pyspark_deep_learning

October 8, 2019

```
In [1]: from pyspark import SparkContext
```

```
In [2]: sc = SparkContext("local", "first app")
```

```
In [4]: from pyspark.sql import SparkSession
```

```
In [5]: spark = SparkSession.builder \
                .master("local") \
                .appName("Neural Network Model") \
                .config("spark.executor.memory", "6gb") \
                .getOrCreate()
```

```
In [6]: sc = spark.sparkContext
```

```
In [8]: df = spark.createDataFrame([('Male', 67, 150), # insert column values
                                    ('Female', 65, 135),
                                    ('Female', 68, 130),
                                    ('Male', 70, 160),
                                    ('Female', 70, 130),
                                    ('Male', 69, 174),
                                    ('Female', 65, 126),
                                    ('Male', 74, 188),
                                    ('Female', 60, 110),
                                    ('Female', 63, 125),
                                    ('Male', 70, 173),
                                    ('Male', 70, 145),
                                    ('Male', 68, 175),
                                    ('Female', 65, 123),
                                    ('Male', 71, 145),
                                    ('Male', 74, 160),
                                    ('Female', 64, 135),
                                    ('Male', 71, 175),
                                    ('Male', 67, 145),
                                    ('Female', 67, 130),
                                    ('Male', 70, 162),
                                    ('Female', 64, 107),
                                    ('Male', 70, 175),
                                    ('Female', 64, 130),
                                    ('Male', 66, 163),
                                    ('Female', 63, 137),
                                    ('Male', 65, 165),
                                    ('Female', 65, 130),
                                    ('Female', 64, 109)],
                                   ['gender', 'height','weight']) # insert header values
```

```
In [9]: df.show()
```

```
+------+------+------+
|gender|height|weight|
+------+------+------+
|  Male|    67|   150|
|Female|    65|   135|
|Female|    68|   130|
|  Male|    70|   160|
|Female|    70|   130|
|  Male|    69|   174|
|Female|    65|   126|
|  Male|    74|   188|
|Female|    60|   110|
|Female|    63|   125|
|  Male|    70|   173|
|  Male|    70|   145|
|  Male|    68|   175|
|Female|    65|   123|
|  Male|    71|   145|
|  Male|    74|   160|
|Female|    64|   135|
|  Male|    71|   175|
|  Male|    67|   145|
|Female|    67|   130|
+------+------+------+
only showing top 20 rows
```

```
In [10]: from pyspark.sql import functions
```

```
In [12]: df = df.withColumn('gender', functions.when(df['gender']=='Female',0).otherwise(1))
```

```
In [13]: df = df.select('height', 'weight', 'gender')
```

```
In [14]: df.show()
```

```
+------+------+------+
|height|weight|gender|
+------+------+------+
|    67|   150|     1|
|    65|   135|     0|
|    68|   130|     0|
|    70|   160|     1|
|    70|   130|     0|
