Jaykumar Nariya(1116571)

Topics in Deep Learning

Home assignment part 1

Q1 : (5 points) Repeat the computer experiment mentioned in the class by using the BP algorithm. This time, however, students need to design a four layers network and carefully train the network with one classification dataset selected from the UCI dataset library. (https://archive.ics.uci.edu/ml/datasets.php). Provide your testing accuracy on the selected dataset

Ans

Ans:

Accuracy on Training set: 76.51%

Accuracy on Testing set: 57.89%

In [25]:

```
import numpy as np #import libraries
import pandas as pd
```

In [26]:

```
def actsig(z): #define sigmoid activation function
    s = 1 / (1 + np.exp(-z))
    return s
```

In [27]:

```
def lay(X, Y): #define input and output layer shape or no of inout and output nodes

    no_input = X.shape[0]
    no_output = Y.shape[0]
    return no_input, no_output
```

In [28]:

```
def init(no_input, n_h1, n_h2, n_h3, no_output): #intialize weights and bisas at each l
ayer

    np.random.seed(1)
    W1 = np.random.randn(n_h1, no_input) * 0.01    #weigth for inputlayer--->Hiddenlaye
r1
    b1 = np.random.rand(n_h1, 1)                #bias for inputlayer--->Hiddenlayer1
    W2 = np.random.rand(n_h2, n_h1)             #weight for hiddenlayer1---->HiddenLaye
r2
    b2 = np.random.rand(n_h2, 1)                 #bias for hiddenlayer1---->HiddenLayer
2
    W3 = np.random.rand(n_h3, n_h2)             #weight for HiddenLayer2--->HiddenLayer
3
    b3 = np.random.rand(n_h3, 1)                #bias for  HiddenLayer2--->HiddenLayer3
    W4 = np.random.rand(no_output, n_h3)        #weight for HiddenLayer3---->OutputLaye
r
    b4 = np.random.rand(no_output, 1)         #bias for HiddenLayer3---->OutputLayer
    para = {"W1": W1,
                    "b1": b1,
                    "W2": W2,
                    "b2": b2,
                    "W3": W3,
                    "b3": b3,
                    "W4": W4,
                    "b4": b4}

    return para
```

In [29]:

```
def forwardpo(X, para):   #define Forward Prpogation
    W1 = para['W1']
    b1 = para['b1']
    W2 = para['W2']
    b2 = para['b2']
    W3 = para['W3']
    b3 = para['b3']
    W4 = para['W4']
    b4 = para['b4']

    Z1 = np.dot(W1, X) + b1    #Calculate output from inputlayer--->Hidden1layer1
    A1 = np.tanh(Z1)           #apply Activation Function at HiddenLayer1 and its input
for Hidden1Layer1
    Z2 = np.dot(W2, A1) + b2     #Calculate output from Hidden1layer1--->Hidden1layer2
    A2 = np.tanh(Z2)            #apply Activation Function at HiddenLayer2 and its inpu
t for Hidden1Layer2
    Z3 = np.dot(W3, A2) + b3   #Calculate output from Hidden1layer2--->Hidden1layer3
    A3 = np.tanh(Z3)         #apply Activation Function at HiddenLayer3 and its input f
or Hidden1Layer3
    Z4 = np.dot(W4, A3) + b4           #Calculate output from Hidden1layer3--->OutputLay
er
    A4 = actsig(Z4)           #apply Activation Function atOutputLayer
    cache = {"Z1": Z1,
             "A1": A1,
             "Z2": Z2,
             "A2": A2,
             "Z3": Z3,
             "A3": A3,
             "Z4": Z4,
             "A4": A4}

    return A4, cache
```

In [30]:

```
def computcost(A4, Y, para):  #define Cost computation
    m = Y.shape[1]
    W1 = para['W1']
    W2 = para['W2']
    W3 = para['W3']
    W4 = para['W4']
    logprobs = np.multiply(np.log(A4), Y) + np.multiply(np.log(1 - A4), (1 - Y))
    cost = - np.sum(logprobs) / m
    cost = np.squeeze(cost)

    return cost
```

