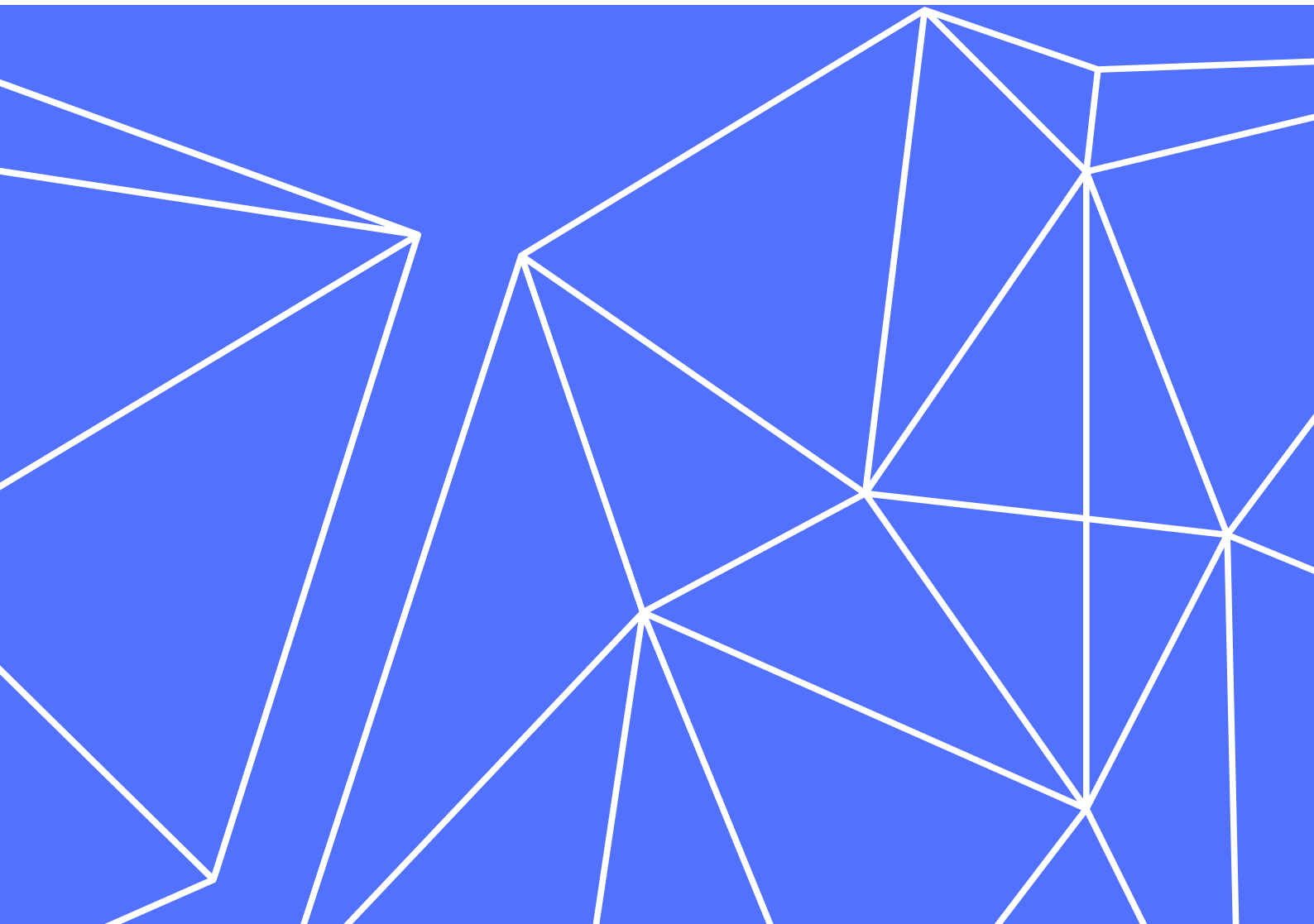
OCTOBER 2022

Report of the CI

CI - Exercice of the LAB 2

MADE BY :

Hajar LACHHEB

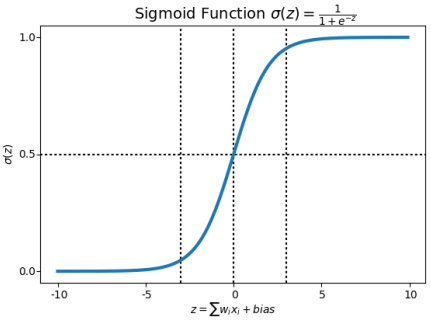


1. Choosing and Building the Models

In the description of the work, we needed to define two different models. Each model was built and defined taking in consideration the characteristics bellow :

- **Model 1 :** The first model need to use logsigmoid for the hidden layer, logsigmoid for the output layer and mean squared error (MSE).

