Machine Learning Project Report
Type A*

Giacomo Antonioli†
(Dated: October 28, 2021)

Implementation of the course contents by developing a simple Neural Network without using ML libraries available on the market. Each aspect of the net is implemented in the given code, from the creation of the **net**, the **feed-forward process**, the **loss calculation** and the **weight update**. Different optimization strategies were applied in order to improve the results and efforts were take in order to avoid over-fitting. The code is written in Python and as a benchmark tool to asses its correctness the **MONK data-set** will be used and the results will be reported. The implemented code will be then applied to another data-set, the **CUP data-set**, in order to enter the competition.

1. INTRODUCTION

This report's aim is to describe the development and realization of a learning system simulator and validating it by reporting its benchmarking results. The system was tuned by searching different hyper-parameters and evaluating the relative models in order to obtain the best performing models for both the data-sets.

2. MODEL DESCRIPTION

The learning simulator under analysis is a simple implementation of a Multi Layer Perceptron model. A brief description of each part of what was implemented will now follow to permit the reader to acquire a better comprehension of the development aim of this report. Each subsection will introduce all the different classes developed in order to build each aspect of the simulator.

2.1. Layers

The *Layers* class is the main building block of the given simulator. It implements a single layer of a MLP and specializes into two different types: a Dense layer and a Dropout layer. This two specializations both implement a `forward_pass` method to propagate the computation from the input to the output and also a `backwards_pass` method to propagate to the origin the error.

2.1.1. *Dense Layer*

This is the true implementation of a MLP layer and contains matrix for both weights and biases, based on the input dimension and the number of units contained in this layer. The input dimensions of each layer, except from the first one, are retrieved from the previous dense layer. The weight and biases matrices are initialized using different initialization methods and there are different options from which to choose. To the internal propagation of the input through the layer an activation function is applied to the final computational result. Each layer has the possibility to implement some kinds of regularization independently from each other.

2.1.2. *Dropout Layer*

On the other hand the dropout layer is a fake layer that only changes the input of the successive dense layer zeroing some random inputs. It can be initialized with an user selected dropout-ratio and with a specific seed for the random generator. This zeroing process is only applied to the feed-forward propagation.

2.2. Biases and Weights Initializers

In order to prevent layer activation output to explode or vanish some strategies to initialize both weights and biases were implemented. Each dense layer can chose between different strategies and apply different matrices initialization heuristics. Some of the implemented heuristics are the ***constant*** initialization, the ***glorot_uniform, glorot_normal*** and the ***he_normal, he_uniform***.

---

* Implementation of a Simple Neural Network model in Python.

† Also at Computer Science Department, UNIPI University.; g.antonioli3@studenti.unipi.it

2.3. Model

This class collects all the parts used to run the simulator and compiles the simulator with all the required parameters. If some of these are not given, it sets them to some default. This class also implements a fitting method, to compute the training of the network, an evaluating method, to determine the quality of the model and a predicting method, to use what the simulator learned to perform inference.

2.4. Activation Functions

Many different activation functions were implemented in order to have a different linear and non-linear transformations applicable to the feed-forward process. Some of the most relevant ones are the ***linear***, the ***sigmoid*** and the ***ReLU.*** Different functions work better with different aspects of different issues so different combinations were tried in the experimental part.

2.5. Loss Functions

Loss function are used by the simulator to measure the distance between the predicted value of the net and the target values of the labeled training set. These distance functions have been implemented in their normal form and in their derivative form in order to be used even during the back-propagation phase. To do this a *Function* class was crated and specialized in an *DerivableFunction* class in order to access to the needed version of the function at the needed time.

2.6. Regularizers

Different regularization techniques were implemented in order to keep the overfitting phenomenon under control. Bot the ***Lasso regression*** and the ***Ridge regression*** were implemented as regularization functions to be given to each layer with a specific parameter as the penalty term.

Another regularization technique implemented is the ***Early Stopping*** that automatically stops the training based on both the monitored value (e.g. loss or validation loss) and the monitoring mode:

1. **growth mode**: if the monitored value grows in a patience range (for n times consecutively w.r.t. the t-n+1th value) then the training is stopped;
2. **absolute growth mode**: if the last monitored value is bigger than the previous one then the training is stopped;
3. **stationary mode**: if the monitored value doesn't change over a tolerance value in a patience range (for n times consecutively w.r.t. the t-n+1th value) then the training is stopped.

