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 1.525e-10

2.1.3 Parte 3

En esta última parte, y tras haber comprobado que las dos anteriores se habían desarrollado correctamente, utilizamos la función scipy.optimize.minimize, y obtuvimos una precisión de entorno al 93.5%.

2.2 Implementación

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.lib.function_base import gradient

from scipy.io import loadmat

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

from scipy.optimize import minimize

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

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

    return a

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

    return np.sum((-(y_ones) * np.log(res)) - ((1 - y_ones) * np.log(1-res)
    )) / x.shape[0]

def costeRegul(x, y_ones, num_capas, thetas, reg):
    cost = coste(x, y_ones, num_capas, thetas)

    val = 0

    for i in range(num_capas):
        val += np.sum(np.power(thetas[i][1:], 2))

    regul = val * (reg / (2*x.shape[0]))

    return cost + regul

def y_oneHot(yR, numLabels, m):
    yR = (yR - 1)
    yOneH = np.zeros((m, numLabels))  # 5000 x 10
    for i in range(m):
        yOneH[i][yR[i]] = 1

    return yOneH

```

```
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]

    numExamples = 100
    numCapas = 2
    numLabels = 10

    yOneH = y_oneHot(yR, numLabels, m)

    pesos = loadmat("Data/ex4weights.mat")

    # red neuronal 400 neuronas input
    # 25 hidden
    # 10 output
    theta1, theta2 = pesos['Theta1'], pesos['Theta2']

    thetas = np.array([theta1, theta2], dtype='object')

    print("Coste sin regular: " + str(coste(x, yOneH, numCapas, thetas)))

    costReg = costeRegul(x, yOneH, numCapas, thetas, 1)

    print("Coste regulado: " + str(costReg))

    #sample = np.random.choice(m, numExamples)

    #displayData(x[sample, :])

    #displayImage(x[700, :])

    #plt.show()

"""
    ==========================================================
    ==========================================================
    ==========================================================
    ==========================================================
"""

def grad_Delta(m, Delta, theta, reg):
    gradient = Delta

    col0 = gradient[0]
    gradient = gradient + (reg/m)*theta
    gradient[0] = col0

```

```
    return gradient

def lineal_back_prop(x, y, thetas, reg):
    m = x.shape[0]
    Delta1 = np.zeros_like(thetas[0])
    Delta2 = np.zeros_like(thetas[1])

    hThetaTot = forwardProp(x, thetas.shape[0], thetas)

    for t in range(m):
        a1t = hThetaTot[0][t, :] # (401,)
        a2t = hThetaTot[1][t, :] # (26,)
        ht = hThetaTot[2][t, :] # (10,)
        yt = y[t] # (10,)

        d3t = ht - yt # (10,)
        d2t = np.dot(thetas[1].T, d3t) * (a2t * (1 - a2t)) # (26,)

        Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
        Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

    Delta1 = Delta1 / m
    Delta2 = Delta2 / m

    gradient1 = grad_Delta(m, Delta1, thetas[0], reg)
    gradient2 = grad_Delta(m, Delta2, thetas[1], reg)

    return np.append(gradient1, gradient2).reshape(-1)

def vect_back_prop(x, y, thetas, reg):
    m = x.shape[0]

    hThetaTot = forwardProp(x, thetas.shape[0], thetas)

    dlts = np.empty_like(thetas)
    Deltas = np.empty_like(thetas)

    dlts[-1] = hThetaTot[-1] - y

    for i in range(1, thetas.shape[0]):
        a = hThetaTot[-(i+1)]

        delta = np.dot(thetas[-i].T, dlts[-i].T).T

        delta = delta * a * (1-a)

        delta = delta[:,1:]

        dlts[-(i+1)] = delta

    res = []

