一．训练集

x1　x2　x3　x4　Y

```
5.1 3.5 1.4 0.2 1
4.9 3.0 1.4 0.2 1
4.7 3.2 1.3 0.2 1

.    .    .   .   .

7.0 3.2 4.7 1.4 2
6.4 3.2 4.5 1.5 2
6.9 3.1 4.9 1.5 2

.    .    .   .   .

5.8 2.7 5.1 1.9 3
7.1 3.0 5.9 2.1 3
5.9 3.0 5.1 1.8 3
```

共150组数据四个输入特征，共分为1,2,3三类

二．代码

1. 数据导入函数

```python
def loadData (filename):
    data = np.loadtxt(filename)
    data = np.array(data)
    return data
```

2. 偏执插入函数

```python
def indexBia (X):
    m = np.size(X, 1)
    X = np.row_stack(((np.ones((1,m))), X))
    return X
```

3. 数据整形

```python
def reshapeData(data):
    m = np.size(data, 0)
    n = np.size(data, 1)
    print('MxN = ', m, 'X', n)
    X = np.array(data[:, 0 : n - 1])#列取值0 1 2 3， n-1 =4，左闭右开取值
    Y = np.array(data[:, n - 1]).reshape((m,1))
    Y = np.where(Y == 1, [1, 0, 0], Y)
    Y = np.where(Y == 2, [0, 1, 0], Y)
    Y = np.where(Y == 3, [0, 0, 1], Y)
    # print(Y)
    return X, Y
```

4. 激活函数

```
def sigmoid(Z):
    unitValue = 1/(1 + np.exp( -Z))
    return unitValue
```

5. 权重 theta，反向误差 Delta 初始化

```
def initparaments (L_0, L_1, L_2, feature_num, option):
    if option == 'theta':
        parament = np.random.rand((feature_num + 1) * L_0 + (L_0 + 1) * L_1
+ (L_1 + 1) * L_2)\
                *(2*INIT_EPSILON) - INIT_EPSILON
    elif option == 'delta':
        parament = np.random.rand((feature_num + 1) * L_0 + (L_0 + 1) * L_1
+ (L_1 + 1) * L_2)
    else:
        parament = np.zeros((feature_num + 1) * L_0 + (L_0 + 1) * L_1 + (L_1
+ 1) * L_2)

    parament_0 = parament[0 : L_0 * (feature_num + 1)].reshape((L_0,
(feature_num + 1)))

    parament_1 = parament[L_0 * (feature_num + 1) : \
                    (L_0 * (feature_num + 1) + (L_0 + 1) *
L_1)].reshape((L_1, L_0 + 1))

    parament_2 = parament[(L_0 * (feature_num + 1) + (L_0 + 1) * L_1) : \
                    ((feature_num + 1) * L_0 + (L_0 + 1) * L_1 + (L_1 + 1) *
L_2)].reshape((L_2, L_1 + 1)) #拆分

    return np.array([parament_0, parament_1, parament_2])
```

6. 前向传导

```
def forwardProp (ctheta, clayerUint):
        clayerUint = indexBia(clayerUint)
        Z = np.dot(ctheta, clayerUint)
        tlayerUint = sigmoid(Z)
        return tlayerUint, Z
```

7. 反向传导

```
def backwardProp (ttheta, clayerDelta, tunitValue):
    m, n = np.shape(tunitValue)
    tunitValue = indexBia(tunitValue)
    tlayerDelta =  np.multiply(np.dot(ttheta.T, clayerDelta), np.multiply\
                                                            (tunitValue,
(np.ones(( (m + 1), n)) - tunitValue)))
    return tlayerDelta
```

8. 利用反传计算各层反传误差 Delta

```
def computDelta (theta, unitValue, Y):
    delta2 = unitValue[2] - Y
    delta1 = backwardProp(theta[2], delta2, unitValue[1])
    delta0 = backwardProp(theta[1], delta1[1:, :], unitValue[0])
    return np.array([delta0, delta1, delta2])
```

