neural_net

September 9, 2018

1 Importing the packages

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

2 Parameters

In [2]:
```
input_dim = 2
out_dim = 1
hidden_nodes = 10
lr = 1e-3
epochs = 10000
training_samples = 1000
```

3 Declaring functions

In [3]:
```
# Sigmoid

def sigmoid(x):
    return 1/(1 + np.exp(-x))

#derivative of sigmoid
def deriv_sigmoid(x):
    return np.exp(-x)/((1+ np.multiply(np.exp(-x),np.exp(-x))))

# softmax

def softmax(x):
    out = np.zeros(x.shape)
    for i in range(0,x.shape[0]):
        for j in range(0,x.shape[1]):
            out[i,j] = np.exp(x[i,j])/np.sum(np.exp(x[i]))
    return out

# sum of Squared error
```

```
def squared_error(y_train, y_predicted):
    return np.sum(np.multiply(y_train - y_predicted , y_train - y_predicted))
```

In [4]:
```
## fitting the model

def net_fit(x_train , y_train , epochs = 100 , hidden_nodes = 2 , lr = 1e-3):
    input_dim = x_train.shape[1]
    training_samples = x_train.shape[0]
    output_dim = y_train.shape[1]
    costs = []
    x_train = np.hstack((np.ones((training_samples , 1)), x_train))
    #initializig the parameters
    alpha = np.asmatrix(np.random.rand(input_dim + 1 , hidden_nodes))
    beta = np.asmatrix(np.random.rand(hidden_nodes+1 , out_dim))

    #looping for number of itretions
    for epoch in range(0,epochs):
        #finding z matrix
        z_raw = x_train * alpha
        z = sigmoid(z_raw)
        z_biased = np.asmatrix(np.hstack((np.ones((training_samples,1)),z)))

        #finding y matrix
        y_raw = z_biased * beta
        y_predicted = sigmoid(y_raw)

        ##finding the cost
        cost = squared_error(y_train , y_predicted)
        costs.append(cost)

        #finding gradient w.r.t beta
        delta = np.multiply((y_predicted - y_train), deriv_sigmoid(y_raw))
        d_beta = np.zeros(beta.shape)
        for i in range(0,d_beta.shape[0]):
            for j in range(0,d_beta.shape[1]):
                d_beta[i,j] = np.sum(np.multiply(delta[:,j],z_biased[:,i]))

        temp_beta = beta[1:,:]

        #finding gradient w.r.t alpha
        ss = np.multiply((delta * temp_beta.T),deriv_sigmoid(z_raw))
        d_alpha = np.zeros(alpha.shape)
        for i in range(0,d_alpha.shape[0]):
            for j in range(0,d_alpha.shape[1]):
                d_alpha[i,j] = np.sum(np.multiply(ss[:,j],x_train[:,i]))

        #updating the weights
        beta = beta - lr * d_beta
```

```
            alpha = alpha - lr*d_alpha
#           print("\n\nEpoch: " + str(epoch+1) + "   cost : " + str(cost))
        return alpha , beta , costs
```

In [5]:
```
#prediction

def net_predict(x_test , alpha , beta ):
    testing_samples = x_test.shape[0]
    #adding bias
    x_test = np.hstack((np.ones((testing_samples , 1)), x_test))

    #finding z matrix
    z_raw = x_test * alpha
    z = sigmoid(z_raw)
    z_biased = np.asmatrix(np.hstack((np.ones((testing_samples,1)),z)))

    #finding Y matrix (predicting the outputs)
    y_raw = z_biased * beta
    y_predicted = sigmoid(y_raw)
    y_predicted = np.round(y_predicted)   ##comment it if solving for regression
    return y_predicted
```

4 Generating training data

In [6]:
```
variance = 0.1
n = int(training_samples/2**input_dim)
x_00 = np.hstack((np.random.normal(0,variance,(n,1)) , np.random.normal(0,variance,(n,1)
x_01 = np.hstack((np.random.normal(0,variance,(n,1)) , np.random.normal(1,variance,(n,1)
x_10 = np.hstack((np.random.normal(1,variance,(n,1)) , np.random.normal(0,variance,(n,1)
x_11 = np.hstack((np.random.normal(1,variance,(n,1)) , np.random.normal(1,variance,(n,1)
x_Train = np.asmatrix(np.concatenate((x_00,x_01,x_10,x_11)))
print(x_Train.shape)
# y_train = np.asmatrix(np.append(np.zeros((3*n,1)),np.ones((n,1)))).T
y_train_and = np.asmatrix(np.append(np.zeros((3*n,1)),np.ones((n,1)))).T
y_train_or = np.asmatrix(np.append(np.zeros((n,1)),np.ones((3*n,1)))).T
y_train_xor = np.asmatrix(np.concatenate((np.zeros((n,1)),np.ones((2*n,1)),(np.zeros((n,

### Trying to solve classification problem , so commenting the below noise
# y_train_and += np.random.normal(1,variance,(4*n,1))
# y_train_or += np.random.normal(1,variance,(4*n,1))
# y_train_xor += np.random.normal(1,variance,(4*n,1))
print(y_train_and.shape , y_train_or.shape , y_train_xor.shape)
```

```
(1000, 2)
(1000, 1) (1000, 1) (1000, 1)
```

