בס"ד

PROJECT 1

Parts C and D

Taliya shitreet 314855099

Renana Rimon 207616830

Talia Meizlish 207883430

Esther Binnes 211328109

~All members of the group are on evening class~

12.05.2022

Backpropagation Algorithm

In machine learning, backpropagation is an extremely useful algorithm for training feedforward neural networks. You can read more about it here: https://en.wikipedia.org/wiki/Backpropagation

Part C:

In this part, we used **MLPClassifier** taken from 'sklearn.neural_network' library,

MLPClassifier is an implementation of the **Backpropagation** Algorithm.

DataSet B:

Train: 1000 points

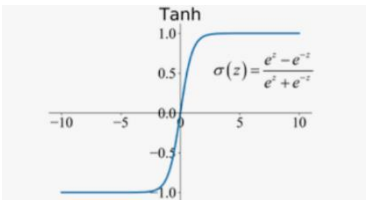
Test: 250 points

To test the algorithm properly, we made sure that at least 50 points would fall within the circle: $4 \leq x^2 + y^2 \leq 9$

```
[36]: def createDataSetB(num: int) -> (np.ndarray, np.ndarray):
          dataSet = np.zeros((num, 3))
          num_in=0
          while num_in<50:
              m = random.randint(-10000, 10000)
              n = random.randint(-10000, 10000)
              x = m/ 100
              y = n/100
              if 4 <= ((x ** 2) + (y ** 2)) <= 9:
                  dataSet[num_in, 0] = x
                  dataSet[num_in, 1] = y
                  dataSet[num_in, 2] = 1
                  num_in+=1
          for i in range(num_in,num):
              m = random.randint(-10000, 10000)
              n = random.randint(-10000, 10000)
              x = m/ 100
              y = n/100
              dataSet[i, 0] = x
              dataSet[i, 1] = y
              dataSet[i, 2] = 1 if 4 <= ((x ** 2) + (y ** 2)) <= 9 else -1

          np.random.shuffle(dataSet)
          return dataSet[:, 0:2], dataSet[:, 2]
```

MLPClassifier's Parameters:

- *activation function*: **'tanh'**, we used this function because its values are [-1,1], same as our Adaline algorithm.
  (Instead of 'sigmoid' that give values [0,1])



- *hidden layer sizes*: (8,2).

last layer with 2 neurons to get the appropriate input for Adaline –
x: neuron 1, y: neuron 2.

MLPClassifier fit & predict

```
[38]: X_trainB, y_trainB  = createDataSetB(1000)
      X_testB, y_testB = createDataSetB(250)
      clfB = MLPClassifier(hidden_layer_sizes=(8,2),activation="tanh",random_state=1,max_iter=240).fit(X_trainB,y_trainB)

      y_predB =clfB.predict(X_trainB)
      print("B- ",clfB.score(X_testB, y_testB))

      B-  0.908
```

Score: 0.908

~BackPropagation: 0.908 **VS** Adaline: 0.48~

we can see a high score in contrast to part B- there we got a score of 0.48 with Adaline algorithm only. The reason for it is that the MLT can also classify non-linear data due to the layers of neurons, back and forward propagation calculate.

DataSet A:

Train: 1000 points

Test: 250 points

Linearly classified Data (n/100)

```
def createDataSetA(num: int) -> (np.ndarray, np.ndarray):
    dataSet = np.ndarray((num, 3))
    for i in range(num):
        m = random.randint(-10000, 10000)
        n = random.randint(-10000, 10000)
        dataSet[i, 0] = m / 100
        dataSet[i, 1] = n / 100
        dataSet[i, 2] = 1 if n / 100 > 1 else -1
    return dataSet[:, 0:2], dataSet[:, 2]
```

```
X_trainA, y_trainA  = createDataSetA(1000)
X_testA, y_testA = createDataSetA(250)
clfA = MLPClassifier(hidden_layer_sizes=(8,2),activation="tanh",random_state=1,max_iter=240).fit(X_trainA,y_trainA)

y_predB =clfA.predict(X_trainA)
print("A- ",clfA.score(X_testA, y_testA))

A-  0.996
```

Score : 0.996

~BackPropagation: 0.996 **VS** Adaline: 0.994~

Because the data is linear, there is no significant difference in accuracy percentages.

geometric diagram

For part C we were asked to show a **geometric diagram** in terms of the inputs of the training set for the output of each neuron separately in the neural network as well as for the output neuron.

