機器學習 HW5

姓名:林士恩

學號:B043011031

系級:108 電機甲

一、原始程式碼:

```
import numpy as np
from scipy.io import loadmat
from sklearn.preprocessing import OneHotEncoder
from scipy.optimize import minimize

def sigmoid(z) :
    return 1 / (1 + np.exp(-z))  #np.exp() Calculate the exponential of all elements in the input array

def sigmoid_gradient(z):
    return np.multiply(sigmoid(z), (1 - sigmoid(z)))

def forward_propagate(X, theta1, theta2) :
    m1 = X.shape[0]                     # #of rows
    a1 = np.insert(X, 0, values = np.ones(m1), axis = 1)     #在每組data補上bias項
    z2 = a1 * theta1.T                 #z2為5000x1之vector
    m2 = z2.shape[0]
    a2 = np.insert(sigmoid(z2), 0, values = np.ones(m2), axis= 1)
    z3 = a2 * theta2.T
    output = sigmoid(z3)
    return a1, z2, a2, z3, output

def computCost(params, input_size, hidden_size, num_labels, X, Y, learning_rate):
    m = X.shape[0]
    X = np.matrix(X)
    Y = np.matrix(Y)

    #reshape the theta matrix, cuz the dimensions of hidden_size and input_size is unknown
    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, output = forward_propagate(X, theta1, theta2)

    J = 0
    for i in range(m):
        first_term = np.multiply(-Y[i, :], np.log(output[i, :]))      #計算y(i)*logh(i)
        second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
        J += np.sum(first_term + second_term)

    J /= m
    #with regulation term
    J += (float(learning_rate) / (2 * m)) * (np.sum(np.power(theta1[:, 1:], 2)) + np.sum(np.power(theta2[:, 1:], 2)))
    return J                                                #column從1開始是為了排除bias case

def back_propagate(params, input_size, hidden_size, num_labels, X, Y, learning_rate):
    m = X.shape[0]
    X = np.matrix(X)
    Y = np.matrix(Y)

    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, output = forward_propagate(X, theta1, theta2)

    #intializations
    J = 0
    delta1 = np.zeros(theta1.shape)
```

```python
    delta2 = np.zeros(theta2.shape)
    J = computCost(params, input_size, hidden_size, num_labels, X, Y, learning_rate)

    #back propagation
    for t in range(m):      #共有m組資料，為求出m筆資料下平均的gradient，最後可以用於gradient descent
        a1t = a1[t, :]      #有包含bias case   1x401
        z2t = z2[t, :]      #z2 1x25
        a2t = a2[t, :]      #a2 1x26
        outputt = output[t, :]    #1x10
        Yt = Y[t, :]        #y取一組資料1x10

        d3t = outputt - Yt        #output error, delta3   1x10

        z2t = np.insert(z2t, 0, values = np.zeros(1))    #加入bias case才有辦法進行乘法
        d2t = np.multiply((theta2.T * d3t.T).T, sigmoid_gradient(z2t))   #26x10 10x1相乘，再跟1x26 elementwise相乘
        #Accumulate the gradient from this data set
        delta1 = delta1 + (d2t[:, 1:]).T * a1t           #bias case要排除 d2t為1x26維度,處理後變成25x1 , delta1為25x401維
        delta2 = delta2 + (d3t.T * a2t)                  #delta2 10x26

    delta1 = delta1 / m      #求出平均微分值(gradient)
    delta2 = delta2 / m      #求出平均微分值(gradient)

    delta1[:,1:] = delta1[:,1:] + (theta1[:,1:] * learning_rate) /m   #bias case需要排除，只需更新非bias項
    delta2[:,1:] = delta2[:,1:] + (theta2[:,1:] * learning_rate) /m

    grad = np.concatenate((np.ravel(delta1), np.ravel(delta2)))     #25*401+10*26

    return J, grad

data = loadmat('data/ex4data1.mat')
print(data['X'].shape, data['y'].shape)     #共5000比資料，一筆xi資料大小為400x1的vector

y = data['y']
encoder = OneHotEncoder(sparse = False)
y_oneshot = encoder.fit_transform(data['y'])        #把y擴充成10x1的vector，並把原本的值map成0和1
print(y_oneshot.shape)
#print(y_oneshot[0, :])                    #用slice檢查結果
#print(y.shape)

input_size = 400
hidden_size = 50
num_labels = 10
learning_rate = 1
                                          #np.random.random Return random floats narray in the half-open interval[0.0, 1.0)
params = (np.random.random(size = hidden_size * (input_size + 1) + num_labels * (hidden_size + 1)) - 0.5)
#print(hidden_size * (input_size + 1) + num_labels * (hidden_size + 1))     params為10285x1vector
#print(params)

X = data['X']

m = X.shape[0]
X = np.matrix(X)

#theta1, theta2 用params來隨機取值,theta1大小要為25x(400+1) theta2為 10x(25+1)

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))))

#print(theta1.shape, theta2.shape)
#print(theta1[:, 1:])
#a1, z2, a2, z3, output = forward_propagate(X, theta1, theta2)
#print(a1.shape, z2.shape, a2.shape, z3.shape, output.shape)
#print(computCost(params, input_size, hidden_size, num_labels, X, y_oneshot, learning_rate))
J, grad = back_propagate(params, input_size, hidden_size, num_labels, X, y_oneshot, learning_rate)
print(J, grad.shape)

fmin = minimize(fun = back_propagate, x0 = params, args = (input_size, hidden_size, num_labels, X, y_oneshot, learning_rate)
, method = 'TNC', jac = True, options = {'maxiter':250})        #最多訓練250次
print(fmin)        #輸出x為theta1 ,theta2 optimal

theta1 = np.matrix(np.reshape(fmin.x[:hidden_size * (input_size + 1)], (hidden_size, (input_size + 1))))
theta2 = np.matrix(np.reshape(fmin.x[hidden_size * (input_size + 1):], (num_labels, (hidden_size + 1))))
a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)
y_pred = np.array(np.argmax(h, axis=1) + 1)

correct = [1 if a == b else 0 for (a, b) in zip(y_pred, y)]
accuracy = (sum(map(int, correct)) / float(len(correct)))
print('accuracy = {0}%'.format(accuracy * 100))
```

