

# 《机器学习基础》实验报告

## 一、实验目的

掌握 BP 算法原理并编程实践。

## 二、实验项目内容

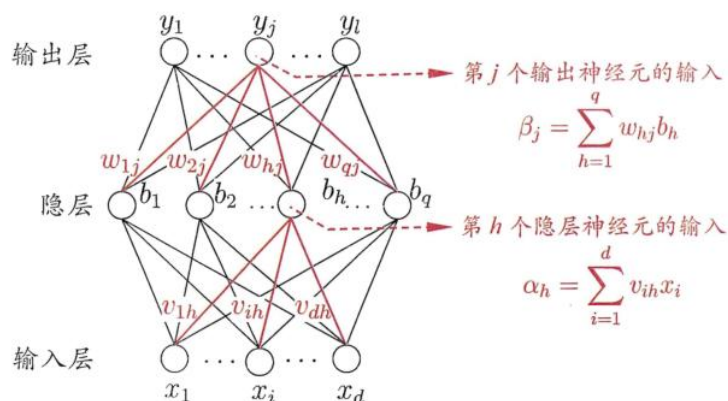
1. 理解并描述 BP 算法原理；
2. 编程实践算法并将其应用于分类任务，要求算法至少用于 iris\_data 以及 winequality\_data 两个数据集，实现对鸢尾花类别以及红酒质量标签的分类；
3. 结果分析要求使用统一的评价指标（分类任务的好坏）和图表结果的规范化（图例，坐标轴标签等）；
4. 可以体现探索性尝试：
  - 实验目标的深入，基础要求线性二分类的基础上进行问题难度的加深；
  - 数据探索深入（数据预处理，哑变量构建，分类特征的选择）；
  - 方法上的深入：不同优化方法间的效果对比，同一方法不同训练方法以及不同超参数的效果对比。

## 三、实验过程或算法（源程序）

### 1. 神经网络模型

实验一使用对数几率回归模型解决了特征数较少情况下的二分类问题，本质上只能解决线性分类。当特征数增多，模型变得复杂，线性分类就不再能够满足样本的分类需要。为了适应更加复杂的模型分类，有人在神经元模型和感知机的基础上，提出了神经网络模型。

神经网络模型在输入层和输出层之间加入了隐藏层，对输入数据做了空间变换，将其从线性不可分转换为近似的线性可分。根据输入数据的不同复杂程度（本实验中即为特征数量），可以选择不同层数的隐藏层，来更好地拟合复杂函数。如图为一个只含一层隐含层的神经网络模型：



假设样本的数据特征有  $d$  类，分类标签有  $l$  类，则构建一个含有  $d$  个神经元输入层， $l$  个神经元输出层的前馈神经网络。为了使网络更好地拟合样本，加入了一层含有  $q$  个神经元隐藏层（ $q$  为超参数，可以人为设置）。

模型中输入层->隐藏层间的连接权值为  $v_{ih}$ ，隐藏层第  $h$  个神经元的阈值为  $\gamma_h$ 。对于输入层的输入，隐藏层第  $h$  个神经元接收的输入为：

$$\alpha_h = \sum_{i=1}^d v_{ih} * x_i$$

设激活函数为  $f(x)$ ，则隐藏层第  $h$  个神经元向输出层的输出为  $b_h = f(\alpha_h - \gamma_h)$ 。隐藏层的激活函数一般使用 **Sigmoid** 函数。

模型中隐藏层->输出层的连接权值为  $w_{hj}$ ，输出层第  $j$  个神经元的阈值为  $\theta_j$ 。对于隐藏层的输入，输出层第  $j$  个神经元接收的输入为：

$$\beta_j = \sum_{h=1}^q w_{hj} * b_h$$

设激活函数为  $g(x)$ ，则隐藏层第  $h$  个神经元向输出层的输出为  $y_j = g(\beta_j - \theta_j)$ 。对于多分类问题，输出层的激活函数一般使用 **Softmax** 函数。但由于 **Softmax** 激活在反向传播时导数不连续，导致损失函数上下波动，有可能不收敛。因此还是使用 **Sigmoid** 函数，取预测结果的最大值作为最终的分

类结果，仍然可以较好的实现多分类。

每经过一次前向传播，可以得到模型的损失函数：

$$E = \frac{1}{2} \sum_{j=1}^l (\hat{y}_j - y_j)^2$$

其中  $\hat{y}_j$  为输出层第  $j$  个神经元的预测值， $y_j$  为其真实标签。

## 2. BP 算法原理

前馈神经网络直接的连接权重和神经元的阈值并没有给定，需要根据训练样本训练得到。通过逆向传播算法迭代学习参数是比较成功的训练多层前馈神经网络的学习方法。

逆向传播算法就是使用梯度下降法，通过对损失函数求导，反向更新参数的过程。BP 算法的过程推导此处不加赘述，每次迭代对参数的调整如下：

$$w_{hj} = w_{hj} - \eta * b_h * g_j$$

$$\theta_j = \theta_j + \eta * g_j$$

$$v_{ih} = v_{ih} - \eta * e_h * x_i$$

$$\gamma_h = \gamma_h + \eta * e_h$$

其中：

$$g_j = \hat{y}_j(1 - \hat{y}_j)(y_j - \hat{y}_j)$$

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

以上便是神经网络通过逆向传播算法迭代学习参数的过程，不断迭代即可逐渐收敛。由于上述过程是基于一个样本的损失函数推导的，因此每轮迭代时需要在样本集中遍历，以保证每个样本都用上。

