Introduction to Machine Learning

Program Assignment #4

Neural Network

0416098 王于哲

1. Screenshot of Forward-propagate code

這個函式要注意的地方就是 a1,a2 的 Matrix 前面要多加一個全是 1 的 column

```python
def forward_propagate(X, theta1, theta2):
    m = X.shape[0]
    #Write codes here
    a1 = np.concatenate((np.ones((X.shape[0],1)),X), axis = 1 ) # the first column is all one, used for bias
    z2 = a1.dot(theta1.T)
    a2 = np.concatenate((np.ones((z2.shape[0],1)),sigmoid(z2)), axis = 1 )
    z3 = a2.dot(theta2.T)
    h = sigmoid(z3)

    return a1, z2, a2, z3, h
```

2. Screenshot of Back-propagate code

Cost function 其實已經有附在上方，但如果直接去呼叫的話 scipy.optimize.minimize 這個函式會報錯，因此就直接複製下來。

一開始是使用 no vectorized 的方式，也就是每個 data row 去算一次 gradient 再相加；但是其實矩陣的運算可以直接一次算好整個 data 的 gradient，比較省時。

```python
def backprop(params, input_size, hidden_size, num_labels, X, y, learning_rate):
    m = X.shape[0]
    theta1 = np.matrix(np.reshape(params[:hidden_size * (input_size + 1)], (hidden_size, (input_size + 1))))
    theta2 = np.matrix(np.reshape(params[hidden_size * (input_size + 1):], (num_labels, (hidden_size + 1))))
    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

    J = 0

    for i in range(m):
        first_term = np.multiply(-y[i,:], np.log(h[i,:]))
        second_term = np.multiply((1 - y[i,:]), np.log(1 - h[i,:]))
        J += np.sum(first_term - second_term)

    J = J / m

    # add the cost regularization term
    J += (float(learning_rate) / (2 * m)) * (np.sum(np.power(theta1[:,1:], 2)) + np.sum(np.power(theta2[:,1:], 2)))


    grad2 = np.zeros(theta1.shape)
    grad3 = np.zeros(theta2.shape)
    ''' # No Vectorized
    for i in range(m):
        d3 = -(y[i,:] - h[i,:]) # 1 x 10
        d2 = np.multiply(d3.dot(theta2[:,1:]) , sigmoid_gradient(z2[i,:]) )

        grad3 = grad3 + d3.T * a2[i,:] # 10 x 1 dot 1 x 401 = 10 x 401
        grad2 = grad2 + d2.T * a1[i,:] # 10 x 1 dor 1 x 11 = 10 x 11
    '''
    # Vectorized method
    d3 = -(y - h) # 1 x 10
    d2 = np.multiply(d3.dot(theta2[:,1:]) , sigmoid_gradient(z2) )
    grad3 = (d3.T * a2)/m # 10 x m dot m x 401 = 10 x 401
    grad2 = (d2.T * a1)/m # 10 x m dor m x 11 = 10 x 11


    # add the gradient regularization term
    grad2[:,1:] = grad2[:,1:] + (theta1[:,1:] * learning_rate) / m
    grad3[:,1:] = grad3[:,1:] + (theta2[:,1:] * learning_rate) / m

    grad = np.concatenate((np.ravel(grad2), np.ravel(grad3)))
    return J,grad
```

3. Accuracy

```
accuracy = 96.36%
```