**Assignment 2**

**Artificial Neural Network**

**Submitted by**: **Muhmmad Naeem Ur Rehman (17-SE-01)**

**Dated: 25/04/2021**

Part B: Run a code in Python implementing backprop, you can use existing methods but apply this to a real problem (e.g., MNIST classification). Prepare a report summarizing your choice of parameters and working of the backprop algorithms.

**Backpropagation**, aka backward propagation of errors, is an algorithm for supervised learning of artificial neural networks using gradient descent. Given, an artificial neural network and an error function(loss), it calculates the gradient of the error function to update the weights of each layer. Here, this is my attempt to explain how it works with a concrete example Iris Dataset.

**1- Introduction**

**1.1 Dataset**

The data set Iris consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. However, for simplifying task, I removed 50 'Iris virginica' samples to convert my job to a binary classification. Let's have a look at Iris data.  
5.1,3.5,1.4,0.2,Iris-setosa ---->the first sample  
7.0,3.2,4.7,1.4,Iris-versicolor ---->the second sample  
Given the first sample, we got x1=5.1 x2=3.5 x3=4.0 x4=0.2 as 4 features and y='Iris-setosa'(after one\_hot encoding y=1.0).  
For the second sample, x1=7.0 x2=3.2 x3=4.7 x4=1.4 y=0.0.   
Finaly,100 samples should be divided up into training set (80 samples) and test set (20 samples).

**1.2 Network Architectures**

I have used a neural network with four inputs corresponding to four features x1 x2 x3 x4, six hidden neurons, one output neuron. Additionally, the hidden and output neurons will include a bias. I didn't give the output layer an active function, such as sigmoid function, since I want to simplify the derivate processing. Actually, the g(z) in hidden neurons is sigmoid function.

def BackwardPropagation(y\_hat,y):

  oe = y\_hat - y

  he = SigmoidDerivative(out\_HL[:, 1:]) \* np.dot(oe, ows.T[:, 1:])

  d\_HL = out\_IL[:, :, np.newaxis] \* he[: , np.newaxis, :]

  d\_hws = np.average(d\_HL,axis=0)

  d\_OL = out\_HL[:, :, np.newaxis] \* oe[:, np.newaxis, :]

  d\_ows = np.average(d\_OL,axis=0)

  return d\_hws,d\_ows

In the backpropagation step we are first calculating output layer error and then hidden layer error.

In the next step we are updating the weights on the hidden layer and the input layer.

OUTPUT:

At start of training the loss was too much.

Accuracy:50.0%

------------Iterative Done------

------------Iterative 1 ------

Loss:41.754135273562724

Accuracy:50.0%

------------Iterative Done------

------------Iterative 2 ------

Loss:34.33519850874117

Accuracy:50.0%

------------Iterative Done------

------------Iterative 3 ------

Loss:31.098234351502988

Accuracy:50.0%

------------Iterative Done------

------------Iterative 4 ------

Loss:29.281283007741116

Accuracy:47.5%

------------Iterative Done------

------------Iterative 5 ------

Loss:28.033118771485235

Accuracy:42.5%

------------Iterative Done------

In the middle of training: Accuracy was 100% but loss was not fully optimized.

**Streaming output truncated to the last 5000 lines.**

------------Iterative 234 ------

Loss:0.20528664329709767

Accuracy:100.0%

------------Iterative Done------

------------Iterative 235 ------

Loss:0.20519013488266058

Accuracy:100.0%

------------Iterative Done------

------------Iterative 236 ------

Loss:0.20509406132527142

Accuracy:100.0%

------------Iterative Done------

------------Iterative 237 ------

Loss:0.20499839839754452

Accuracy:100.0%

------------Iterative Done------

------------Iterative 238 ------

Loss:0.20490312345480316

Accuracy:100.0%

------------Iterative Done------

------------Iterative 239 ------

Loss:0.2048082153314547

Accuracy:100.0%

------------Iterative Done------

------------Iterative 240 ------

Loss:0.20471365424413995

Accuracy:100.0%

------------Iterative Done------

------------Iterative 241 ------

Loss:0.20461942170121655

Accuracy:100.0%

------------Iterative Done------

------------Iterative 242 ------

Loss:0.20452550041816064

Accuracy:100.0%

------------Iterative Done------

------------Iterative 243 ------

Loss:0.20443187423850168

Accuracy:100.0%

------------Iterative Done------

------------Iterative 244 ------

Loss:0.20433852805993177

Accuracy:100.0%

------------Iterative Done------

------------Iterative 245 ------

Loss:0.20424544776524656

Accuracy:100.0%

------------Iterative Done------

------------Iterative 246 ------

Loss:0.20415262015780872

Accuracy:100.0%

------------Iterative Done------

Testing on test data proved to be on 100% accuracy.

------------ TEST Result ------

Loss:0.05949899996475772

Accuracy:100.0%

------------ TEST Result ------