## 3. 实验过程

（以下分析步骤只展示主要功能代码，完整代码见附录）

- 反向传播神经网络模型 BPNN.py

(1) 设计 BPNN 类时，定义其属性如下：

```
class BPNN(object):
    def __init__(self,X,Y,q,l,epoch_times=1000,Eta=0.001,visible=False):
        '''args:
            X(n*d):样本集属性的np数组,包含n个样本,每个样本包含d个属性
            Y(n*1):样本集标签的np数组
            gama:隐藏层神经元的阈值[1*q]
            theta:输出层神经元的阈值[1*1]
            n:样本数
            d:属性数
            q:隐藏层中神经元数
            l:分类数
            v:输入层->隐藏层的权重向量[(d+1)*q]
            w:隐藏层->输出层的权重向量[q*1]
            epoch_times:梯度下降法中的迭代次数
            Eta:梯度下降法中的学习率
            visible:训练过程是否可视化
        '''
        self.X=X
        self.Y=Y
        self.n,self.d=self.X.shape
        self.q=q
        self.l=l
        self.v,self.w=self.Init_wgt()
        self.gama,self.theta=self.Init_threshold()
        self.epoch_times=epoch_times
        self.Eta=Eta
        self.visible=visible
```

(2) 定义 BPNN 类的辅助函数，用于扩展输入的一个样本 X、初始化模型的权重矩阵 v、w 和各神经元的阈值  $\gamma$ 、 $\theta$ ：

```
def Add_bias(self,X):
    #对X(1*d)添加偏置项->X_new(1*(d+1))
    X_new=np.ones((X.shape[0]+1))
    X_new[1:]=X
    return X_new
def Init_wgt(self):
    #初始化v和w
    v=np.random.uniform(-1.0,1.0,size=(self.d+1)*self.q)
    v=v.reshape(self.d+1,self.q)
    w=np.random.uniform(-1.0,1.0,size=self.q*(self.l))
    w=w.reshape(self.q,self.l)
    return v,w
def Init_threshold(self):
    #初始化gama和theta
    gama=np.random.uniform(-1.0,1.0,size=self.q)
    gama=gama.reshape(1,self.q)
    theta=np.random.uniform(-1.0,1.0,size=self.l)
    theta=theta.reshape(1,self.l)
    return gama,theta
```

这三个函数需要最先定义，用于初始化成员属性。

(3) 定义 BPNN 类的激活函数，并计算其梯度，这里采用 Sigmoid：

```
def Sigmoid(self,z):          #激活函数
    return expit(z)
def Sigmoid_gradient(self,z):  #激活函数的梯度
    sg=self.Sigmoid(z)
    return sg*(1-sg)
```

(4) 定义 BPNN 类的正向传播函数，用于将输入的样本正向传播，并返回每一层的输入和输出：

```
def FP(self,X):               #将输入的训练/测试样本正向传播
    X_new=self.Add_bias(X)    #X_new:1*(d+1)
    alpha=np.matmul(X_new,self.v) #alpha:1*q
    b=self.Sigmoid(alpha-self.gama) #b:1*q
    beita=np.matmul(b,self.w)    #beita:1*1
    y=self.Sigmoid(beita-self.theta)#y:1*1
    return alpha,b,beita,y
```

这里的样本不仅是训练样本，还可以是测试样本。

(5) 定义 BPNN 类的损失函数，用于记录训练过程中损失函数的下降过程：

```
def Get_cost(self,y,label):   #损失函数
    tmp=(y-label)*(y-label)/2
    cost=np.mean(tmp)
    return cost
```

(6) 定义 BPNN 类的梯度下降过程，通过最小化损失函数来求解  $v$ 、 $w$ 、 $\gamma$ 、 $\theta$ ：

```
def GD(self,X,b,y,label):     #反向传播,修正v,w,gama,theta
    g=y*(1-y)*(y-label)       #g:1*1
    tmp1=np.matmul(g,self.w.T) #计算Σw[h][j]*g[j],tmp:1*q
    e=b*(1-b)*tmp1            #b:1*q,e:1*q
    X_new=self.Add_bias(X)

    self.w-=self.Eta*np.matmul(b.T,g)
    self.theta+=self.Eta*g
    tmp2=X_new.T.reshape(-1,1)
    self.v-=self.Eta*np.matmul(tmp2,e)
    self.gama+=self.Eta*e
    return self
```

(7) 定义 BPNN 模型对测试样本的预测分类结果：

```
def Pred(self,X):             #对输入的`测试样本集`预测分类结果
    pred=[]
    for i in range(X.shape[0]):
        alpha,b,beita,y=self.FP(X[i])
        pred_tmp=np.argmax(y,axis=-1) #预测值取最大的分类作为标签
        pred.append(pred_tmp[0])
    return pred
```

(8) 定义 BPNN 类的训练过程，由于模型推导中的损失函数是基于一个样本进行推导的，因此每轮迭代时需要在样本集中遍历，使用每个样本




进行，以保证每个样本都用上：

```
def Train(self):
    costs=[] #存储每次训练后的损失函数
    for i in range(self.epoch_times):
        if self.visible:
            sys.stderr.write("\rEpoch times: %d/%d"%(i+1,self.epoch_times)+" "*(i//((self.epoch_times//20))))
            sys.stderr.flush()
        cost_tmp=[]
        for j in range(len(self.X)):#对每个样本进行训练
            alpha,b,beita,y=self.FP(self.X[j])
            cost=self.Get_cost(y,self.Y[j])
            cost_tmp.append(cost)
            self.GD(self.X[j],b,y,self.Y[j])
        costs.append(np.mean(np.array(cost_tmp)))
    return costs
```

由于迭代次数过大时训练时间较长，为了实验者掌握模型训练进度，加入了打印进度条，将训练过程可视化。

Epoch times: 34177/100000 

Epoch times: 63831/100000 

### • 预测鸢尾花数据集 Iris.py

(1) 对于鸢尾花数据集，读取 excel 中数据，并按 7:3 划分为训练集和测试集：

```
#读取数据并将其按7:3划分为训练集和测试集
path=r"ML\实验2\data\iris_data.xlsx"
book=open_workbook(path)
sheet=book.sheets()[0] #打开sheet0
nrow=sheet.nrows #行数
ncol=sheet.ncols #列数
label_dic={'Iris-setosa':0,'Iris-versicolor':1,'Iris-virginica':2} #标签序列化
data_train=[] #训练集属性(n*d)
data_test=[] #训练集标签(n*1)
label_train=[] #测试集属性
label_test=[] #测试集标签(one-hot)
labeln_test=[] #测试集标签(分类值)
for i in range(1,nrow):
    row=sheet.row_values(i)
    one_hot=[0]*len(label_dic) #样本分类标签的one-hot向量
    label=label_dic[row[4]]
    one_hot[label]=1
    row.pop(4) #保留样本属性,删除标签
    if(i%10<7):
        data_train.append(row)
        label_train.append(one_hot)
    else:
        data_test.append(row)
        label_test.append(one_hot)
        labeln_test.append(label)
data_train=np.array(data_train)
data_test=np.array(data_test)
label_train=np.array(label_train)
label_test=np.array(label_test)
labeln_test=np.array(labeln_test)
```