In [31]:

```
def backpro(para, cache, X, Y):  #define Backpropogation
    m = Y.shape[1]
    W1 = para['W1']
    W2 = para['W2']
    W3 = para['W3']
    W4 = para['W4']
    A1 = cache['A1']
    A2 = cache['A2']
    A3 = cache['A3']
    A4 = cache['A4']


    dZ4 = A4 - Y       #Goto backword and with the derivative of output w.r.t weights f
ind error in weight at every layer
    dW4 = (1 / m) * np.dot(dZ4, A3.T)  #OutputLayer--->HiddenLayer3
    db4 = (1 / m) * np.sum(dZ4, axis=1, keepdims=True)

    dZ3 = np.multiply(np.dot(W4.T, dZ4), 1 - np.square(A3))  #HiddenLayer3--->HiddenLay
er2
    dW3 = (1 / m) * np.dot(dZ3, A2.T)
    db3 = (1 / m) * np.sum(dZ3, axis=1, keepdims=True)

    dZ2 = np.multiply(np.dot(W3.T, dZ3), 1 - np.square(A2))  #HiddenLayer2--->HiddenLay
er1
    dW2 = (1 / m) * np.dot(dZ2, A1.T)
    db2 = (1 / m) * np.sum(dZ2, axis=1, keepdims=True)

    dZ1 = np.multiply(np.dot(W2.T, dZ2), 1 - np.square(A1)) #HiddenLayer1--->InputLayer
    dW1 = (1 / m) * np.dot(dZ1, X.T)
    db1 = (1 / m) * np.sum(dZ1, axis=1, keepdims=True)

    grads = {"dW1": dW1,
             "db1": db1,
             "dW2": dW2,
             "db2": db2,
             "dW3": dW3,
             "db3": db3,
             "dW4": dW4,
             "db4": db4}

    return grads
```

In [32]:

```
def upd_para(para, grads, alpha):   #Update parameters Regarding Backpropogation
    W1 = para['W1']
    b1 = para['b1']
    W2 = para['W2']
    b2 = para['b2']
    W3 = para['W3']
    b3 = para['b3']
    W4 = para['W4']
    b4 = para['b4']

    dW1 = grads['dW1']
    db1 = grads['db1']
    dW2 = grads['dW2']
    db2 = grads['db2']
    dW3 = grads['dW3']
    db3 = grads['db3']
    dW4 = grads['dW4']
    db4 = grads['db4']

    W1 = W1 - alpha * dW1
    b1 = b1 - alpha * db1
    W2 = W2 - alpha * dW2
    b2 = b2 - alpha * db2
    W3 = W3 - alpha * dW3
    b3 = b3 - alpha * db3
    W4 = W4 - alpha * dW4
    b4 = b4 - alpha * db4

    para = {"W1": W1,
                  "b1": b1,
                  "W2": W2,
                  "b2": b2,
                  "W3": W3,
                  "b3": b3,
                  "W4": W4,
                  "b4": b4}
    return para
```

In [33]:

```
def model(X, Y, n_h1, n_h2, n_h3, num_iters, alpha, print_cost): #Built Model To train
 and test dataset
    np.random.seed(3)
    no_input = lay(X, Y)[0]  #define No of nodes at input layer and output layer
    no_output = lay(X, Y)[1]

    para = init(no_input, n_h1, n_h2, n_h3, no_output)  # calll parametrs
    W1 = para['W1']
    b1 = para['b1']
    W2 = para['W2']
    b2 = para['b2']
    W3 = para['W3']
    b3 = para['b3']
    W4 = para['W4']
    b4 = para['b4']

    costs = []
    for i in range(0, num_iters):

        A4, cache = forwardpo(X, para)  #call forawrd propogation function in range of
 num of iteration

        cost = computcost(A4, Y, para)
        grads = backpro(para, cache, X, Y)  #call backpropogation function
        if (i > 20000):
            alpha1 = (20000 / i) * alpha
            para = upd_para(para, grads, alpha1) #call update pameter function
        else:
            para = upd_para(para, grads, alpha)



    X_test = pd.read_csv('test_cancer_data.csv') #read dataset for testing
    X_test = np.array(X_test)
    X_test = X_test.T
    Y_test = pd.read_csv('test_cancer_data_y.csv')
    Y_test = np.array(Y_test)
    Y_test = Y_test.T

    predictionions = prediction(para,  X)
    print('Accuracy on training set: %.2f' % float(
         (np.dot(Y, predictionions.T) + np.dot(1 - Y, 1 - predictionions.T)) / float(Y.
size) * 100) + '%')


    predictionions = prediction(para, X_test)
    print('Accuracy on test set: %.2f' % float(
        (np.dot(Y_test, predictionions.T) + np.dot(1 - Y_test, 1 - predictionions.T)) /
float(Y_test.size) * 100) + '%')



    return para
```