The patience value and the tolerance value can be defined by the user but already have a default value.

2.7. Optimizer

Three different optimizers have been implemented during the project's development. These are:

- Stochastic Gradient Descent (***SGD***) with a ***learning rate*** and ***momentum*** parameter. Also the ***Nesterov Accelerated Momentum*** method has been implemented and it can be toggled by a Boolean parameter.
- Root Mean Square Propagation (***RMSProp***) with a ***learning rate***, ***momentum*** and ***rho*** parameter. This algorithm advantage is that it divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.
- Adaptive Moment Estimation (***Adam***) with a ***learning rate***, ***momentum***, ***beta1***, ***beta2*** and ***epsilon*** parameter. *T*his algorithm is for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

All the optimizers have a method called `optimize` that cycles on the given epochs and check if the early stopping regularization is used. Then based on the batch_size it splits the training data-set into $\frac{data\_set}{batch\_size}$ subsets and cycles on every subset. For each subset it propagates forward every input of the batch it accumulates the gradient and the depending on the optimizer, it applies different regularization techniques. Successively the optimizer updates the biases and weights matrices of each dense layer. Once this is done, then it proceeds, if specified in the parameters, to the evaluation of the model using a validation data-set. Finally it reports the results of all the epochs based on the specified loss and metric, of the training phase and of the validation phase (if a validation data-set was fed to the model).

2.8. Model Selection

In order to perform model selection different types of validation techniques were developed. A ***Simple Holdout*** method was implemented in order to divide the given data-set into training_set, validation_set and test_set. A ***KFold*** method was also implemented, which divides the data-set into K different subset and iterates over them in order to use different partitions as different training sets and a single partitions, different for all iterations, to validate the constructed training set.

A Hyper-parameter selection method such as ***Grid Search*** was implemented to give the user a way to fine tune the models. This algorithm uses one of the two above illustrated data-sets splitters and if the KFold method is used the final scores of the loss and validation are averaged over all the K iterations. This method stores the best parameter configuration and the best evaluator based on the validation metrics result

3. EXPERIMENTS

In this section the experimental part of the project will be introduced and illustrated to give an insight of how the simulator performs on two different data-sets.

3.1. MONK

The Monk data-set is used as a baseline to evaluate the correct functioning of the simulator.

3.1.1. *Dataset Introduction*

The MONK's problem were the basis of a first international comparison of learning algorithms. It is made of 6 different attributes that were one-hot encoded and two different possible classifications. This data-set so was used to determine the classification power of the simulator and due to the fact that these data-sets are guaranteed to obtain approximately a 100% accuracy, this information was exploited to see if the simulator performed correctly the classification.

3.1.2. *Validation*

The MONK's data-sets were tested using a grid search with using KFold with a value of K=3. In this way 66% of the data-set was used for training and 33% was used for validation. Previously a 10% of the original data-set was excluded to be the internal test-set.

3.2. CUP

The CUP data-set represents a regression problem and it is made of 10 different attributes and two different scores. Two different models were implemented to simulate the model the two different regressions.

3.2.1. *Data-set Introduction*

This data-set for us students is unknown. No information about what the data is used for and how it was collected was given so the approach to it is totally blind.

3.2.2. *Validation*

The CUP data-set was tested using a grid search with using KFold with a value of K=3. In this way 66% of the data-set was used for training and 33% was used for validation. A previous 10% of the

4. RESULTS

The principal aim of this project was to implement a simulator system that could autonomously learn from given inputs and due to a less efficient implementation w.r.t off the shelf products, the training and fine-tuning required an important amount of time. So due too the many developed parts the fine-tuning and testing parts results in being less deep and doesn't cover all the developed parts for simplicity.

4.1. MONK Results

This data-set, as it has been already said, is used for a baseline comparison of the simulator and to guarantee the effectiveness of it.

Different configurations of learning_rate,moments ,activation functions, neuron configurations and optimizers were tested.

Best results for each data-set will be reported below.

