《机器学习》编程作业 3

题目 1.1，2.1

一、题目理解

1.1 题目 1.1

编程实现标准 BP 算法，在西瓜数据集 3.0 上训练一个单隐层网络。

即针对西瓜数据集 3.0，采用 BP 神经网络算法，设计一个单隐层网络，即包含输入层、隐藏层以及输出层。对西瓜数据集进行分割，得到训练集以及测试集，网络在训练集上进行训练，更新网络权重后达到网络更新的结束条件，（训练轮数达到指定值或者误差小于特定值）利用测试集得到准确率、召回率、F1 分数等等，对网络进行评价。

1.2 题目 2.1

复用所实现的 BP 算法，在一种 UCI 数据集上评价性能。

使用上述题目实现的 BP 网络，在 UCI 数据集（本次实验我选用的是乳腺癌数据集）上，通过改变训练轮数、结束迭代条件、学习率等参数，进行多次训练，并进行网络评估，给出训练的准确率、召回率以及 F1 分数等等。

二、算法原理阐述

2.1 BP 神经网络算法原理

BP（Back Propagation，反向传播）神经网络是一种基于误差反向传播的多层前馈神经网络，它在输入和输出层之间引入了一个或多个隐藏层，能够学习复杂的映射关系。BP 神经网络的训练过程包括正向传播和反向传播两个阶段。

2.1.1 向前传播

在正向传播阶段，输入信息从输入层经过隐藏层传播到输出层，并产生网络输出。BP 神经网络的每一层包含神经元节点，每个节点的计算可以通过加权求和、加偏置后应用激活函数得到输出。

假设输入层有 n 个节点，隐藏层有 h 个节点，输出层有 m 个节点。

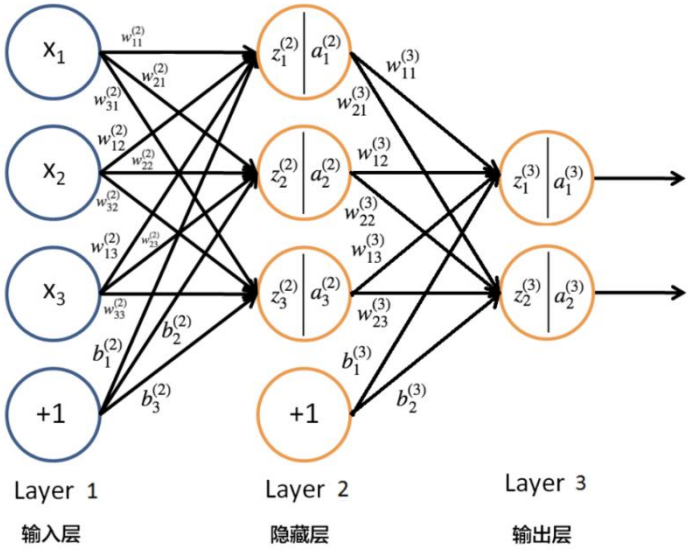


图 1 BP 单隐层神经网络结构图

（1）输入层到隐藏层

假设输入层节点的输出为 x_i（i=1,2,…,n），隐藏层节点的输入为：

$$z_j = \sum_{i=1}^{n} w_{ij}^{(1)} x_i + b_j^{(1)}$$

其中，$w_{ij}{}^{(1)}$表示输入层到隐藏层第 j 个节点的权重，$b_j{}^{(1)}$是偏置。隐藏层节点的输出 h_j 通过激活函数 f，即 Sigmoid 函数计算得到：

