**大连海事大学**

**智能信息处理实验大纲**

实验一 BP神经网络实验

# 实验目的

* 熟悉和掌握BP神经网络的设计和实现技术。
* 掌握BP神经网络训练和模拟技术；

# 实验内容

* + 1. 选定一个合适的线性不可分应用，设计BP神经网络解决该问题。
    2. 对比分析各种BP神经网络改进算法，比如消除样本顺序影响的算法等。
    3. 分析比较不同的激励函数、隐含层数、隐含层节点数、学习率、误差等参数对神经网络的影响。

# 实验要求

* + 1. 可以用MATLAB实现或者基于现有的神经网络开源框架或代码用JAVA或Python实现；
    2. 可参考网络资源，但不能照搬照抄，一定要有自己的改造和创新。

# 实验步骤

## 问题描述

流媒体音乐提供商通常会应用大数据、深度神经网络等技术，根据用户的点击频次、聆听时长等为用户推荐更加个性化的内容和服务，这样可以显著增加用户黏着度与自身产品竞争力。本应用旨在利用BP神经网络预测用户对音乐的喜好：通过给音乐作品集的每一首音乐打上标签，再结合它的作曲家，利用神经网络得出结果。

## BP神经网络设计

本神经网络的实现可以接受任意的隐藏层结构，例如：[3], [5], [3,4,3]

在这个应用中，音乐样本具有属性列：composer作曲家，Genre1:第一个流派，Genre2：第二个流派，然后是目标列：1表示喜欢，0表示不喜欢。

音乐样本集存储在CSV文件中，并且有一个专门的python源程序（data\_generator.py）可以反复生成不同体积的样本集。源程序中，为不同的作曲家和流派赋予权重，三个权重总和超过阈值时，这个音乐被标记为喜欢（目标列为1），这样就模拟了采样或者人工标记过程。并且方便反复用不同的样本集调试神经网络。

在神经网络中，我们对每个样本就根据这三个属性来尝试得出liked，因此神经网络有三个输入，有一个输出。

在代码中用二维列表weights[]存储权值矩阵，biases[]存储偏置向量，所有节点都选择sigmoid函数。正向传播完毕后，用梯度下降法修正权值和偏置。最后由于此问题相当于二分类问题，使用阈值分类整理网络输出outputs[]

向前传播和向后传播的过程：

向前传播：

向后传播backpropagation:

表示Hadamard积

算法部分伪代码：

DEFINE FUNCTION initialize\_weights\_and\_biases(self):

SET sizes TO [self.INPUT\_size] + self.hidden\_sizes + [self.output\_size]

SET weights TO [np.random.randn(sizes[i], sizes[i + 1]) FOR i IN range(len(sizes) - 1)]

SET biases TO [np.zeros((1, sizes[i + 1])) FOR i IN range(len(sizes) - 1)]

RETURN weights, biases

DEFINE FUNCTION sigmoid(self, x):

RETURN 1 / (1 + np.exp(-x))

DEFINE FUNCTION sigmoid\_derivative(self, x):

RETURN x \* (1 - x)

DEFINE FUNCTION forward(self, INPUT\_data):

SET self.INPUTs TO []

SET self.outputs TO []

SET self.activations TO []

SET current\_INPUT TO INPUT\_data

FOR i IN range(len(self.weights)):

self.INPUTs.append(current\_INPUT)

SET current\_output TO np.dot(current\_INPUT, self.weights[i]) + self.biases[i]

self.outputs.append(current\_output)

SET current\_activation TO self.sigmoid(current\_output)

self.activations.append(current\_activation)

SET current\_INPUT TO current\_activation

RETURN current\_INPUT

DEFINE FUNCTION backward(self, target, learning\_rate):

# 反向传播

SET errors TO [target - self.activations[-1]]

SET deltas TO [errors[-1] \* self.sigmoid\_derivative(self.activations[-1])]