{'opt': 'sgd', 'mom': 0.7, 'lr': 0.1, 'metrics': 'binary', 'loss': 'squared', 'epochs': 100, 'batch-size': 1}

Parameter	Value	Parameter	Value
n_units_1	4	n_units2	4
act_1	tanh	act_2	sigmoid
learning rate	0.1	momentum	0.7
optimizer	SGD	batch size	1
epochs	100		

Table I. Best Model configuration for Monk1

Parameter	Value	Parameter	Value
n_units_1	4	n_units2	1
act_1	tanh	act_2	sigmoid
learning rate	0.1	momentum	0.7
optimizer	SGD	batch size	1
epochs	100		

Table II. Best Model configuration for Monk2

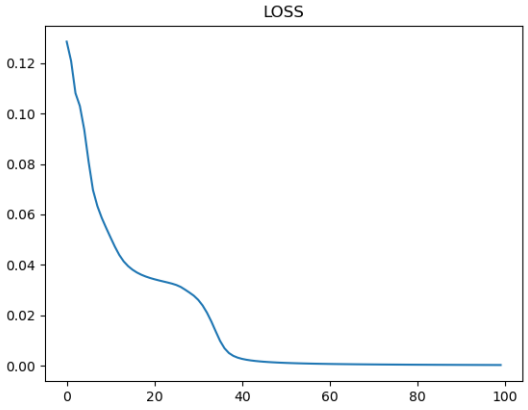

Figure 1. Best model Monk 1 Loss (MSE)

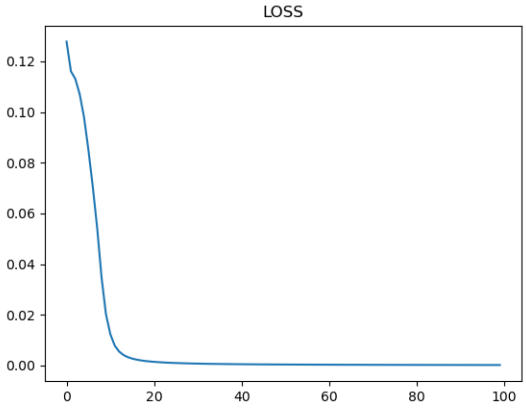

Figure 3. Best model Monk 2 Loss (MSE)

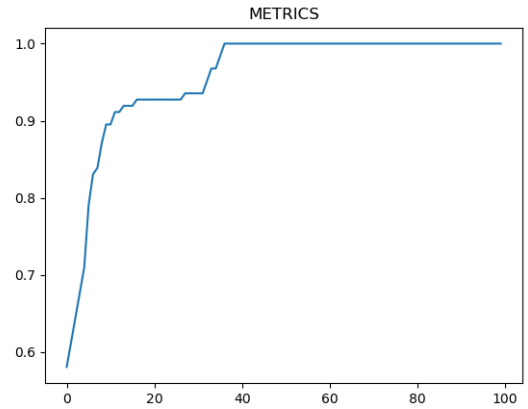

Figure 2. Best model Monk 1 Accuracy

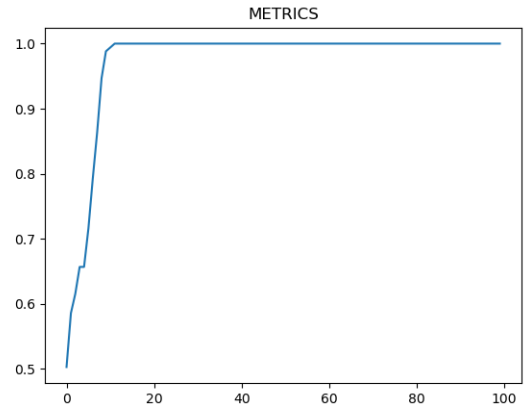

Figure 4. Best model Monk 2 Accuracy

Parameter	Value	Parameter	Value
n_units_1	4	n_units2	1
act_1	tanh	act_2	sigmoid
learning rate	0.1	momentum	0.7
optimizer	SGD	batch size	1
epochs	100		

Table III. Best Model configuration for Monk3

Data-set	Loss TR (MSE)	Loss Ts (MSE)	Accuracy TR (%)	Accuracy TS(%)
monk 1	0.000283	0.000418	100%	100%
monk 2	0.0001453	5.887e-05	100%	100%
monk 3	0.00439	0.0198058	99.18%	94.21%