5 Training the Model

5.1 And Gate

5.1.1 training

```
In [7]: alpha , beta , losses = net_fit(x_Train , y_train_and , hidden_nodes = hidden_nodes , ep
        print("\nalpha:\n",alpha ,"\nbeta:\n", beta,"\n" ,"\nloss:\n", losses[9999])
```

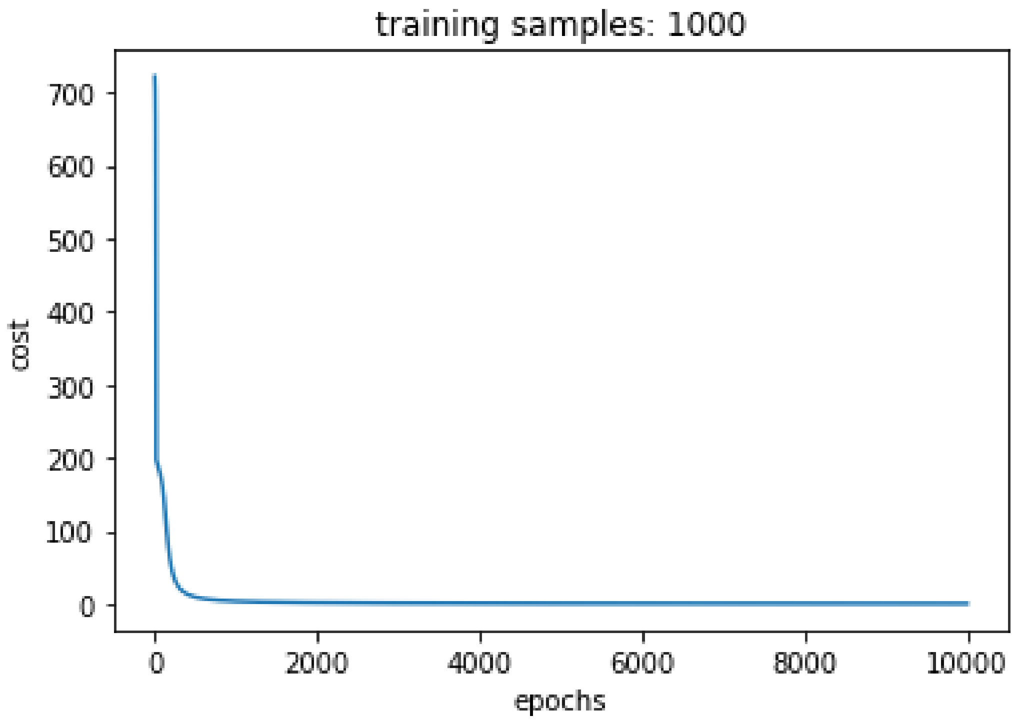
```
alpha:
 [[-3.51307423 -4.63465242  1.22520601  0.7982468  -0.24351985  0.66671741
   2.83871054 -2.69294093  4.25837015  0.11330976]
 [ 2.40884505  3.19422337  0.85820346  0.77768616  0.78663338 -0.89653705
  -1.93309024  2.09334228 -2.9443485   1.07293019]
 [ 2.41639271  3.09642561  0.48413305  0.69610432  0.64122463 -0.50088637
  -2.03072593  1.68855293 -2.85099551  0.22973448]]
beta:
 [[-2.06287244]
 [ 3.51790811]
 [ 4.86943312]
 [-0.39416143]
 [-0.09943295]
 [ 0.65310177]
 [-1.09686714]
 [-3.2592442 ]
 [ 2.73540354]
 [-4.90146692]
 [ 0.47527758]]

loss:
 0.43016720873004366
```

