Forward_prop:

```python
def forward_propagate(X, theta1, theta2):
    m = X.shape[0]
    #Write codes here

    X_ndarr = np.matrix(X) #(5000,400)

    bias = np.array(np.zeros(m), ndmin=2)
    X_ndarr = np.concatenate((bias.T, X_ndarr), axis=1)

    a1= X_ndarr #(5000,401)

    z2= np.matmul(a1, theta1.T) #(5000,25)

    a2= sigmoid(z2)

    a2= np.concatenate((bias.T, a2), axis=1) #(5000, 26)

    z3= np.matmul(a2, theta2.T) #(5000,10)

    h= sigmoid(z3) #(5000, 10)

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

Back_prop:

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

    #Write codes here

    theta1_b= np.matrix(np.reshape(params[:hidden_size * (input_size + 1)], (hidden_size, (input_size + 1))))
    theta2_b= np.matrix(np.reshape(params[hidden_size * (input_size + 1):], (num_labels, (hidden_size + 1))))

    delta1= np.zeros((hidden_size, input_size+1))
    delta2= np.zeros((num_labels, hidden_size+1))

    theta1= theta1_b[:, 1:]  #(25, 400)
    theta2= theta2_b[:, 1:]  #(10, 25)

    for i in range(m):

        bias= np.array([0], ndmin=2) #(1, 1))

        a1_1= np.concatenate((bias, X[i].T), axis=0) #(401*1)

        z2_2= np.matmul(theta1_b, a1_1) #(25*1)

        a2_2= np.concatenate((bias, sigmoid(z2_2)), axis=0) #(26*1)

        z3_3= np.matmul(theta2_b, a2_2) #(10*1)

        a3_3= sigmoid(z3_3) #(10*1)

        d3= np.subtract(a3_3, y[i].reshape(10, 1)) #(10*1)

        d2= np.multiply(np.matmul(theta2.T, d3), sigmoid_gradient(z2_2)) #(25*1)


        delta1= delta1 + np.matmul(d2, a1_1.T)  # (25*401)
        delta2= delta2 + np.matmul(d3, a2_2.T)  # (10*26)


    theta1_g= (1 / m) * delta1   #(25, 401)
    theta2_g= (1 / m) * delta2   # (10*26)

    theta1_g[:, 1:] = ((learning_rate / m) * theta1) + theta1_g[:, 1:]
    theta2_g[:, 1:] = ((learning_rate / m) * theta2) + theta2_g[:, 1:]


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

    grad= np.concatenate((theta1_g.flatten(), theta2_g.flatten()), axis=1)

    return J, grad
```

Accuracy:

```
harry@harry-X550JX:~/Desktop$ python3 0516072.py
/home/harry/.local/lib/python3.6/site-packages/sklearn/externals/joblib/externals/cloudpickle/cloudpickle.py:47: Deprecati
onWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
/home/harry/.local/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py:363: FutureWarning: The handling of inte
ger data will change in version 0.22. Currently, the categories are determined based on the range [0, max(values)], while
in the future they will be determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the
OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
accuracy = 99.62%
harry@harry-X550JX:~/Desktop$ python3 0516072.py
/home/harry/.local/lib/python3.6/site-packages/sklearn/externals/joblib/externals/cloudpickle/cloudpickle.py:47: Deprecati
onWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
/home/harry/.local/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py:363: FutureWarning: The handling of inte
ger data will change in version 0.22. Currently, the categories are determined based on the range [0, max(values)], while
in the future they will be determined based on the unique values.
If you want the future behaviour and silence this warning, you can specify "categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the
OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
accuracy = 99.58%
harry@harry-X550JX:~/Desktop$
```