22b2251

Yash Tangri

In [20]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [28]:
```
def relu(z):
    if z < 0:
        return 0
    else :
        return z
```

In [29]:
```
x = np.linspace(-5,5,50)
y = relu(x)
plt.plot(x,y)
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 2
      1 x = np.linspace(-5,5,50)
----> 2 y = relu(x)
      3 plt.plot(x,y)
      4 plt.show()

Cell In[28], line 2, in relu(z)
      1 def relu(z):
----> 2     if z.all < 0:
      3         return 0
      4     else :

TypeError: '<' not supported between instances of 'builtin_function_or_metho
d' and 'int'
```

In [ ]:

In [ ]:

In [ ]:

Function for initializing parameters

In [5]:
```
def initialize_params(layer_sizes):
    params = {}
    for i in range(1,len(layer_sizes)):
        params['w' + str(i)] = np.random.randn(layer_sizes[i],layer_sizes[i-1]
        params['b' + str(i)] = np.random.randn(layer_sizes[i],1)*0.01
    return params
```

In [13]:
```
def forward_propagation(X_train, params):
    layers = len(params)//2
    values = {}
    for i in range(1,layers+1):
        if i ==1:
            values['z'+ str(i)]= np.dot(params['w'+str(i)],X_train)+params['b'
            values['A'+str(i)]= relu(values['z'+str(i)])
        else :
            values['z'+ str(i)]= np.dot(params['w'+str(i)],values['A'+str(i-1)
            if i == layers:
                values ['A'+str(i)]=values['A'+str(i)]
            else:
                values ['A'+str(i)]= relu(values['A'+str(i)])
    return values
```

$$J = 1/(2m)\Sigma(J_{true} - J_{pred})^2 :$$

why 2 here? no physical significance. just for mathematical convenience.

In [16]:
```
def compute_cost(values, Y_train):
    layers = len(values)//2
    Y_pred = values['A'+str(layers)]
    cost = 1/(2*len(Y_train))*np.sum(np.square(Y_pred-Y_train))
    return cost
```

In [18]:
```
#back propagation
#gradient descent
#assume all of this given to you
```

pseudo code for model function

In [25]:
```
#model (learning_rate,intial_params, X_train,Y_train,layers)
#then we will iterate
#for 1 to len(layers)
#predictions = forward_propagation(X_train,Y_train, intial_params)
#cost = compute_cost(predictions, Y_train)
#gradients = backward_propagation(X_train,Y_train, predictions, intial_params)
#parameters = gradient_descent(intial_params, gradients, learning_rate)
#error.append(cost)
#
```

In [ ]:
```
```