In [34]:

```
def prediction(para, X): #define prediction
    A4, cache = forwardpo(X, para)
    predictionions = np.round(A4)

    return predictionions
```

In [35]:

```
def main(): #main method
    traino_input = pd.read_csv('cancer_data.csv') #read training dataset
    traino_input = np.array(traino_input)
    traino_output = pd.read_csv('cancer_data_y.csv')
    traino_output = np.array(traino_output)

    d = model(traino_input.T, traino_output.T, n_h1=20, n_h2=30, n_h3=20, num_iters=500
00, alpha=0.0002, print_cost=True)
```

In [36]:

```
main()
racy on test set: 57.89%
```

```
Accuracy on training set: 76.51%
Accuracy on test set: 57.89%
```

Q2: (3 points) Select the same classification dataset used in the above. Then design a single-layer network trained by a fixed ELM method. Provide your testing accuracy on the selected data

Ans

- **Download elm.py from** https://www.ntu.edu.sg/home/egbhuang/elm_codes.html

Training accuracy of dataset: 0.9887

Testing Acccuracy of dataset: 0.9833

In [6]:

```
import numpy as np #import libraries
import scipy
```

In [7]:

```
from elm import ELMClassifier # download elm.py and random_layer.py from https://www.ntu.ea

from sklearn.model_selection import train_test_split # import sklearn for split data
```

In [8]:

```
import pandas as pd
import numpy as np
irx = pd.read_csv("cancer_data.csv") #read  dataset
irx = np.array(irx)

iry = pd.read_csv("cancer_data_y.csv") #read data lables
iry = np.array(iry)
```

In [9]:

```
irx_train, irx_test, iry_train, iry_test = train_test_split(irx, iry, test_size=0.4, random
```

In [10]:

```
elmc = ELMClassifier(n_hidden=100, activation_func='gaussian', alpha=0.0, random_state=0) #
elmc.fit(irx,iry)
print (" Trainning accurracy of datset:- ",elmc.score(irx_train, iry_train)) #return traini
```

```
 Trainning accurracy of datset:-  0.9887640449438202
```

In [11]:

```
elmc = ELMClassifier(n_hidden=100, activation_func='gaussian', alpha=0.0, random_state=0)
elmc.fit(irx,iry)
print (" Testing accurracy of datset:- ",elmc.score(irx_test, iry_test)) #return testing ac
```

```
 Testing accurracy of datset:-  0.9833333333333333
```

Q3: (2 points) Discuss how your training parameters have been set, including learning rate, learning momentum, number of hidden neurons, learning epoch, etc.

Ans:

Learning rate(alpha) in my code is set to 0.0002 because if learning rate small then higher accuracy chances get increased. In my code for first 20000 epochs I have set 0.0002 learning rate and for others it will be alpha1 which is alpha divide by i (count of epochs after 20000). So, in my case learning rate is decreasing when epochs increased.

Learning momentum is nothing but technique which can improves training speed and accuracy in this process it can help in learning rate that we can't stuck on local minima.

Number of hidden neurons is nodes at each layer. If number of neurons is more then network is deepest. Deepest network gives more accuracy than less one. I took respectively 20 30 20 neurons for hidden layer1 to hidden layer 3. More hidden neurons give you more accuracy but also increase training time.

Learning epoch means one cycle for pass entire dataset in Network with forward and backward propagation is called one epoch and from number of epochs when network train one time it called one iteration. More number of epochs and iteration also gives more accuracy.

Other training parameters like Number of input node it can be qual to Number of features in dataset and No of output node is should be qual to number of classes f dataset. In my case input nodes is around 30 and output is 2.