FOR i IN range(len(self.weights) - 1, 0, -1):

errors.append(deltas[-1].dot(self.weights[i].T))

deltas.append(errors[-1] \* self.sigmoid\_derivative(self.activations[i - 1]))

# 更新权重和偏置

FOR i IN range(len(self.weights) - 1, -1, -1):

self.weights[i] += learning\_rate \* self.INPUTs[i].T.dot(deltas[len(self.weights) - 1 - i])

self.biases[i] += learning\_rate \* np.sum(deltas[len(self.weights) - 1 - i], axis=0, keepdims=True)

DEFINE FUNCTION threshold\_classify(self, output, threshold=0.5):

# 二元分类

RETURN (output > threshold).astype(int)

对于隐藏层，多次尝试，每次使用不同的结构，比如：[4], [5], [6], [4,3], [6,4]

为了找到更优架构，加入了nas神经架构搜索。

sklearn.model\_selection.GridSearchCV是scikit-learn库中的一个类，用于在给定的参数值范围内对估计器进行穷举搜索。以下是关于GridSearchCV的一些重要信息：

* 1. 目的：GridSearchCV用于优化估计器的参数，通过交叉验证对参数网格进行搜索，以找到最佳的参数组合。
  2. 成员方法：GridSearchCV实现了“fit”和“score”方法，以及其他一些方法，如“score\_samples”、“predict”、“predict\_proba”、“decision\_function”、“transform”和“inverse\_transform”。
  3. 参数：