Console：

```
8.50708401941 (10285,)
      fun: 0.34231678063982912
      jac: array([ -4.92231661e-04,   7.98980885e-07,  -2.88112992e-06,
...,
        -7.86320005e-05,  -3.42269103e-05,  -8.25864936e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 18
  status: 3
 success: False
       x: array([ 1.59812166,  0.0039949 , -0.01440565, ...,  3.7362753
,
       -0.51142129, -0.23502754])
accuracy = 99.26%
```

(由上面的結果可以得知 forward, back propagation 共跑到上限共 250 次)

二、問題討論：

1. 調整 hyper parameter, 並討論其影響：

```
7.86224892888 (12340,)
      fun: 0.33537440391259021
      jac: array([  1.19026377e-04,  -1.28748241e-07,
-1.12814157e-06, ...,
         6.48421704e-05,   1.35919053e-04,  -5.67541297e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 22
  status: 3
 success: False
       x: array([  4.81948509e-01,  -6.43741205e-04,
-5.64070785e-03, ...,
         1.77376130e+00,  -3.75695034e-01,  -9.17932922e-01])
accuracy = 99.42%
```

(hidden_unit = 30 lambda = 1)

```
6.64770711836 (8230,)
      fun: 0.36277605648452838
      jac: array([  2.81970681e-04,  -2.28968669e-06,  -9.87083231e-08,
...,
         1.05135812e-05,  -2.45017566e-05,  -7.86908510e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 19
  status: 3
 success: False
       x: array([  1.74631551e+00,  -1.14484335e-02,  -4.93541615e-04,
...,
        -2.37669615e+00,   2.89793282e-01,  -1.38317695e+00])
accuracy = 98.88%
```

(hidden_unit = 20 lambda = 1 , theta1 和 theta2 兩攤平成 vector = 20*401+10*21 = 8230)

```
8.04405605854 (4120,)
     fun: 0.46839357596449305
     jac: array([  3.95119824e-04,  -2.58511845e-06,   1.75714968e-06,
...,
         5.62128709e-05,   1.53237321e-05,   1.41286001e-04])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 19
  status: 3
 success: False
       x: array([ 1.12936832, -0.01292559,  0.00878575, ...,  3.5589876
,
       -2.83932374, -0.11490965])
accuracy = 97.34%
```

(hidden_unit = 10, lambda = 1)

```
        -3.29851199e+00,  -1.32624623e+00,  -2.40234955e+00,
        -2.10438072e+00,  -1.38878806e+00,  -3.71022540e-01,
        -3.70277410e+00,  -5.00278575e+00,  -3.27601880e+00,
         4.69729201e+00,  -1.08466615e+00,  -3.30881190e+00,
        -2.94335318e+00,  -3.49779136e+00,  -5.69050548e+00,
         3.33792290e+00,  -5.65502180e+00,   3.05151724e+00,
         3.43916432e+00,  -1.49537408e+00,   3.73521405e-01,
        -4.13877933e+00,  -5.65757423e+00,   2.79498201e-01,
         4.67102064e+00,  -9.20695084e-01,  -5.09428391e+00,
        -4.16830983e+00])
accuracy = 46.44%
```