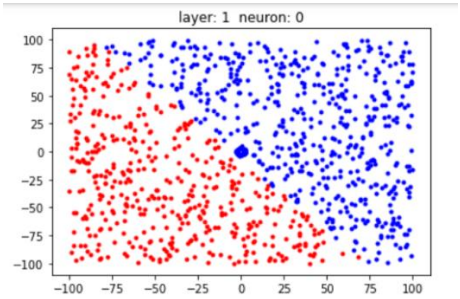
Below you can see the code for plotting the geometric diagram:

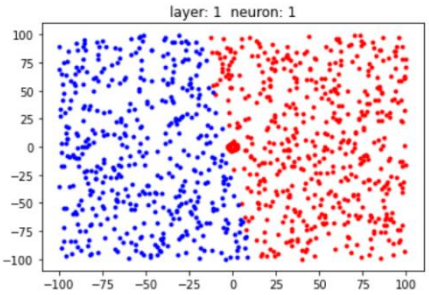
- *Result_neuron()*:
  Input: list of neurons, classifier (MLP in this case), the desired layer number, and index of the wanted neuron from the list input.
  Output: if this neuron classification is 1 or -1 based on the threshold.
- *Cal_result_neuron()*:
  Input: dataset X , classifier (MLP in this case) , the desired layer number, and index of the wanted neuron from the list input.
  Output: two lists – one for all points were classified as 1 and the other for the points were classified as -1.

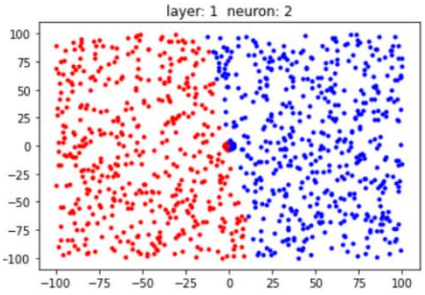
```python
def result_neuron(neuron_list, clf, layer, index:int):
    neuron_list = result_layer(neuron_list, clf, layer)
    if neuron_list[index]<0:
        val= -1
    else:
        val= 1
    return val

def cal_result_neuron(X, clf, layer, index:int):
    valmin= []
    valmax= []
    for i in range(len(X)):
        v= result_neuron(X[i], clf, layer, index)
        if v==1:
            valmax.append(X[i])
        else:
            valmin.append(X[i])
    return np.asarray(valmax),np.asarray(valmin)
```

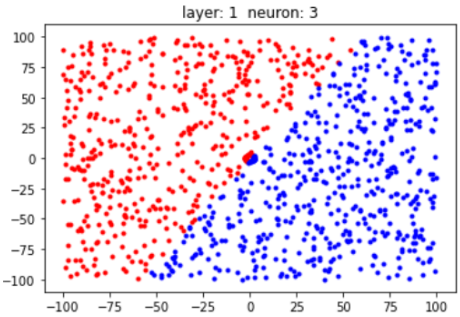
Plotting the geometric diagram for each neuron:

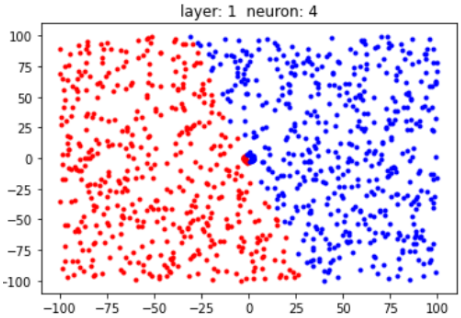
Layer 1:

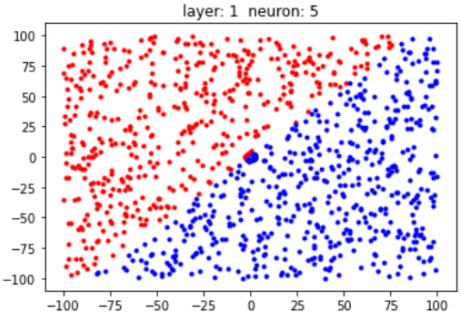


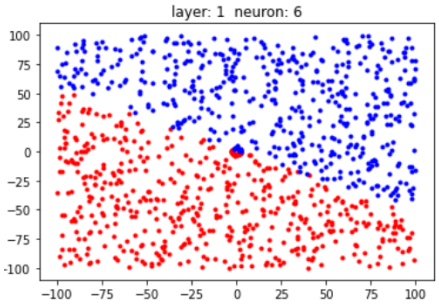


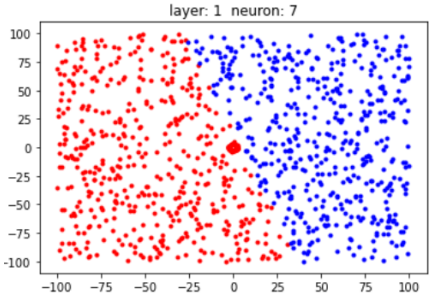




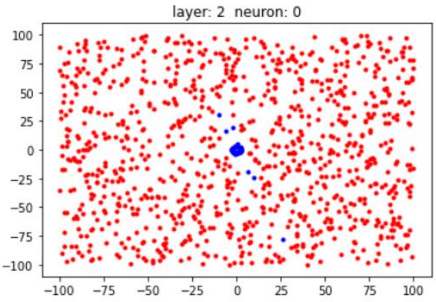


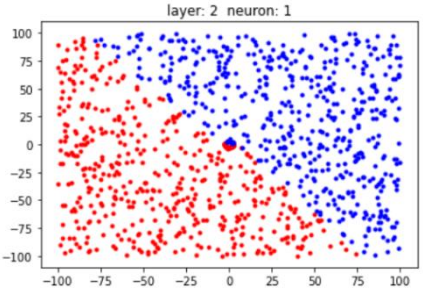






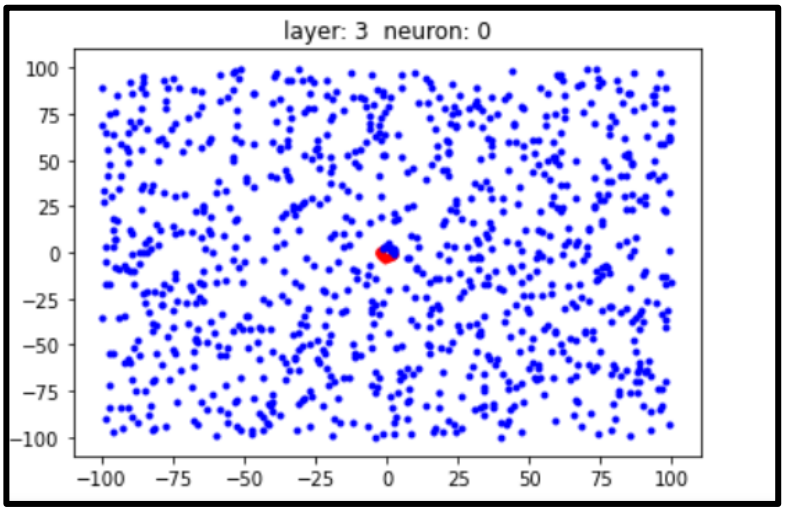
Layer 2





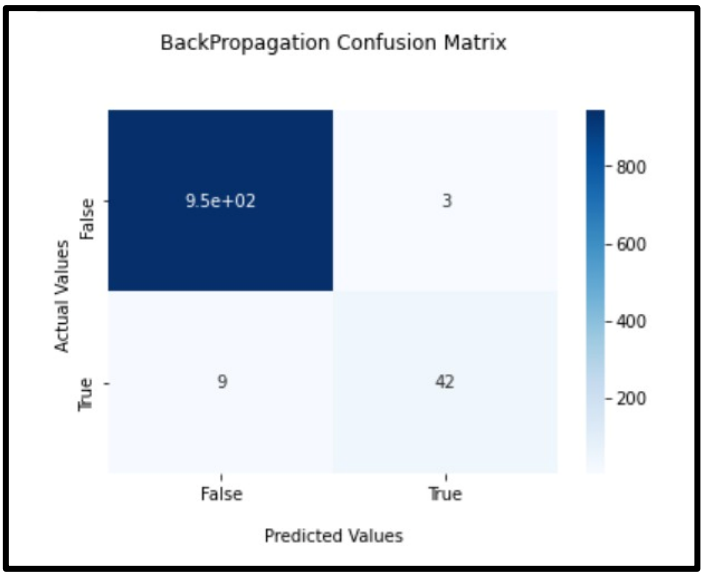
Layer 3:

This layer it's not a hidden layer - it's the classification layer so we can see the points that "fell down" into the radius. It seems like very tiny circle because of the small radius $4 \leq x^2 + y^2 \leq 9$ is the red points on the image.



Confusion Matrix:

We can see that only 9 'true' points were classified as false and only 3 'false' point as true(!)



Part D:

In this part we were asked for use the trained neurons from the next to last level of Part C as input and only an Adaline for the output.

code below:

- *htan*(): calculate the tanh function
- *result_layer()*: this function returns list of neurons of required layer by calculate the neuron's value of each neuron in this layer- this function actually "mimics" the MLP algorithm calculations in the forward propagation.

```
[53]: def htan(x):
          return (np.exp(x) - np.exp(-x))/(np.exp(x) + np.exp(-x))

      def htan_list(data):
          for i in range(len(data)):
              data[i]= htan(data[i])
          return data

      def result_layer (neuron_list, clf, layer: int):

          for l in range(layer):
              w= clf.coefs_[l].T
              neuron_list = (neuron_list*w).sum(axis=1)  # Double the weights
              neuron_list = neuron_list + clf.intercepts_[l]  #add the bais
              neuron_list = htan_list(neuron_list)
          if layer==3:
              if neuron_list[0]>0:
                  neuron_list[0]=1
              else:
                  neuron_list[0]=-1

          return neuron_list
```

We added our implementation of Adaline algorithm from part A and created new X_Train By the function *to_adaline()* above by the output of the last layer of the neural network.

```
def newTrain1(clf, X): #Returns the classification that the function result_layer returns
    X_1 = []
    for i in range(1000):
        res = result_layer(X[i], clf, 3)
        x = res[0]
        X_1.append(x)
    return np.asarray(X_1)

def to_adeline(clf, X, num): #Returns the classification that the function result_layer returns
    X_1 = []
    for i in range(num):
        x,y = result_layer(X[i], clf, 2)
        X_1.append((x,y))
    return np.asarray(X_1)

def adalineBack(X_train, y_train, X_test, y_test):
    adaline = Adaline(learning_rate=0.01, n_iter= 100)
    adaline.fit(X_train, y_train)
    print(adaline.score(X_test, y_test))
```

We can see a high score of 95% accuracy due to the neuron's value that the MLT algorithm has already calculated for " opening credits" for the Adaline.

```
x_ad_train= to_adeline(clfB, X_trainB, 1000)
x_ad_test= to_adeline(clfB, X_testB, 250)

adalineBack(x_ad_train, y_trainB, x_ad_test, y_testB)
```

```
0.952
```

Score: 0.952

~BackPropagation & Adaline: 0.952 **VS** Adaline: 0.48~

As expected, the combination of the 2 algorithms gave a much better result than the Adaline algorithm (in part B). This is because Adaline works on a single neuron, so can only compute linearly classified data.