5.1.2 Predicting

```
In [8]: #testing samples
        x_test = np.matrix([[0,0],[0,1],[1,0],[1,1]])
        print(x_test.shape)
        #predicting the output
        res = net_predict(x_test , alpha , beta)
        print(res)
        # print(losses.shape)
        #ploting the cost vs epochs
        plt.plot(np.arange(epochs),losses)
        plt.title("training samples: " + str(training_samples))
        plt.xlabel("epochs")
        plt.ylabel("cost")
        plt.show()
```

```
(4, 2)
[[0.]
 [0.]
 [0.]
 [1.]]
```



5.2 Xor Gate

5.2.1 training

```
In [9]: alpha , beta , loss = net_fit(x_Train , y_train_xor , hidden_nodes = hidden_nodes , epoc
        print("\nalpha:\n",alpha ,"\nbeta:\n", beta,"\n","\nloss:\n", losses[9999])
```

```
alpha:
 [[-3.71883113  1.11747416  0.7002032   1.01022159  0.28228737 -2.09510014
   0.47747518  1.32104959  1.34809111  0.62993386]
 [ 2.58457781  0.88917357  2.22633808  5.24818766 -1.80649148  5.44323584
   4.98511908  1.11036849 -4.31150943  2.12772933]
 [ 2.44675438  1.13997084  2.21535201 -3.37590706  3.73155728  5.32201025
  -2.65333072  1.13761712  6.36057377  2.10873965]]
beta:
 [[ 1.74473387]
```

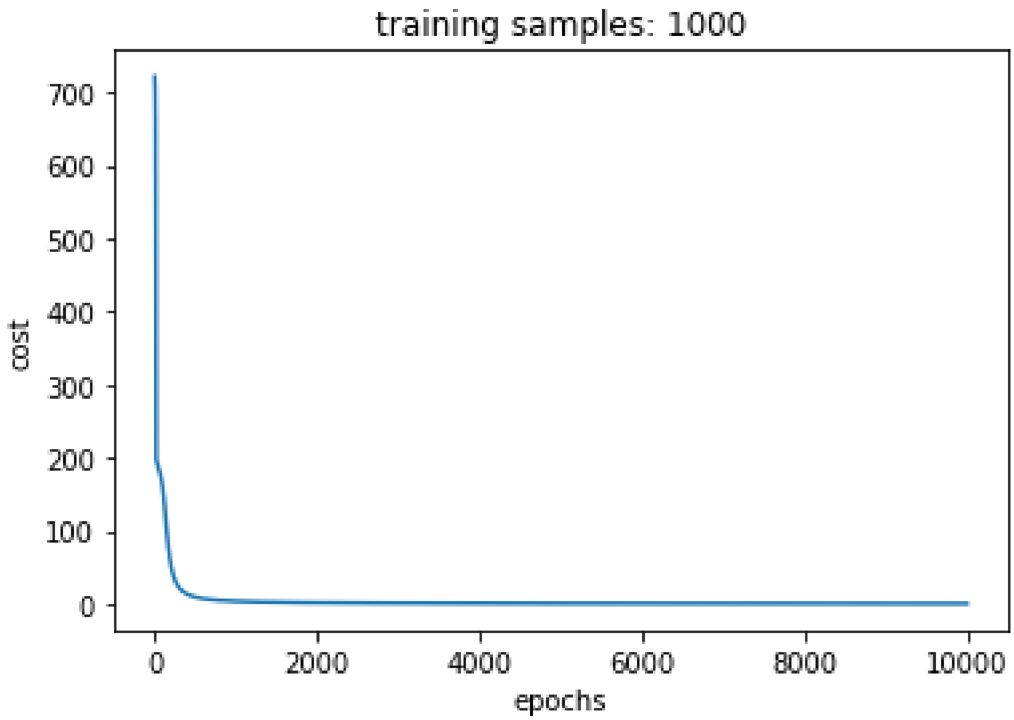
```
 [-3.48457082]
 [ 0.94442015]
 [ 2.46592664]
 [-4.50142825]
 [-2.20006634]
 [ 6.66347943]
 [-4.10288441]
 [ 1.28602994]
 [-6.36196538]
 [ 2.00915264]]

loss:
 0.43016720873004366
```

5.2.2 Predicting

```
In [10]: #testing samples
         x_test = np.matrix([[0,0],[0,1],[1,0],[1,1]])
         print(x_test.shape)
         #predicting the output
         res = net_predict(x_test , alpha , beta)
         print(res)
         # print(losses.shape)
         #ploting the cost vs epochs
         plt.plot(np.arange(epochs),losses)
         plt.title("training samples: " + str(training_samples))
         plt.xlabel("epochs")
         plt.ylabel("cost")
         plt.show()
```

```
(4, 2)
[[0.]
 [1.]
 [1.]
 [0.]]
```