$$h_j = f(z_j)$$

（2）隐藏层到输出层

输出层节点的输入为：

$$y_k = \sum_{j=1}^{h} w_{jk}^{(2)} h_j + b_k^{(2)}$$

其中，$w_{jk}{}^{(2)}$表示隐藏层到输出层第 k 个节点的权重，$b_k{}^{(2)}$是偏置。

输出层节点的输出o_k通过激活函数 g，即 Sigmoid 函数，计算得到：

$$o_k = g(y_k)$$

2.1.2 反向传播

反向传播阶段通过计算网络输出与目标值之间的误差来更新网络的权重和偏置，从而最小化损失函数。通常使用均方误差（MSE）作为损失函数：

$$E = \frac{1}{2} \sum_{k=1}^{m} (t_k - o_k)^2$$

其中，t_k表示第 k 个输出节点的目标值，o_k表示第 k 个输出节点的预测值。

(1)计算输出层的误差

输出层每个节点的误差项δ_k为：

$$\delta_k = (t_k - o_k) \cdot g'(y_k)$$

其中，$g'(y_k)$是输出层激活函数的导数。

(2)计算隐藏层的误差

隐藏层每个节点的误差项δ_j为：

$$\delta_j = \left(\sum_{k=1}^{m} \delta_k w_{jk}^{(2)} \right) \cdot f'(z_j)$$

其中，$f'(z_j)$是隐藏层激活函数的导数。

(3)更新权重和偏置

通过梯度下降法更新权重和偏置，学习率为η。

· 输出层权重更新：

$$w_{jk}^{(2)} = w_{jk}^{(2)} + \eta \cdot \delta_k \cdot h_j$$

· 隐藏层权重更新：

$$w_{ij}^{(1)} = w_{ij}^{(1)} + \eta \cdot \delta_j \cdot x_i$$

· 偏置更新：

$$b_k^{(2)} = b_k^{(2)} + \eta \cdot \delta_k$$

$$b_j^{(1)} = b_j^{(1)} + \eta \cdot \delta_j$$

2.1.3 迭代训练过程

- **初始化权重和偏置**：随机设置网络中所有权重和偏置的初始值。
- **正向传播**：给定输入，通过网络计算输出。
- **计算误差**：通过损失函数计算输出误差。
- **反向传播**：计算输出层和隐藏层的误差，并更新权重和偏置。
- **检查收敛性**：若损失函数的值小于某一设定的阈值或达到最大迭代次数，终止训练，否则返回步骤 2。

2.2 整数编码的原理

整数编码（Integer Encoding）是一种将分类变量转换为数值表示的技术。分类变量通常是字符串类型的数据，例如颜色、城市、品牌等。这些变量不能直接用于许多机器学习算法，因为这些算法通常只能处理数值数据。整数编码通过将每个唯一的分类值映射到一个唯一的整数，使得分类变量可以被机器学习模型处理。

三、算法设计思路

3.1 数据集处理思路

（1）西瓜数据集 3.0

使用 pandas 读取 watermelon3.0 数据集，其中包含多个特征和一个类别标签。将类别标签“好瓜”列中的值“是”和“否”映射为数值标签，分别为 1 和 0，便于神经网络处理二值分类问题。对非数值特征进行整数编码。利用整数编码每个类别特征编码为整数，以方便后续的数值运算和输入神经网络。

采用 train_test_split 将数据划分为训练集和测试集，并使用不同的测试集比例（0.1、0.2、0.25 和 0.3）来评估模型的性能稳定性。

（2）乳腺癌数据集

由于 sklearn.datasets 中自带乳腺癌数据集，故直接调用 load_breast_cancer()函数加载数据。并且该数据集中均为数值类型，也无需做编码处理，直接提取 X 值与标签值即可。

3.2 算法框架设计思路

（1） 数据预处理：

读取西瓜数据集，将类别标签“好瓜”列中的“是”和“否”转换为 1 和 0（数值标签），以便进行分类。对非数值特征进行整数编码，以便神经网络能够处理所有特征。

（2）BP 神经网络定义：

设计一个包含输入层、隐藏层、输出层的 BP 神经网络，并设置网络的初始化参数，包括输入层节点数 input_size、隐藏层节点数 hidden_size、输出层节点数 output_size，以及学习率 learning_rate。

初始化权重和偏置矩阵，设定激活函数为 Sigmoid，来完成前向传播和反向传播过程。

（3）模型训练：

实现 train 方法，通过多次迭代，调用前向传播（forward 方法）和反向传播（backward 方法）来不断更新模型的权重和偏置，以最小化误差。

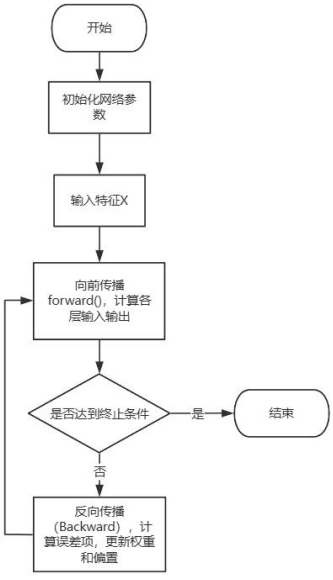


图 2 BP 神经网络框架流程图

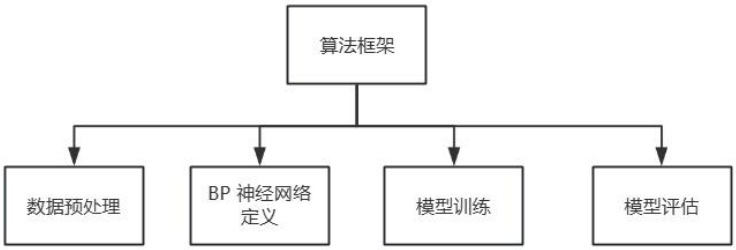
（4）模型评估：

在不同的测试集比例（0.1, 0.2, 0.25, 0.3）下，分别划分训练集和测试集。使用训练集训练 BP 神经网络模型，并在测试集上进行预测。计算模型的准确率（Accuracy）、精确率（Precision）、召回率（Recall）和 F1 分数（F1 Score），输出结果。

