Tarea7_NeuralNets

December 14, 2017

1 Tarea 7. Redes Neuronales.

L.E. Rojón
138442

```
In [31]: import tensorflow as tf
         import numpy as np
         import matplotlib.pyplot as plt
         import random
         import matplotlib.patches as patches
         import matplotlib.path as path
         import math
```

2 Regresión Logística con la función AND y la función OR

Primero, la función AND:

```
In [2]: ANDinput = np.asarray([[0,0],[0,1],[1,0],[1,1]])
        ANDoutput = np.asarray([[0],[0],[0],[1]])
```

Las entradas para las gráficas de Tensorflow.

```
In [3]: x = tf.placeholder(tf.float32, [None, 2])
        y = tf.placeholder(tf.float32, [None,1])
```

Pesos iniciales

```
In [4]: W= tf.Variable(tf.random_uniform([2,1], -1, 1), name="W")
        b = tf.Variable(tf.zeros([1]), name="b")

        predict = tf.nn.sigmoid(tf.matmul(x,W)+b)
```

Función de pérdida

```
In [5]: loss = tf.reduce_mean(tf.reduce_sum((y-predict)**2))
```

```
In [6]: learning_rate = 0.01
        epochs = 5000
        optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
        init = tf.global_variables_initializer()
        sess = tf.Session()
        sess.run(init)
        for i in range(epochs):
          sess.run(optimizer, feed_dict={x: ANDinput, y: ANDoutput})
        print("Predict: ", sess.run(predict, feed_dict={x:[[0.8,0.5]]}))
        print(sess.run(W, feed_dict={x: ANDinput, y: ANDoutput}))
```

```
('Predict: ', array([[ 0.34308243]], dtype=float32))
[[ 2.61386251]
 [ 2.61802483]]
```

```
In [7]: correct_prediction = tf.equal(tf.round(predict),y)
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        print(sess.run(accuracy, feed_dict={x: ANDinput, y: ANDoutput}))
```

```
1.0
```

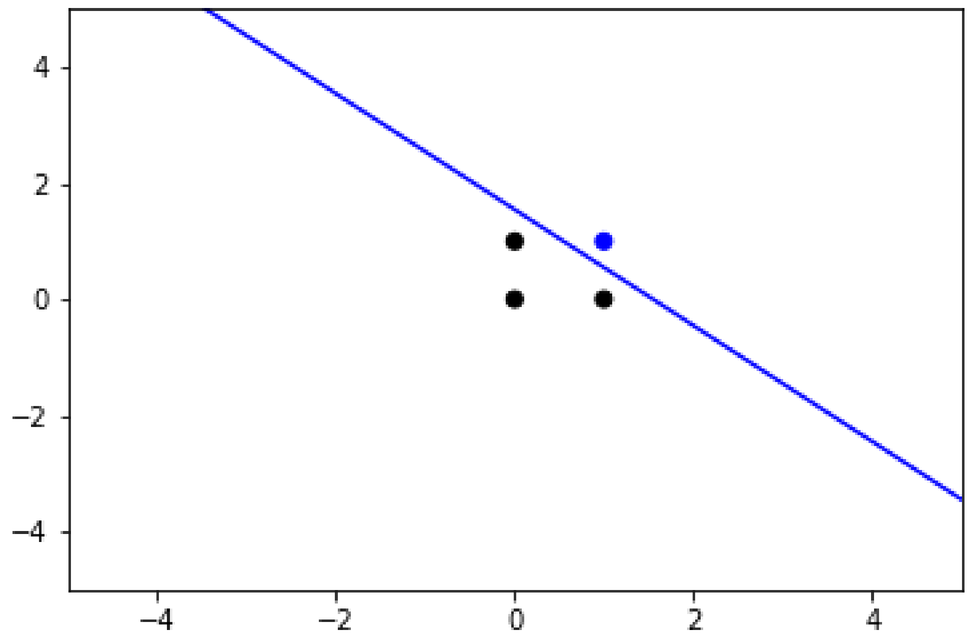
```
In [9]: w=[sess.run(b, feed_dict={x: ANDinput, y: ANDoutput})[0]]+[i[0] for i in sess.run(W, fee
        w0 = w[0]
        w1 = w[1]
        w2 = w[2]
        print w0, w1, w2
```

```
-4.04969 2.61386 2.61802
```

```
In [10]: diffX = (-w0 / w1 - 0)
         diffY = (0 - (-w0 / w2))
         gradient = (diffY)/(diffX)
         print gradient
```

```
-0.998410196655
```

```
In [11]: plt.scatter(ANDinput[:,0],ANDinput[:,1],color=['blue' if i==1 else 'black' for i in AND
         randompoints = np.linspace(-10,10,10) # 100 numeros espaciados
         plt.plot(randompoints,-w[0]/w[2] + gradient*randompoints ,color='blue')
         plt.ylim([-5,5])
         plt.xlim([-5,5])
         plt.show()
```