Table IV. Loss and Accuracy for Monk Datasets

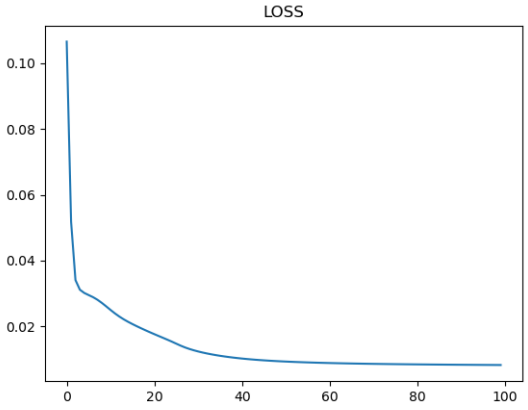

Figure 5. Best model Monk 3 Loss (MSE)

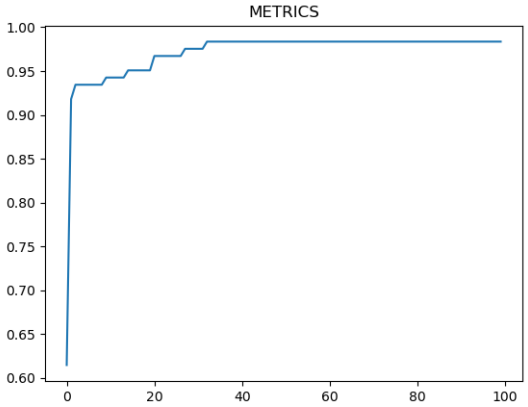

Figure 6. Best model Monk 3 Accuracy

4.2. CUP Results

Two different models were implemented for the CUP data-set, due to the nature of it. One model was trained to predict the first target and the second was trained to predict the second target.

Model	Training set MEE	Validation set MEE	Internal Test set MEE
model 1	2.79305	2.20227	5.2965
model 2	Todo	todo	todo
model 3	todo	todo	todo-

Table V. MEE results for CUP

The parameters used for the best model are:

Parameter	Value	Parameter	Value
n_units_1	10	n_units2	4
n_units_3	1	act_1	sigmoid
act_2	relu	act_3	linear
regularizer	L2	learning rate	0.01
regularizer param	0.001	dropout rate	10%
optimizer	RMSProp	momentum	0.1
epochs	50	batch size	10

Table VI. Best Model configuration for CUP

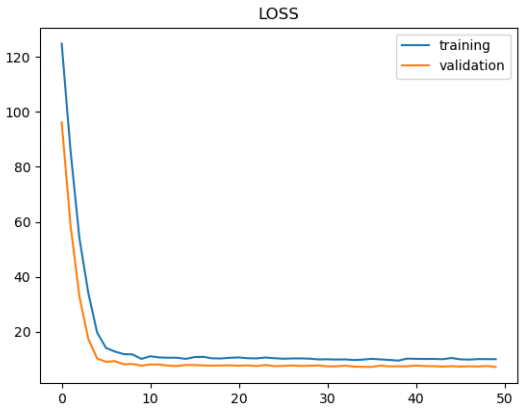

Figure 7. Best CUP 2 Model Loss

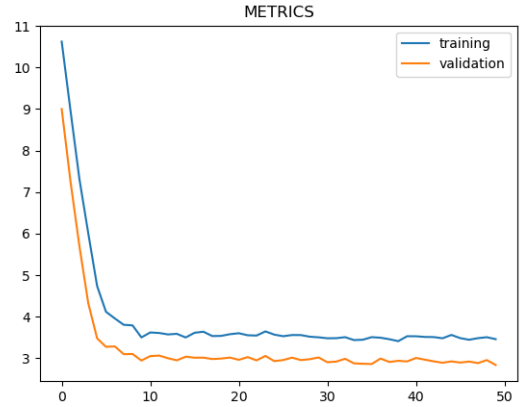

Figure 8. Best CUP 2 Model Accuracy

Model	Training set MEE	Validation set MEE	Internal Test set MEE
model 1	3.4621534	2.840094	2.78521
model 2	3.6036494	2.911343	
model 3	3.5027666	2.921879	

Table VII. MEE results for CUP for the second label

5. CONCLUSIONS

This project was really fun to do and it game me the opportunity to explore more in depth the course's contents as well as having a practical point of view of what I saw during the lessons.

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