# 机器学习学习报告

周珊琳

上海电力大学
上海, 2019-07-23, 中国

## 第一节　算法概述

算法名称：BP（Back Propagation）算法

算法原理：BP（Back Propagation）算法也叫反向传播算法，使用这种算法的神经网络称为 BP 神经网络。它由前向传递和反向传播所组成，它的前向传递过程与前馈神经网络相似，大致上都是模仿了人脑的视觉皮层。因此 BP 神经网络实际上就是在前馈神经网络的基础上加上了反向传播的功能，即将前馈最终得到的预测值与实际值之间的差值以误差的形式告知之前经过的隐含层，误差通过数学公式自后向前传递由此来更新神经元上的权重，让误差梯度下降，最终反复训练得到最小误差及其相对应的模型。所谓的 BP 只是一种形式即有反馈的算法就是 BP 算法。

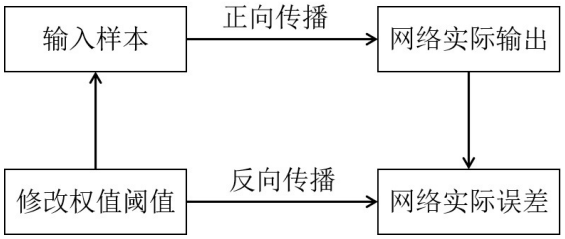


图 1: BP 流程

相关公式：

$$\hat{y}_j^k = f(\beta_j - \theta_j) \tag{1.1}$$

$$g_i = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \tag{1.2}$$

$$e_h = b_h(1 - b_h)\sum_{j=1}^{l} w_{hj}g_j \tag{1.3}$$

$$\triangle w_{hj} = \eta g_j b_h \tag{1.4}$$

$$\triangle \theta_j = -\eta g_j \tag{1.5}$$

$$\triangle v_{ih} = \eta e_h x_i \tag{1.6}$$

$$\triangle \gamma_h = -\eta e_h \tag{1.7}$$

## 第二节　算法设计

### 2.1　算法流程

**Algorithm 1** k-means 算法

输入：训练集 $D = (x_k, y_k)_{k=1}^{m}$, 学习率 $\eta$

输出：连接权和阈值确定的多层前馈神经网络

过程：

在 (0,1) 范围内随机初始化网络中的所有连接权和阈值
**repeat**
    **for** all $(x_k, y_k) \in D$ **do**
        根据当前参数和式 (1.1) 计算当前样本的输出
        根据式 (1.2) 计算输出层神经元的梯度项 $g_j$
        根据式 (1.3) 计算隐层神经元的梯度项 $e_h$
        根据式 (1.4)-(1.8) 更新连接权 $w_{hj}$，$v_{ih}$ 与阈值 $\theta_j$，$\gamma_h$
    **end for**
**until** 达到停止条件

### 2.2　核心代码

源代码 1:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug 1 18:40:31 2019

@author: zsl
"""

import numpy as np
import matplotlib.pyplot as plt
import math
```

```
import datetime
from sklearn.metrics import classification_report
from sklearn.datasets import load_iris,load_wine
from sklearn import preprocessing

def process_data(data):
    min_max_scaler =preprocessing.MinMaxScaler()
    return min_max_scaler.fit_transform(data)
#iris,wine
def loadData(data):
    #load
    dataSet =data.data
    target =data.target
    #shuffle
    num_example=dataSet.shape[0]
    array =np.arange(num_example)
    np.random.shuffle(array)
    dataSet =process_data(dataSet)
    dataSet =dataSet[array]
    target =target[array]

    return dataSet,target

##iris
data =load_iris()
data, label=loadData(data)

#wine
data =load_wine()
data, label=loadData(data)

print('=====read===ok======')

num_example =data.shape[0]
# 训练集：验证集 = 7：3，考虑到样本较少，验证集的结果可以反映测试集结果
sample =np.int(num_example *0.7)
x_train =data[: sample]
y_train =label[: sample]
x_test =data[sample:]
y_test =label[sample:]

#激活函数
def sigmoid(x):
    if type(x)!=np.ndarray:
       return 1/(1+math.exp(-x))
    return 1/(1+np.exp(-x))

#激活函数的偏导数
def sigDer(x):
    return sigmoid(x)*(1-sigmoid(x))