estimator：估计器对象，必须实现scikit-learn估计器接口。估计器需要提供一个评分函数或者需要传递scoring参数。

param\_grid：字典或字典列表，包含参数名称（字符串）作为键，要尝试的参数设置列表作为值。这允许在任何参数设置序列上进行搜索。

scoring：用于评估交叉验证模型在测试集上性能的策略。可以是单个字符串或多个评分的列表、元组或字典。

n\_jobs：并行运行的作业数。

refit：是否在整个数据集上使用找到的最佳参数重新拟合估计器。

## BP神经网络训练

学习曲线描述了训练样本损失losses和循环轮数epoch的关系，这个曲线不断下降时说明训练正常进行。训练样本损失就是均方误差。

下面展示了使用不同隐藏层的输出列表和学习曲线，BP神经网络训练的初始训练轮数epoch定为10000，学习率alpha为0.01

1. Hidden\_layers: (4,)

Epoch 0, Training Loss: 0.13772397045918536, Test Accuracy: 0.885

Epoch 1000, Training Loss: 0.10159638829336196, Test Accuracy: 0.895

Epoch 2000, Training Loss: 0.10027168462712403, Test Accuracy: 0.89

Epoch 3000, Training Loss: 0.09826147731179886, Test Accuracy: 0.915

Epoch 4000, Training Loss: 0.09526613664231949, Test Accuracy: 0.915

Epoch 5000, Training Loss: 0.09243149368904582, Test Accuracy: 0.92

Epoch 6000, Training Loss: 0.09032334344147618, Test Accuracy: 0.915

Epoch 7000, Training Loss: 0.08884938408567074, Test Accuracy: 0.915

Epoch 8000, Training Loss: 0.08784804051860187, Test Accuracy: 0.92

Epoch 9000, Training Loss: 0.0871576355721264, Test Accuracy: 0.92

测试准确率: 0.92

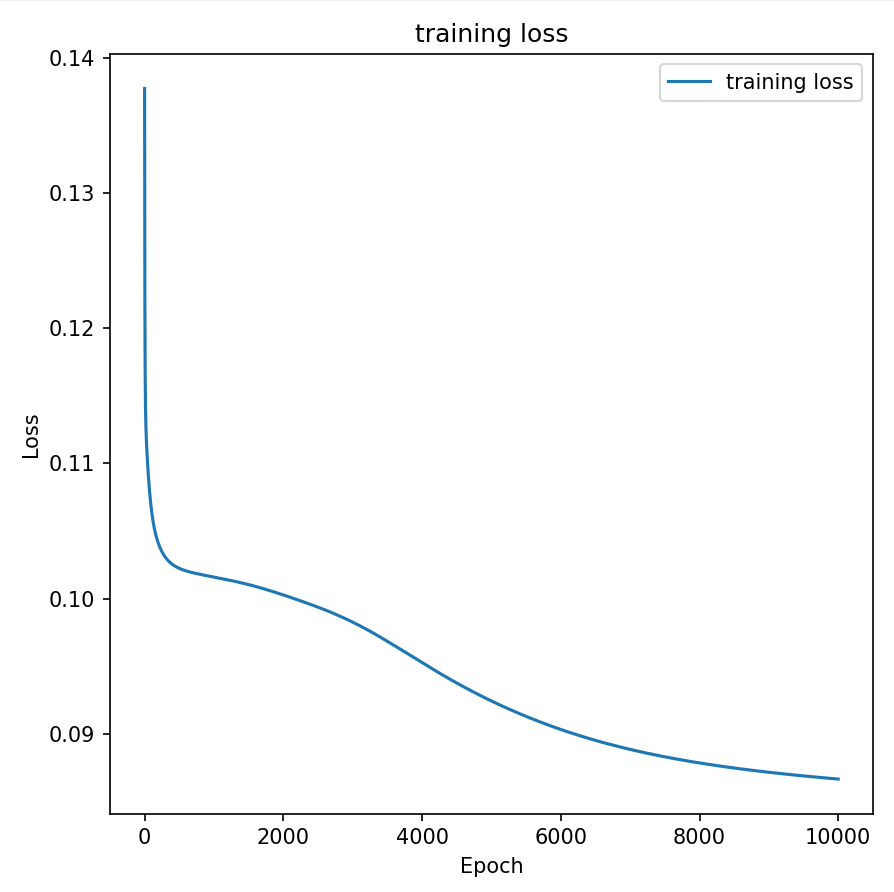


图 4‑1‑‑1： 单隐层学习曲线

1. hidden\_layers: (5,)