(hidden_unit = 2, lambda = 1)

```
6.98954948191 (4120,)
     fun: 1.1377988727442414
     jac: array([  4.90296352e-05,   5.98470762e-07,
-2.29208499e-07, ...,
         4.12154471e-05,   1.82206362e-04,   1.18942358e-04])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 23
  status: 3
 success: False
       x: array([  2.36132264e-01,   2.99235381e-04,
-1.14604250e-04, ...,
        -1.31255927e+00,   2.73796596e+00,   2.12664728e+00])
accuracy = 93.12%
```

(hidden_unit = 10, lambda = 10)

```
7.50563831362 (8230,)
     fun: 1.0227191822852857
     jac: array([ -3.52401592e-04,   5.66335492e-08,
4.94786552e-07, ...,
         4.98671657e-04,   1.00149146e-04,   3.54941398e-04])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 18
  status: 3
 success: False
       x: array([  1.31483295e+00,   2.83167746e-05,
2.47393276e-04, ...,
         8.62459020e-01,  -6.62916749e-01,   9.73733007e-01])
accuracy = 93.96%
```

(hidden_unit = 20, lambda = 10)

```
15.6313818394 (8230,)
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:13: RuntimeWarning:
overflow encountered in exp
  return 1 / (1 + np.exp(-z))  #np.exp() Calculate the exponential of
all elements in the input array
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
divide by zero encountered in log
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
invalid value encountered in multiply
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
     fun: 2.6099818421132355
     jac: array([ -3.66875820e-03,   4.04308668e-05,  -3.35128362e-05,
...,
         1.06328465e-02,   1.27222599e-02,   9.16550586e-03])
 message: 'Linear search failed'
    nfev: 164
     nit: 10
  status: 4
 success: False
       x: array([ 0.10319628,  0.00202154, -0.00167564, ...,  0.052069
,
       -0.05122081,  0.06703103])
accuracy = 81.82000000000001%
```

(hidden_unit = 20, lambda = 100)

```
9.27358416132 (10285,)
     fun: 0.99801554100490009
     jac: array([ -9.62876464e-04,   1.11315474e-07,  -2.76920708e-09,
...,
        -1.83906667e-04,  -6.14493312e-05,  -3.53011832e-04])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 18
  status: 3
 success: False
       x: array([ -8.99267002e-01,   5.56577368e-05,  -1.38460354e-06,
...,
         1.34604925e+00,  -5.18594071e-01,   8.18997440e-01])
accuracy = 94.26%
```

(hidden_unit = 25, lambda = 10)

```
16.8606563139 (10285,)
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
divide by zero encountered in log
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
invalid value encountered in multiply
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
     fun: 2.6681455658259665
     jac: array([ -5.79929797e-03,  -2.19264612e-05,   8.80026610e-05,
...,
        -9.98461587e-04,  -7.71259907e-03,  -8.95916904e-03])
 message: 'Linear search failed'
    nfev: 126
     nit: 7
  status: 4
 success: False
       x: array([ 0.08978215, -0.00109632,  0.00440013, ...,
-0.53826753,
        0.20416498,  0.14142817])
accuracy = 80.28%
```

(hidden_unit = 25, lambda = 100)

```
9.50243672326 (10285,)
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
divide by zero encountered in log
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
C:/Users/user1/Desktop/Dick Learning_HW/HW5.py:42: RuntimeWarning:
invalid value encountered in multiply
  second_term = np.multiply(Y[i, :] - 1, np.log(1 - output[i, :]))
     fun: 0.11237394989217969
     jac: array([ -1.19321690e-04,  -3.91713129e-06,   5.99151071e-06,
...,
         2.42642687e-05,   9.10142045e-05,  -4.78053116e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 17
  status: 3
 success: False
       x: array([ 0.83635503, -0.19585656,  0.29957554, ...,
-3.01664685,
       -1.81858298, -2.62498999])
accuracy = 99.88%
```

(hidden_unit = 25, lambda = 0.1)

```
8.73532563656 (20560,)
     fun: 0.29887769654632512
     jac: array([ -5.25123585e-04,  -3.46321123e-06,   8.45642464e-07,
...,
         3.55130479e-04,   1.01099516e-04,   6.62300492e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 22
  status: 3
 success: False
       x: array([-0.24495967, -0.01731606,  0.00422821, ...,
2.59825472,
       -1.06655384, -1.96962856])
accuracy = 99.66000000000001%
```