(2) 实例化模型进行训练，然后使用测试样本进行测试：

```

#实例化BPNN对象并进行训练
d=len(data_train[0])          #属性数
l=len(label_dic)              #分类数
n=nrow                        #样本数
ep=10000                      #训练次数
q=10                          #隐藏层神经元数
bpnn=BPNN(data_train,label_train,q,l,ep,0.001,True)
cst=bpnn.Train()              #训练过程的损失函数
pred=bpnn.Pred(data_test)     #训练后的分类结果

```

其中 `ep` 为训练迭代次数，`q` 为隐藏层神经元数，都是超参数，其取值会影响模型的准确性。

(3) 将测试结果与真实标签进行比对，计算准确率：

```

#计算准确率
acc=0
for i in range(len(pred)):
    if(pred[i]==labeln_test[i]):
        acc+=1
acc/=len(pred)
print('\nAccuracy of classification: %.2f%%'%(acc*100))

```

(4) 为了更好的比对预测结果，将预测标签值  $y$  和真实标签值  $y_{真}$  绘制成折线图；并以 `sepal_length` 和 `sepal_width` 属性为例绘制多分类结果的散点图（鸢尾花数据集有 4 种属性，因此无法通过散点坐标将其汇入一张图）。同时，为了量化训练过程，将损失函数下降的过程记录并绘制成折线图：

```

#绘制预测值/标签值-编号的图像
plt.plot(pred,marker='+')
plt.plot(labeln_test,marker='*')
plt.ylabel('Prediction & Label')
plt.xlabel('number of sample')
plt.legend(['Pred_label','Real_Label'])
plt.show()

```

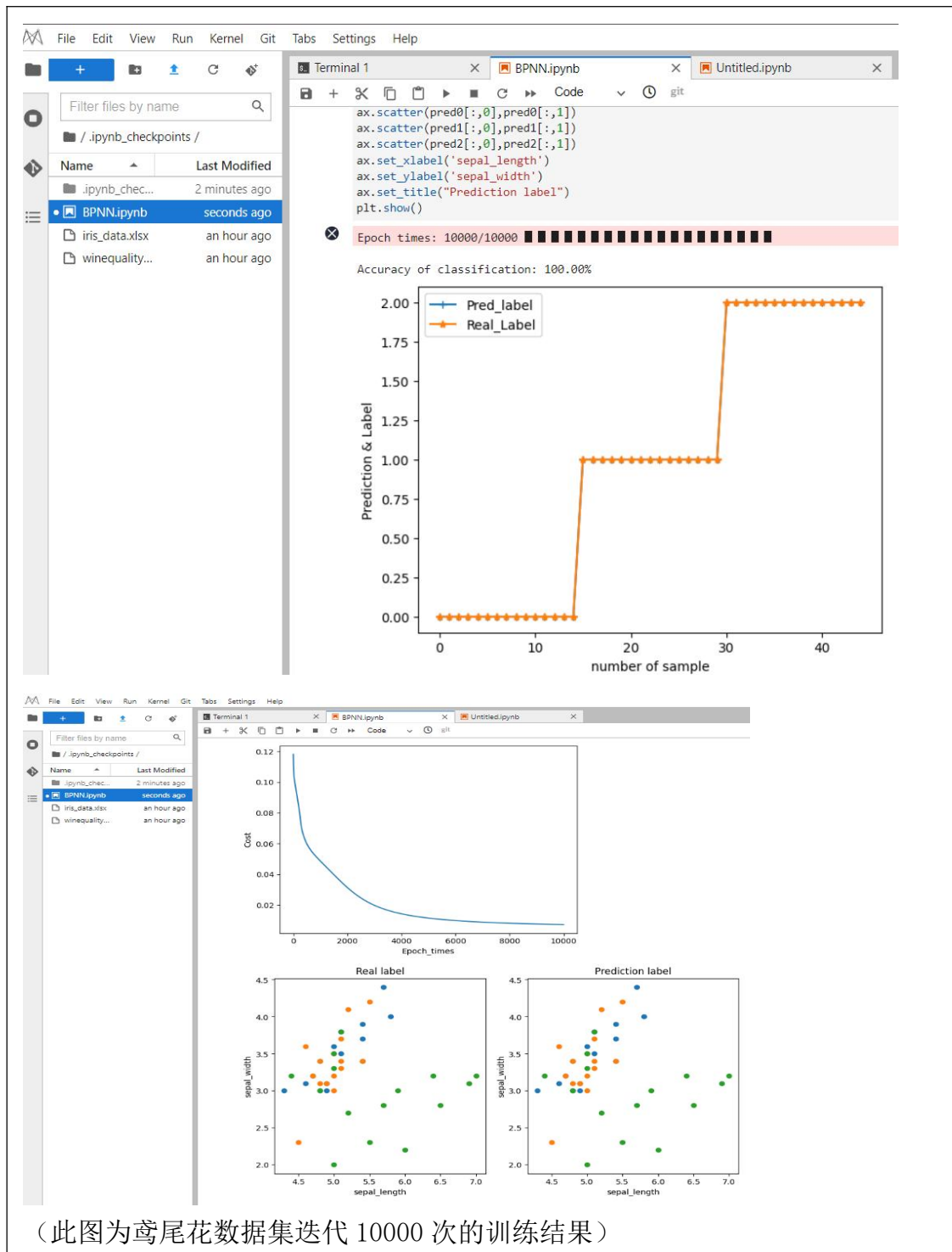
```

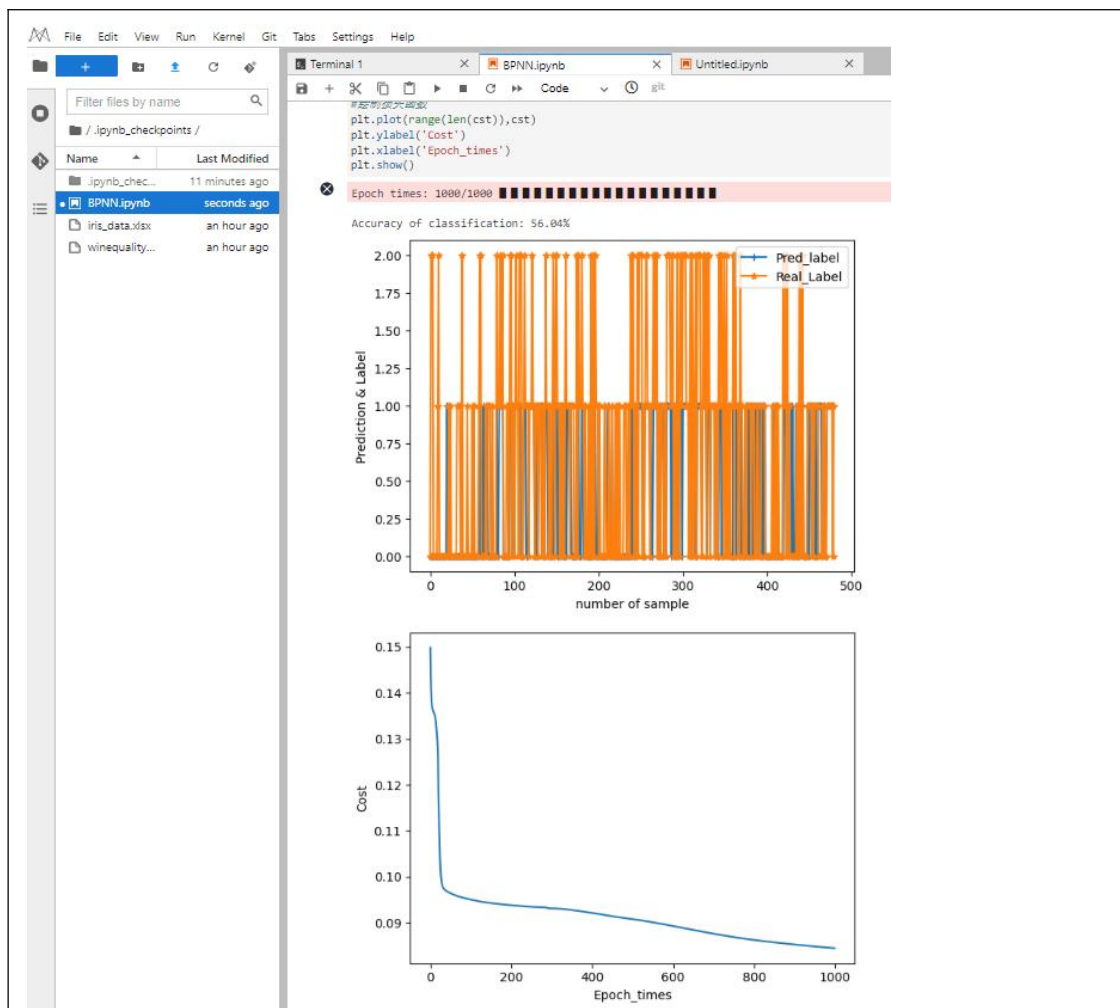
#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()