Epoch 0, Training Loss: 0.23058327073448268, Test Accuracy: 0.625

Epoch 1000, Training Loss: 0.10130904238803133, Test Accuracy: 0.905

Epoch 2000, Training Loss: 0.09978544545214958, Test Accuracy: 0.915

Epoch 3000, Training Loss: 0.0969828285287362, Test Accuracy: 0.92

Epoch 4000, Training Loss: 0.09041004800978455, Test Accuracy: 0.92

Epoch 5000, Training Loss: 0.08707868555153292, Test Accuracy: 0.92

Epoch 6000, Training Loss: 0.08460124363009744, Test Accuracy: 0.925

Epoch 7000, Training Loss: 0.08218592633913799, Test Accuracy: 0.925

Epoch 8000, Training Loss: 0.07967227668913753, Test Accuracy: 0.915

Epoch 9000, Training Loss: 0.07742651609269421, Test Accuracy: 0.915

测试准确率: 0.915

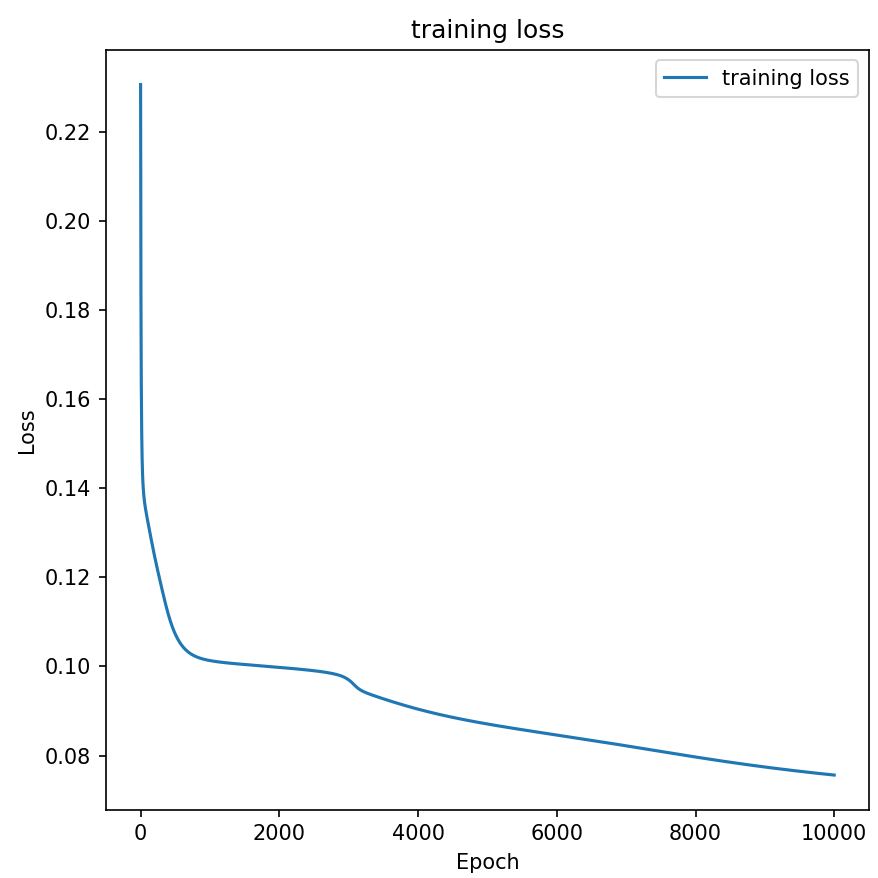


图 4‑1‑‑2： 单隐层5神经元学习曲线

本次训练观察到：虽然训练误差小于上一次训练，但训练损失函数有上升，验证集准确率也有下降，推测出现了过拟合。

1. hidden\_layers: (6,)

