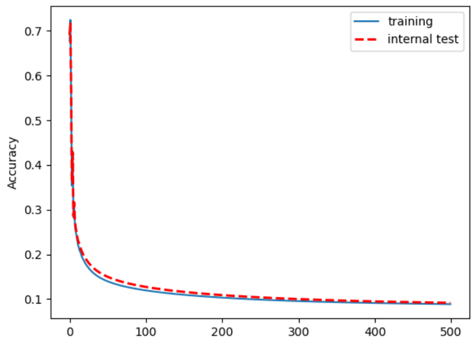


(f) Accuracy of 6° model

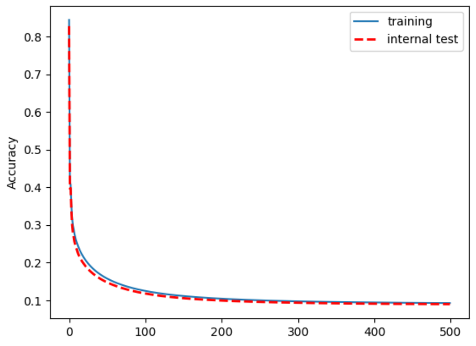
Figure 2: Learning curves obtained with the final retraining of the best 10 models. Note that plots are normalized.



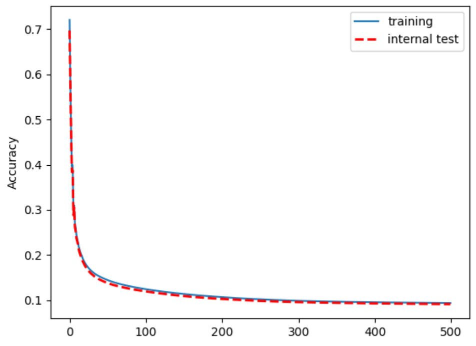
(g) Accuracy of 7° model



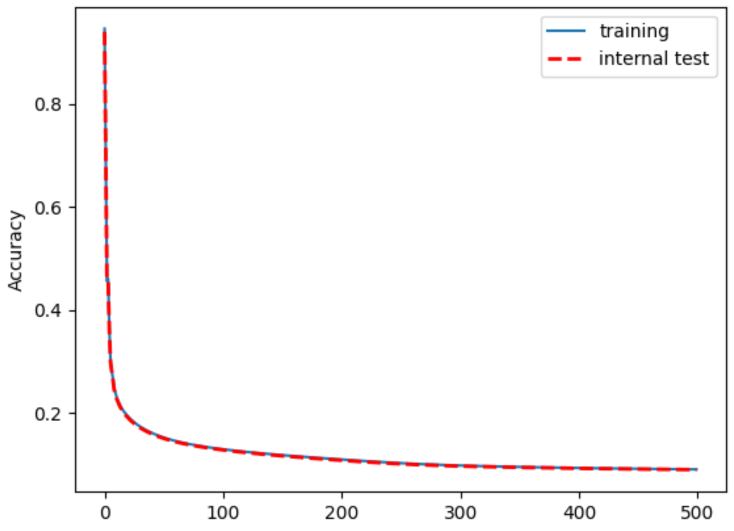
(h) Accuracy of 8° model



(i) Accuracy of 9° model



(j) Accuracy of 10° model



(k) Accuracy of ensemble model

Figure 2: Learning curves obtained with the final retraining of the best 10 models. Note that plots are normalized.