#神经网络前向传播过程
def forward(x,w1,w2):
```

```
    #layer1
    hin1 =x.dot(w1)
    hout1 =sigmoid(hin1)
    #layer2
    # 将h层数据传到o层，并计算输出 将输出的o层激活
    oin =hout1.dot(w2)
    out =sigmoid(oin)

    return out


#定义网络初始参数
'''
N:批量样本数
layer1_in:输入向量维数，样本特征数
layer_out:输出向量维数
'''
if __name__ =="__main__":

    starttime =datetime.datetime.now()
#    iris
#    N, layer1_in, layer2_in, layer_out = 10, 4,8,1
#    wine
    N, layer1_in, layer2_in, layer_out =10, 13,52,1
    #参数初始化，正态分布
    w1 =np.random.normal(size=[layer1_in,layer2_in])
    w2 =np.random.normal(size=[layer2_in,layer_out])


    #存放损失数据
    losses =[]
    #学习速率以 0.01 ~ 0.001 为宜。
    learning_rate =0.001
    flag =0
    for step in range(50):
        if flag:

            # 计算h层输出,将输出的h层激活
            #layer1
            Houter1=np.zeros(shape=(np.shape(x_train)[0],np.shape(w1)[1]))
            i =0
            for line in x_train:
                hin1 =line.dot(w1)
                hout1 =sigmoid(hin1)
                Houter1[i]=hout1
                i +=1

            # 将h层数据传到o层，并计算输出 将输出的o层激活
            Y_=np.zeros(shape=(np.shape(x_train)[0],np.shape(w2)[1]))
            i =0
            for line in Houter1:
                oin =line.dot(w2)
                out =sigmoid(oin)
                Y_[i]=out
```

```
            i +=1

        #扁平化处理
        y_tr=np.reshape(y_train, [np.shape(y_train)[0], -1])

        print(round((np.square(Y_ -y_tr).sum())/N, 6))
        if step % 1 ==0:
            losses.append(round((np.square(Y_ -y_tr).sum())/N, 6))

        #计算w2的梯度损失
        grad_y_pred =(2 *(Y_ -y_tr))
        grad_o_sig =sigDer(out)
        grad_o =grad_y_pred *grad_o_sig
        grad_w2 =Houter1.T.dot(grad_o)

        #计算w1的梯度损失
        grad_h_relu =grad_y_pred.dot(w2.T)
        grad_h_sig =sigDer(hout1)
        grad_h =grad_h_relu *grad_h_sig
        grad_w1 =x_train.T.dot(grad_h)


        # 更新梯度
        w1 -=learning_rate *grad_w1
        w2 -=learning_rate *grad_w2

    flag =1
    #使用训练集进行测试
    right_n =0
    all_n =len(x_train)
    for i in range(all_n):
        Y_ =forward(x_train[i],w1,w2)
        if np.argmax(Y_)==np.argmax(y_train[i]):
            right_n=right_n+1

    print("总数---"+str(all_n))
    print("正确个数"+str(right_n))
    print("准确率----")
    print(float(right_n/all_n))

#使用测试集进行测试
pre =[]
right_n =0
all_n =len(x_test)
for i in range(len(x_test)):
    Y_ =forward(x_test[i],w1,w2)
    pre.append(np.argmax(y_test[i]))
#       取元素最大值所对应的索引
    if np.argmax(Y_)==np.argmax(y_test[i]):
        right_n=right_n+1

