**第10章 神经网络**

1. 概述BP神经网络的基本原理，并分析其优势和不足；梯度消失和梯度爆炸是神经网络常见问题，如何避免？

**BP神经网络：**

BP神经网络就是将感知机的激励函数改为tanh或者阶跃函数。通过输入的特征值和初始的学习参数，将其计算出结果直接和，根据分类函数，判断其标签类型，并且将得出的结果和实际结果进行比对，得出误差，将误差返回，求其关于学习参数的偏导，这样就再根据随机梯度下降，计算出迭代后的学习参数，进行迭代，然后反复进行以上计算最终结果，并且根据误差进行反向传播重新迭代学习参数。优势可以实现解决：线性问题和一些非线性的问题。可以解决很多复杂的问题。

**常见问题：**

梯度消失：梯度趋近于零，网络权重无法更新或更新的很微小，网络训练再久也不会有效果；  
梯度爆炸：梯度呈指数级增长，变的非常大，然后导致网络权重的大幅更新，使网络变得不稳定。

梯度剪切这个方案主要是针对梯度爆炸提出的，其思想是设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸。

权重正则化也是解决度爆炸的手段，比较常见的是L1正则，和L2正则，正则化是通过对网络权重做正则，防止w过大，限制过拟合。

1. 利用BP神经网络实现鸢尾花数据分类。

1）针对当前数据、模型，**详叙数据预处理、模型参数调整过程及效果比对**；

2）代码要列出，重点代码加注释说明，特别是自己调试过程中的自我理解；

3）运行结果要截图，结果要文字说明；

4）注意排版

*import* math

*import* random

*import* pandas *as* pd

*import* matplotlib.pyplot *as* plt

*import* numpy *as* np

*import* matplotlib.patches *as* mpatches

*# 载入鸢尾花数据集*

iris = pd.read\_csv('./iris.csv')

all\_data = iris.values

*# 提取特征数据，包括4个特征和1个标签*

*# 4个特征分别是：萼片长度、萼片宽度、花瓣长度、花瓣宽度*

all\_feature = all\_data[1:, 0:5]

*# 可视化鸢尾花数据集的前两个特征（萼片长度和萼片宽度）*

*# 使用不同颜色和标记区分三种鸢尾花类别*

*for* i *in* range(len(all\_feature)):

*if* all\_feature[i,4]==0:

        plt.scatter(all\_feature[i,0], all\_feature[i,1], *color*='red', *marker*='s')  *# setosa类*

*elif* all\_feature[i,4]==1:

        plt.scatter(all\_feature[i,0], all\_feature[i,1], *color*='green', *marker*='o')  *# versicolor类*

*else*:

        plt.scatter(all\_feature[i,0], all\_feature[i,1], *color*='blue', *marker*='d')  *# virginica类*

*# 设置图例和中文显示*

labels = ['setosa', 'versicolor', 'virginica']

color = ['red','blue','lightskyblue']

patches = [ mpatches.Patch(*color*=color[i], *label*="{:s}".format(labels[i]) ) *for* i *in* range(len(color)) ]

plt.legend(*handles*=patches, *loc*=2)

plt.rcParams['font.sans-serif'] = ['SimHei']  *# 设置中文字体*

plt.rcParams['axes.unicode\_minus'] = False

plt.show()

print('plt.show()')

*# 数据预处理：将标签转换为one-hot编码*

*# one-hot编码将类别标签转换为二进制向量*

data = []

*for* i *in* range(len(all\_feature)):

    newdata = []

    newdata.append(list(all\_feature[i]))  *# 添加特征数据*

*if* all\_data[i][4] == 'setosa':

        newdata.append([1, 0, 0])  *# setosa -> [1,0,0]*

*elif* all\_data[i][4] == 'versicolor':

        newdata.append([0, 1, 0])  *# versicolor -> [0,1,0]*

*else*:

        newdata.append([0, 0, 1])  *# virginica -> [0,0,1]*

    data.append(newdata)

*# 随机打乱数据并划分训练集和测试集*

*# 使用随机打乱来确保数据的随机性*

random.shuffle(data)

train = data[0:100]  *# 前100个样本作为训练集*

test = data[101:]    *# 剩余样本作为测试集*

print('data shuffle')

