Machine Learning
Coursework 1
Mattia Gennaro

1)The dataset

To create the dataset I used two different methods, defined in the *generateMatrices()* and *generateMatricesDifferent()*.
The first method uses the np.random.uniform(), wich allows to generate **n** random points in a chosen range, which is what we need to create two rectangles.
The second method uses the np.random.multivariate_normal(), that creates **n** points given the mean and the covariance.

I created the matrices as follows:

```
#intialise the matrices
matrix = generateMatrices(2,5,1,4,1,500)
matrix_2 = generateMatrices(1,3,-5,-1,2,500)
matrix_3 = generateMatricesDifferent(-2,-3,0.5,0,0,3,3,500)
matrix_4 = generateMatricesDifferent(-4,-1,3,0.5,0.5,0.5,4,500)
```

After rotating the first two, I plot them in the graph, which gives:

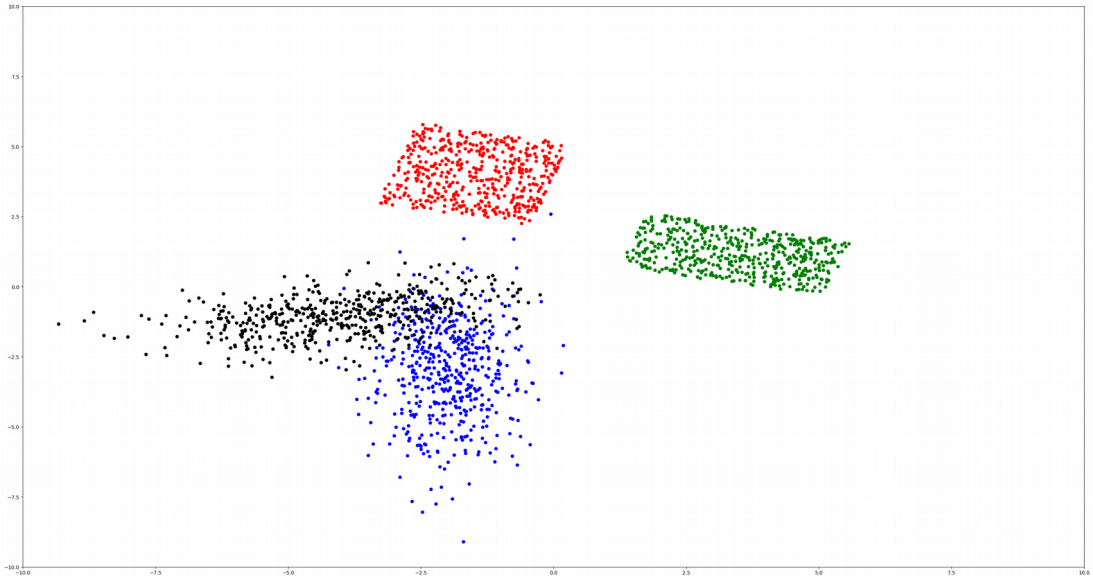
```
rotated1 = rotatedMatrix(matrix, 75)
rotated2 = rotatedMatrix(matrix_2, 75)

plt.axis([-10, 10, -10, 10])

plt.plot(rotated1[0], rotated1[1], 'or')
plt.plot(rotated2[0], rotated2[1], 'og')
plt.plot(matrix_3[0], matrix_3[1], 'ob')
plt.plot(matrix_4[0], matrix_4[1], 'ok')

plt.show()
```

Graph:



I used the values [2,5] and [1,4] for the first matrix and [1,3], [-5,1] for the second one such that they distance is at most two.
For the third and fourth one I used parameters such that they overlap.

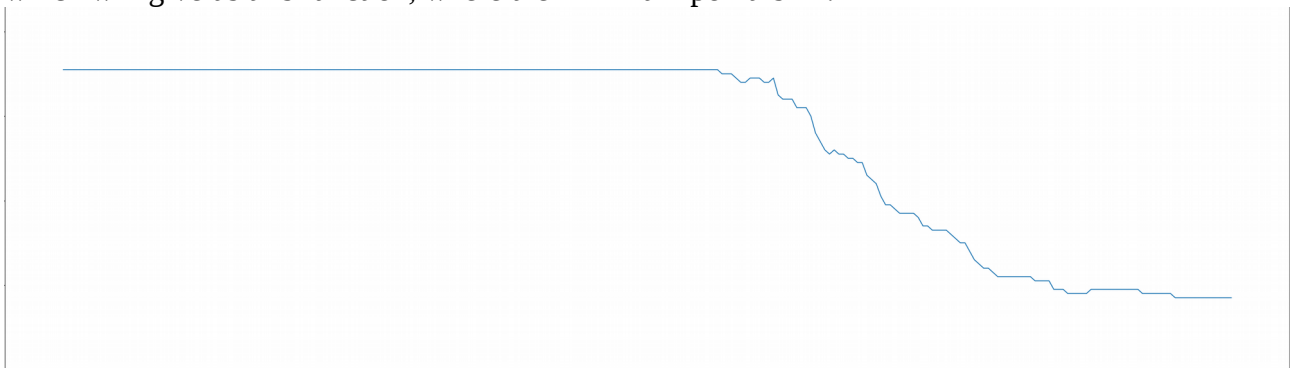
N.B: I commented the code that plots the initial dataset in *script()*, to avoid the code from stopping. To see the it , just uncomment the code.

