

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 Θ_1 y Θ_2 .

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 entre el 90% y el 95%.

2.2 Implementación

```
1000 import numpy as np
1001 import matplotlib.pyplot as plt
1002 from numpy.lib.function_base import gradient
1003
1004 from scipy.io import loadmat
1005
1006 from checkNNGradients import checkNNGradients
1007 from displayData import displayData
1008 from displayData import displayImage
1009
1010 from scipy.optimize import minimize
1011
1012 def sigmoide(z):
1013     return 1 / (1 + np.exp(-z))
1014
1015 def forwardProp(x, num_capas, thetas):
1016     a = np.empty(num_capas + 1, dtype="object")
1017     a[0] = x
1018     for i in range(num_capas):
1019         aNew = np.hstack([np.ones([x.shape[0], 1]), a[i]])
1020         a[i] = aNew
1021         a[i+1] = sigmoide(np.dot(aNew, thetas[i].T))
1022
1023     return a
1024
1025 def coste(x, y_ones, num_capas, thetas):
1026     res = forwardProp(x, num_capas, thetas)[num_capas]
1027
1028     return np.sum(-(y_ones) * np.log(res)) - ((1 - y_ones) * np.log(1-res)) / x.shape[0]
1029
1030 def costeRegul(x, y_ones, num_capas, thetas, reg):
1031     cost = coste(x, y_ones, num_capas, thetas)
1032
1033     val = 0
1034
1035     for i in range(num_capas):
1036         val += np.sum(np.power(thetas[i][1:], 2))
1037
1038     regul = val * (reg / (2*x.shape[0]))
1039
1040     return cost + regul
1041
1042 def y_oneHot(yR, numLabels, m):
1043     yR = (yR - 1)
1044     yOneH = np.zeros((m, numLabels)) # 5000 x 10
1045     for i in range(m):
1046         yOneH[i][yR[i]] = 1
1047
1048     return yOneH
1049
```

```

1050 def partel():
1051     data = loadmat("Data/ex4data1.mat")
1052
1053     x = data['X']
1054     y = data['y']
1055     yR = np.ravel(y)
1056
1057     m = np.shape(x)[0]
1058     n = np.shape(x)[1]
1059
1060     numExamples = 100
1061     numCapas = 2
1062     numLabels = 10
1063
1064     yOneH = y_oneHot(yR, numLabels, m)
1065
1066     pesos = loadmat("Data/ex4weights.mat")
1067
1068     # red neuronal 400 neuronas input
1069     # 25 hidden
1070     # 10 output
1071     theta1, theta2 = pesos['Theta1'], pesos['Theta2']
1072
1073     thetas = np.array([theta1, theta2], dtype='object')
1074
1075     print("Coste sin regular: " + str(coste(x, yOneH, numCapas, thetas)))
1076
1077     costReg = costeRegul(x, yOneH, numCapas, thetas, 1)
1078
1079     print("Coste regulado: " + str(costReg))
1080
1081     #sample = np.random.choice(m, numExamples)
1082
1083     #displayData(x[sample, :])
1084
1085     #displayImage(x[700, :])
1086
1087     #plt.show()
1088
1089     """
1090     =====
1091     =====
1092     =====
1093     =====
1094     """
1095
1096 def grad_Delta(m, Delta, theta, reg):
1097     gradient = Delta
1098
1099     col0 = gradient[0]
1100     gradient = gradient + (reg/m)*theta
1101     gradient[0] = col0
1102

```

```

1103     return gradient
1104
1105 def lineal_back_prop(x, y, thetas, reg):
1106     m = x.shape[0]
1107     Delta1 = np.zeros_like(thetas[0])
1108     Delta2 = np.zeros_like(thetas[1])
1109
1110     hThetaTot = forwardProp(x, thetas.shape[0], thetas)
1111
1112     for t in range(m):
1113         a1t = hThetaTot[0][t, :] # (401,)
1114         a2t = hThetaTot[1][t, :] # (26,)
1115         ht = hThetaTot[2][t, :] # (10,)
1116         yt = y[t] # (10,)
1117
1118         d3t = ht - yt # (10,)
1119         d2t = np.dot(thetas[1].T, d3t) * (a2t * (1 - a2t)) # (26,)
1120
1121         Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
1122         Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])
1123
1124     Delta1 = Delta1 / m
1125     Delta2 = Delta2 / m
1126
1127     gradient1 = grad_Delta(m, Delta1, thetas[0], reg)
1128     gradient2 = grad_Delta(m, Delta2, thetas[1], reg)
1129
1130     return np.append(gradient1, gradient2).reshape(-1)
1131
1132 def vect_back_prop(x, y, thetas, reg):
1133     m = x.shape[0]
1134
1135     hThetaTot = forwardProp(x, thetas.shape[0], thetas)
1136
1137     dlts = np.empty_like(thetas)
1138     Deltas = np.empty_like(thetas)
1139
1140     dlts[-1] = hThetaTot[-1] - y
1141
1142     for i in range(1, thetas.shape[0]):
1143         a = hThetaTot[-(i+1)]
1144
1145         delta = np.dot(thetas[-i].T, dlts[-i].T).T
1146
1147         delta = delta * a * (1-a)
1148
1149         delta = delta[:, 1:]
1150
1151         dlts[-(i+1)] = delta
1152
1153     res = []
1154
1155     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,
1199 yR, laps, reg):
1200     pesosSize = (num_entradas + 1) * num_ocultas[0] + (num_ocultas[-1] + 1)
1201     * num_etiquetas
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], (
1215 num_entradas + 1)))
1216
1217     for i in range(1, num_ocultas.shape[0]):
1218         thetas[i] = np.reshape(out.x[pointer : pointer + num_ocultas[i] * (
1219 num_ocultas[i-1] + 1)], (num_ocultas[i], (num_ocultas[i-1] + 1)))
1220         pointer += num_ocultas[i] * (num_ocultas[i-1] + 1)
1221
1222     thetas[-1] = np.array(np.reshape(out.x[pointer :], (num_etiquetas, (
1223 num_ocultas[-1] + 1))), dtype='float')
1224
1225     res = forwardProp(x, thetas.shape[0], thetas)[-1]
1226
1227     maxIndices = np.argmax(res, axis=1) + 1
1228
1229     acertados = np.sum(maxIndices==yR)
1230     print("Porcentaje acertados: " + str(acertados*100/np.shape(res)[0]) +
1231 "%")
1232
1233 def parte3():
1234     data = loadmat("Data/ex4data1.mat")
1235
1236     x = data['X']
1237     y = data['y']
1238     yR = np.ravel(y)
1239
1240     m = np.shape(x)[0]
1241     n = np.shape(x)[1]
1242
1243     numCapas = 2
1244     numLabels = 10
1245
1246     yOneH = y_oneHot(yR, numLabels, m)

```

```
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