Práctica 4: Entrenamiento de Redes Neuronales

Ana Martín Sánchez, Nicolás Pastore Burgos

21/09/2021

1 Descripción de la práctica

En esta práctica, se pedía implementar la función de coste de una red neuronal para un conjunto de datos de entrenamiento. Posteriormente, se debía implementar el gradiente para esa red neuronal y, una vez comprobada su corrección, se pedía entrenar la red neuronal para obtener los valores óptimos de Theta1 y Theta2.

Para ello, se utilizan los mismo datos de entrenamiento que en la práctica anterior: un conjunto de 500 imágenes de números escritos a mano, en el que cada imagen (de 20x20 píxeles) se representa como una matriz de 20x20 números, donde cada número indica la intensidad en escala de grises del píxel.

2 Solución propuesta

2.1 Resultados obtenidos

2.1.1 Parte 1

En esta parte, teníamos que implementar una función de coste para una red neuronal, utilizando un algoritmo de propagación hacia delante. Con nuestra implementación, obtuvimos un coste de 0.287629.

Posteriormente, añadimos el término de regularización al coste, con lo que el coste subió hasta 0.384470.

2.1.2 Parte 2

En esta segunda parte, necesitábamos calcular el gradiente de una red neuronal de tres o más capas. Para hacerlo, implementamos una función de retro-propagación. Primero, ejecutamos una propagación hacia delante para calcular la salida de la red, y posteriormente se ejecuta la retro-propagación, para calcular cuánto contribuye cada nodo al error total. Al hacerlo, obtuvimos un error de AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.

2.2 Implementación

```
import numpy as np
import matplotlib.pyplot as plt

from scipy.io import loadmat

from checkNNGradients import checkNNGradients
from displayData import displayData
from displayData import displayImage

def sigmoide(z):
    return 1 / (1 + np.exp(-z))

def forwardProp(x, num_capas, thetas):
    a = x
    for i in range(num_capas):
        aNew = np.hstack([np.ones([x.shape[0], 1]), a])
        a = sigmoide(np.dot(aNew,thetas[i].T))

    return a

def coste(x, y_ones, num_capas, thetas):
    res = forwardProp(x, num_capas, thetas)

    aux = -(y_ones) * np.log(res)

    aux2 = (1 - y_ones) * np.log(1-res)

    return np.sum(aux - aux2) / x.shape[0]

def costeRegul(x, y_ones, num_capas, thetas, l):

    cost = coste(x, y_ones, num_capas, thetas)

    vals = np.zeros(num_capas)

    for i in range(num_capas):
        vals[i] = np.sum(thetas[i] ** 2) - np.sum(thetas[i][:, 0] ** 2)

    regul = np.sum(vals) * l / (2*x.shape[0])

    return cost + regul

def parte1():
    data = loadmat("Data/ex4data1.mat")

    x = data['X']
    y = data['y']
    yR = np.ravel(y)

    m = np.shape(x)[0]
    n = np.shape(x)[1]
```

```
1051
1052        numExamples = 100
1053        numCapas = 2
1054        numLabels = 10
1055
1056        yR = (yR - 1)
1057        y_onehot = np.zeros((m, numLabels))  # 5000 x 10
1058        for i in range(m):
1059            y_onehot[i][yR[i]] = 1
1060
1061        pesos = loadmat("Data/ex4weights.mat")
1062
1063        # red neuronal 400 neuronas input
1064        # 25 hidden
1065        # 10 output
1066        theta1, theta2 = pesos['Theta1'], pesos['Theta2']
1067
1068        thetas = np.array([theta1, theta2], dtype='object')
1069
1070        cost = costeRegul(x, y_onehot, numCapas, thetas, 1)
1071
1072        print(cost)
1073
1074        #sample = np.random.choice(m, numExamples)
1075
1076        #displayData(x[sample, :])
1077
1078        #displayImage(x[700, :])
1079
1080        #plt.show()
1081
1082    """
1083        ==========================================================
1084        ==========================================================
1085        ==========================================================
1086        ==========================================================
1087    """
1088
1089    def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, x, y, reg
        ):
1090        # backprop devuelve una tupla (coste, gradiente) con el coste y el
        gradiente de
1091        # una red neuronal de tres capas, con num_entradas, num_ocultas nodos
        en la capa
1092        # oculta y num_etiquetas nodos en la capa de salida. Si m es el numero
        de ejemplos
1093        # de entrenamiento, la dimension de 'X' es (m, num_entradas) y la de 'y
        '' es
1094        # (m, num_etiquetas)
1095
1096        if(num_ocultas.shape[0] + 2 < 3):
1097            print("ERROR: num_capas incorrect, must have an input, at least one
         hidden and an output layer")
```

```
1098            return (0,0)
1099
1100        # calculo de thetas
1101        thetas = np.empty(num_ocultas.shape[0] + 1, dtype='object')
1102        pointer = num_ocultas[0] * (num_entradas + 1)
1103
1104        thetas[0] = np.reshape(params_rn[: pointer] , (num_ocultas[0], (
            num_entradas + 1)))
1105
1106        for i in range(1, num_ocultas.shape[0]):
1107            thetas[i] = np.reshape(params_rn[pointer : pointer + num_ocultas[i]
             * (num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
1108            pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1109
1110        thetas[num_ocultas.shape[0]] = np.reshape(params_rn[pointer :] , (
            num_etiquetas, (num_ocultas[-1] + 1)))
1111
1112        # theta1 = np.reshape(params_rn[: num_ocultas * (num_entradas + 1)], (
            num_ocultas, (num_entradas + 1)))
1113        # theta2 = np.reshape(params_rn[num_ocultas * (num_entradas + 1) :], (
            num_etiquetas, (num_ocultas + 1)))
1114
1115        # thetas = np.array([theta1, theta2], dtype='object')
1116
1117        hTheta = forwardProp(x, thetas.shape[0], thetas)
1118
1119        #print(hTheta.shape)
1120
1121        delta3 = hTheta - y
1122
1123        #test = a2 *
1124
1125        delta2 = np.dot(delta3, thetas[1])
1126
1127        print(delta2.shape)
1128
1129        # print(thetas.shape)
1130
1131        # print(thetas[0].shape)
1132        # print(thetas[1].shape)
1133
1134        cost = costeRegul(x, y, thetas.shape[0], thetas, 1)
1135
1136        return (cost,0)
1137
1138
1139  def parte2():
1140        print(checkNNGradients(backprop, 1))
1141
1142  """
1143        ==========================================================
1144        ==========================================================
1145        ==========================================================
```

```
1146      ==========================================================
1147 """
1148
1149 def parte3():
1150     print("aun no llegamos")
1151
1152
1153 def main():
1154     #parte1()
1155     parte2()
1156     #parte3()
1157
1158 if __name__ == "__main__":
1159     main()
```

main.py