9. 通过各层反传误差求总误差对于各层权重的偏导

```
def computeGradient (Delta, unitValue, theta, tempDelta, LAMBDA, X):
    X = indexBia(X)
    unitValue[0] = indexBia(unitValue[0])
    unitValue[1] = indexBia(unitValue[1])
    n, m = np.shape(X)
    Delta0 = Delta[0] + np.dot(tempDelta[0][1:, :], X.T)
    Delta1 = Delta[1] + np.dot(tempDelta[1][1:, :], unitValue[0].T)
    Delta2 = Delta[2] + np.dot(tempDelta[2], unitValue[1].T)
    D0 = Delta0/m + LAMBDA*theta[0]
    D1 = Delta1/m + LAMBDA*theta[1]
    D2 = Delta2/m + LAMBDA*theta[2]
    D0[:, 0] = Delta0[:, 0]/m
    D1[:, 0] = Delta1[:, 0]/m
    D2[:, 0] = Delta2[:, 0]/m
    return np.array([Delta0, Delta1, Delta2]), np.array([D0, D1, D2])
```

10. 总的误差函数

```
def costFunction(unitValue, Y, theta):
    n, m = np.shape(unitValue)
    A = (np.concatenate((theta[0].reshape(20),theta[1].reshape(30),\
                      theta[2].reshape(21)))).flatten() #重构
    cost = -(np.sum(np.multiply(Y.T, np.log(unitValue)) + \
        np.multiply((1 - Y).T, np.log(1 - unitValue)))) / m
    return cost
```

11. 权重保存函数

```
def saveArray(filename, data):
    file = open(filename, 'wb')
    np.save(file, theta)
    file.close()
    file = open(filename, 'rb')
    save_data = np.load(file)
    file.close()
    print(save_data,'\n',
          filename,'\n',
          'SAVED IN CURRENT DIRECTORY!!','\n')
```

12. 测试函数

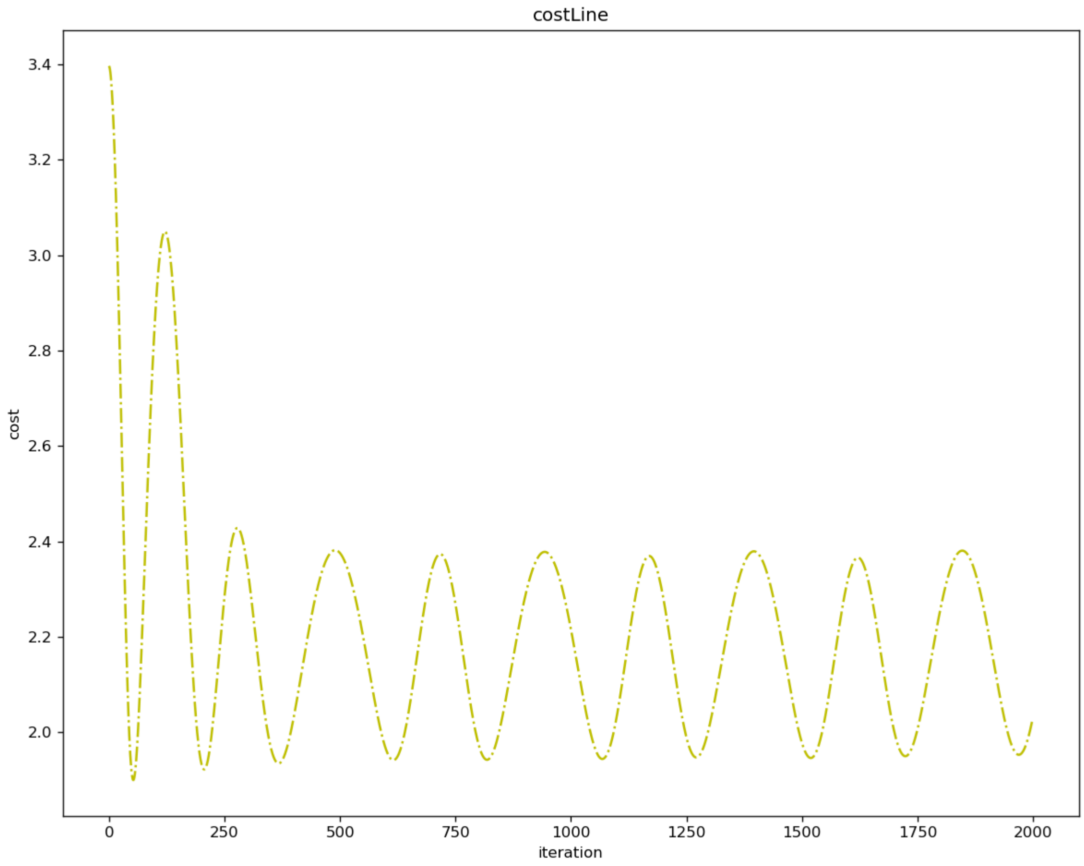
```
def weightsCheckOut(weight, X):
    unitValue, z = computUnitValue(weight, X.T)
    return unitValue[2]
```