Epoch 0, Training Loss: 0.2575781178136836, Test Accuracy: 0.625

Epoch 1000, Training Loss: 0.09969793561589471, Test Accuracy: 0.915

Epoch 2000, Training Loss: 0.09757949071571267, Test Accuracy: 0.915

Epoch 3000, Training Loss: 0.09340985081319339, Test Accuracy: 0.93

Epoch 4000, Training Loss: 0.09114155387304114, Test Accuracy: 0.935

Epoch 5000, Training Loss: 0.08976433711405644, Test Accuracy: 0.92

Epoch 6000, Training Loss: 0.08846467161110928, Test Accuracy: 0.92

Epoch 7000, Training Loss: 0.08739419340552003, Test Accuracy: 0.92

Epoch 8000, Training Loss: 0.0865181789140507, Test Accuracy: 0.92

Epoch 9000, Training Loss: 0.08565922392754823, Test Accuracy: 0.92

测试准确率: 0.92

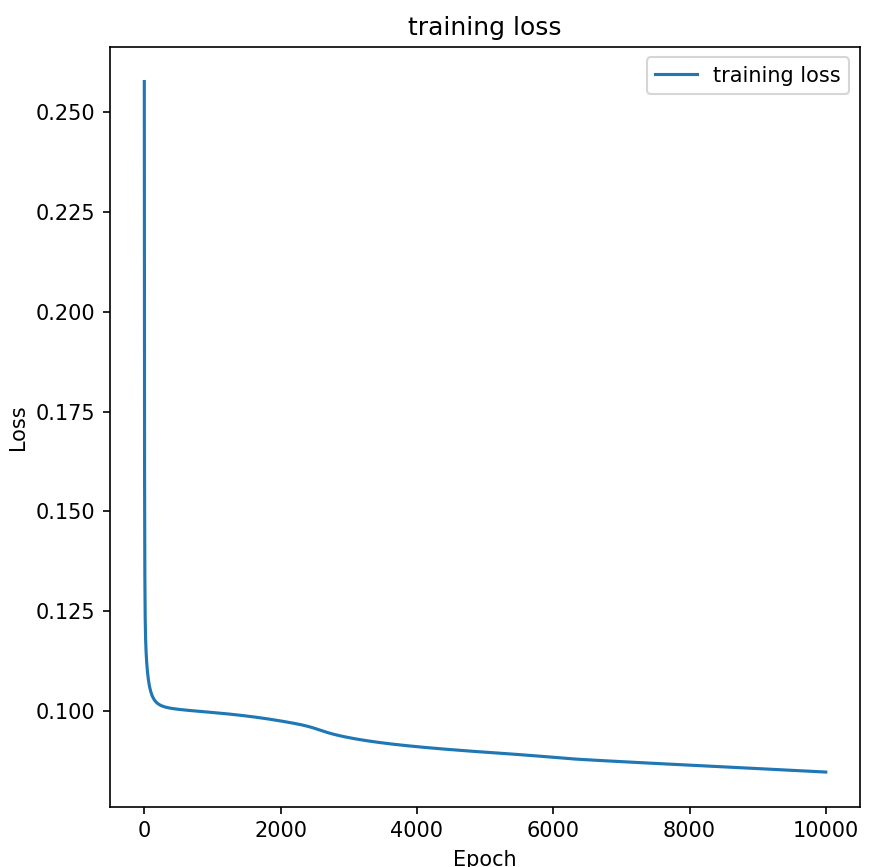


图 4‑1‑‑3： 单隐层6神经元学习曲线

这次训练中，观察到损失跟预期一样一直下降，但准确率却并非最优。

1. hidden\_layers: (4,3)