    for i in range(thetas.shape[0]):
```

```
1156            Deltas[i] = np.dot(dlts[i].T, hThetaTot[i]) / m
1157            Deltas[i] = np.append(Deltas[i][0], Deltas[i][1:] + (reg/m) *
        thetas[i][1:])
1158            res = np.append(res, Deltas[i]).reshape(-1)
1159
1160        return res
1161
1162    def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, x, y, reg
        ):
1163        # backprop devuelve una tupla (coste, gradiente) con el coste y el
        gradiente de
1164        # una red neuronal de tres capas, con num_entradas, num_ocultas nodos
        en la capa
1165        # oculta y num_etiquetas nodos en la capa de salida. Si m es el numero
        de ejemplos
1166        # de entrenamiento, la dimension de 'X' es (m, num_entradas) y la de 'y
        '' es
1167        # (m, num_etiquetas)
1168
1169        if(num_ocultas.shape[0] + 2 < 3):
1170            print("ERROR: num_capas incorrect, must have an input, at least one
         hidden and an output layer")
1171            return (0,0)
1172
1173        # calculo de thetas
1174        thetas = np.empty(num_ocultas.shape[0] + 1, dtype='object')
1175        pointer = num_ocultas[0] * (num_entradas + 1)
1176
1177        thetas[0] = np.reshape(params_rn[: pointer] , (num_ocultas[0], (
        num_entradas + 1)))
1178
1179        for i in range(1, num_ocultas.shape[0]):
1180            thetas[i] = np.reshape(params_rn[pointer : pointer + num_ocultas[i]
         * (num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
1181            pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1182
1183        thetas[num_ocultas.shape[0]] = np.reshape(params_rn[pointer :] , (
        num_etiquetas, (num_ocultas[-1] + 1)))
1184
1185        #return costeRegul(x, y, thetas.shape[0], thetas, reg),
        lineal_back_prop(x, y, thetas, reg)
1186        return costeRegul(x, y, thetas.shape[0], thetas, reg), vect_back_prop(x
        , y, thetas, reg)
1187
1188    def parte2():
1189        print(np.sum(checkNNGradients(backprop, 1)))
1190
1191    """
1192        ==========================================================
1193        ==========================================================
1194        ==========================================================
1195        ==========================================================
1196    """
```

```
1197
1198 def optm_backprop(eIni, num_entradas, num_ocultas, num_etiquetas, x, yOneH,
     yR, laps, reg):
1199
1200     pesosSize = (num_entradas + 1) * num_ocultas[0] + (num_ocultas[-1] + 1)
     * num_etiquetas
1201
1202     for i in range(1, num_ocultas.shape[0]):
1203         pesosSize = pesosSize + ((num_ocultas[i-1] + 1) * num_ocultas[i])
1204
1205     pesos = np.random.uniform(-eIni, eIni, pesosSize)
1206
1207     out = minimize(fun = backprop, x0= pesos,
1208         args = (num_entradas, num_ocultas, num_etiquetas, x, yOneH, reg),
1209         method='TNC', jac = True, options = {'maxiter': laps})
1210
1211     thetas = np.empty(shape=[num_ocultas.shape[0] + 1], dtype='object')
1212     pointer = num_ocultas[0] * (num_entradas + 1)
1213
1214     thetas[0] = np.reshape(out.x[: pointer] , (num_ocultas[0], (
     num_entradas + 1)))
1215
1216     for i in range(1, num_ocultas.shape[0]):
1217         thetas[i] = np.reshape(out.x[pointer : pointer + num_ocultas[i] * (
     num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
1218         pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1219
1220     thetas[-1] = np.array(np.reshape(out.x[pointer :] , (num_etiquetas, (
     num_ocultas[-1] + 1))), dtype='float')
1221
1222     res = forwardProp(x, thetas.shape[0], thetas)[-1]
1223
1224     maxIndices = np.argmax(res,axis=1) + 1
1225
1226     acertados = np.sum(maxIndices==yR)
1227     print("Porcentaje acertados: " + str(acertados*100/np.shape(res)[0]) +
     "%")
1228
1229 def parte3():
1230     data = loadmat("Data/ex4data1.mat")
1231
1232     x = data['X']
1233     y = data['y']
1234     yR = np.ravel(y)
1235
1236     m = np.shape(x)[0]
1237     n = np.shape(x)[1]
1238
1239     numCapas = 2
1240     numLabels = 10
1241
1242     yOneH = y_oneHot(yR, numLabels, m)
1243
```

```
1244      num_etiquetas = 10
1245      num_ocultas = np.array([25])
1246      num_entradas = np.shape(x)[1]
1247
1248      eIni = 0.12
1249      laps = 70
1250      reg = 1
1251
1252      optm_backprop(eIni,
1253      num_entradas, num_ocultas, num_etiquetas,
1254      x, yOneH, yR,
1255      laps, reg)
1256
1257  def main():
1258      parte1()
1259      parte2()
1260      parte3()
1261
1262  if __name__ == "__main__":
1263      main()
```

main.py