2) Multi Layer Perceptron

Experimenting with network size: To experiment, I chose eta to be **0.1**.
The first experiment is done with 2 hidden layers:

```
#initialising the values
x = training[0,:2]
h = 2
eta = 0.1
```

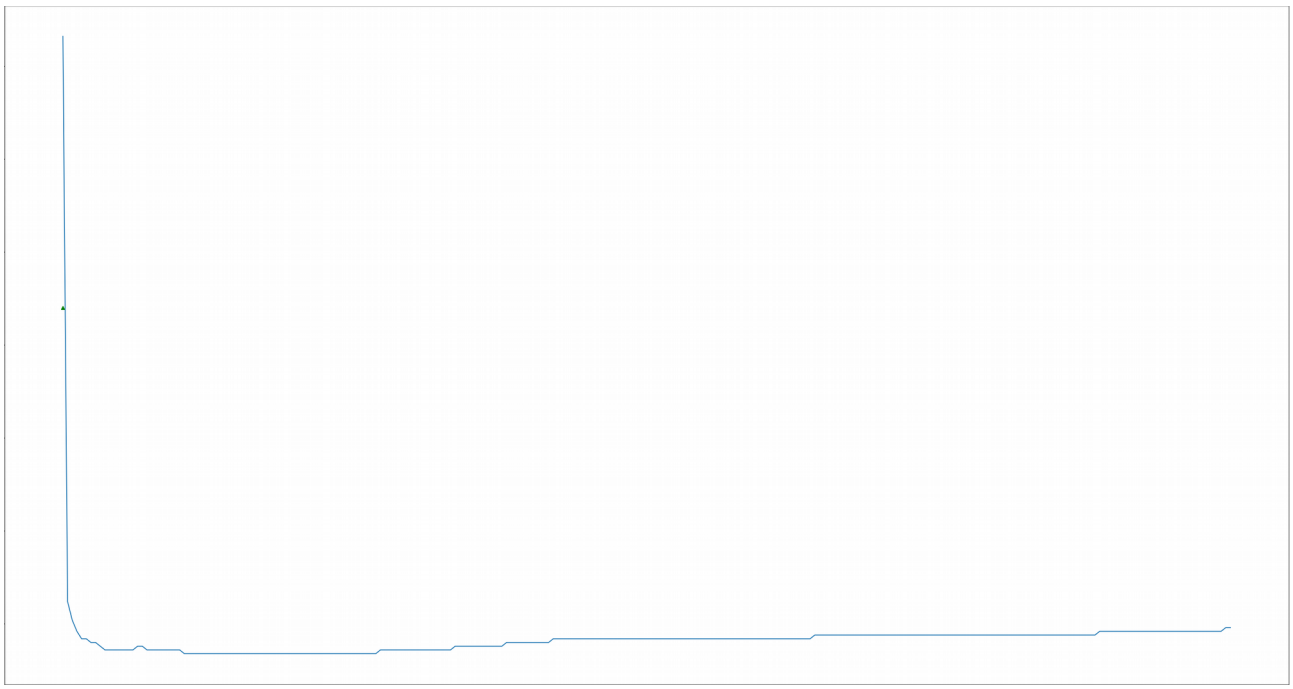
which will give us this function, where the minimum point is **77**:



The second experiment is done with 4 hidden layers:

```
#initialising the values
x = training[0,:2]
h = 4
eta = 0.1
```

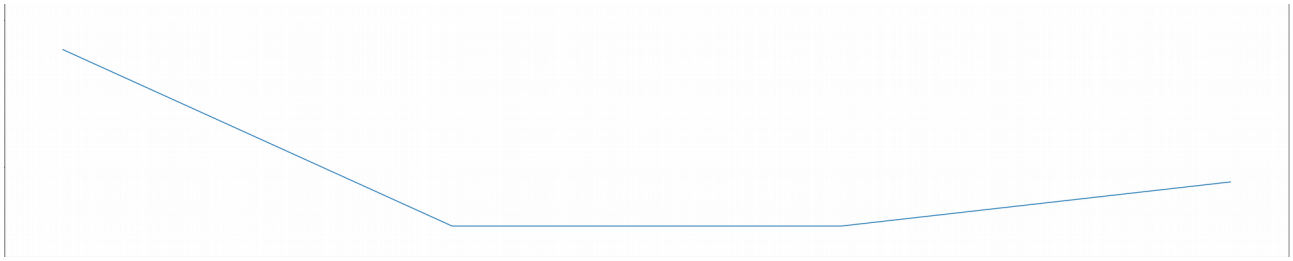
Which will give us this funcion, where the minimum point is **92**:



The third and last experiment is done with 8 hidden layers:

```
#initialising the values
x = training[0,:2]
h = 8
eta = 0.1
```

Which will give us the function, where the minimum point is **78** :



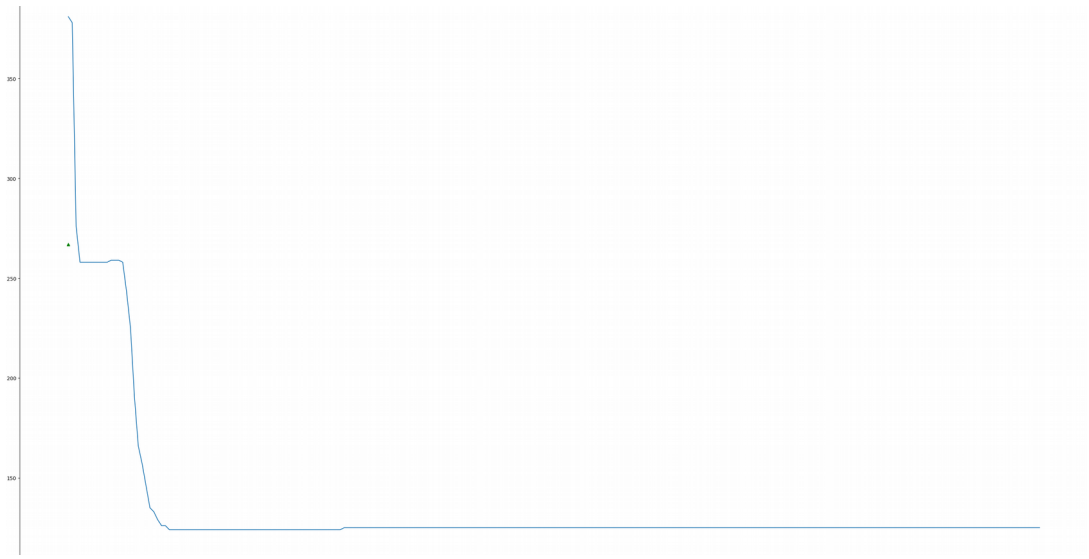
As we can notice, we got our best MLP on the first experiment, since the error value is the smallest among all, that is **77.**

Using my best MLP, I will now change the learning rate, which is eta.

The first experiment will be done with a very small value for eta: **0.01**.

```
x = training[0,:2]
h = 2
eta = 0.01
```

Which will give us this function, where the minimum error is: **124**:



The second experiment will be done with a larger value for eta, which is **0.09:**

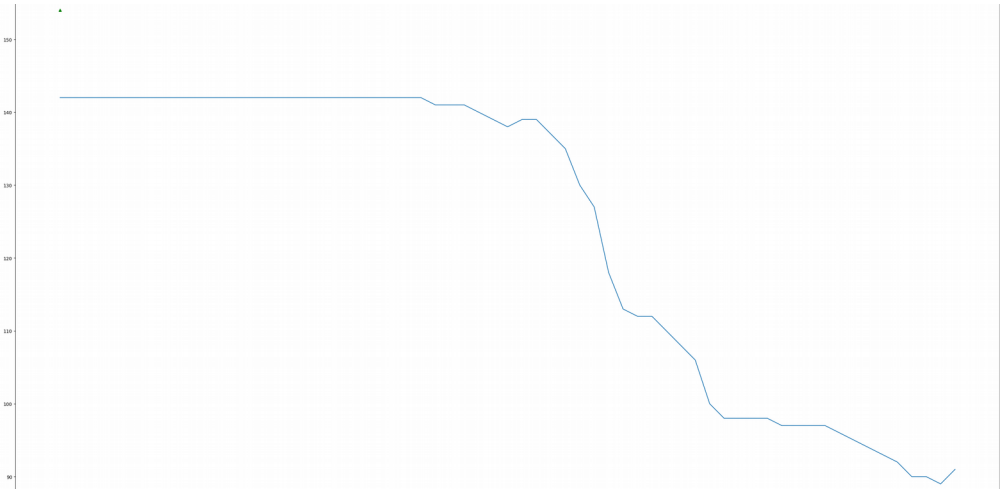
```
x = training[0,:2]
h = 2
eta = 0.09
```

Which will give us this function, where the minimum error is **133**:

The last test will be done with the maximum value for eta, which is **0.2:**

```
x = training[0,:2]
h = 2
eta = 0.2
```

Which will give us this function, where the minimum error is **89**:



Its noticeble that an high learning rate increases the chances to get a lower error, with a significant smaller number of iterations.
For the first and second experiment, 250 iterations where necessary, while for the last one, only 62.
Notice that the green triangle present in the graphs is the error predicted by the test set.

3) Plotting of the test set