四、代码结构及核心部分介绍

4.1 整体代码结构

代码主要分为数据预处理模块、BP 神经网络定义模块、模型训练模块、模型评估模块。



4.2 核心部分介绍

4.2.1 BP 神经网络定义

初始化输入层、隐藏层和输出层节点数，以及学习率等神经网络参数，并使用正态分布随机初始化权重矩阵 v（输入层到隐藏层）、w（隐藏层到输出层），偏置矩阵 gamma 和 theta 则初始化为零矩阵。

```
# 定义 BP 神经网络类
class BPNeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.1):
        self.input_size = input_size #输入层节点数
        self.hidden_size = hidden_size #隐藏层节点数
        self.output_size = output_size #输出层节点数
        self.learning_rate = learning_rate #学习率

        # 随机初始化权重矩阵
        self.v = np.random.randn(input_size, hidden_size)  # 输入层到隐藏层的权重
        self.gamma = np.zeros((1, hidden_size))                  # 隐藏层的偏置
        self.w = np.random.randn(hidden_size, output_size) # 隐藏层到输出层的权重
        self.theta = np.zeros((1, output_size))                 # 输出层的偏置
```

4.2.2 前向传播

sigmoid 函数用于激活神经元，是前向传播中的核心部分。该函数通过公式计算出各层的输入与输出，最后得到当前轮次的 y_hat。

```
def forward(self, X):
    self.alpha = np.dot(X, self.v) + self.gamma          # 计算隐藏层加权输入
```

```
    self.b = self.sigmoid(self.alpha)                    # 计算隐藏层输出
    self.beta = np.dot(self.b, self.w) + self.theta     # 计算输出层加权输入
    self.y_hat = self.sigmoid(self.beta)                 # 计算输出层输出（预测值）
    return self.y_hat
```

4.2.3 反向传播

backward 函数实现了反向传播算法，通过计算误差项并利用梯度更新权重和偏置。即通过同时计算得到输出层误差项 self.g 和隐藏层误差项 self.e。并依据各层的权重更新公式，使用梯度下降法更新各层的权重以及偏置。

```
    def backward(self, X, y, y_hat):
    # 计算输出层误差项
    self.g = y_hat * (1 - y_hat) * (y - y_hat)
    # 计算隐藏层误差项
    self.e = self.b * (1 - self.b) * np.dot(self.g, self.w.T)

    # 更新权重和偏置
    self.w += self.learning_rate * np.dot(self.b.T, self.g)  # 更新隐藏层到输出层的权重
    self.theta += self.learning_rate * np.sum(self.g, axis=0, keepdims=True)  # 更新输出层偏置
    self.v += self.learning_rate * np.dot(X.T, self.e)  # 更新输入层到隐藏层的权重
    self.gamma += self.learning_rate * np.sum(self.e, axis=0, keepdims=True)  # 更新隐藏层偏置
```

五、题目 1.1 实验流程、测试结果及分析

4.1 流程与测试结果

给出不同的训练集测试集划分比例，分别为 0.1,0.15.0.2,0.3。进行多次训练，同时利用测试集给出正确率、召回率、F1 分数等不同指标进行评估。得到下列结果：

划分比例	Accuracy	Precision	Recall	F1 Score
0.1	100.00%	100.00%	100.00%	100.00%
0.15	75.00%	100.00%	66.67%	80.00%
0.2	80.00%	75.00%	100.00%	85.71%
0.3	50.00%	50.00%	100.00%	66.67%

4.2 结果分析

当划分比例为 0.1 时，模型的准确率、精确率、召回率和 F1 分数均为 100.00%，此时模型在所有样本上都预测正确，且在正类样本的预测上表现完美。

当划分比例为 0.15 时，模型的准确率为 75.00%，精确率为 100.00%，召回率为 66.67%，F1 分数为 80.00%。精确率很高，但召回率较低，此时模型在预测正类样本时非常准

确，但可能漏掉了一些实际为正类的样本。

当模型的准确率为 80.00%，精确率为 75.00%，召回率为 100.00%，F1 分数为 85.71%。召回率很高，但精确率较低，此时模型能够正确预测所有实际为正类的样本，但可能将一些负类样本错误地预测为正类。

当划分比例为 0.3 时，模型的准确率为 50.00%，精确率为 50.00%，召回率为 100.00%，F1 分数为 66.67%。虽然召回率很高，但精确率和准确率都很低，此时模型能够正确预测所有实际为正类的样本，但可能将大部分负类样本错误地预测为正类。

