Jose Daniel Rave Robayo
Ángel López Benítez

Practica 4

Redes Neuronales
Multi-Clase

Para esta práctica hemos implementado los métodos oneVsAll para obtener los pesos (Θ) de cada una de las clases de los datos de entrada(X) y la variable independiente 'b'. En este caso de los números del 1 al 9. (Fase de entrenamiento)

Una vez obtenidos estos dos datos para cada clase, los usamos para predecir si dichos valores al ser introducidos en la función sigmoide se acercan a los valores reales de entrenamiento (Y). Para ellos hemos usado la función sigmoide, la regresión logística, descenso de gradiente regresivo de las prácticas anteriores.

La práctica consta en dos partes.La parte A consta en predecir los datos de salida usando los datos de entrenamiento obtenidos. Dicho predicción tiene que ser del 92% aproximadamente.
La parte B consta en usar unos pesos de entrada y los datos de entrada para ser introducidos en una red neuronal, la cual debería volver a predecir los datos de salida(Y) pero con mayor acierto. En particular el porcentaje de acierto debería ser del 97%.

```
PS C:\hlocal\AprendizajeAutomatico\P4\p4> python .\main.py
Matplotlib is building the font cache; this may take a moment.
OneVsAll...
Predicting...
A: 93.44%
B: 97.52%
```

CÓDIGO FUENTE DE LA PARTE A:

```python
def our_test_A():
    X , Y = readData("ex3data1.mat")

    n_label = 10
    alpha = 0.001
    print("OneVsAll...")
    all_theta = mC.oneVsAll(X, Y, n_label, alpha)
    # print(all_theta)
    print("Predicting...")
    p = mC.predictOneVsAll(all_theta, X)

    percentage = compareEquals(Y, p)

    print(f"A: {percentage}%")
```

Jose Daniel Rave Robayo
Ángel López Benítez

CÓDIGO FUENTE DE LA PARTE B:

```python
def our_test_B():
    data = loadmat('data/ex3data1.mat', squeeze_me=True)
    X = data['X']
    Y = data['y']

    weights = loadmat('data/ex3weights.mat')
    theta1, theta2 = weights['Theta1'], weights['Theta2']

    result = mC.predict(theta1, theta2, X)

    percentage = compareEquals(Y, result)

    print(f"B: {percentage}%")
```

IMPLEMENTACIÓN

```python
def oneVsAll(X, y, n_labels, lambda_):
    all_theta = np.zeros((n_labels, X.shape[1] + 1))

    alpha = 1

    for i in range(n_labels):
        newRow =  np.where(y == i, 1, 0)

        w_init = np.zeros(X.shape[1])
        b = 0
        theta_i, b_b, hisotry = lr.gradient_descent(X, newRow, w_init, b, lr.compute_cost_reg,
        lr.compute_gradient_reg, alpha, 1500, lambda_)

        all_theta[i, 0] = b_b
        all_theta[i, 1:] = theta_i

    return all_theta
```

Jose Daniel Rave Robayo
Ángel López Benítez

Predicciones:

Predicción estándar

```python
def predictOneVsAll(all_theta, X):
    # Para cada uno tienes que pasar el 0 de su linea = b
    # Con W, que es el resto de la linea
    # Y con X
    m = len(X)

    p = np.zeros(m)

    #Iterative
    # for i in range(m):
    #     # p[i] = lr.fun_wb(X[i], all_theta[i, 1:])
    #     label = 0
    #     max = 0
    #     for j in range(all_theta.shape[0]):
    #        n_Result = lr.fun_wb(X[i], all_theta[j, 1:], all_theta[j, 0])
    #        if(n_Result > max) :
    #             max = n_Result
    #             label = j
    #     print(label)
    #     p[i] = label

    # return p
    #Vectorized
    p_ = np.argmax(lr.fun_wb(X , all_theta[:, 1:].T , all_theta[:, 0]), 1)
    # print(np.array_equal(p , p_))
    return p_
```

Predicciióón de red neuronal:

```python
def predict(theta1, theta2, X):
    m = X.shape[0]
    #Input matrix
    X1s = np.c_[np.ones(m), X]
    #First layer
    first = np.dot(theta1, X1s.T)
    #Sigmoid of first multiplication
    X2s = lr.sigmoid(first)
    #new input
    X2s = np.c_[np.ones(len(X2s[0])), X2s.T]
    #second layer
    second = np.dot(theta2, X2s.T)
    #output
    p = lr.sigmoid(second)
    #Final output
    p = np.argmax(p.T, 1)
    return p
```