$z = \sum_{i=1}^{m} w_i x_i + bias$
Sigmoid Function is: $\sigma(z) = \frac{1}{1+e^{-z}}$



- **Model 2 :** The second model need to use logsigmoid for the hidden layer, softmax for the output layer and cross-entropy.

In fact, we could also mention that the difference between cross-entropy and MSE, is actually that the cross-entropy measures the performance of our classification model whose output is a probability value between 0 and 1. It is actually preferred for classification, while mean squared error (MSE) is one of the best choices when it comes to regression. So we will take this into consideration while doing our tests.
We defined two files, each file contain the code of the model adequat to the descriptions in the exercice requirement.
We will create two other files to perform a grid search algorithm to find the best parameters and then use them to run the respective model with these parameters.

To optimize our models, we will be using Gradient Descent with Momentum and Adaptive Learning Rate. We could use the Adam optimizer as well as an an Adaptive LR so as to computes individual learning rates for different parameters.

2. Describing the different configurations

To start with, we applied a hyperparameter grid search algorithm to get the best parameters for each of the models. This is why we have two different files, each one is proper to each model.
The objective behind the hyperparameter grid search is to be able to find the optimal values of each of the following parameters : Number of Epochs, Learning Rate and Momentum.
To be able to do so, we will train and use the validation set, and this by trying different parameter values and checking if the accuracy resulted is good or no.
This was built in a function in both of the Lab_Exercice files, we will be using different If boucles, to go over all the parameter's values and then compare the resulted accuracy. Whenever the accuracy resulted is better than the previous one, the parameters used are counted as the best parameters.

The grid search algorithms for our case will take a lot of time to be computed.

So as to find the best results and best parameters, we configured the number of tests to 10, so our configuration will be executed 10 times.
The total combinations will be 90 combinations. It's definitely taking a lot of time.

- **Model 1** : The best parameters we had are :

Epochs Number	Learning Rate	Momentum	Accuracy Achieved
500	**0.001**	**0.30**	**28%**

- **Model 2** : The best parameters we had are :

Epochs Number	Learning Rate	Momentum	Accuracy Achieved
500	**0.001**	**0.30**	**28%**

3. Results of our trainings

- **Model 2 :** First we will start by sharing the result of our training. It took us approximately more than 3 hours of training.

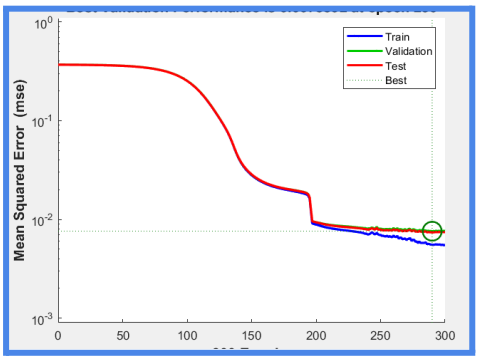
We are using the best set of parameters we got earlier and that resulted from the grid search hyperparameter algorithm.

```
Testing using Hidden units=200 TrainDataRatio=0.800000 ValDataRatio=0.100000 TestDataRatio=0.100000
Obtained accuracies: Train=0.318120 Validation=0.313495 Test=0.294810
Testing using Hidden units=200 TrainDataRatio=0.400000 ValDataRatio=0.200000 TestDataRatio=0.400000
Obtained accuracies: Train=0.330787 Validation=0.295905 Test=0.305767
Testing using Hidden units=200 TrainDataRatio=0.100000 ValDataRatio=0.100000 TestDataRatio=0.800000
Obtained accuracies: Train=0.408535 Validation=0.302884 Test=0.310278
Testing using Hidden units=500 TrainDataRatio=0.800000 ValDataRatio=0.100000 TestDataRatio=0.100000
Obtained accuracies: Train=0.303099 Validation=0.288812 Test=0.300346
Testing using Hidden units=500 TrainDataRatio=0.400000 ValDataRatio=0.200000 TestDataRatio=0.400000
Obtained accuracies: Train=0.282127 Validation=0.263091 Test=0.259285
Testing using Hidden units=500 TrainDataRatio=0.100000 ValDataRatio=0.100000 TestDataRatio=0.800000
```

Hidden units	TrainDataRatio	ValDataRatio	TestDataRatio	Train Accuracy
50	0.8	0.1	0.1	18%
50	0.4	0.2	0.4	25%
50	0.1	0.1	0.8	24%
200	0.8	0.1	0.1	31%
200	0.4	0.2	0.4	33%
200	**0.1**	**0.1**	**0.8**	**40%**
500	0.8	0.1	0.1	30%
500	0.4	0.2	0.4	28%
500	**0.1**	**0.1**	**0.8**	**40%**

We notice that when the performance is better when the set is defined as 0.1/0.1/0.8. We get an accuracy that reaches 40%.