Ahora, construímos y usamos la función OR:

```
In [12]: XORinput = np.asarray([[0,0],[0,1],[1,0],[1,1]])
         XORoutput = np.asarray([[0],[1],[1],[0]])
```

Las entradas para las gráficas de Tensorflow.

```
In [13]: x = tf.placeholder(tf.float32, [None, 2])
         y = tf.placeholder(tf.float32, [None,1])
```

Pesos iniciales

```
In [14]: W= tf.Variable(tf.random_uniform([2,1], -1, 1), name="W")
         b = tf.Variable(tf.zeros([1]), name="b")
         predict = tf.nn.sigmoid(tf.matmul(x,W)+b)
```

Función de pérdida

```
In [15]: loss = tf.reduce_mean(tf.reduce_sum((y-predict)**2))

In [17]: learning_rate = 0.01
         epochs = 5000
         optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
         init = tf.global_variables_initializer()
         sess = tf.Session()
         sess.run(init)
         for i in range(epochs):
           sess.run(optimizer, feed_dict={x: XORinput, y: XORoutput})
         print("Predict: ", sess.run(predict, feed_dict={x:[[0.8,0.5]]}))
         print(sess.run(W, feed_dict={x: XORinput, y: XORoutput}))
```

```
('Predict: ', array([[ 0.49993813]], dtype=float32))
[[-0.00066309]
 [ 0.00117939]]
```

```
In [18]: correct_prediction = tf.equal(tf.round(predict),y)
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: XORinput, y: XORoutput}))
```

```
0.5
```

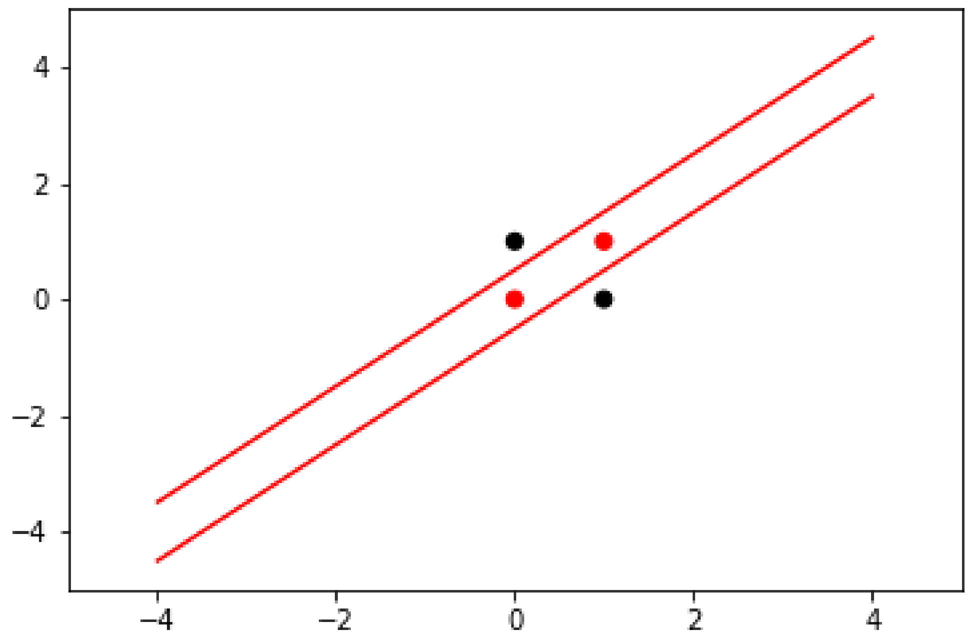
```
In [20]: w=[sess.run(b, feed_dict={x: XORinput, y: XORoutput})[0]]+[i[0] for i in sess.run(W, fe
         w0 = w[0]
         w1 = w[1]
         w2 = w[2]
         print w0, w1, w2
```

```
-0.000306496 -0.000663092 0.00117939
```

```
In [21]: diffX = (-w0 / w1 - 0)
         diffY = (0 - (-w0 / w2))
         gradient = (diffY)/(diffX)
         print gradient
```

```
0.562232158412
```

```
In [33]: boundary = np.linspace(-4,4,10)
         plt.plot(boundary,boundary - gradient/2,c='red')
         plt.plot(boundary,boundary + gradient/2,c='red')
         plt.scatter(XORinput[:,0],XORinput[:,1],color=['black' if i==1 else 'red' for i in XORo
         plt.ylim([-5,5])
         plt.xlim([-5,5])
         plt.show()
```