Epoch 0, Training Loss: 0.5060655187758029, Test Accuracy: 0.115

Epoch 1000, Training Loss: 0.12451101391289399, Test Accuracy: 0.885

Epoch 2000, Training Loss: 0.10289939875864196, Test Accuracy: 0.885

Epoch 3000, Training Loss: 0.09955200354991106, Test Accuracy: 0.905

Epoch 4000, Training Loss: 0.09725745004324995, Test Accuracy: 0.915

Epoch 5000, Training Loss: 0.09390646638145281, Test Accuracy: 0.915

Epoch 6000, Training Loss: 0.09072296020561912, Test Accuracy: 0.915

Epoch 7000, Training Loss: 0.08931522101521798, Test Accuracy: 0.915

Epoch 8000, Training Loss: 0.0885136627531814, Test Accuracy: 0.915

Epoch 9000, Training Loss: 0.08790582145071418, Test Accuracy: 0.915

测试准确率: 0.915

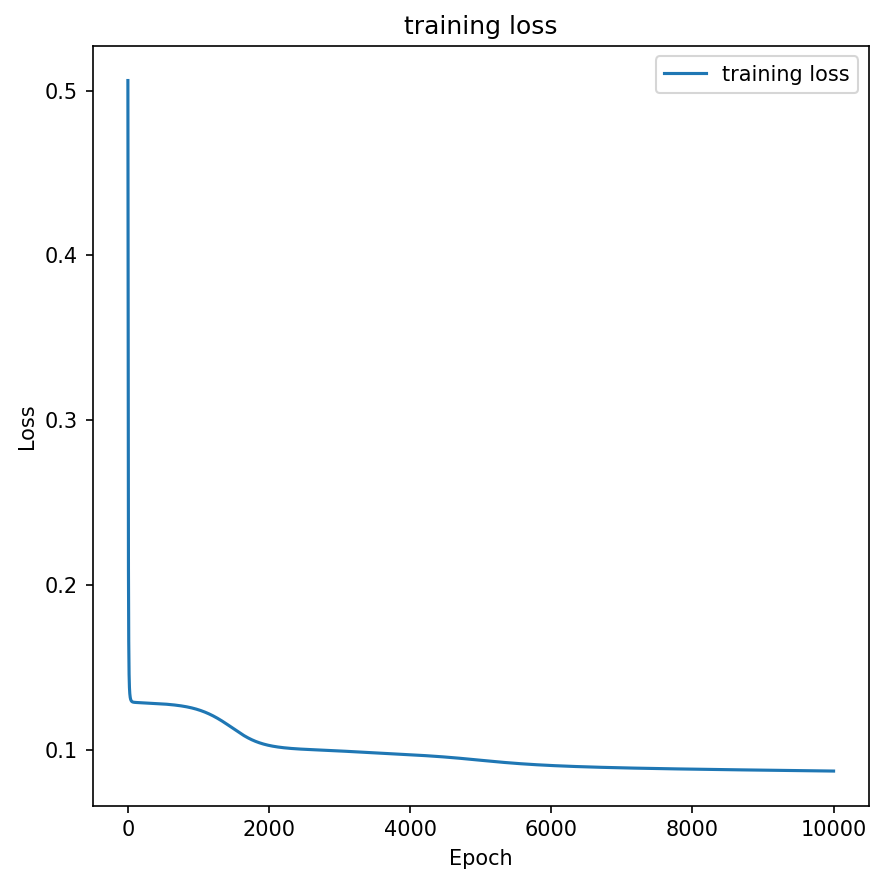


图 4‑1‑‑4：2隐层，4，3神经元的学习曲线

1. hidden\_layers: (6,4)

