**HOMEWORK 3 REPORT**

**MEHUL SURESH KUMAR U52982215**

**Q1) Create a neural network which allows you to specify the number of input / hidden /output layers and measure performance differences**

Check q1.py for the program

We will perform the tests with data set 4 and hence the number of inputs and outputs will always remain at 3 and 1 respectively but we can change the number of hidden layers

|  |  |
| --- | --- |
| NUMBER OF HIDDEN LAYERS | AVERAGE ERROR OVER THE ENTIRE DATASET |
| 1 | 38.5940974917 |
| 2 | 37.3333232292 |
| 3 | 36.9215506014 |
| 4 | 36.5110537911 |
| 10 | 38.0177895529 |
| 20 | 41.0271523791 |
| 50 | 60.9547552555 |

**Q2) Implement tanH,Sigmoid and ReLu fuctions and report performance**

**ReLu fuction is being used in all test cases at outputnode because the output has to be in the range 1 to 2**

**Note: use q2.py**

|  |  |
| --- | --- |
| **Fuction Used** | **Performance** |
| **Sigmoid** | **35.0280815857** |
| **TanH** | **57.7190936403** |

Code used for activation functions:

**Tan h**

Activation

math.tanh(x)

Derivative

1. - x\*x

**Sigmoid**

Activation

1.0/(1.0+np.exp(-x)

Derivative

x \* (1.0 - x)

**ReLu**

Activation

np.log(1.0 + np.exp(x))

Derivative

expit(x)

**Q3)** **Program a neural network applying Softmax on the outputs and using cross entropy as the loss function for the backpropagation.**

**Note: Look at q3.py**

**Code Snippet**

Cross Entropy Function given by

np.sum(np.nan\_to\_num(-y\*np.log(a)-(1-y)\*np.log(1-a)))

Softmax is given by

def softmax(x):

vec = np.exp(x)

return vec / np.sum(vec)

def softmaxDerivative(x):

return x\*(1.0-x)

**Q4) Modify your gradient descent backpropagation algorithm to stochastic gradient descent algorithm for the neural network training.**

**Note: Look at q4.py**

**Code snippet**

**random.shuffle(patterns)**

**mini\_batches = [patterns[k:k+mini\_batch\_size]for k in range(0, n,mini\_batch\_size)]**

**for mini\_batch in mini\_batches:**

**for p in mini\_batch:**

**self.calculate(p[0])**

**error += self.backPropagation(p[1])**

**pass**

**self.learning\_rate = self.learning\_rate \* (self.learning\_rate / (self.learning\_rate + (self.learning\_rate \* self.rate\_decay)))**

**pass**

**Q5) Preprocess your data before training. Report the training results with different preprocessing approaches. Report performance change if any.**

Note: check q5.py

Average Error is 34.9212815902

Average error with it disabled is 37.3333232292

**Q6. Implement regularization for the weights in your neural network training. Report performance change if any.**

Note: check q6.py

Performance change is not apparently noticeable for 100 epochs

**Q7. Implement dropout in your neural network. Report performance change if any.**

**Here we randomly deactivate neurons in the hidden layer**

Note: check q7.py

Average Error with dropout enabled is 34.8798884913

Average error with it disabled is 37.3333232292