print("总数---"+str(all_n))
print("正确个数"+str(right_n))
print("准确率----")
```

```
173     print(float(right_n/all_n))
174
175
176     plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
177     plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
178     print(losses)
179     plt.title('wine训练过程中loss变化情况')
180     plt.plot(losses)
181     plt.show()
182
183     print (classification_report(y_test, pre))
184     endtime =datetime.datetime.now()
185     print ("time",endtime -starttime)
```

## 第三节　选用数据

iris 行数：150 列数：5

列属性及取值:

1) 萼片长度 cm，数值型

2) 萼片宽度 cm，数值型

3) 花瓣长度 cm，数值型

4) 花瓣宽度 cm 数值型

类别：

Iris Setosa

Iris Versicolour

Iris Virginica

wine，行数：178，列数：13

属性：

1) Alcohol

2) Malic acid

3) Ash

4) Alcalinity of ash

5) Magnesium

6) Total phenols

7) Flavanoids

8) Nonflavanoid phenols

9) Proanthocyanins

10) Color intensity

11) Hue

12) OD280/OD315 of diluted wines

13) Proline

类别：

Alcohol 1，2，3

## 第四节　实验结果展示

在对 iris 数据集进行实验时，使用 4/12/1，学习率 0.001，30 发现实验结果较为理想。

在进行 wine 实验时发现，使用 13/52/1，学习率 0.001，进行 30 次实验，结果表现为震荡，为找到其原因。对从算法与数据两个方面进行探究。