```

- 预测红酒品质数据集 `WineQuality.py`  
过程同鸢尾花数据集，此处不加赘述。

- 开发平台  
华为云 ModelArts JupyterLab:  
这次选择了 8 核 32GB 的 CPU 做训练：





（此图为红酒数据集迭代 3000 次的结果）

#### 四、实验结果及分析

##### • 鸢尾花数据集

对于**鸢尾花数据集**，因为属性值较少，隐层神经元数量影响较小。因此迭代次数 **ep** 很大程度上决定了模型的精度：

**ep=100** 时，准确率只有 **68.89%**；

**ep=1000** 时，准确率提高到 **84.44%**；

**ep=10000** 时，准确率为 **97.78%**；

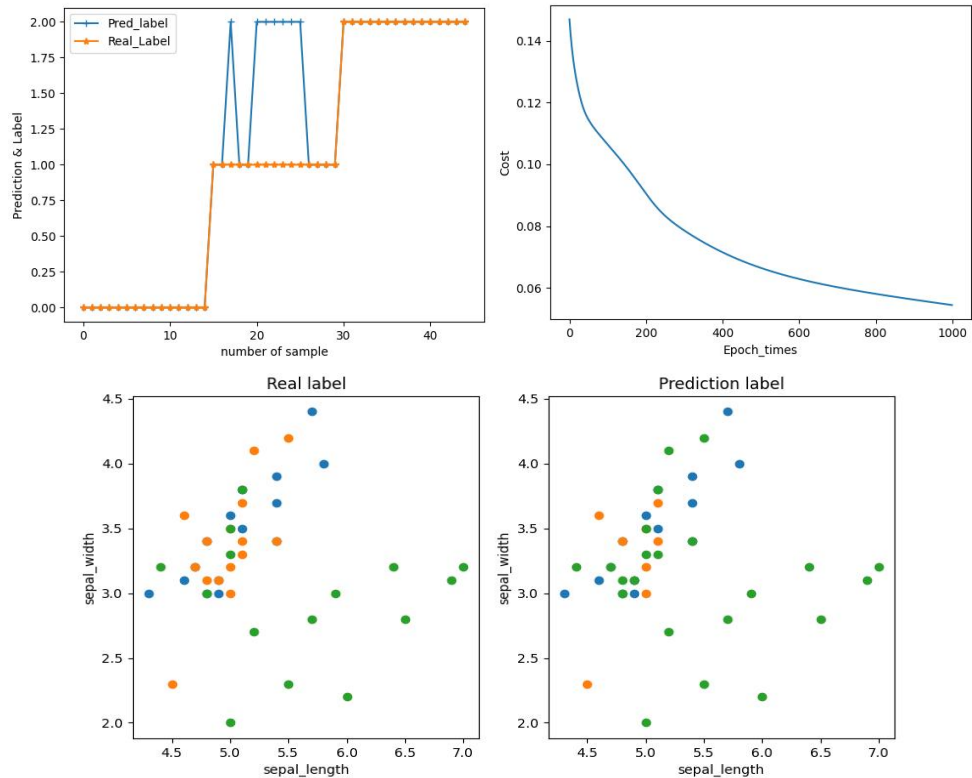
**ep=20000** 时，准确率达到 **100.00%**；

以迭代次数为 **1000** 和 **20000** 为例，终端输出模型准确率，预测标签和真实标签，损失函数，分类散点图如下：


**ep=1000:**

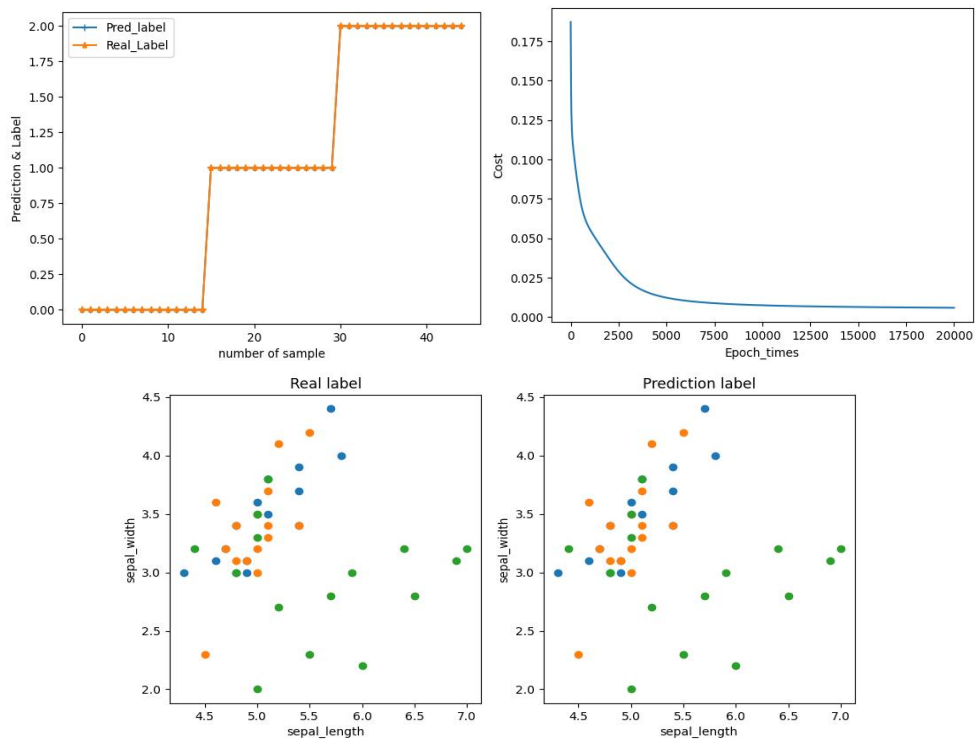
```
Epoch times: 1000/1000
Accuracy of classification: 84.44%
```





ep=20000:

Epoch times: 20000/20000   