*# 定义标签映射*

iris\_lables = ['setosa', 'versicolor', 'virginica']

def **generate\_matrix**(*rows*, *cols*):

    """生成指定大小的矩阵"""

*return* [[0.0 *for* \_ *in* range(*cols*)] *for* \_ *in* range(*rows*)]

def **rand**(*a*, *b*):

    """生成指定范围内的随机数"""

*return* (*b* - *a*) \* random.random() + *a*

def **sigmoid**(*x*):

    """sigmoid激活函数"""

*return* 1.0 / (1.0 + math.exp(-*x*))

def **dsigmoid**(*x*):

    """sigmoid函数的导数"""

*return* *x* \* (1.0 - *x*)

class MyBPNetwork:

    """ 三层BP神经网络实现

    网络结构：

    输入层(4个特征) -> 隐藏层 -> 输出层(3个类别)

    使用sigmoid作为激活函数

    使用反向传播算法更新权重

    """

    def **\_\_init\_\_**(*self*, *in\_num*, *hidden\_num*, *output\_num*):

*# 初始化网络结构参数*

*self*.in\_num = *in\_num* + 1    *# 输入层神经元个数（+1为偏置项）*

*self*.hidden\_num = *hidden\_num* + 1  *# 隐藏层神经元个数*

*self*.output\_num = *output\_num*      *# 输出层神经元个数*

*# 初始化各层神经元激活值*

*self*.in\_layer = [1.0] \* *self*.in\_num      *# 输入层*

*self*.hid\_out = [1.0] \* *self*.hidden\_num   *# 隐藏层*

*self*.out\_out = [1.0] \* *self*.output\_num   *# 输出层*

*# 初始化权重矩阵*

*# W1: 输入层到隐藏层的权重矩阵*

*# W2: 隐藏层到输出层的权重矩阵*

*self*.W1 = generate\_matrix(*self*.in\_num, *self*.hidden\_num)

*self*.W2 = generate\_matrix(*self*.hidden\_num, *self*.output\_num)

*# 随机初始化权重值*

*# 使用随机值初始化权重，范围在(0.1,0.9)之间*

*for* i *in* range(*self*.in\_num):

*for* j *in* range(*self*.hidden\_num):

*self*.W1[i][j] = rand(0.1, 0.9)

*for* j *in* range(*self*.hidden\_num):

*for* k *in* range(*self*.output\_num):

*self*.W2[j][k] = rand(-0.5, 1)

    def **train**(*self*, *sample\_data*, *iter*=1000, *alpha*=0.1):

        """ 训练神经网络

        训练过程：

        1. 前向传播：计算网络输出

        2. 反向传播：计算误差并更新权重

        3. 重复以上步骤直到达到最大迭代次数

        Args:

            sample\_data: 训练数据

            iter: 迭代次数

            alpha: 学习率（控制权重更新的步长）

        """

        errors = []

*for* it *in* range(*iter*):

            error = 0.0

*# 对每个训练样本进行前向传播和反向传播*

*for* f *in* *sample\_data*:

                feature = f[0]    *# 特征*

                labels = f[1]     *# 标签*

*self*.update(feature)  *# 前向传播*

                error = error + *self*.back\_propagate(labels, *alpha*)  *# 反向传播*

*if* it % 100 == 0:

                print('error: %-.8f' % error)

            errors.append(error)

*self*.show\_curve(errors)  *# 显示训练误差曲线*

    def **update**(*self*, *feature*):

        """ 前向传播过程

        前向传播步骤：

        1. 输入层：接收特征输入

        2. 隐藏层:计算加权和并应用sigmoid激活函数

        3. 输出层:计算加权和并应用sigmoid激活函数

        Args:

            feature: 输入特征

        Returns:

            输出层的激活值

        """

*# if len(feature) != self.in\_num - 1:*

*#     raise ValueError('与输入层神经元个数不相等！')*

*# 激活输入层*

*for* i *in* range(*self*.in\_num - 1):

*self*.in\_layer[i] = *feature*[i]