It's maybe indicating that our model is indeed overfitting which is not a good sign for our training.



If we observe the performance plot, we will see that around the 200 epoch, the MSE of the training set is decreasing while the validation and the test one are remaining stable, which is an overfitting issue.

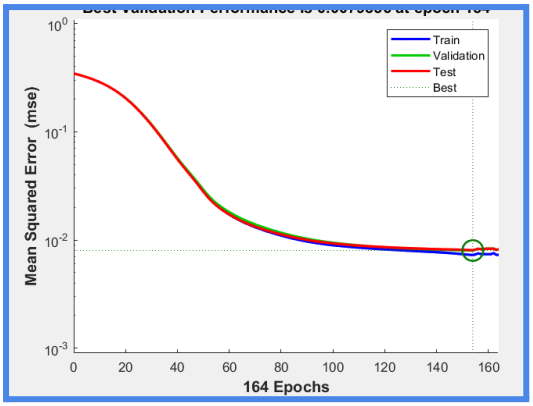
- **Model 1 :** Second we will proceed to share the result of our second model training. It took us approximately more than 3 hours of training as well.

We are using the best set of parameters we got earlier and that resulted from the grid search hyperparameter algorithm.

```
Testing using Hidden units=50 TrainDataRatio=0.800000 ValDataRatio=0.100000 TestDataRatio=0.100000
Obtained accuracies: Train=0.208693 Validation=0.207728 Test=0.207958
Testing using Hidden units=50 TrainDataRatio=0.400000 ValDataRatio=0.200000 TestDataRatio=0.400000
Obtained accuracies: Train=0.266878 Validation=0.264187 Test=0.253633
Testing using Hidden units=50 TrainDataRatio=0.100000 ValDataRatio=0.100000 TestDataRatio=0.800000
Obtained accuracies: Train=0.294118 Validation=0.246021 Test=0.245596
Testing using Hidden units=200 TrainDataRatio=0.800000 ValDataRatio=0.100000 TestDataRatio=0.100000
Obtained accuracies: Train=0.296641 Validation=0.299885 Test=0.282814
Testing using Hidden units=200 TrainDataRatio=0.400000 ValDataRatio=0.200000 TestDataRatio=0.400000
Obtained accuracies: Train=0.259700 Validation=0.239562 Test=0.243973
```

Hidden units	TrainDataRatio	ValDataRatio	TestDataRatio	Train Accuracy
50	0.8	0.1	0.1	20%
50	0.4	0.2	0.4	26%
50	0.1	0.1	0.8	29%
200	0.8	0.1	0.1	29%
200	0.4	0.2	0.4	25%
200	**0.1**	**0.1**	**0.8**	**42%**
500	0.8	0.1	0.1	29%
500	0.4	0.2	0.4	33%
500	**0.1**	**0.1**	**0.8**	**34%**

We notice that when the performance is better when the set is defined as 0.1/0.1/0.8. We get an accuracy that reaches 40% and 34%.
It's maybe indicating that our model is slighlty overfitting but not like the second model.



If we observe the performance plot, we will see that around the 150 epochs, the MSE of the training set is decreasing a bit while the validation and the test one are remaining stable, which is a slighlty overfitting issue.

➔ **The difference between the performance and results of Model 1 and Model 2 is not too much. Actually, there is a difference when it comes to the accuracy, but it's slightly noticable. We can indeed choose the second model Model 2 since it performs better on our dataset. And probably, this can be explained by the architecture LOGSIG + SOFTMAX that performs better with multi classification type of problems.**

4. Conclusions

In this Lab Exercice, we performed a hyperparameter grid search to choose the best parameters to use for out training. We are indeed working with two multilayer perceptron models. Experimenting with two different built in models made us pay close attention to the fact that the configuration of theses models, the training and the different parameters can definitely affect the overal results.

One of the best known effects of the parameter change is the close relation between overfitting and training set size. This help us get to the conclusion that the general properties of neural networks are highly data dependent. Another conclusion we got is definitely the fact that the number of neurons does not always improve the result, because the architecture has its limits.
More to be added to our conclusion is that there is no one-size-fits-all configuration for neural network models, they are complex systems that require specific configurations. And it requires time and ressource to be able to find the best configuration that fits our data.

In addition, we also notice that the overall accuracy is not the best we can get even while using the best parameters and two different built in models. This may be caused by the type of our dataset. Maybe the neural network dont perform that good when the data is flatened which is definitely our case.

Maybe the approach that used could work and perform better with a set of images, where we have to compare the pixels and how similar they are like a general image classification problem.
Let's not forget the computational effort and time that the training took us. By executing the configuration time 10 times, we can get more results indeed but it's time and ressource consuming.