Accuracy of classification: 100.00%



可以看到，训练次数较少时，分类效果较差；随着训练次数增加，损失逐渐降低，分类效果趋于完美。

## • 红酒数据集

对于**红酒数据集**，属性值较多，隐层神经元数量也会影响模型的准确性。但迭代次数 **ep** 对模型准确性的影响还是占主导地位（隐藏层神经元默认 10 个）：

**ep=100** 时，准确率只有 47.29%；

**ep=1000** 时，准确率提高到 56.25%；

**ep=10000** 时，准确率为 58.33%；

**ep=100000** 时，准确率为 60.42%；

为了比较隐层神经元数量对模型准确性的影响，实验了 **ep=10000** 时，不同隐藏层神经元数量下的模型准确性：

**q=10** 时，准确率为 58.33%；

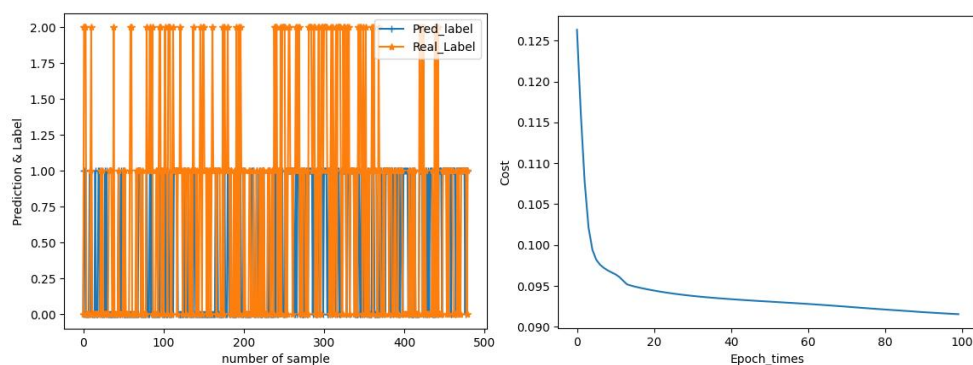
**q=20** 时，准确率为 59.17%；

可以看到，隐层神经元数量对模型准确性有一定影响，在样本属性较多时更加明显。

以迭代次数为 100 和 100000，**q=10** 为例，终端输出模型准确率，预测标签和真实标签，损失函数如下：

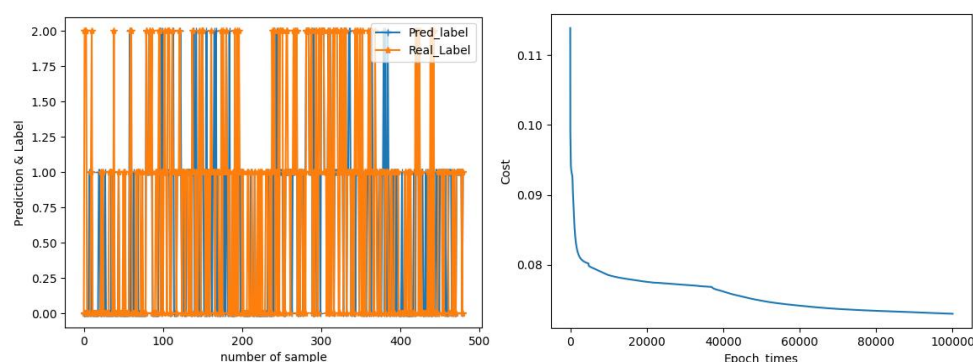
**ep=100:**

```
Epoch times: 100/100 ██████████
Accuracy of classification: 47.29%
```