3 Red de ANN para XOR

```
In [23]: XORinput = np.asarray([[0,0],[0,1],[1,0],[1,1]])
         XORoutput = np.asarray([[0],[1],[1],[0]])
```

Capa de entrada.

```
In [24]: inputlayer = tf.placeholder(tf.float32, shape=[4,2])
         inputweights = tf.Variable(tf.random_uniform([2,2], -1, 1))
         inputbias = tf.Variable(tf.zeros([2]))
```

Capa de salida.

```
In [25]: outputlayer = tf.placeholder(tf.float32, shape=[4,1])
         outputweights = tf.Variable(tf.random_uniform([2,1], -1, 1))
         outputbias = tf.Variable(tf.zeros([1]))
```

Capa central(escondida).

```
In [26]: hiddenlayer = tf.sigmoid(tf.matmul(inputlayer, inputweights) + inputbias)
         outputformula = tf.sigmoid(tf.matmul(hiddenlayer, outputweights) + outputbias)
```

Calculamos la pérdida

```
In [27]: loss = tf.reduce_mean(((outputlayer * tf.log(outputformula)) +((1 - outputlayer) * tf.l
```

Entrenamos

```
In [28]: learning_rate = 0.03
         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
         init = tf.global_variables_initializer()
         sess = tf.Session()
         sess.run(init)
         epochs = 1000
         for i in range(epochs):
             sess.run(train_step, feed_dict={inputlayer: XORinput, outputlayer: XORoutput})
         w0 = sess.run(outputbias[0])
         w1 = sess.run(outputweights[0][0])
         w2 = sess.run(outputweights[1][0])
         print w0, w1, w2
```

```
0.0587005 0.535055 -0.530501
```

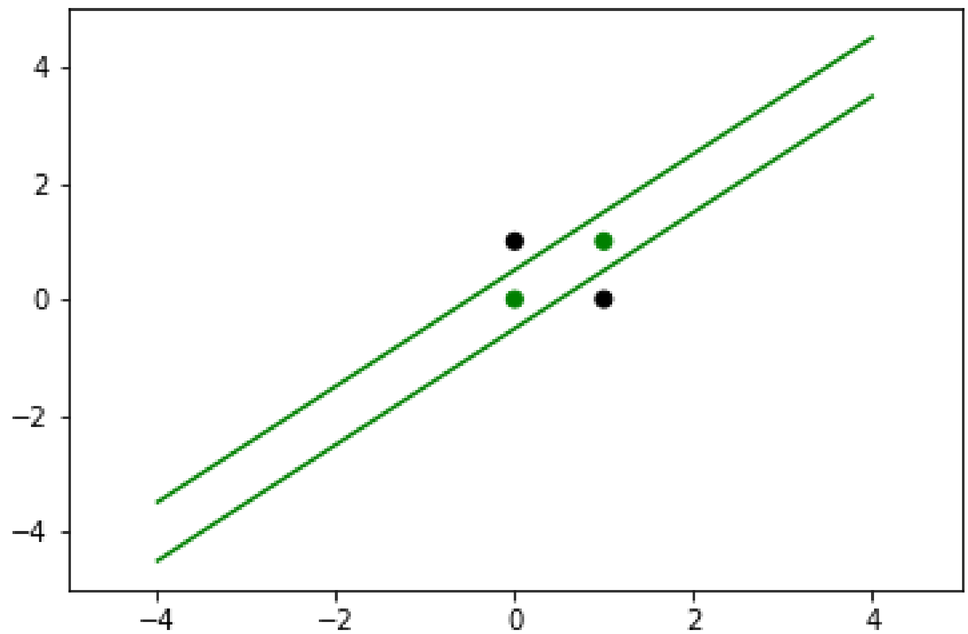
El valor del gradiente

```
In [29]: diffX = (-w0 / w1 - 0)
         diffY = (0 - (-w0 / w2))
         gradient = (diffY)/(diffX)
         print gradient
```

```
1.00858585251
```