|    69|   174|     1|
|    65|   126|     0|
|    74|   188|     1|
|    60|   110|     0|
|    63|   125|     0|
|    70|   173|     1|
|    70|   145|     1|
|    68|   175|     1|
|    65|   123|     0|
|    71|   145|     1|
|    74|   160|     1|
|    64|   135|     0|
|    71|   175|     1|
|    67|   145|     1|
|    67|   130|     0|
+------+------+------+
only showing top 20 rows
```

```
In [15]: import numpy as np

In [16]: df.select("height", "weight", "gender").collect()

Out[16]: [Row(height=67, weight=150, gender=1),
          Row(height=65, weight=135, gender=0),
          Row(height=68, weight=130, gender=0),
          Row(height=70, weight=160, gender=1),
          Row(height=70, weight=130, gender=0),
          Row(height=69, weight=174, gender=1),
          Row(height=65, weight=126, gender=0),
          Row(height=74, weight=188, gender=1),
          Row(height=60, weight=110, gender=0),
          Row(height=63, weight=125, gender=0),
          Row(height=70, weight=173, gender=1),
          Row(height=70, weight=145, gender=1),
          Row(height=68, weight=175, gender=1),
          Row(height=65, weight=123, gender=0),
          Row(height=71, weight=145, gender=1),
          Row(height=74, weight=160, gender=1),
          Row(height=64, weight=135, gender=0),
          Row(height=71, weight=175, gender=1),
          Row(height=67, weight=145, gender=1),
          Row(height=67, weight=130, gender=0),
          Row(height=70, weight=162, gender=1),
          Row(height=64, weight=107, gender=0),
          Row(height=70, weight=175, gender=1),
          Row(height=64, weight=130, gender=0),
          Row(height=66, weight=163, gender=1),
          Row(height=63, weight=137, gender=0),
          Row(height=65, weight=165, gender=1),
          Row(height=65, weight=130, gender=0),
          Row(height=64, weight=109, gender=0)]
```

```
In [19]: data_array = np.array(df.select("height", "weight", "gender").collect())
```

```
In [20]: data_array
```

```
Out[20]: array([[ 67, 150,   1],
                [ 65, 135,   0],
                [ 68, 130,   0],
                [ 70, 160,   1],
                [ 70, 130,   0],
                [ 69, 174,   1],
                [ 65, 126,   0],
                [ 74, 188,   1],
                [ 60, 110,   0],
                [ 63, 125,   0],
                [ 70, 173,   1],
                [ 70, 145,   1],
                [ 68, 175,   1],
                [ 65, 123,   0],
                [ 71, 145,   1],
                [ 74, 160,   1],
                [ 64, 135,   0],
                [ 71, 175,   1],
                [ 67, 145,   1],
                [ 67, 130,   0],
                [ 70, 162,   1],
                [ 64, 107,   0],
                [ 70, 175,   1],
                [ 64, 130,   0],
                [ 66, 163,   1],
                [ 63, 137,   0],
                [ 65, 165,   1],
                [ 65, 130,   0],
                [ 64, 109,   0]])
```

```
In [21]: data_array.shape
```

```
Out[21]: (29, 3)
```

```
In [22]: data_array[0]
```

```
Out[22]: array([ 67, 150,   1])
```

```
In [23]: data_array[28]
```

```
Out[23]: array([ 64, 109,   0])
```

```
In [24]: print(data_array.max(axis=0))
         print(data_array.min(axis=0))
```