**ep=100000:**

```
Epoch times: 100000/100000 ██████████
Accuracy of classification: 60.42%
```



为了达到更高的准确率，将迭代次数改为  $10^5$ ，隐藏层神经元数赋为 30，得到的准确率是 61.46%：

```
Epoch times: 100000/100000 ██████████
Accuracy of classification: 61.46%
```

将迭代次数改为  $10^6$ ，隐藏层神经元数赋为 30，得到的准确率是 **62.83%**：

Epoch times: 1000000/1000000 ██████████  
Accuracy of classification: 62.83%

该用例耗时近 20h。

调试过程:

### 1. 问题：使用 xlrd 读取 Excel 数据失败：

解决方法: xlrd 版本升级后不再支持 xlsx 文件读取, 降低版本可以解决该问题:

```
pip uninstall xlrd
```

```
pip install xlrd==1.2.0
```

2. **问题：**接问题 1，本地解决后部署到华为云 notebook 上再次出问题：

```
-----
ModuleNotFoundError                                Traceback (most recent call last)
/tmp/ipykernel_1124/3117148932.py in <module>
      6     呈现训练过程,绘制分类结果,评估模型准确性
      7 '''
----> 8 import xlrd
      9 from xlrd import open_workbook
     10 import numpy as np

ModuleNotFoundError: No module named 'xlrd'
```

**解决方法:** 在文件开头插入 code 单元, 安装指定版本的包:

```
1.3 s [7] !pip install xldr==1.2.0
```

Looking in indexes: <http://repo.myhuaweicloud.com/repository/pypi/simple>  
Requirement already satisfied: xldr==1.2.0 in /home/ma-user/anaconda3/envs/PyTorch-1.8/lib/python3.7/site-packages (1.2.0)

3. **问题：**刚完成代码在测试的过程中出现了一大堆 not defined 的问题：

```
Epoch times: 1/10000000 Traceback (most recent call last):
  File "d:\PyLearn\ML\实验2\Iris.py", line 42, in <module>
    cst=bpnn.Train()
  File "d:\PyLearn\ML\实验2\BPNN.py", line 104, in Train
    self.GD(b,y,self.Y)
  File "d:\PyLearn\ML\实验2\BPNN.py", line 81, in GD
    tmp=np.matmul(g,w.T) #计算 $\sum w[h][j]*g[j]$ , tmp:n*q
NameError: name 'w' is not defined
```

**解决方法：**类中方法需要加上 **self** 才可以调用。

4. **问题：** 计算  $d \times 1$  矩阵和  $1 \times q$  矩阵的乘法时，发生错误：

```
ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)
->(n?,m?) (size 1 is different from 4)
```

解决方法：numpy 包将  $d \times 1$  矩阵自动压缩为  $d$  维向量，导致无法使用 `np.matmul()` 矩阵乘法。

5. **问题:** 绘制预测结果的散点图时, 出现维数不匹配问题:

```
ax.scatter(pred1[:,0],pred1[:,1])
IndexError: too many indices for array: array is 1-dimensional, but 2 were indexed
```

**解决方法：**该错误并不是每次都出现，一开始怀疑是轻量级编译器不稳定造成。后来经过测试发现，此问题都是在训练次数较少时出现，原因是预测结果中没有包含指定的类，**pred1 数组为空**，因此不存在该检索。在绘制散点图前加入非空判断即可。

## 五、模型总结与思考

反向传播神经网络模型通过梯度下降法大量训练求解模型参数，最终收敛到稳定模型，输出预测结果。取预测结果激活后的最大值替代 softmax 激活，有效解决了 softmax 导数不连续造成的损失函数不稳定甚至模型不收敛问题。

通过实验发现，模型超参数和数据集样本分布都会影响模型的训练效果和准确率：

- **模型超参数：**本实验中的模型超参数主要是训练次数 **ep** 和隐藏层神经元数 **q**。当训练次数较少时，模型训练参数未收敛，准确率较低；随着训练次数增多，**v、w、γ、θ** 逐渐收敛，准确率不断提高；当训练次数过多，一旦超过测试集拐点，可能会出现过拟合现象。当训练次数固定时，隐藏层神经元数也会对模型准确率产生影响，在适当范围内，**q** 越大，准确率越高。

- **随机初始化矩阵：**由于模型在初始化 **v、w、γ、θ** 时采用随机数，因此在同一超参数下模型的训练准确率会上下波动。经实验发现，同一超参数下模型随机初始化参数矩阵可能使准确率上下波动 5%。

- **数据集：**对比一下同一模型在两个数据集上的训练效果：鸢尾花数据集数据的分布效果较规律，迭代 10000 次模型就可以接近完美；红酒数据集样本分布有些混乱，迭代 100000 次也只能达到 60% 左右的准确率。为了验证猜想，调用 **pytorch** 中随机森林、**svm** 等模型，以及在 **BPNN** 中加入多层隐藏层、优化器、**DropOut** 层等方法，最高只能达到 70% 的准确率。事实证明，该数据集的样本分布规律性较少，因此模型准确率较低。

实验中还有两个一直困扰我的问题，一个就是在计算 **d\*1** 矩阵和 **1\*q** 矩阵的乘法时，总是发生 **mismatch** 错误。当时很是奇怪，打印出来检查也没有问题。后来发现 **numpy** 包将 **d\*1** 矩阵自动压缩为 **d** 维向量，导致无法使用 **np.matmul()** 矩阵乘法，需要将 **(n,)** 转为 **(n,1)** 才可以顺利运行；还有一个就是矩阵乘法的参数不匹配，通过转置以将其匹配。