从算法参数上，进行了调整训练次数、学习率、隐含层节点数等方面进行调整，结果表现不理想，仍旧震荡。从数据上，进行归一化处理，结果表现震荡消失，loss 变化表现为趋势下降的光滑曲线。为进一步改良算法，对之前的算法参数又做了调整和对比。

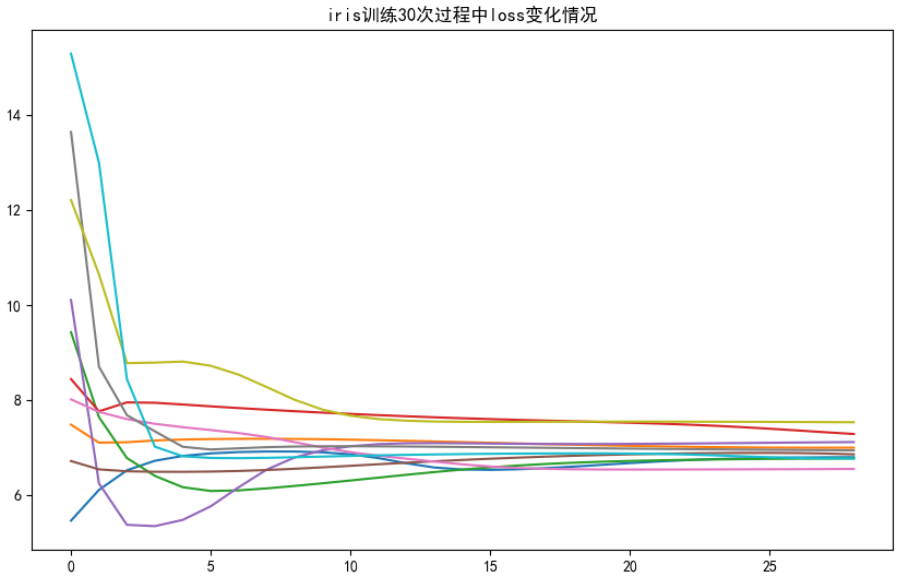


图 2: iris 训练 30 次过程 loss 变化图

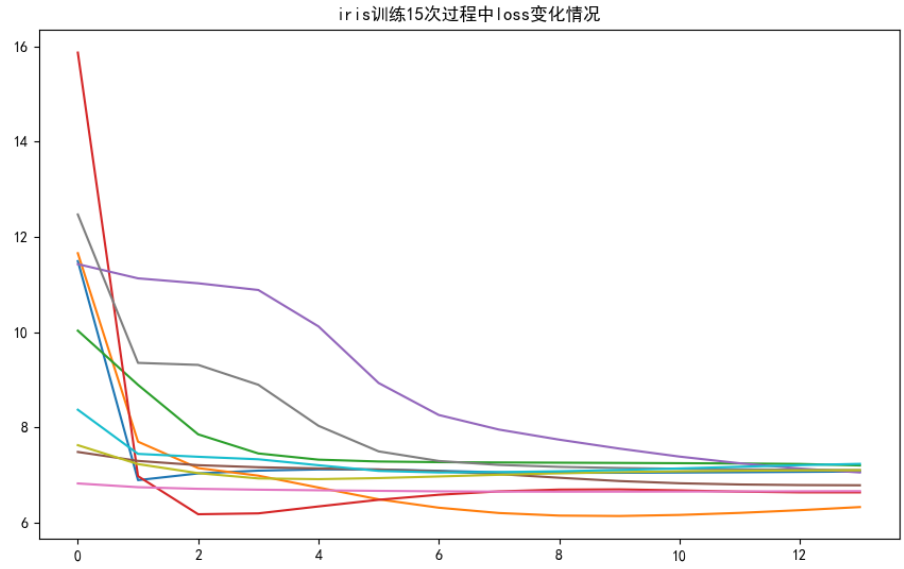


图 3: iris 训练 15 次过程 loss 变化图

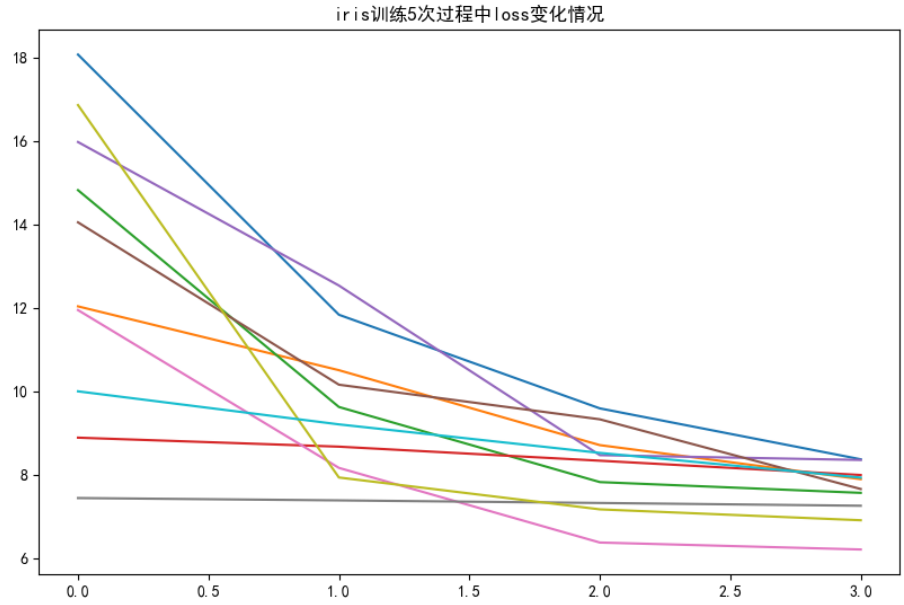


图 4: iris 训练 5 次过程 loss 变化图

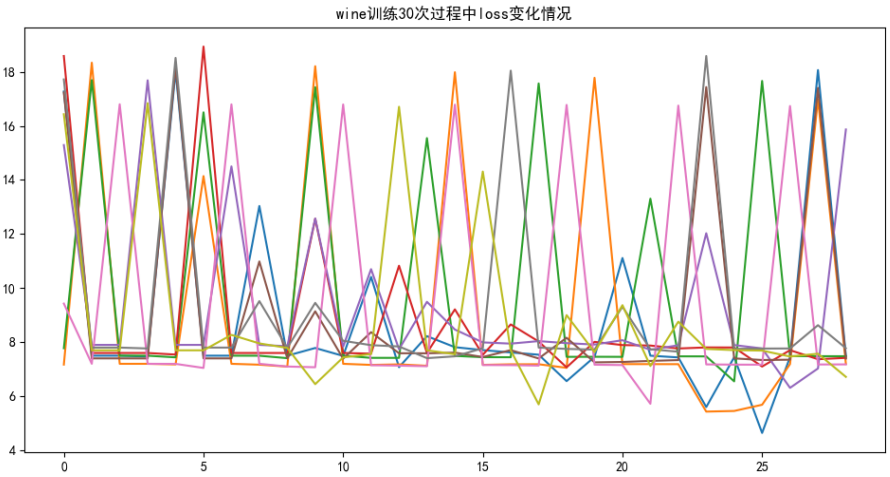


图 5: wine 训练 30 次过程 loss 变化图

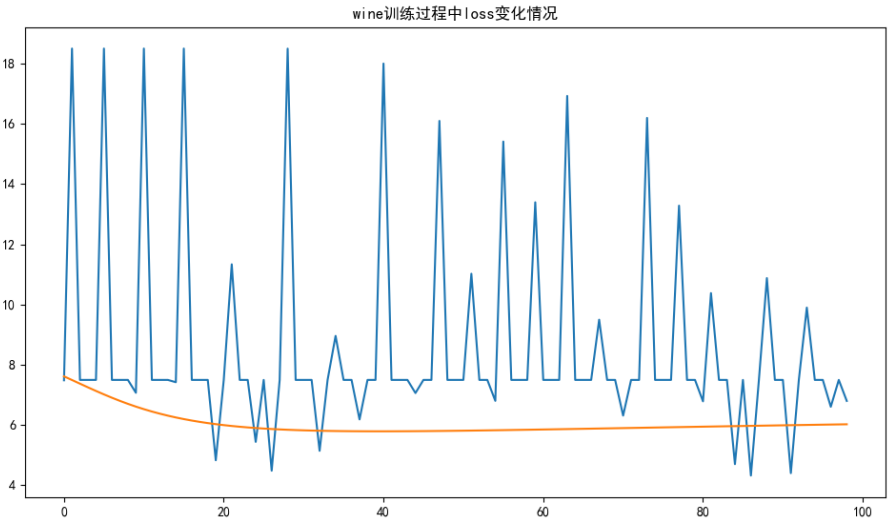


图 6: wine 数据归一化前后，训练 100 次过程 loss 变化图

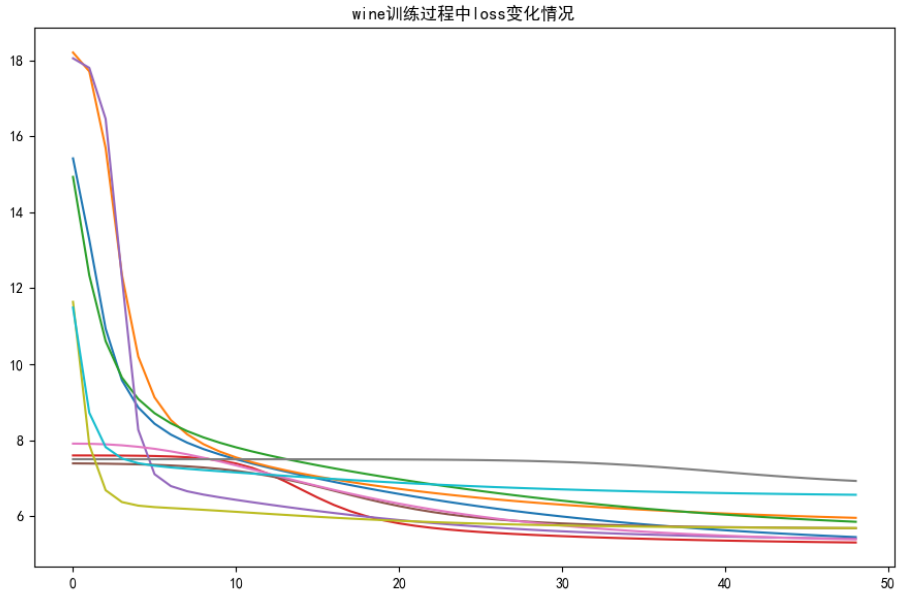


图 7: wine 数据归一化后训练 50 次过程 loss 变化图

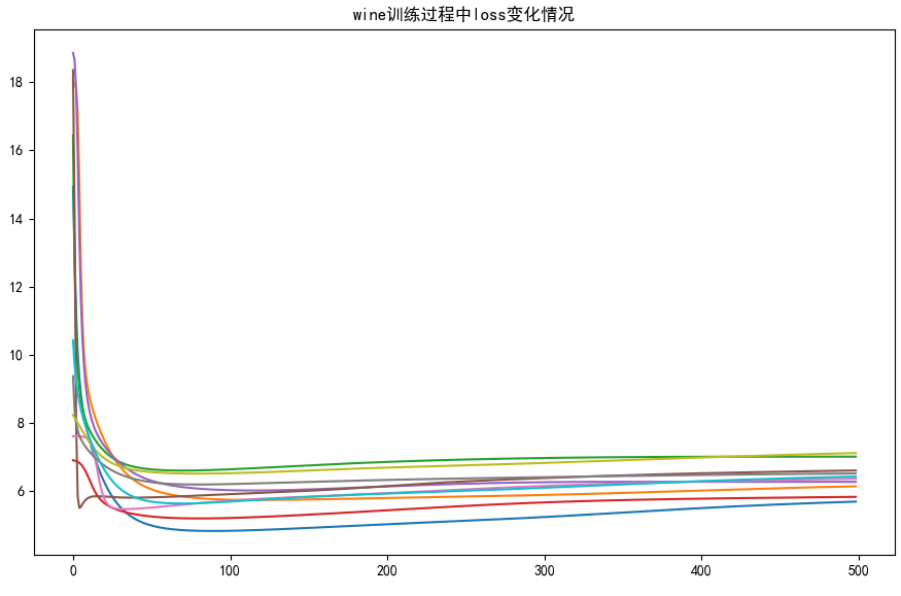


