H.W 2

1. XOR problem:

Given; $J(\theta) = \frac{1}{4} \sum_{x \in X} \left(f^*(x) - f(x;\theta)\right)^2$; ——(1)

$X = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$;

$w = [w_1, w_2]$;

$f(x;\theta) = f(x;w,b) = x^T w + b$.

→ Solving for equation (1):

$$J(\theta) = \frac{1}{4} \sum_{x \in X} \left(f^*(x) - (x^T w + b)\right)^2$$

$$= \frac{1}{4} \sum_{x \in X} \left(f^*(x) - (x^T w + b)\right)^T \left(f^*(x) - (x^T w + b)\right)$$

Take Gradient w.r.t w,

$$\nabla_w J = \frac{1}{4} \sum_{x \in X} (-2) \left(f^*(x) - (x^T w + b)\right)^T \cdot x^T$$

Equating to 0,

$$\therefore \sum \left(f^*(x) - (x^T w + b)\right)^T \cdot x^T = 0 \qquad \text{——(2)}$$

We know that for XOR,

$$\text{Inputs} \rightarrow \text{Outputs}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$\therefore$ Transforming equation (2) in its XOR form & using w_1 & w_2:

$$\therefore \sum \left(\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \begin{bmatrix} b \\ b \\ b \\ b \end{bmatrix} \right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = 0$$

$$\rightarrow \sum \left(\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ w_2 \\ w_1 \\ (w_1+w_2) \end{bmatrix} - \begin{bmatrix} b \\ b \\ b \\ b \end{bmatrix} \right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = 0$$

$$\rightarrow \sum \left(\begin{bmatrix} -b \\ 1-w_2-b \\ 1-w_1-b \\ -w_1-w_2-b \end{bmatrix} \right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = 0$$

$$\rightarrow \sum \left(\begin{bmatrix} -b \\ 1-w_2-b \\ 1-w_1-b \\ -w_1-w_2-b \end{bmatrix} \right)^T \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = 0$$

$\rightarrow \begin{bmatrix} -b & 1-w_2-b & 1-w_1-b & -w_1-w_2-b \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = 0$

$\rightarrow \therefore \quad 1-w_1-b-w_1-w_2-b = 0 \quad \text{———} (3)$

$\& \quad 1-w_2-b-w_1-w_2-b = 0 \quad \text{———} (4)$

Taking Gradient w.r.t b, & equating to 0,

$\nabla_b J = 0$

$\therefore \frac{1}{4} \sum_{x \in X} (-2.) \left(f^*(x) - (x^T w + b) \right)^T = 0$

$\therefore \sum \left(\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \begin{bmatrix} b \\ b \\ b \\ b \end{bmatrix} \right)^T = 0$

$\therefore 2(1-w_1-w_2-2b) = 0$

$\cancel{\therefore -2w_1-2w_2}$

$\therefore 1-w_1-w_2-2b = 0$

$\therefore w_1+w_2+2b = \cancel{0}1 \quad \text{———} (5)$

Substituting equation (5) in (3);

$\cancel{1-w_1} \quad 1-2w_1-w_2-2b = 0$

$\therefore \cancel{2}\, w_1 + w_1 + w_2 + 2b = 1$

$\therefore w_1 + 1 = 1$

$\therefore \boxed{w_1 = 0}$

Similarly, substituting equation (3) in (4);

$\therefore \quad 1 - w_1 - 2w_2 - 2b = 0$

$\therefore \quad w_1 + 2w_2 + 2b = 1$

$\therefore \quad w_2 + (w_1 + w_2 + 2b) = 1$

$\therefore \quad w_2 + 1 = 1$

$\therefore \boxed{w_2 = 0}$

Substituting values of w_1 & w_2 in equation (5),

$\therefore 0 + 0 + 2b = \cancel{2}\ 1$

$\therefore \quad b = \frac{1}{2}$

$\boxed{\therefore b = 0.5}$

Hence, we can say that the values for $w_1 = 0$, $w_2 = 0$ & $b = 0.5$ ~~minize th~~ minimize the XOR function.