通过本次实验，我加深了对反向传播神经网络模型原理的理解，在实践中收获了许多理论知识学习时难以发现的细节问题。

## 六、代码附录

BPNN.py

```
'''神经网络模型
    输入层输入:X
    输入层输出:X添加一列偏置项以将 X*v+b1 简化为(X,1)*(V,b)
    隐藏层输入:alpha=X*v,X:n*(d+1),v:(d+1)*q
    隐藏层输出:b=f(alpha-gama),b:n*q,其中 gama 为阈值,需要学习得到,f 为激活函数
```



```

        输出层输入:beita=b*w,b:n*q,w:q*1
        输出层输出:y=g(beita-theta),y:n*1
        损失函数:E=1/2*Σ(y 真-y)^2
    ...

import sys
import numpy as np
from scipy.special import expit

class BPNN(object):
    def __init__(self,X,Y,q,l,epoch_times=1000,Eta=0.001,visible=False):
        '''args:
            X(n*d):样本集属性的 np 数组,包含 n 个样本,每个样本包含 d 个属性
            Y(n*1):样本集标签的 np 数组
            gama:隐藏层神经元的阈值[1*q]
            theta:输出层神经元的阈值[1*1]
            n:样本数
            d:属性数
            q:隐藏层中神经元数
            l:分类数
            v:输入层->隐藏层的权重向量[(d+1)*q]
            w:隐藏层->输出层的权重向量[q*1]
            epoch_times:梯度下降法中的迭代次数
            Eta:梯度下降法中的学习率
            visible:训练过程是否可视化
        ...

        self.X=X
        self.Y=Y
        self.n,self.d=self.X.shape
        self.q=q
        self.l=l
        self.v,self.w=self.Init_wgt()
        self.gama,self.theta=self.Init_threshold()
        self.epoch_times=epoch_times
        self.Eta=Eta
        self.visible=visible

    def Add_bias(self,X):
        #对 X(1*d)添加偏置项->X_new(1*(d+1))
        X_new=np.ones((X.shape[0]+1))
        X_new[1:]=X
        return X_new

    def Init_wgt(self):
        #初始化 v 和 w
        v=np.random.uniform(-1.0,1.0,size=(self.d+1)*self.q)
        v=v.reshape(self.d+1,self.q)
        w=np.random.uniform(-1.0,1.0,size=self.q*(self.l))

```

```

w=w.reshape(self.q,self.l)
return v,w

def Init_threshold(self):          #初始化 gama 和 theta
    gama=np.random.uniform(-1.0,1.0,size=self.q)
    gama=gama.reshape(1,self.q)
    theta=np.random.uniform(-1.0,1.0,size=self.l)
    theta=theta.reshape(1,self.l)
    return gama,theta

def Sigmoid(self,z):              #激活函数
    return expit(z)

def Sigmoid_gradient(self,z):     #激活函数的梯度
    sg=self.Sigmoid(z)
    return sg*(1-sg)

def FP(self,X):                  #将输入的训练/测试样本正向传播
    X_new=self.Add_bias(X)        #X_new:1*(d+1)
    alpha=np.matmul(X_new,self.v) #alpha:1*q
    b=self.Sigmoid(alpha-self.gama) #b:1*q
    beita=np.matmul(b,self.w)     #beita:1*l
    y=self.Sigmoid(beita-self.theta)#y:1*1
    return alpha,b,beita,y

def Get_cost(self,y,label):       #损失函数
    tmp=(y-label)*(y-label)/2
    cost=np.mean(tmp)
    return cost

def GD(self,X,b,y,label):        #反向传播,修正 v,w,gama,theta
    g=y*(1-y)*(y-label)          #g:1*1
    tmp1=np.matmul(g,self.w.T)    #计算Σw[h][j]*g[j],tmp:1*q
    e=b*(1-b)*tmp1               #b:1*q,e:1*q
    X_new=self.Add_bias(X)

    self.w-=self.Eta*np.matmul(b.T,g)
    self.theta+=self.Eta*g
    tmp2=X_new.T.reshape(-1,1)
    self.v-=self.Eta*np.matmul(tmp2,e)
    self.gama+=self.Eta*e
    return self

def Pred(self,X):                #对输入的`测试样本集`预测分类结果
    pred=[]
    for i in range(X.shape[0]):

```

```

        alpha,b,beita,y=self.FP(X[i])
        pred_tmp=np.argmax(y,axis=-1)    #预测值取最大的分类作为标签
        pred.append(pred_tmp[0])
    return pred

def Train(self):
    costs=[]                               #存储每次训练后的损失函数
    for i in range(self.epoch_times):
        if self.visible:
            sys.stderr.write("\rEpoch times: %d/%d
"%(i+1,self.epoch_times)+"█"*(i//((self.epoch_times//20)))
            sys.stderr.flush()
            cost_tmp=[]
            for j in range(len(self.X)):#对每个样本进行训练
                alpha,b,beita,y=self.FP(self.X[j])
                cost=self.Get_cost(y,self.Y[j])
                cost_tmp.append(cost)
                self.GD(self.X[j],b,y,self.Y[j])
            costs.append(np.mean(np.array(cost_tmp)))
    return costs

```

## Iris.py

```

''' 鸢尾花数据集
    150 个样本,按 7:3 划分为训练集和测试集
    3 个类别('setosa', 'versicolor', 'virginica'),
    4 个属性('sepal length','sepal width','petal length','petal width'),
    训练神经网络模型并测试准确率
    呈现训练过程,绘制分类结果,评估模型准确性
...
import xlrd
from xlrd import open_workbook
import numpy as np
import matplotlib.pyplot as plt
from BPNN import BPNN

#读取数据并将其按 7:3 划分为训练集和测试集
path=r"ML\实验 2\data\iris_data.xlsx"
book=open_workbook(path)
sheet=book.sheets()[0]                #打开 sheet0
nrow=sheet.nrows                       #行数
ncol=sheet.ncols                       #列数
label_dic={'Iris-setosa':0,'Iris-versicolor':1,'Iris-virginica':2} #标签序列化
data_train=[]                          #训练集属性(n*d)