*# 激活隐藏层（使用sigmoid激活函数）*

*# sigmoid(x) = 1 / (1 + e^(-x))*

*for* j *in* range(*self*.hidden\_num):

            z = 0.0

*for* i *in* range(*self*.in\_num):

                z = z + *self*.in\_layer[i] \* *self*.W1[i][j]

*self*.hid\_out[j] = sigmoid(z)

*# 激活输出层*

*for* k *in* range(*self*.output\_num):

            z = 0.0

*for* j *in* range(*self*.hidden\_num):

                z = z + *self*.hid\_out[j] \* *self*.W2[j][k]

*self*.out\_out[k] = sigmoid(z)

*return* *self*.out\_out

    def **back\_propagate**(*self*, *y\_label*, *alpha*):

        """ 反向传播过程

        反向传播步骤：

        1. 计算输出层误差：δ = (y - ŷ) \* f'(z)

        2. 计算隐藏层误差：δ = Σ(δ \* w) \* f'(z)

        3. 更新权重:w = w + α \* δ \* x

        Args:

            y\_label: 真实标签

            alpha: 学习率

        Returns:

            当前样本的误差

        """

*# 计算输出层的误差*

        delta\_output = [0.0] \* *self*.output\_num

*for* index *in* range(*self*.output\_num):

            error = *y\_label*[index] - *self*.out\_out[index]

*# dsigmoid(x) = sigmoid(x) \* (1 - sigmoid(x))*

            delta\_output[index] = dsigmoid(*self*.out\_out[index]) \* error

*# 计算隐藏层的误差*

        delta\_hidden = [0.0] \* *self*.hidden\_num

*for* j *in* range(*self*.hidden\_num):

            error = 0.0

*for* k *in* range(*self*.output\_num):

                error = error + delta\_output[k] \* *self*.W2[j][k]

            delta\_hidden[j] = dsigmoid(*self*.hid\_out[j]) \* error

*# 更新输出层权重*

*for* j *in* range(*self*.hidden\_num):

*for* k *in* range(*self*.output\_num):

                change = delta\_output[k] \* *self*.hid\_out[j]

*self*.W2[j][k] = *self*.W2[j][k] + *alpha* \* change

*# 更新输入层权重*

*for* i *in* range(*self*.in\_num):

*for* j *in* range(*self*.hidden\_num):

                change = delta\_hidden[j] \* *self*.in\_layer[i]

*self*.W1[i][j] = *self*.W1[i][j] + *alpha* \* change

*# 计算误差（均方误差）*

        error = 0.0

*for* k *in* range(*self*.output\_num):

            error += 0.5 \* (*y\_label*[k] - *self*.out\_out[k]) \*\* 2

*return* error

    def **test**(*self*, *patterns*):

        """ 测试网络性能

        测试步骤：

        1. 对每个测试样本进行前向传播

        2. 选择输出层中最大激活值对应的类别作为预测结果

        3. 计算预测准确率

        Args:

            patterns: 测试数据

        """

        count = 0

*for* p *in* *patterns*:

            target = iris\_lables[(p[1].index(1))]  *# 获取真实标签*

            result = *self*.update(p[0])  *# 前向传播*

            index = result.index(max(result))  *# 预测结果（选择最大激活值对应的类别）*

            print(p[0], ':', target, '->', iris\_lables[index])

            count += (target == iris\_lables[index])

        accuracy = float(count / len(*patterns*))

        print('accuracy: %-.8f' % accuracy)

    def **show\_curve**(*self*, *errors*):

        """显示训练误差曲线"""

        plt.figure()

        plt.plot(*errors*)

        plt.title('训练误差曲线')

        plt.xlabel('迭代次数')

        plt.ylabel('误差')

        plt.show()

print('bp be created')

*# 主程序：训练和测试BP神经网络*

*# 创建BP神经网络实例（4个输入神经元，8个隐藏层神经元，3个输出神经元）*

bp = MyBPNetwork(4, 8, 3)

*# 训练网络*

print("开始训练...")

bp.train(train, *iter*=1000, *alpha*=0.1)

*# 测试网络*

print("\n开始测试...")

bp.test(test)