13. 主函数

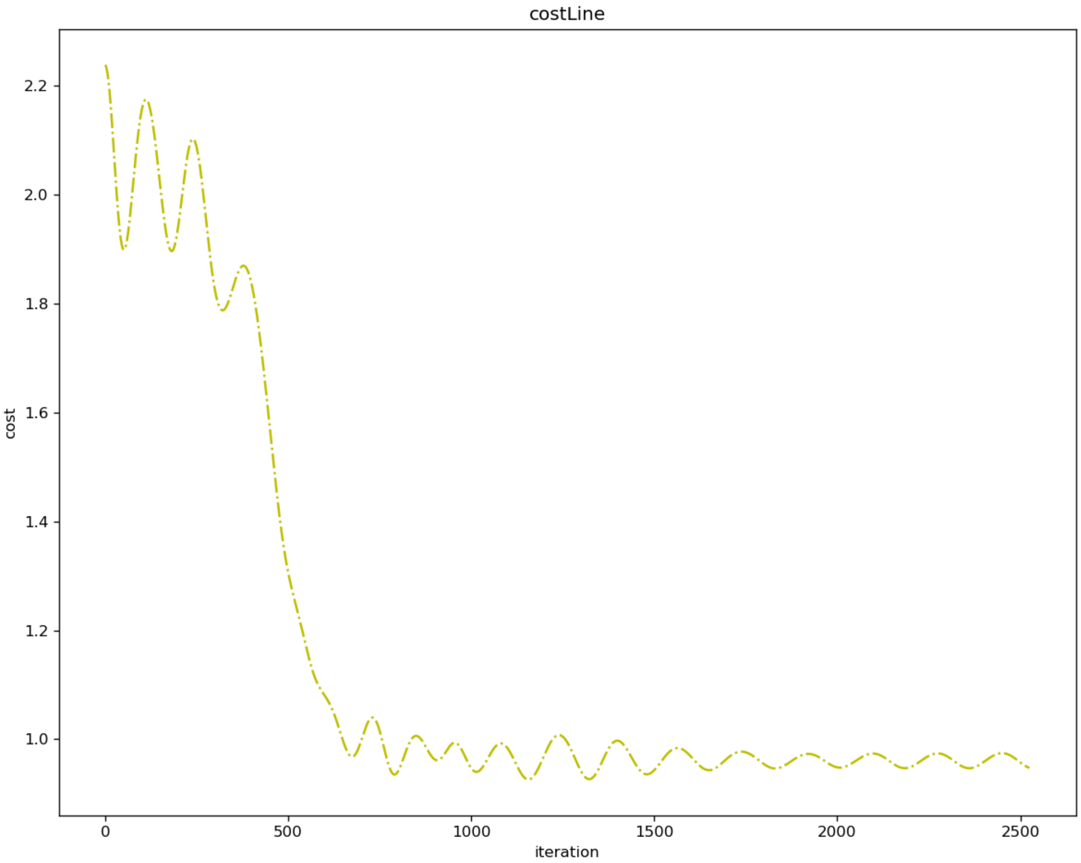
```
if __name__ == '__main__':
#网络参数初始化
    Layer_num = 3
    Unit_L0 = 4
    Unit_L1 = 6
    Unit_L2 = 3
    INIT_EPSILON = 1
    LAMBDA = 2.56
    ALPHA = 0.001
    ITERATION = 15000

#网络数据初始化
    load_data = loadData('setdata.txt')
    cost = []
    X, Y = reshapeData(load_data)
    feature_num = np.size(X.T, 0)
    theta = initparaments(Unit_L0, Unit_L1, Unit_L2, feature_num, 'theta')
    Delta = initparaments(Unit_L0, Unit_L1, Unit_L2, feature_num, 'delta')

#开始训练
    for i in range(ITERATION):
        unitValue, z = computUnitValue(theta, X.T)
        tempDelta = computDelta(theta, unitValue, Y.T)
        Delta, D = computeGradient(Delta, unitValue, theta, tempDelta,
LAMBDA, X.T)
        theta = theta - ALPHA*D
        cost = np.append(cost, costFunction(unitValue[2], Y, theta))

#保存训练好的权重
    saveArray('Theta.npy', theta)
    np.savetxt('theta0.txt', theta[0])
    np.savetxt('theta1.txt', theta[1])
    np.savetxt('theta2.txt', theta[2])

#载入需要测试的权重
    weights = np.load('Theta.npy')#
    test_data = loadData('test.txt')
    X, Y = reshapeData(test_data)
    result = weightsCheckOut(weights, X)
    print(np.around(np.column_stack((result.T,Y)), decimals=2))

#绘制误差图
    fig = plt.figure(1, figsize=(10, 8), dpi=120)
    chart = fig.add_subplot(1, 1, 1)
    x1 = np.arange(0,ITERATION, 1)
    x2 = cost
    costLine = chart.plot(x1,x2,c = 'y', linestyle = '-.')
    plt.xlabel('iteration')
    plt.ylabel('cost')
    plt.title('costLine')
    plt.savefig("costLine.png")
    plt.show()
```