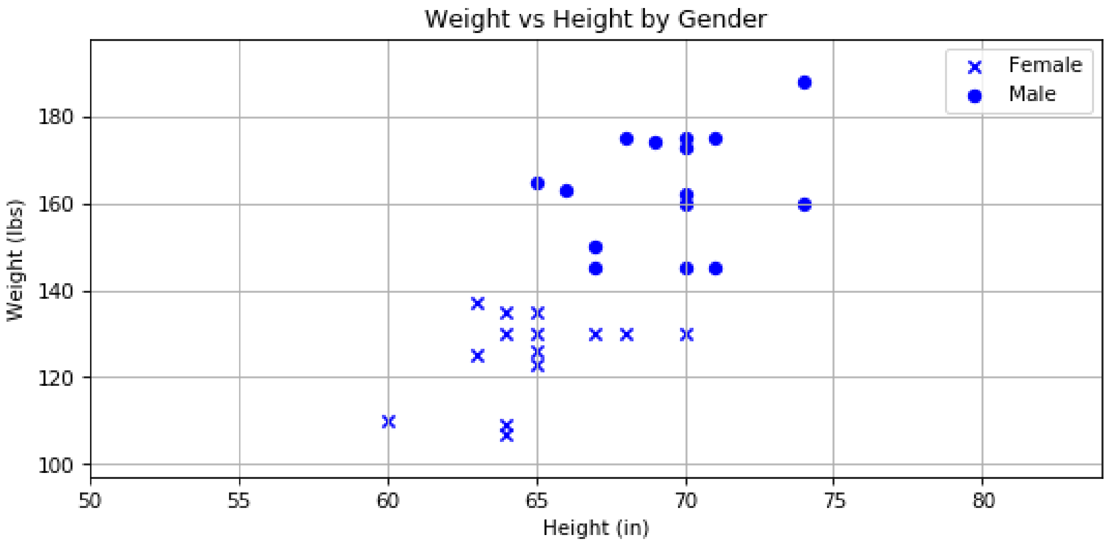
```
[ 74 188   1]
[ 60 107   0]
```

```
In [25]: import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [26]: min_x = data_array.min(axis=0)[0]-10
         max_x = data_array.max(axis=0)[0]+10
         min_y = data_array.min(axis=0)[1]-10
         max_y = data_array.max(axis=0)[1]+10

         print(min_x, max_x, min_y, max_y)
```

```
50 84 97 198
```

```
In [28]: plt.figure(figsize=(9,4), dpi=75)
         plt.axis([min_x, max_x, min_y, max_y])
         plt.grid()
         for i in range(len(data_array)):
             value = data_array[i]
             # assign labels values to specific matrix elements
             gender = value[2]
             height = value[0]
             weight = value[1]
             #filter data points by gender
             a = plt.scatter(height[gender==0], weight[gender==0], marker='x', c='b', label='Fe
             b = plt.scatter(height[gender==1], weight[gender==1], marker='o', c='b', label='Ma
             #plot values, title, legend, x and y axis
             plt.title('Weight vs Height by Gender')
             plt.xlabel('Height (in)')
             plt.ylabel('Weight (lbs)')
             plt.legend(handles=[a,b])
```



```
In [29]: np.random.seed(12345)

In [30]: w1=np.random.randn()
         w2=np.random.randn()
         b=np.random.randn()

In [31]: print(w1, w2, b)

-0.20470765948471295 0.47894333805754824 -0.5194387150567381


In [47]: X = data_array[:,:2]
         y = data_array[:,2]
         print(X, y)

[[ 67 150]
 [ 65 135]
 [ 68 130]
 [ 70 160]
 [ 70 130]
 [ 69 174]
 [ 65 126]
 [ 74 188]
 [ 60 110]
 [ 63 125]
 [ 70 173]
 [ 70 145]
 [ 68 175]
 [ 65 123]
 [ 71 145]
 [ 74 160]
 [ 64 135]
 [ 71 175]
 [ 67 145]
 [ 67 130]
 [ 70 162]
 [ 64 107]
 [ 70 175]
 [ 64 130]
 [ 66 163]
 [ 63 137]
 [ 65 165]
 [ 65 130]
 [ 64 109]] [1 0 0 1 0 1 0 1 0 0 1 1 1 0 1 1 0 1 1 0 1 0 1 0 1 0 1 0 0]
```

```
In [48]: def normalize(X):
             x_mean = X.mean(axis=0)
             x_std = X.std(axis=0)
             X = (X - x_mean)/x_std
             return X
```

```
In [49]: X = normalize(X)
         print(X)
```

```
[[-0.06163661  0.21460055]
 [-0.65745714 -0.4618577 ]
 [ 0.23627366 -0.68734378]
 [ 0.8320942   0.66557271]
 [ 0.8320942  -0.68734378]
 [ 0.53418393  1.29693375]
 [-0.65745714 -0.86773265]
 [ 2.02373527  1.92829478]
 [-2.14700848 -1.58928812]
 [-1.25327768 -0.91282987]
 [ 0.8320942   1.25183653]
 [ 0.8320942  -0.01088554]
 [ 0.23627366  1.34203096]
 [-0.65745714 -1.0030243 ]
 [ 1.13000446 -0.01088554]
 [ 2.02373527  0.66557271]
 [-0.95536741 -0.4618577 ]
 [ 1.13000446  1.34203096]
 [-0.06163661 -0.01088554]
 [-0.06163661 -0.68734378]
 [ 0.8320942   0.75576715]
 [-0.95536741 -1.72457977]
 [ 0.8320942   1.34203096]
 [-0.95536741 -0.68734378]
 [-0.35954687  0.80086436]
 [-1.25327768 -0.37166327]
 [-0.65745714  0.8910588 ]
 [-0.65745714 -0.68734378]
 [-0.95536741 -1.63438533]]
```

```
In [50]: print('standard deviation')
         print(round(X[:,0].std(axis=0),0))
         print('mean')
         print(round(X[:,0].mean(axis=0),0))
```

```
standard deviation
1.0
mean
-0.0
```