Epoch 0, Training Loss: 0.15087321305523113, Test Accuracy: 0.885

Epoch 1000, Training Loss: 0.10429980123806352, Test Accuracy: 0.91

Epoch 2000, Training Loss: 0.10015018828938821, Test Accuracy: 0.91

Epoch 3000, Training Loss: 0.09823780639786736, Test Accuracy: 0.91

Epoch 4000, Training Loss: 0.09573580248972104, Test Accuracy: 0.91

Epoch 5000, Training Loss: 0.09305522129940368, Test Accuracy: 0.92

Epoch 6000, Training Loss: 0.09084279419625886, Test Accuracy: 0.92

Epoch 7000, Training Loss: 0.0893245414850492, Test Accuracy: 0.925

Epoch 8000, Training Loss: 0.08826101320968181, Test Accuracy: 0.925

Epoch 9000, Training Loss: 0.08745121694350043, Test Accuracy: 0.925

测试准确率: 0.925

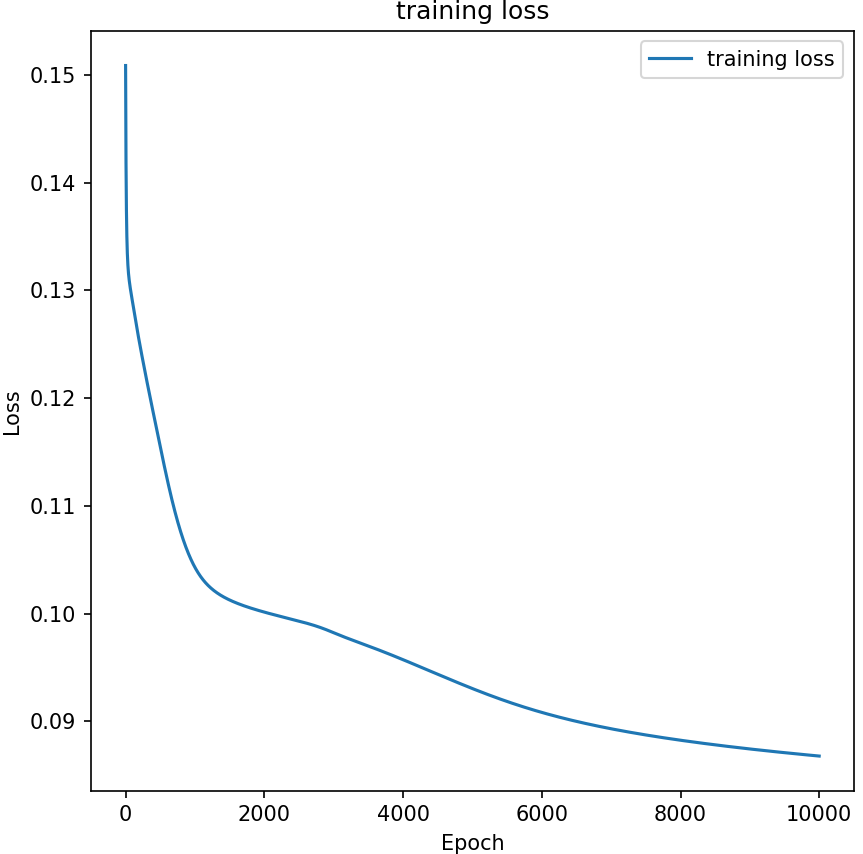


图 4‑1‑‑5：2隐藏层，6、4神经元学习曲线

下面进行神经架构搜索nas：

运行nas.py

指定搜索空间，结合上面的问题，指定搜索多种隐藏层架构，学习率。

由于准确率可能会像上面一样上升后下降，指定多种迭代次数。

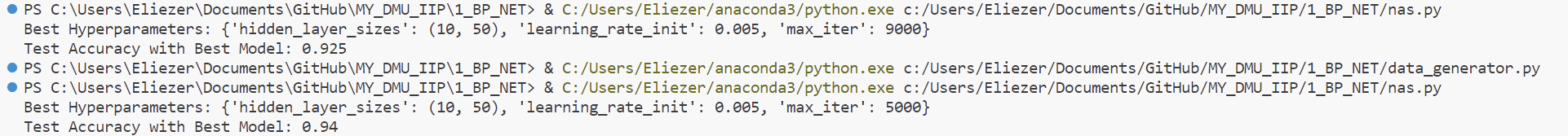
param\_grid = {

        'hidden\_layer\_sizes': [ (10), (10,50)],

        'learning\_rate\_init': [0.002,0.001,0.005],

        'max\_iter': [9000,5000]

    }



搜索到最优结果：hidden\_layers: (10,50)

按照搜索结果设置神经网络：(经过测试学习率改为0.002)

…

# 创建神经网络

    input\_size = train\_features.shape[1]

    hidden\_sizes = [10,50]

    output\_size = 1

    nn = NeuralNetwork(input\_size=input\_size, hidden\_sizes=hidden\_sizes, output\_size=output\_size)

    # 训练BP神经网络

    epochs = 5000

    learning\_rate = 0.002

…

结论：隐藏层(10,50)以及0.002的学习率，5000轮迭代次数的BP神经网络可以较好的实现这个预测模型的任务。

## BP神经网络测试

1. 运行pd.py, 测试1

Epoch 0, Training Loss: 0.1582907771861319, Test Accuracy: 0.87

Epoch 1000, Training Loss: 0.07839147739798386, Test Accuracy: 0.8675

Epoch 2000, Training Loss: 0.061909871298793794, Test Accuracy: 0.885

Epoch 3000, Training Loss: 0.05325445614132475, Test Accuracy: 0.925

Epoch 4000, Training Loss: 0.04775199244897882, Test Accuracy: 0.935

测试准确率: 0.935

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[0], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[0]

……

图表, 折线图

描述已自动生成

图表 4‑6测试1准确率曲线

1. 运行测试2

Epoch 0, Training Loss: 0.15868084439866878, Test Accuracy: 0.87

Epoch 1000, Training Loss: 0.08382264252969668, Test Accuracy: 0.875

Epoch 2000, Training Loss: 0.06941138697271797, Test Accuracy: 0.9

Epoch 3000, Training Loss: 0.06292334840221363, Test Accuracy: 0.9175

Epoch 4000, Training Loss: 0.05322846989085039, Test Accuracy: 0.9275

测试准确率: 0.9275

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[1]