图 8: wine 数据归一化后训练 500 次过程 loss 变化图

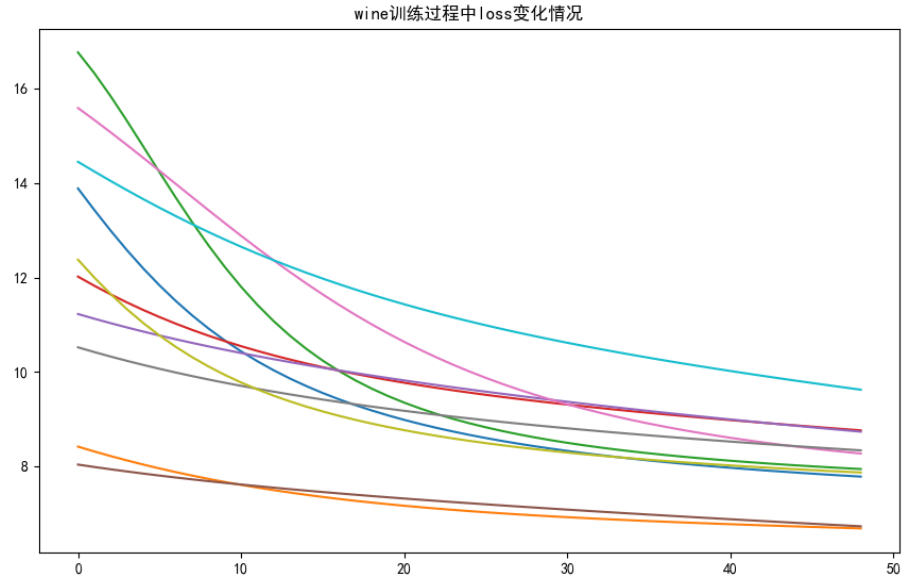


图 9: 4 个神经元训练 50 次过程 loss 变化图

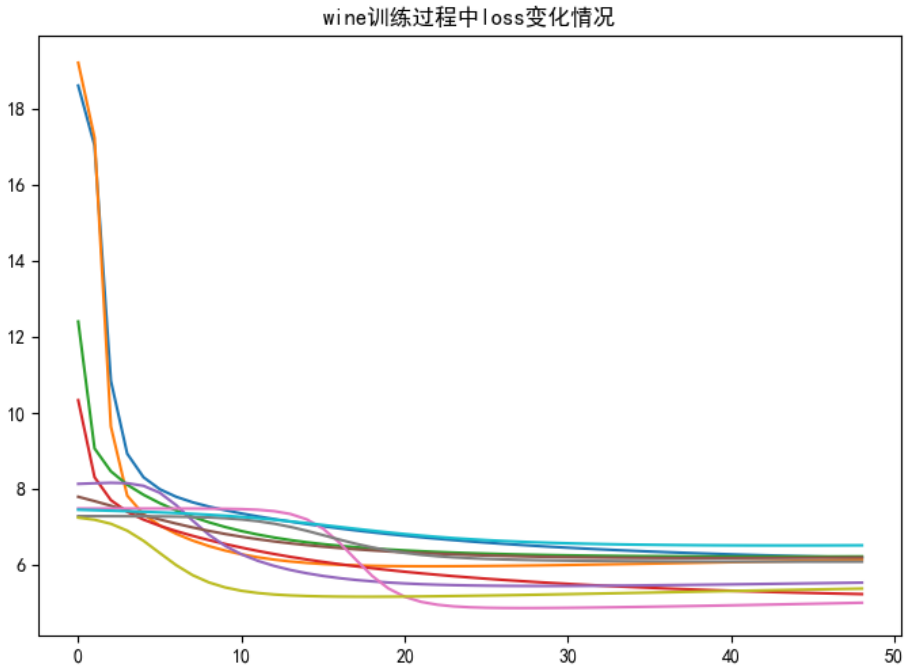


图 10: 100 个神经元训练 50 次过程 loss 变化图

## 第五节　评价方法

准确率：准确率是指在给定的测试数据集上进行模型训练，模型能够正确分类的样本数和总样本数之比。

## 第六节　实验分析和比较

猜测是数据集比较简单，因此最终得到的准确率均为 100，有点诡异。具体的比较内容在实验结果展示中已经提及。BP 神经网络有强非线性能力和柔性的网络结构。其层数和各层的神经元的个数还可以根据具体情况任意设定。然而，这种网络结构也存在着一定的缺陷，如学习速度缓慢，学习的过程中易陷入局部极小值等。值得注意的是，学习速度指的是训练的轮数，它并不等于训练时间。在面对一个简单的问题时，这种网络可能也需要几百甚至上千次的学习才能收敛。

另外，在一定范围内看似得到了一个误差最小的训练模型，实际上并不是误差最小，这种情况称为陷入局部极小值。如何加速收敛速度以及如何尽可能的避免陷入局部极小值，这些是目前对 BP 的改进研究最多的地方。