Graficamos

```
In [34]: x = np.asarray(XORinput)[:,0]
         y = np.asarray(XORinput)[:,1]
         boundary = np.linspace(-4,4,10)
         plt.plot(boundary,boundary - gradient/2,c='green')
         plt.plot(boundary,boundary + gradient/2,c='green')
         plt.scatter(x, y, color=["black" if i==1 else "green" for i in XORoutput])
         plt.ylim([-5,5])
         plt.xlim([-5,5])
         plt.show()
```



4 ANN para el Círculo

Primero, generamos puntos aleatorios.

```
In [35]: numberOfPoints = 10000
         randompoints = np.random.rand(numberOfPoints,2)*2
         xcor = randompoints[:,0]
         ycor = randompoints[:,1]
```

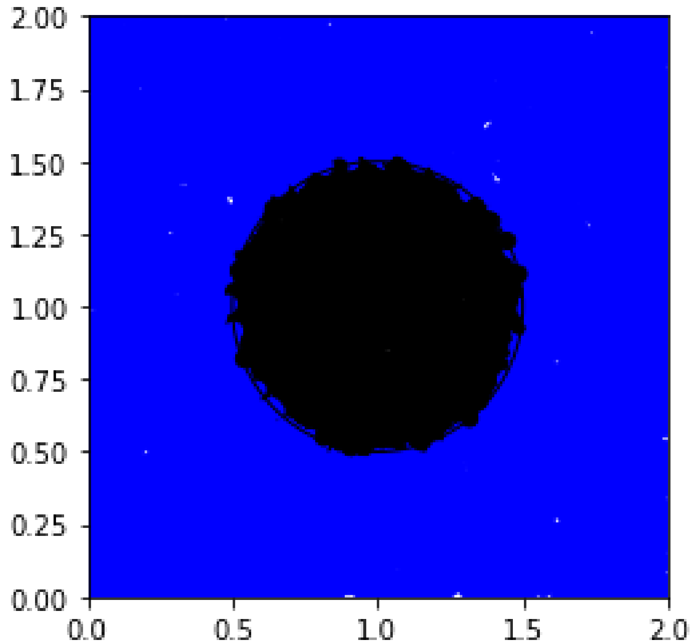
Ahora, los parámetros del círculo.

```
In [36]: radius = 0.5
         centre = (1,1)
```

Graficamos el círculo:

```
In [37]: upperlimit = centre[0] + radius
         lowerlimit = centre[0] - radius
         circledatainput = np.zeros(shape=(len(xcor),2))
         circledataoutput = np.zeros(shape=(len(xcor),1))
         for i,j,count in zip(xcor,ycor, range(len(xcor))):
             dist = math.sqrt((centre[0] - i) ** 2 + (centre[1] - j) ** 2)
             circledatainput[count] = i,j
             if (dist<=radius):
                 circledataoutput[count] = 1
```

```
    else:
        circledataoutput[count] = 0

circledatainput = np.asarray(circledatainput)
circledataoutput = np.asarray(circledataoutput)
fig1 = plt.figure()
ax1 = fig1.add_subplot(111, aspect='equal')
ax1.add_patch(patches.Circle(centre, radius, fill= False))
plt.ylim([0,2])
plt.xlim([0,2])
plt.scatter(circledatainput[:,0],circledatainput[:,1],color=['black' if i==1 else 'blue
plt.show()
```