(hidden_unit = 50, lambda = 1)

```
7.60666936616 (20560,)
     fun: 0.075503642933452686
     jac: array([  3.40180082e-05,  -1.89315941e-06,  -2.51697088e-06,
...,
         6.03581733e-06,   5.60279855e-05,   1.80013131e-05])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 22
  status: 3
 success: False
       x: array([ 0.13511475, -0.09465797, -0.12584854, ...,
-0.90423135,
       -1.16778706,  0.35371297])
accuracy = 100.0%
```

(hidden_unit = 50, lambda = 0.1)

```
(5000, 400) (5000, 1)
(5000, 10)
8.72277515307 (20560,)
     fun: 0.31146633772726279
     jac: array([ -2.02617567e-04,   1.80157465e-06,   4.73278950e-06,
...,
        -1.22445158e-04,  -1.60190259e-04,  -1.17906379e-04])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 21
  status: 3
 success: False
       x: array([ 0.20834283,  0.00900787,  0.02366395, ...,
0.78318145,
        1.38478568, -2.08489907])
accuracy = 99.58%
```

(hidden_unit = 50, lambda = 1)

再調參數的最好結果為100%(hidden unit = 50, lambda = 0.1)，其中因為feature夠多(解析度大)，且因為lambda的值不夠大到足以做好regulation，所以為overfiting，若hidden unit = 2時，準確率只有大概46.5%，代表feature數量根本不夠，導致嚴重的underfitting，此訓練出的模組不能使用。

並且由演算法表示法還有上面的結果可以觀察到:

Hidden unit:代表著訓練數學方程式的feature數量(解析度)，可以想成一張照片的解析度問題，1024x760一定比500x300的畫面更好，所以hidden unit就可以想成是其中的小方格，數量越多就代表著切得越精細，準確度也會越高。

Lambda(learning rate):代表著乘上regulation項的常數，如果值越大的話，能使數學模型不至於overfitting，但準確度有可能會因此下降；反之，如果值越小的話，它會導致數學模型對抗overfitting的能力下降，準確度有可能會因此上升許多。

2.調整 hyper parameter, 並討論其影響:

若採用regression的方式而不是NN的方式來訓練模型，regression的用意在於求出確切數值，而NN則是偏向多logistic的問題，探討分類問題，得到的結果為，可以知道較NN的準確度下降不少

```
(5000, 1)
32.1573689739 (10051,)
     fun: 0.34266176260446135
     jac: array([ -9.08308396e-04,  -1.02410470e-05,   2.09100363e-05, ...,
        -1.78171990e-04,   5.26714082e-03,   5.47017862e-03])
 message: 'Max. number of function evaluations reached'
    nfev: 250
     nit: 23
  status: 3
 success: False
       x: array([-0.28888085, -0.05120524,  0.10455018, ...,  1.31649397,
       -0.56268799, -1.63036737])
accuracy = 74.3%
```

可以重新修改的程式碼得知，利用 regression 類似於 NN 的架構，把 hidden unit 看成做 regression 的 feature，而輸出就是數值(類似預測房價的問題)，在 a2 時不需再經過一次 activation function，如下

```
def forward_propagate(X, theta1, theta2) :
    m1 = X.shape[0]                # #of rows
    a1 = np.insert(X, 0, values = np.ones(m1), axis = 1)     #在每組data補上bias項
    z2 = a1 * theta1.T             #z2為5000x1之vector
    m2 = z2.shape[0]
    a2 = np.insert(sigmoid(z2), 0, values = np.ones(m2), axis= 1)
    z3 = a2 * theta2.T

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

並且，在做 regresssion 時並沒有加入 regulation 項，準確度正常情況下不會提升，在 250 次 gradient descent 後得到的答案(級數值)，還須利用 transform function 去做類似量化的方法才能得到最後的答案，如下

```
def transform(h):
    pre = np.zeros(h.shape)
    for i in range(m):
        if h[i,:] > 9.5:
            pre[i, :] = 10
        elif h[i, :] < 1.5:
            pre[i, :] = 1
        else:
            pre[i, :] = int(h[i, :] + 0.5)

```

誤判大部分都在這個 function 中產生，例如手寫圖 8 經過運算後得到 8.8 在經過 transform function 後得到 9，產生誤判。

所以我們可以得到的結論為 regression 並不適合來解決多種類分類的問題，regression 會故意將數值做量化，所以效能並不高，所以 NN 較高一籌。