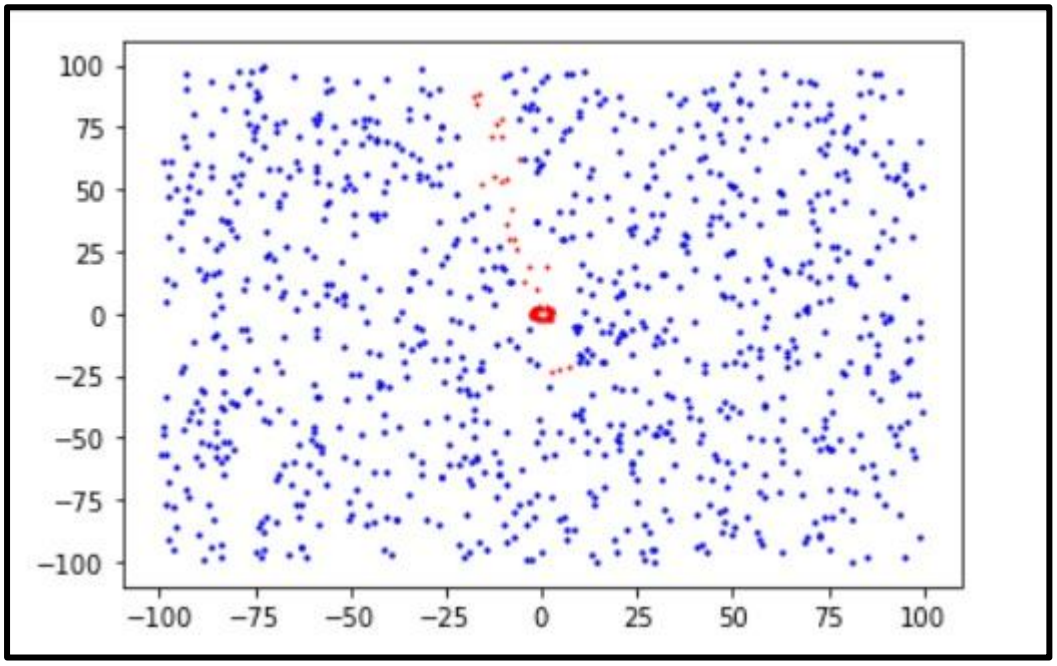
combination of the 2 algorithms:

1. ran the data on **BackPropagation** (neural network)
2. input of Adaline: last layer neurons (the processed data)
3. output: Adaline output

Now, Adaline can compute the non-linear data because it already has processed by the network.

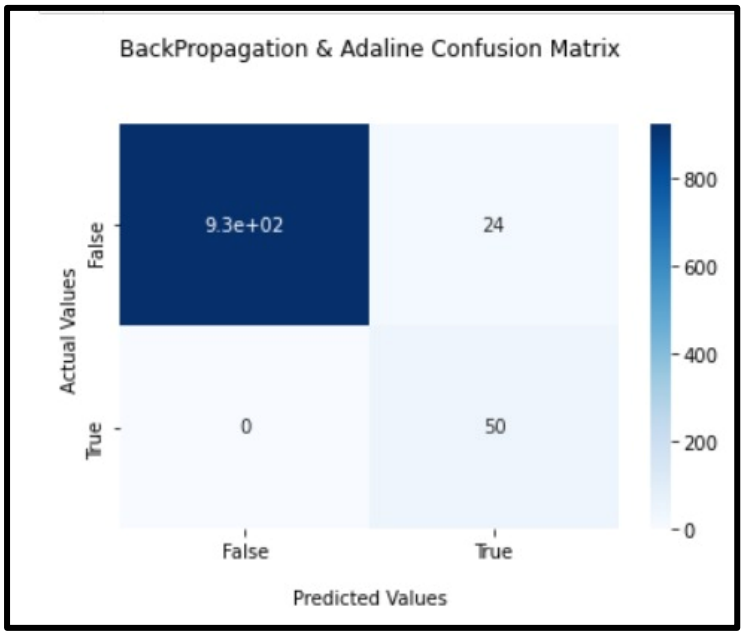
Classification plot:

Below we can see the classification of the combination of MLP & Adaline of the last layer:



Confusion Matrix:

We can see that all "true" points were classified as true(!) and only 24 "false" point as true – we can also see it on the classification plot above.



In conclusion:

You can see the comparison of the prediction for each point, in each of the models: **MLP** and **MLP & Adaline**

	x	y	target	MLP	Adaline
0	77.72	-15.26	-1.0	-1.0	-1.0
1	-25.98	21.41	-1.0	-1.0	-1.0
2	-30.51	-69.68	-1.0	-1.0	-1.0
3	-60.20	16.63	-1.0	-1.0	1.0
4	7.96	-91.27	-1.0	-1.0	-1.0
...	...	...	...	...	...
995	47.95	-59.50	-1.0	-1.0	1.0
996	84.20	7.99	-1.0	-1.0	-1.0
997	93.51	-45.75	-1.0	-1.0	-1.0
998	-98.11	20.81	-1.0	-1.0	1.0
999	7.19	-80.03	-1.0	-1.0	-1.0