Construimos la red para el círculo.

```
In [38]: input_size=2
         hidden_layers=4
         middle_layer=4
         output_size=1
         x = tf.placeholder(tf.float32,shape=[None,input_size])
         y = tf.placeholder(tf.float32,shape=[None,output_size])
```

Puntos de entrenamiento

```
In [39]: W1 = tf.Variable(tf.random_uniform([input_size,middle_layer], -1, 1), name="W1")
         b1 = tf.Variable(tf.zeros([middle_layer]), name="b1")
```

Puntos de predicción

```
In [40]: W2 = tf.Variable(tf.random_uniform([hidden_layers,output_size], -1, 1), name="W2")
         b2 = tf.Variable(tf.zeros([output_size]), name="b2")
```

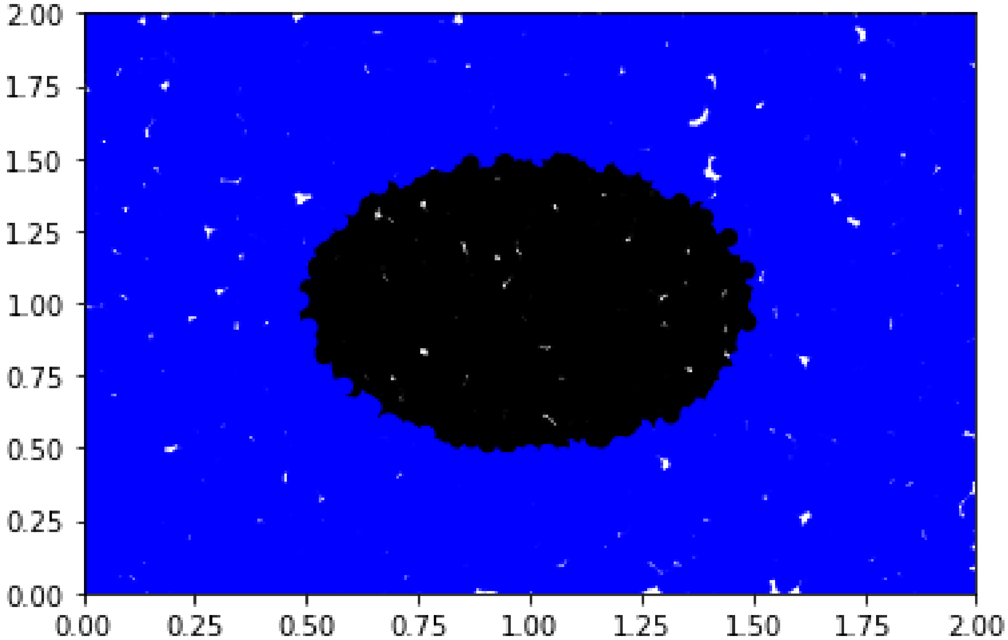
Entrenamos

```
In [43]: hidden_1 = tf.nn.sigmoid(tf.matmul(x,W1)+b1)
         predict = tf.nn.sigmoid(tf.matmul(hidden_1,W2)+b2)
         loss = tf.reduce_mean(( (y * tf.log(predict) + ((1 - y) * tf.log(1.0 - predict)) ) * -1

         learning_rate = 0.01
         train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
         init = tf.global_variables_initializer()
         sess = tf.Session()
         sess.run(init)
         epochs = 1000

         for i in range(epochs):
             sess.run(train_step, feed_dict={x: circledatainput, y: circledataoutput})

         correct_prediction = tf.equal(tf.round(predict),y)
         accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
         print(sess.run(accuracy, feed_dict={x: circledatainput, y: circledataoutput}))
```

```
0.8055
```

Graficamos la predicción

```
In [44]: prediction = sess.run(predict,feed_dict={x:circledatainput})
         plt.scatter(circledatainput[:,0],circledatainput[:,1],color=['black' if i==1 else 'blue
         plt.ylim([0,2])
         plt.xlim([0,2])
         plt.show()
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00