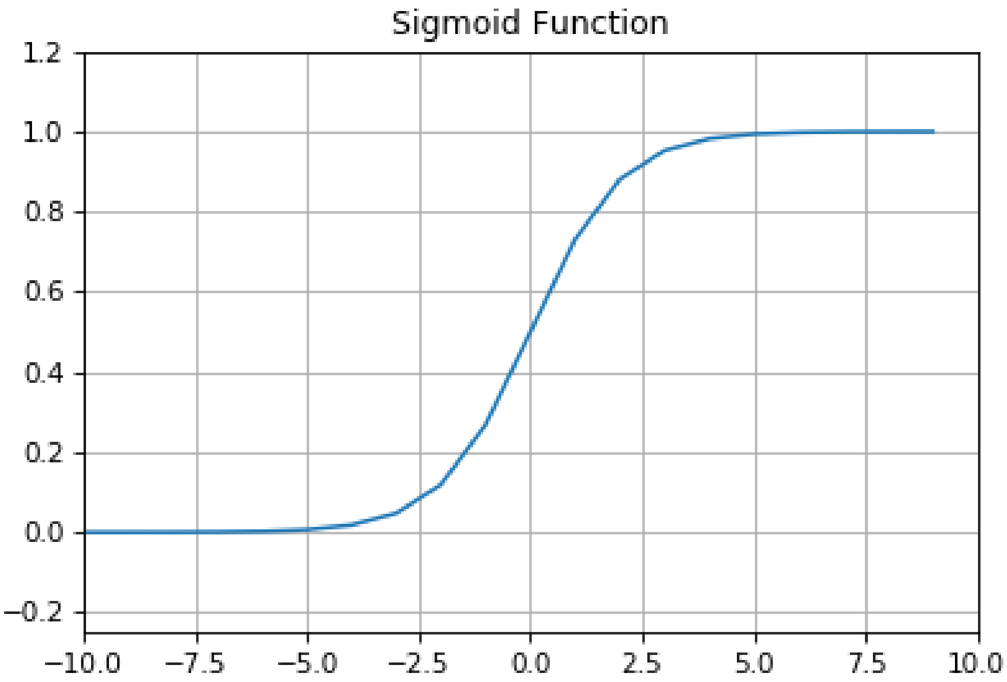
```
In [51]: data_array = np.column_stack((X[:,0], X[:,1],y))
         print(data_array)
```

```
[[-0.06163661  0.21460055  1.        ]
 [-0.65745714 -0.4618577   0.        ]
 [ 0.23627366 -0.68734378  0.        ]
 [ 0.8320942   0.66557271  1.        ]
 [ 0.8320942  -0.68734378  0.        ]
 [ 0.53418393  1.29693375  1.        ]
 [-0.65745714 -0.86773265  0.        ]
 [ 2.02373527  1.92829478  1.        ]
 [-2.14700848 -1.58928812  0.        ]
 [-1.25327768 -0.91282987  0.        ]
 [ 0.8320942   1.25183653  1.        ]
 [ 0.8320942  -0.01088554  1.        ]
 [ 0.23627366  1.34203096  1.        ]
 [-0.65745714 -1.0030243   0.        ]
 [ 1.13000446 -0.01088554  1.        ]
 [ 2.02373527  0.66557271  1.        ]
 [-0.95536741 -0.4618577   0.        ]
 [ 1.13000446  1.34203096  1.        ]
 [-0.06163661 -0.01088554  1.        ]
 [-0.06163661 -0.68734378  0.        ]
 [ 0.8320942   0.75576715  1.        ]
 [-0.95536741 -1.72457977  0.        ]
 [ 0.8320942   1.34203096  1.        ]
 [-0.95536741 -0.68734378  0.        ]
 [-0.35954687  0.80086436  1.        ]
 [-1.25327768 -0.37166327  0.        ]
 [-0.65745714  0.8910588   1.        ]
 [-0.65745714 -0.68734378  0.        ]
 [-0.95536741 -1.63438533  0.        ]]
```

```
In [52]: def sigmoid(input):
             return 1/(1+np.exp(-input))
```

```
In [53]: X = np.arange(-10,10,1)
         Y = sigmoid(X)
```