当划分比例为 0.1 时，模型表现较好，可能是由于数据集较小，模型能够完美拟合训练数据。 划分比例为 0.15 和 0.2 时，模型在精确率和召回率上存在一定的权衡，但总体表现较好。划分比例为 0.3 时，模型在召回率上表现很好，但在精确率和准确率上表现较差，可能是由于模型过拟合或数据集划分不合理。

六、题目 2.1 实验流程、测试结果及分析

6.1 流程与测试结果

给出不同的训练集测试集划分比例，分别为 0.1,0.15.0.2,0.3。同时采用两种算法迭代终止条件，分别为“达到一定的迭代次数”和“搜索到一定的误差阈值”。进行多次训练，同时利用测试集给出正确率、召回率、F1 分数等不同指标进行评估。

首先，设置算法的迭代终止条件为：“达到一定的迭代次数”,这里设置为 10000。得到下列结果：

划分比例	Accuracy	Precision	Recall	F1 Score
0.1	70.18%	70.18%	100.00%	82.47%
0.15	62.28%	62.28%	100.00%	76.76%
0.2	62.94%	62.68%	100.00%	77.06%
0.3	63.74%	63.53%	100.00%	77.70%

由于以上采用的迭代结束条件为迭代次数，现在将结束条件改为达到的误差阈值 0.02 后，算法结束，得到的结果如下：

划分比例	Accuracy	Precision	Recall	F1 Score
0.1	70.18%	70.18%	100.00%	82.47%
0.15	62.28%	62.28%	100.00%	76.76%
0.2	62.24%	62.24%	100.00%	76.72%
0.3	63.16%	63.16%	100.00%	77.42%

6.2 结果分析

首先分析不同终止条件的结果，在两种迭代终止条件下，模型的准确率和 F1 分数表现相似，且在不同划分比例下的变化并不显著，模型表现较为稳定。可能是由于数据集特征有限，且相对简单，无法支持更显著的性能差异。模型的隐藏层较浅（仅三个节点），对数据特征的提取能力有限，导致性能提升较为缓慢。两种终止条件下的准确率、精确率和 F1 分数基本一致，表明模型在 10000 次迭代下已基本收敛，继续迭代只会微小调整参数，对结果无明显提升。

再分析相关评估指标，两种迭代条件下 Recall（召回率）始终保持 100%。这表明模型在所有测试集上能够识别出所有的正例（即好瓜样本）。

Precision（精确率）变化不大，始终维持在 60%左右，这表明模型在预测为正例的样本中，有一定比例的样本是误判的。模型的高召回率和较低的精确率组合表明模型偏向于将样本判为正例，这可能是因为正例数据在训练集中占比较大。

F1 Score：模型的 F1 分数与精确率和召回率直接相关，保持在 75%到 82%的水平。F1 分数的变化较小，说明无论是终止条件还是数据划分比例的调整，都未能显著影响模型的均衡性。

从结果来看，测试集比例增加（训练集减少）时模型的精确度略有上升（如从 0.1 到 0.3 比例，准确率从 70.18%提升至 63.74%或 63.16%）。这可能是因为测试集比例越大，训练数据越少，网络难以更好地拟合数据，导致模型的泛化性能稍有提升（即在测试集上不会过拟合）。

八、问题与收获

8.1 遇到的问题

最开始在西瓜数据集上进行训练时，我没有对 X 数据进行编码，形如“颜色=青绿”这样的数据变无法参与计算，导致代码报错。后来我采用的是 sklearn.preprocessin 中的 LabelEncoder 编码方法，虽然能够成功运行，但是给出的结果不尽人意，准确率比较低。后来我查阅资料发现，可以对数据集中非数值数据采用 pandas 中的 factorize 函数进行编码，将每一种非数值型数据转化成“1”、“2”、“3”等多种标签，同时相较于之前的编码方式，正确率提升了很多。

8.2 收获

通过本次实验，我对 BP 神经网络算法的理解更加深刻了。通过实现一个 BP（反向传播）神经网络，掌握了其基本结构，包括输入层、隐藏层和输出层的定义，我对前向传播和反向传播的具体过程的了解更加深刻了：前向传播计算输出，反向传播计算误差并调整权重，以最小化误差。也培养了我在模型训练、评估和分析方面的动手能力。这将为后续更加复杂的深度学习任务打下坚实基础，同时提升了我在机器学习项目中的代码组织和实验分析能力。

九、参考资料

[1]《西瓜书》
[2]一、最简单的神经网络--Bp 神经网络-CSDN 博客
[3]神经网络——Python 实现 BP 神经网络算法（理论+例子+程序） bp 神经网络 python-CSDN 博客
[4]BP 神经网络原理与如何实现 BP 神经网络 bp 神经网络训练原理-CSDN 博客