三．误差图

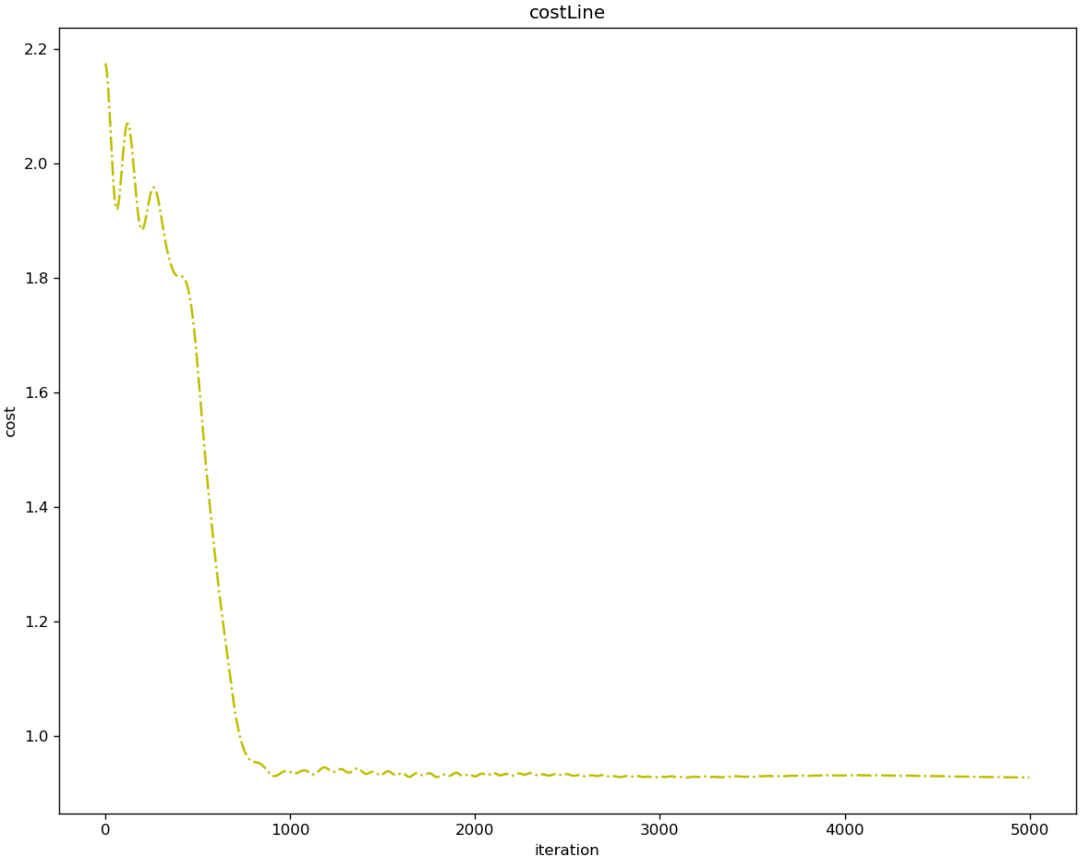
1. 欠拟合形式 1 的 cost 图



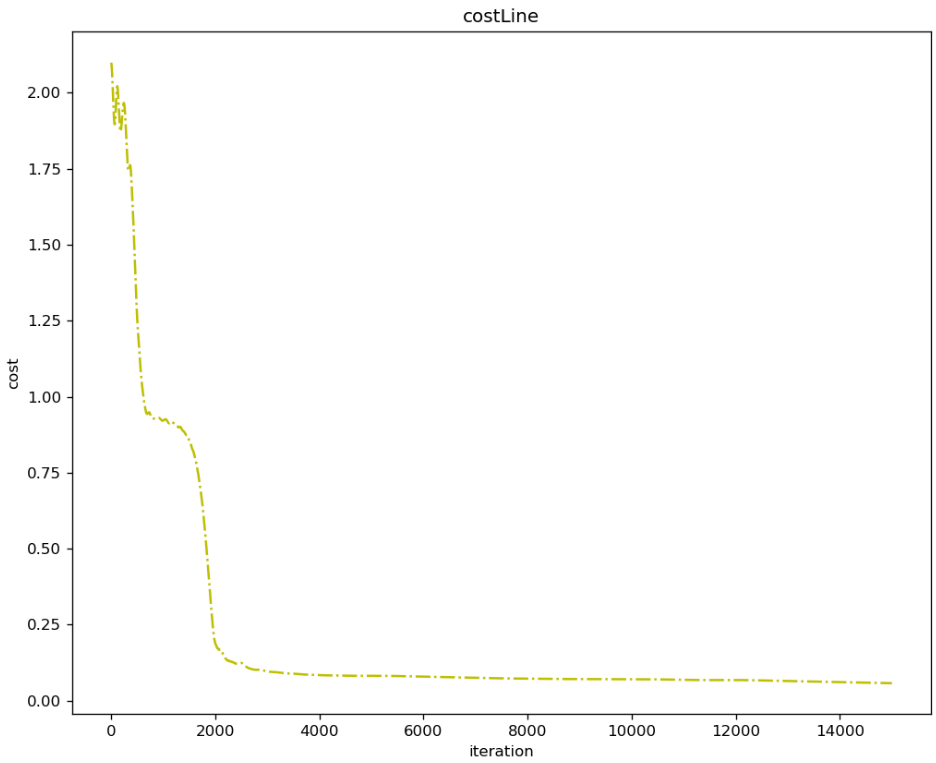
2. 欠拟合形式 2 的 cost 图



3.　欠拟合形式 3 的 cost 图



4. 正确预测的 cost 图



四．测试结果

观测方法：做三列是训练的权重的预测结果，右三列是答案

1. 欠拟合形式 1

```
[[1. 0. 0. 1. 0. 0.]
.................
[1. 0. 0. 0. 1. 0.]
.................
[1. 0. 0. 0. 0. 1.]]
```

2. 欠拟合形式 2

```
[[0.33 0.33 0.33 1.   0.   0.  ]
..................................
[0.33 0.33 0.33 0.   1.   0.  ]
..................................
[0.33 0.33 0.33 0.   0.   1.  ]]
```

3. 欠拟合形式 3

```
[[1.   0.   0.   1.   0.   0.  ]
..................................
[0.   0.49 0.47 0.   1.   0.  ]
..................................
[0.   0.49 0.47 0.   0.   1.  ]]
```

4. 准确拟合

```
[[ 1.     0.     0.      1.     0.     0.    ]
  [0.     1.     0.      0.     1.     0.    ]
  [0.     0.01 1.      0.     0.     1.    ]
  [0.     0.01 0.99   0.     0.     1.    ]
  [0.     0.01 0.99   0.     0.     1.    ]
  [0.     0.01 1.      0.     0.     1.    ]
  [0.     0.01 1.      0.     0.     1.    ]
  [0.     0.01 1.      0.     0.     1.    ]
  [0.     0.02 0.99   0.     0.     1.    ]
  [0.     0.02 0.99   0.     0.     1.    ]
  [0.     0.01 1.      0.     0.     1.    ]
  [0.     0.04 0.95   0.     0.     1.    ]]
```