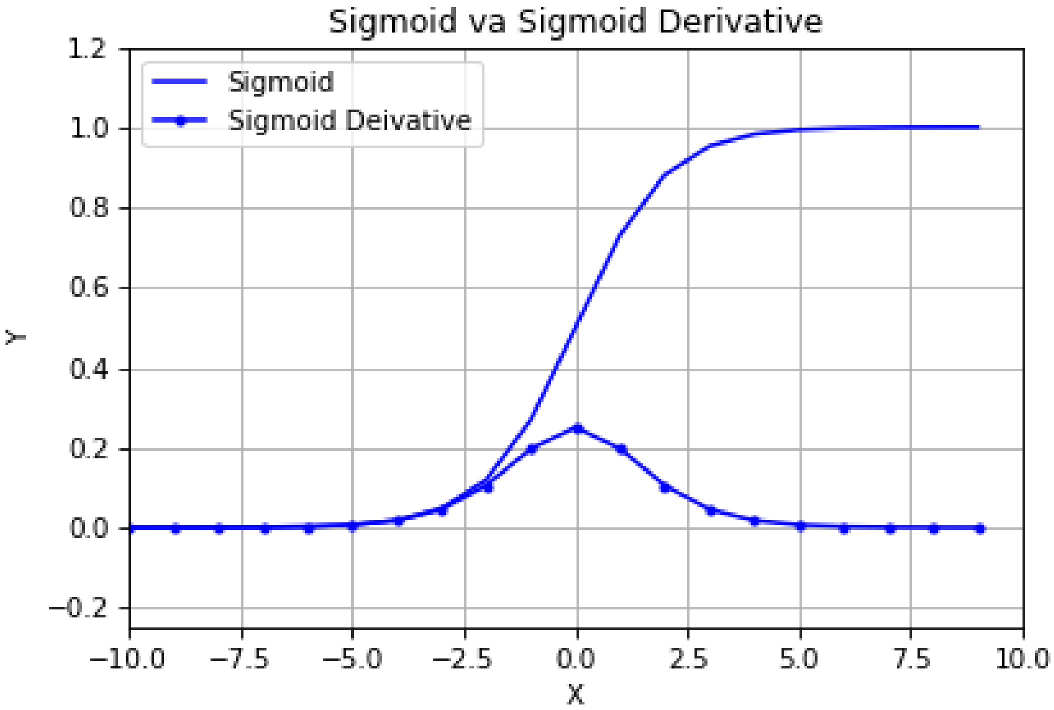
```
In [57]: plt.figure(figsize=(6,4), dpi=75)
         plt.axis([-10,10,-0.25,1.2])
         plt.grid()
         plt.plot(X,Y)
         plt.title('Sigmoid Function')
         plt.show()
```