5.3 Or Gate

5.3.1 training

```
In [11]: alpha , beta , loss = net_fit(x_Train , y_train_or , hidden_nodes = hidden_nodes , epoc
         print("\nalpha:\n",alpha ,"\nbeta:\n", beta,"\n","\nloss:\n", losses[9999])
```

```
alpha:
 [[-0.95879603  1.57479491 -0.5788973   1.31012256 -1.20093351 -1.77966034
  -1.9263469   0.20634353  0.76336794 -0.89312721]
 [ 1.97566829 -3.12739594  1.30861684  0.37082782  2.34635004  3.51479808
   3.82449813  0.86993224 -0.92118877  1.88387417]
 [ 1.70968809 -2.95769162  1.33462911 -0.15729169  2.27549665  3.37310516
   3.64227276  0.58380744 -1.30389912  1.59391367]]
beta:
 [[-2.84438748]
 [ 1.54861331]
 [-4.04636704]
 [ 0.96232151]
 [-0.97758311]
 [ 2.15904539]
 [ 3.62615009]
 [ 4.08981376]
```

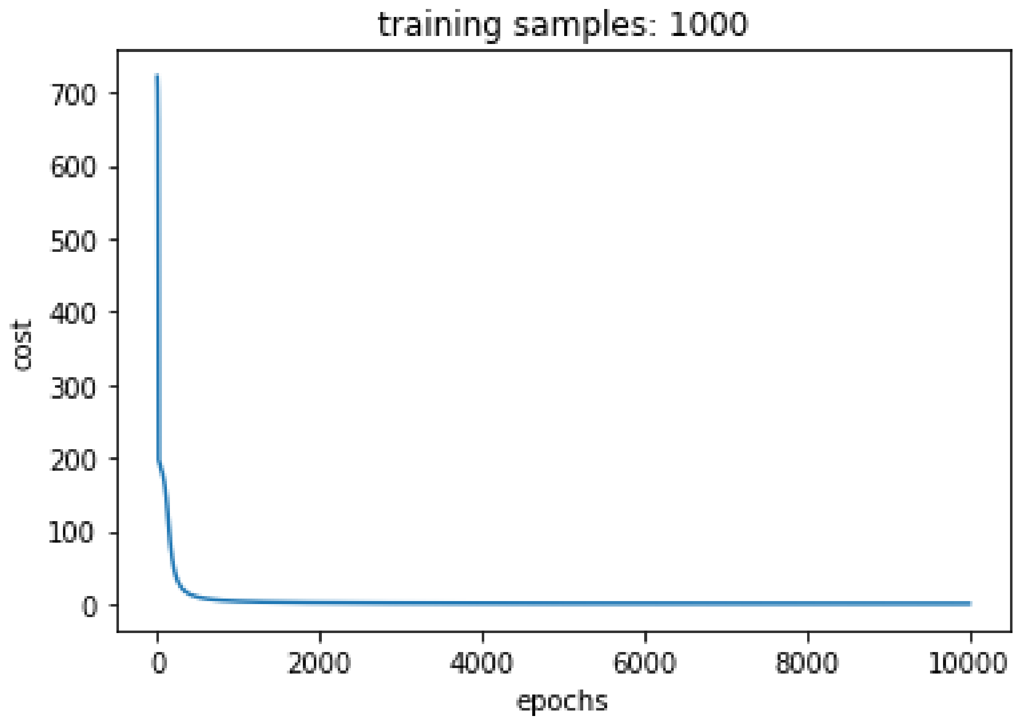
```
 [ 0.11290452]
 [-1.74823723]
 [ 1.41020165]]

loss:
 0.43016720873004366
```

5.3.2 Predicting

```
In [12]: #testing samples
         x_test = np.matrix([[0,0],[0,1],[1,0],[1,1]])
         print(x_test.shape)
         #predicting the output
         res = net_predict(x_test , alpha , beta)
         print(res)
         # print(losses.shape)
         #ploting the cost vs epochs
         plt.plot(np.arange(epochs),losses)
         plt.title("training samples: " + str(training_samples))
         plt.xlabel("epochs")
         plt.ylabel("cost")
         plt.show()
```

```
(4, 2)
[[0.]
 [1.]
 [1.]
 [1.]]
```


training samples: 1000
cost
700
600
500
400
300
200
100
0
0
2000
4000
6000
8000
10000
epochs