```

```

data_test=[] #训练集标签(n*1)
label_train=[] #测试集属性
label_test=[] #测试集标签(one-hot)
labeln_test=[] #测试集标签(分类值)
for i in range(1,nrow):
    row=sheet.row_values(i)
    one_hot=[0]*len(label_dic) #样本分类标签的 one-hot 向量
    label=label_dic[row[4]]
    one_hot[label]=1
    row.pop(4) #保留样本属性,删除标签
    if(i%10<7):
        data_train.append(row)
        label_train.append(one_hot)
    else:
        data_test.append(row)
        label_test.append(one_hot)
        labeln_test.append(label)
data_train=np.array(data_train)
data_test=np.array(data_test)
label_train=np.array(label_train)
label_test=np.array(label_test)
labeln_test=np.array(labeln_test)

#实例化 BPNN 对象并进行训练
d=len(data_train[0]) #属性数
l=len(label_dic) #分类数
n=nrow #样本数
ep=20000 #训练次数
q=10 #隐藏层神经元数
bpnn=BPNN(data_train,label_train,q,l,ep,0.001,True)
cst=bpnn.Train() #训练过程的损失函数
pred=bpnn.Pred(data_test) #训练后的分类结果

#计算准确率
acc=0
for i in range(len(pred)):
    if(pred[i]==labeln_test[i]):
        acc+=1
acc/=len(pred)
print('\nAccuracy of classification: %.2f%%'%(acc*100))

#绘制预测值/标签值-编号的图像
plt.plot(pred,marker='+')
plt.plot(labeln_test,marker='*')

```



```

plt.ylabel('Prediction & Label')
plt.xlabel('number of sample')
plt.legend(['Pred_label', 'Real_Label'])
plt.show()

#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()

#以 sepal_length 和 sepal_width 属性为例绘制多分类结果
#将预测结果按类分开
pred0=[] #预测为 Iris-setosa 类的样本点集合
pred1=[] #预测为 Iris-versicolor 类的样本点集合
pred2=[] #预测为 Iris-virginica 类的样本点集合
for i in range(len(pred)):
    if(pred[i]==0):
        pred0.append(data_train[i][:2])
    elif(pred[i]==1):
        pred1.append(data_train[i][:2])
    else:
        pred2.append(data_train[i][:2])
pred0=np.array(pred0)
pred1=np.array(pred1)
pred2=np.array(pred2)
#绘制真实分类和预测分类
fig0=plt.figure(figsize=(10,5))
ax=plt.subplot(121) #真实分类
ax.scatter(data_train[:15,0],data_train[:15,1])
ax.scatter(data_train[15:30,0],data_train[15:30,1])
ax.scatter(data_train[30:45,0],data_train[30:45,1])
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
ax.set_title("Real label")
ax=plt.subplot(122) #预测分类
ax.scatter(pred0[:,0],pred0[:,1])
ax.scatter(pred1[:,0],pred1[:,1])
ax.scatter(pred2[:,0],pred2[:,1])
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')
ax.set_title("Prediction label")
plt.show()

```

## WineQuality.py

```
'''红酒品质数据集
    1600 个样本,按 7:3 划分为训练集和测试集
    3 个类别('Bad','Normal','Good')
    11 个属性('fixed acidity','volatile acidity','citric acid','residual
sugar','chlorides','free sulfur dioxide','total sulfur
dioxide','density','pH','sulphates','alcohol')
    训练神经网络模型并测试准确率
    呈现训练过程,绘制分类结果,评估模型准确性
'''

import xlrd
from xlrd import open_workbook
import numpy as np
import matplotlib.pyplot as plt
from BPNN import BPNN

#读取数据并将其按 7:3 划分为训练集和测试集
path="ML\\实验 2\\data\\winequality_data.xlsx"
book=open_workbook(path)
sheet=book.sheets()[0]          #打开 sheet0
nrow=sheet.nrows                #行数
ncol=sheet.ncols                #列数
label_dic={'Bad':0,'Normal':1,'Good':2}    #标签序列化
data_train=[]                  #训练集属性(n*d)
data_test=[]                   #训练集标签(n*1)
label_train=[]                 #测试集属性
label_test=[]                  #测试集标签(one-hot)
labeln_test=[]                 #测试集标签(分类值)
for i in range(1,nrow):
    row=sheet.row_values(i)
    one_hot=[0]*len(label_dic)  #样本分类标签的 one-hot 向量
    label=label_dic[row[11]]
    one_hot[label]=1
    row.pop(11)                  #保留样本属性,删除标签
    if(i%10<7):
        data_train.append(row)
        label_train.append(one_hot)
    else:
        data_test.append(row)
        label_test.append(one_hot)
        labeln_test.append(label)
data_train=np.array(data_train)
data_test=np.array(data_test)
label_train=np.array(label_train)
```

```

label_test=np.array(label_test)
labeln_test=np.array(labeln_test)

#实例化 BPNN 对象并进行训练
d=len(data_train[0])          #属性数
l=len(label_dic)              #分类数
n=nrow                        #样本数
ep=100                        #训练次数
q=20                          #隐藏层神经元数
bpnn=BPNN(data_train,label_train,q,l,ep,0.001,True)
cst=bpnn.Train()              #训练过程的损失函数
pred=bpnn.Pred(data_test)     #训练后的分类结果

#计算准确率
acc=0
for i in range(len(pred)):
    if(pred[i]==labeln_test[i]):
        acc+=1
acc/=len(pred)
print('\nAccuracy of classification: %.2f%%'%(acc*100))

#绘制预测值/标签值-编号的图像
plt.plot(pred,marker='+')
plt.plot(labeln_test,marker='*')
plt.ylabel('Prediction & Label')
plt.xlabel('number of sample')
plt.legend(['Pred_label','Real_Label'])
plt.show()

#绘制损失函数
plt.plot(range(len(cst)),cst)
plt.ylabel('Cost')
plt.xlabel('Epoch_times')
plt.show()

```