Introduction to Machine Learning
Assignment #4

ID: 0416208
Name: 黃士軒

1. Forward-propagate code

```python
def forward_propagate(X, theta1, theta2):
    m = X.shape[0] # number of data instances

    #Write codes here
    #Layer1
    extraBias1 = np.ones((m,1))
    a1 = np.append(extraBias1, X, axis=1) #append bias on every data instances
    z2 = np.dot(a1, theta1.transpose())
    # Layer2
    extraBias2 = np.ones((z2.shape[0],1))
    a2 = np.append(extraBias2, sigmoid(z2), axis=1)
    z3 = np.dot(a2, theta2.transpose())
    # Output
    h = sigmoid(z3)

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

2. Back-propagate code

```python
def backprop(params, input_size, hidden_size, num_labels, X, y, learning_rate):
    m = X.shape[0]

    J = cost(params, input_size, hidden_size, num_labels, X, y, learning_rate)

    # reshape the parameter array into parameter matrices for each layer
    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))))
    delta1 = np.zeros(theta1.shape)  # (10, 401)
    delta2 = np.zeros(theta2.shape)  # (10, 11)
    # run the feed-forward pass
    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)

    #print(J)  ## print cost for every iterate

    # perform backpropagation
    for i in range(m):  ## update weight by every instance
        a1Del = a1[i,:] #(1, 401)
        z2Del = z2[i,:] #(1, 10)
        a2Del = a2[i,:] #(1, 11)
        hDel = h[i,:] #(1, 10)
        yDel = y[i,:] #(1, 10)

        grad3Del = hDel - yDel # (1, 10)

        z2Del = np.append(np.ones((1,1)), z2Del, axis=1)  # (1, 11)
        temp = np.dot(theta2.transpose(), grad3Del.transpose()) # (11, 1)
        grad2Del = np.multiply(temp.transpose(), sigmoid_gradient(z2Del))  # (1, 11)

        delta1 = delta1 + np.dot((grad2Del[:,1:]).transpose(), a1Del) ## remove grad2Del[:,0] #(10, 401)
        delta2 = delta2 + np.dot(grad3Del.transpose(), a2Del) #(10, 11)

    delta1 = delta1 / m
    delta2 = delta2 / m

    delta1[:,1:] = delta1[:,1:] + (theta1[:,1:] * learning_rate) / m
    delta2[:,1:] = delta2[:,1:] + (theta2[:,1:] * learning_rate) / m

    gradient = np.concatenate((np.ravel(delta1), np.ravel(delta2)))
    return J, gradient
```

3. Accuracy

jupyter hw4 最后检查: 幾秒前 (自动保存)

File Edit View Insert Cell Kernel Widgets Help

```python
X = np.matrix(X)
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))
```

```
0.458332636852
0.45833263415
0.458332634323
0.458332633669
0.458332634942
0.458332633888
0.458332634147
0.458332634846
0.458332634943
0.458332633996
0.458332634922
0.458332634846
0.458332636022
0.452170205714
0.452170204738
0.452170204972
0.451779704453
0.451779704453
accuracy = 97.64%
```

```python
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = np.array([[2,2,3,4]])
extraBias = np.ones((5000,1))
#Write codes here
extraBias1 = np.ones((5000,1))
```