homework2_ktapedia_jkkothari

January 24, 2018

Deep Learning Home Work 2

0.1 Team Members :-

Krushika Tapedia (ktapedia) Janvi Kothari (jkkothari)

```
In [15]: import numpy as np
         from matplotlib import pyplot as plt
```

```
In [16]: train_faces = np.load('smile_data/trainingFaces.npy')
         train_labels = np.load('smile_data/trainingLabels.npy')
         test_faces = np.load('smile_data/testingFaces.npy')
         test_labels = np.load('smile_data/testingLabels.npy')
         train_labels = train_labels.reshape(2000,1)
         test_labels = test_labels.reshape(test_labels.shape[0],1)
```

```
In [17]: #This function computes the cost 'J' (equation is given in Method 1)
         def J(w,x,y,alpha=0):
             wt = np.matrix.transpose(w)
             j = np.sum((np.dot(x,wt)-y)**2)
             return j/2
```

```
In [18]: #This is a function that reports costs
         def report_cost(w,alpha):
             print("Training Cost :",J(w,train_faces,train_labels,alpha),"\n")
             print("Testing Cost :",J(w,test_faces,test_labels,alpha),"\n")
```

```
In [19]: #This is a function that would calculate the Gradient
         #of Cost function for Methods 1 and 2
         def gradJ(w,x,y,alpha):
             try:
                 wt,xt = np.matrix.transpose(w[:]), np.matrix.transpose(x[:])
                 h = np.dot(x,wt)
                 loss = h-y
                 grad_j = np.dot(xt,loss)
                 return grad_j
             except:
                 print(xt.shape)
```

```
In [20]: # This is the gradient descent function used in Method 3 which requires alpha
         # and a derived from a new cost function
         def gradient_descent(w,x,y,alpha):
             a = np.dot(alpha,np.matrix.transpose(w))
             gradient = gradJ(w,x,y,alpha)+a
             return gradient
```

0.2 Question 2 : Method 1

Set Gradient to 0 and Solve

```
In [21]: def method1(train_faces,train_labels,test_faces,test_labels):
             # Computing 'w' -  weights
             w_transpose = np.matrix.transpose(np.dot(np.linalg.pinv(train_faces),train_labels))
             #Computing Testing Phase
             # y = w transpose * x
             return w_transpose
```

0.3 Question 2 : Method 2

Gradient Descent - using epsilon (learning rate) and tolerance

```
In [22]: def method2(train_faces,train_labels,test_faces,test_labels,alpha):
             w = np.random.randn(1,train_faces.shape[1])
             tolerance = 10
             epsilon = 8e-6
             while tolerance > 0.001:
                 prev_cost = J(w,train_faces,train_labels,alpha)
                 prev_gradJ = gradJ(w,train_faces,train_labels,alpha)
                 u = np.dot(prev_gradJ,epsilon)
                 w = np.subtract(w,u.transpose())
                 curr_cost = J(w,train_faces,train_labels,alpha)
                 tolerance = np.absolute(prev_cost-curr_cost)
             test_cost = J(w,test_faces,test_labels,0)
             return w
```

0.4 Question 2 : Method 3

Using Penalty alpha(alpha = 1000) and epsilon (learning rate = 1e-6)

```
In [23]: def method3(train_faces,train_labels,alpha=1e3):
             w = np.random.randn(1,train_faces.shape[1])
             tolerance = 10
             epsilon = 1e-6
             while tolerance > 0.001:
                 penalty = (alpha/2)*np.dot(w,np.matrix.transpose(w))
                 prev_cost = J(w,train_faces,train_labels,alpha)+penalty
                 prev_gradJ = gradient_descent(w,train_faces,train_labels,alpha)
                 u = np.dot(prev_gradJ,epsilon)
```

```
            w = np.subtract(w,u.transpose())
            curr_cost = J(w,train_faces,train_labels,alpha)+penalty
            tolerance = np.absolute(prev_cost-curr_cost)
        test_cost = J(w,test_faces,test_labels,1e3)
        norm_w = np.linalg.norm(w)
        return w
```

In [26]:
```
print("------Results for Method 1-----\n")
w1 = method1(train_faces,train_labels,test_faces,test_labels)
report_cost(w1,0)
print("------Results for Method 2-----\n")
w2 = method2(train_faces,train_labels,test_faces,test_labels,0)
report_cost(w2,0)
print("------Results for Method 3-----\n")
w3 =  method3(train_faces,train_labels,1e3)
report_cost(w3,1000)
print("----------------------------------")
print("Norm of W in Method 2 : ",np.linalg.norm(w2)**2,"\n")
print("Norm of W in Method 3 : ",np.linalg.norm(w3)**2,"\n")
```

```
------Results for Method 1-----

Training Cost : 112.722612528

Testing Cost : 190.375513314

------Results for Method 2-----

Training Cost : 114.207086934

Testing Cost : 191.921492989

------Results for Method 3-----

Training Cost : 139.874835971

Testing Cost : 149.952408751

----------------------------------
Norm of W in Method 2 :  20.4521547085

Norm of W in Method 3 :  0.052674369434
```

0.4.1 Output Description :

1) The above output shows the Testing and Training costs for all the three Methods

2) Cost for Method 1 is obtained by directly setting the gradient zero and computing the weights
3) We use learning rates like epsilon = 8e-6 for Method 2 and epsilon = 1e-6 for Method 3
4) If you compare method 2 and 3, training cost is higher in method 3, whereas testing cost is lower in Method 3 as compared to Method 2.
5) The Output also has the Norm vales of W in methods 2 and 3