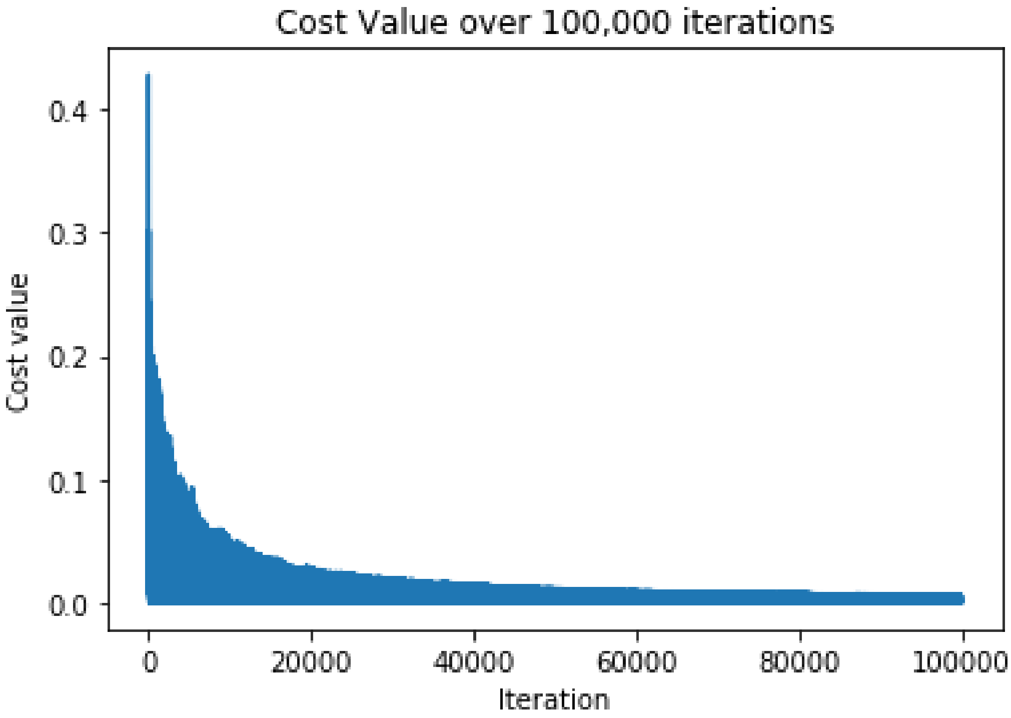
```
In [58]: def sigmoid_derivative(x):
             return sigmoid(x)*(1-sigmoid(x))
```

```
In [64]: plt.figure(figsize(6,4), dpi=75)
         plt.axis([-10,10,-0.25,1.2])
         plt.grid()
         X = np.arange(-10,10,1)
```

```
Y = sigmoid(X)
Y_Prime = sigmoid_derivative(X)
plt.plot(X, Y, label="Sigmoid", c='b')
plt.plot(X, Y_Prime, marker=".", label="Sigmoid Deivative", c='b')
plt.title("Sigmoid va Sigmoid Derivative")
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```