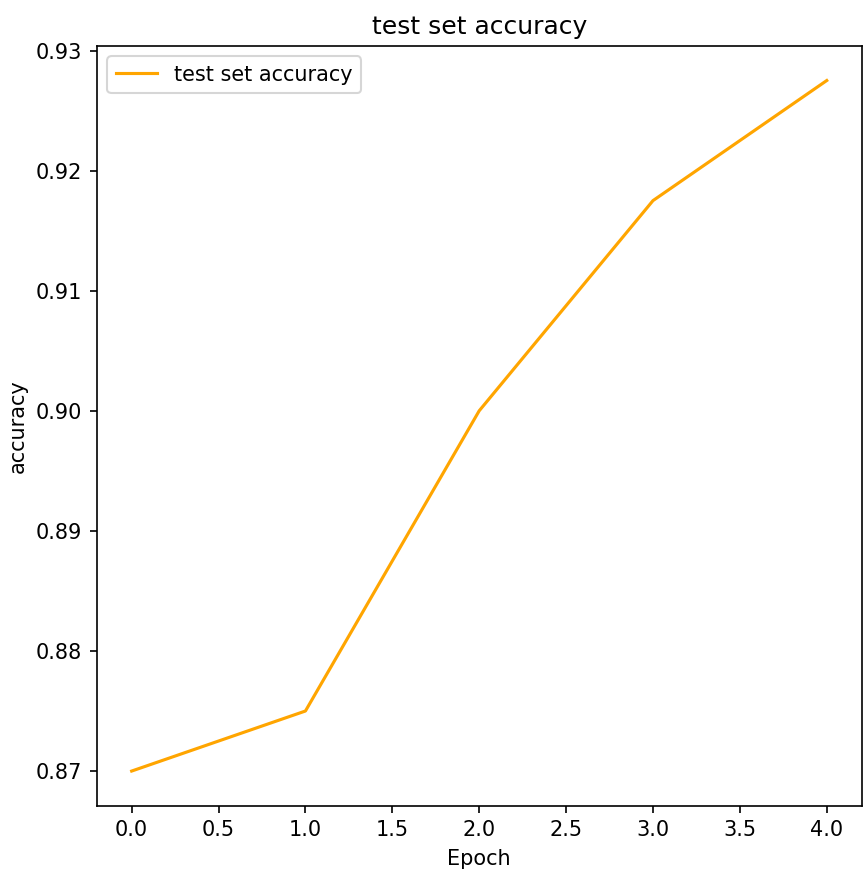
预测：[0], 实际：[1]

预测：[1], 实际：[1]

预测：[1], 实际：[0]

预测：[1], 实际：[1]

……



图表 4‑7测试2准确率曲线

# 总结与体会

本次实验中，我使用python实现了可以自定义架构的BP神经网络，使用源程序生成的数据集，用留出法划分训练和测试集，通过梯度下降法，正向传播，反向传播来优化网络，在scikit-MLPclassifier，GridSearchCV的帮助下得到相对优秀的模型，实现了测试集上较高的准确率。

但是本实验中还存在着一些问题：

1. 我的模型还存在着过拟合风险，学习曲线容易振荡，并且没有使用动态学习率等高级功能，导致准确率还不够
2. 模型未使用自适应学习率、自适应梯度下降等方法，存在数据未添加噪声、数据没有经过强化等问题。

# 参考文献

[1] João Gabriel Lima. backpropagation. Github, 2014. <https://github.com/jgabriellima/backpropagation>

[2] Matt Mazur. simple-neural-network. Github, 2015 <https://github.com/mattm/simple-neural-network>

[3] 2002–2012 John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team; 2012–2023 The Matplotlib development team. <https://matplotlib.org/stable/api/pyplot_summary.html>