Machine Learning Assignment#4 Report

1. Feedforward propagation

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

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

Fig.1 The Feedforward propagation code implemented in python

2. Backpropagation

```python
def backprop(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 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))))
    # run the feed-forward pass
    a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)
    # compute the cost
    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
    #Regulization term
    J += (float(learning_rate) / (2*m) * (np.sum(np.power(theta1[:,1:], 2)) + np.sum(np.power(theta2[:,1:], 2))))


    ...
```

Fig.2 Compute the J(loss)

```python
'''
Start back propagation
'''
#Initialize delta
delta1 = np.zeros(theta1.shape)
delta2 = np.zeros(theta2.shape)

#Compute the loss of all instances
for t in range(m):
    #column vectors, represents all the value for each instaces
    a1t = a1[t,:].T
    z2t = z2[t,:].T
    a2t = a2[t,:].T
    z3t = z3[t,:].T
    ht  = h[t,:].T
    yt  = y[t,:].T

    #Compute error terms
    d3 = (ht - yt)
    d2 = np.multiply(np.dot(theta2.T,d3)[1:,:], sigmoid_gradient(z2t))

    #Adding to grad matricies
    delta2 = delta2 + np.dot(d3, a2t.T)
    delta1 = delta1 + np.dot(d2, a1t.T)

delta2 = delta2/m
delta1 = delta1/m

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

#flatten all the grad matricies and concate them
grad = np.concatenate((delta1.flatten(), delta2.flatten()), axis=1)
grad = np.ravel(grad.T)
return J, grad
```

Fig.3 Backpropagation implemented in python

We need to modify the code in cost function for regularization to:

np.sum (np.power (theta2[:,2:], 2))

3. Accuracy

```
camel% vim 0616023.py
camel% python3.6 0616023.py
/usr/local/lib/python3.6/dist-packages/
le's documentation for alternative uses
  import imp
/usr/local/lib/python3.6/dist-packages/
ed based on the range [0, max(values)],
If you want the future behaviour and si
In case you used a LabelEncoder before
  warnings.warn(msg, FutureWarning)
accuracy = 97.22%
```

The accuracy is 97.22%, in order to improve the accuracy, we may modify the hidden size and add more dense layer.