```
In [65]: learning_rate =0.1
         all_costs = []

         for i in range(100000):
             # set the random data points that will be used to caluate the summation
             random_number = np.random.randint(len(data_array))
             random_person = data_array[random_number]

             # the height and weight from the random individual are selected
             height = random_person[0]
             weight = random_person[1]

             z = w1*height+w2*weight+b
             predictedGender = sigmoid(z)
             actualGender = random_person[2]

             cost = (predictedGender-actualGender)**2

             #the cost value is appended to th list
             all_costs.append(cost)

             # partial derivatives of the cost function and summation are caluated
             dcost_predictedGender = 2*(predictedGender-actualGender)
             dpredictedGender_dz = sigmoid_derivative(z)
             dz_dw1 = height
             dz_dw2 = weight
             dz_db = 1

             dcost_dw1 = dcost_predictedGender * dpredictedGender_dz * dz_dw1
             dcost_dw2 = dcost_predictedGender * dpredictedGender_dz * dz_dw2
             dcost_db = dcost_predictedGender * dpredictedGender_dz * dz_db

             # grdiient descent calculation
             w1 = w1 - learning_rate * dcost_dw1
             w2 = w2 - learning_rate * dcost_dw2
             b  = b  - learning_rate * dcost_db

In [67]: plt.plot(all_costs)
         plt.title('Cost Value over 100,000 iterations')
         plt.xlabel('Iteration')
         plt.ylabel('Cost value')
         plt.show()
```



```
In [68]: print('The final values of w1, w2, and b')
         print('---------------------------------')
         print('w1 = {}'.format(w1))
         print('w2 = {}'.format(w2))
         print('b  = {}'.format(b))
```

```
The final values of w1, w2, and b
---------------------------------
w1 = 1.8207834497316182
w2 = 10.501523489170982
b  = 2.691173312112982
```

```
In [70]: for i in range(len(data_array)):
             random_individual = data_array[i]
             height = random_individual[0]
             weight = random_individual[1]
             z = height*w1 + weight*w2 + b
             predictedGender = sigmoid(z)
             print("Individual #{} actual score: {} predicted score: {}"
                   .format(i+1,random_individual[2], predictedGender))
```

```
Individual #1 actual score: 1.0 predicted score: 0.9920970050491756
Individual #2 actual score: 0.0 predicted score: 0.033695221069093326
Individual #3 actual score: 0.0 predicted score: 0.016354211472413987
Individual #4 actual score: 1.0 predicted score: 0.9999862679477725
Individual #5 actual score: 0.0 predicted score: 0.046890252255051036
Individual #6 actual score: 1.0 predicted score: 0.9999999688230504
Individual #7 actual score: 0.0 predicted score: 0.0004910720098421908
Individual #8 actual score: 1.0 predicted score: 0.999999999997268
Individual #9 actual score: 0.0 predicted score: 1.669743526542196e-08
Individual #10 actual score: 0.0 predicted score: 0.00010339213774166028
Individual #11 actual score: 1.0 predicted score: 0.9999999708970826
Individual #12 actual score: 1.0 predicted score: 0.9835674477053107
Individual #13 actual score: 1.0 predicted score: 0.9999999666012114
Individual #14 actual score: 0.0 predicted score: 0.00011865200379699481
Individual #15 actual score: 1.0 predicted score: 0.9903810250342817
Individual #16 actual score: 1.0 predicted score: 0.9999984316493865
Individual #17 actual score: 0.0 predicted score: 0.019868457681766056
Individual #18 actual score: 1.0 predicted score: 0.9999999934384332
Individual #19 actual score: 1.0 predicted score: 0.9216248756592944
Individual #20 actual score: 0.0 predicted score: 0.00957280115497827
Individual #21 actual score: 1.0 predicted score: 0.9999946742043095
Individual #22 actual score: 0.0 predicted score: 3.5311440795027645e-08
Individual #23 actual score: 1.0 predicted score: 0.9999999887129113
Individual #24 actual score: 0.0 predicted score: 0.0018952627116051622
Individual #25 actual score: 1.0 predicted score: 0.9999709601428179
Individual #26 actual score: 0.0 predicted score: 0.02948902697084434
Individual #27 actual score: 1.0 predicted score: 0.9999806260367572
Individual #28 actual score: 0.0 predicted score: 0.003255752867870676
Individual #29 actual score: 0.0 predicted score: 9.104791596102294e-08
```

```
In [71]: def input_normalize(height, weight):
             inputHeight = (height -x_mean[0])/x_std[0]
             inputWeight = (weight -x_mean[1])/x_std[1]
             return inputHeight, inputWeight
```

```
In [72]: score = input_normalize(70, 180)
```

```
In [75]: def predict_gender(raw_score):
             gender_summation = raw_score[0]*w1 + raw_score[1]*w2 + b
             gender_score = sigmoid(gender_summation)
             if gender_score <= 0.5:
                 gender = 'Female'
             else:
                 gender = 'Male'
             return gender, gender_score
```

```
In [76]: predict_gender(score)
```

```
Out[76]: ('Male', 0.9999999989427069)
```

```
In [77]: score = input_normalize(50, 120)
```

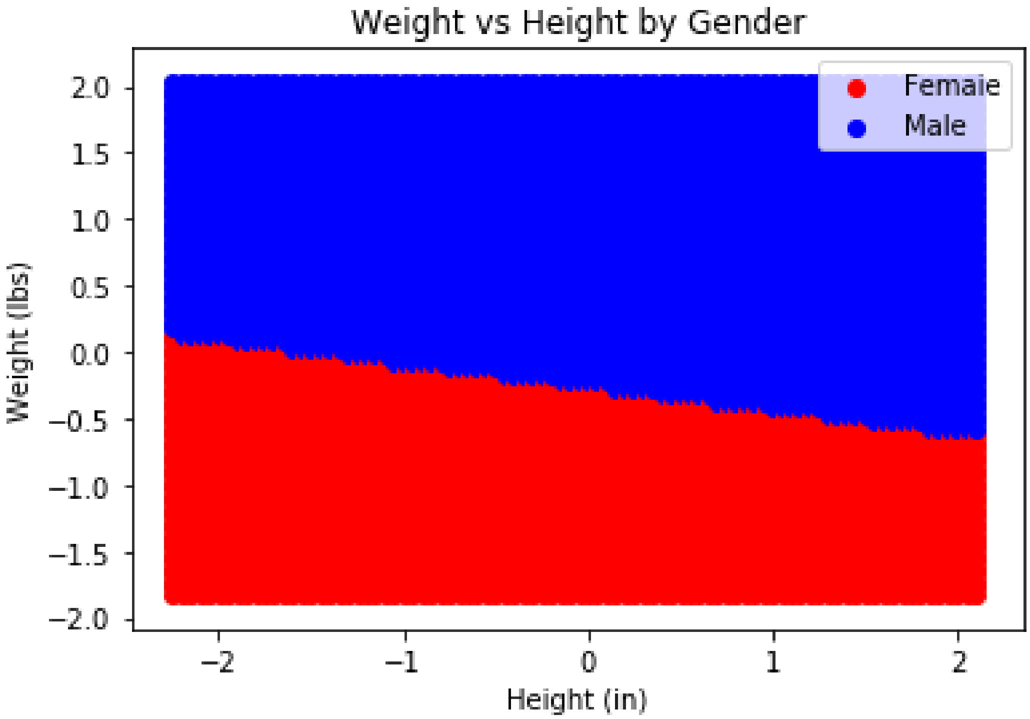
```
In [78]: predict_gender(score)
```

```
Out[78]: ('Female', 8.38839401302328e-09)
```

```
In [79]: x_min=min(data_array[:,0])-0.1
         x_max=max(data_array[:,0])+0.1
         y_min=min(data_array[:,1])-0.1
         y_max=max(data_array[:,1])+0.1
         increment= 0.05
         print(x_min, x_max, y_min, y_max)
```

```
-2.24700848158745 2.1237352673336227 -1.8245797669033634 2.028294779946051
```

```
In [83]: x_data = np.arange(x_min, x_max, increment)
         y_data = np.arange(y_min, y_max, increment)
         xy_data = [[x_all, y_all] for x_all in x_data for y_all in y_data]

         for i in range(len(xy_data)):
             data = (xy_data[i])
             height = data[0]
             weight = data[1]
             z_new = height*w1 + weight*w2 + b
             predictedGender_new = sigmoid(z_new)
             # print(height, weight, predictedGender_new)
             ax = plt.scatter(height[predictedGender_new<=0.5],
                             weight[predictedGender_new<=0.5],
                             marker='o', c='r', label='Femaie')
             bx = plt.scatter(height[predictedGender_new>0.5],
                             weight[predictedGender_new>0.5],
                             marker='o', c='b', label='Male')
             #plot values, title, legend, x and y
             plt.title('Weight vs Height by Gender')
             plt.xlabel('Height (in)')
             plt.ylabel('Weight (lbs)')
             plt.legend(handles=[ax, bx])
```


Weight vs Height by Gender
Female
Male
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Weight (lbs)
−2
−1
0
1
2
Height (in)

```
In [ ]:
```