

# 《人工智能》实验指导书

本实验课程是计算机、智能、物联网等专业学生的一门专业课程，通过实验，帮助学生更好地掌握人工智能中涉及的相关概念、算法，具体包括掌握不同搜索策略的设计思想、步骤、性能；掌握推理的基本方法；掌握神经网络的基本原理和学习机制。实验要求实验前复习课程中的有关内容，准备好实验数据，设计或编程要独立完成，完成实验报告。

## 实验三 神经网络

### 一、实验目的

掌握 BP 神经网络的算法原理，用 BP 神经网络算法对测试集进行预测。

### 二、实验内容

#### 1. BP 神经网络(必做)

通过实验平台 `pd.read_csv` 读取右侧数据集 `datatest.csv`，根据每一步的提示，完成相应代码，依次完成读取数据，提取特征值和标签值，对标签值进行独热编码，划分训练集和测试集，建立模型，并训练模型，对数据进行预测，将概率值转换为标签值，评估预测的准确率。

说明：数据集 `datatest.csv` 有 16 列，前 15 列为特征值，最后 1 列为标签值，并且标签值的类型只有两种。

#### 2. 选做

- (1) 认识 M-P 神经元
- (2) 搭建神经网络前向传输
- (3) 训练神经网络模型
- (4) 使用 `sigmoid` 激活函数
- (5) 使用 `tanh` 激活函数
- (6) 计算交叉熵
- (7) 计算均方误差
- (8) 随机梯度下降计算最小值
- (9) 构建房间预测模型

### 三、实验环境

中南大学 AI 能力支撑平台。



#### 四、实验步骤

- 1.进入神经网络可视化实验环境，基本实验步骤如下：
  - (1)进入实验环境；
  - (2)选择相关的实验模块；
  - (3)根据实验指导进行相应模块代码学习和编写；
  - (4)运行代码；
  - (5)观测运行结果；
  - (6)修改相应地参数观察结果变化。
- 2.编写程序实现所选模块功能。

#### 五、实验结论

- 包括做实验的目的、方法、过程等，具体要写成实验报告，见后附表三。
- 1、BP 网络的基本结构及 BP 算法的代码编写。
  - 2、试述参数变化对 BP 网络推理结果的影响。

附：神经网络实验报告表三

姓名	张文睿	指导老师:黄芳 汪洁	日期:23.6 .5
实验目的	掌握 BP 神经网络的算法原理，用 BP 神经网络算法对测试集进行预测		

读取数据，查看前5行	<pre>import pandas as pd  data=pd.read_csv('/data/shixunfiles/185be7a01dd636dd3fc5a3b6a4dd84cc_1577440382803.csv')  data.head()</pre>	<p><code>data.head()</code>用以查看前五行数据，若需打印前五行，则可改为 <code>print(data.head())</code></p>
提取特征值和标签值	<pre>X = data.iloc[:, :-1].values y = data.iloc[:, -1].values</pre>	<p><code>data.iloc[:, :-1].values</code> 用以提取 <code>data</code> 的所有行，和除了最后一列以外的所有列，将提取出特征变量的数据。</p> <p><code>data.iloc[:, -1].values</code> 用以提取 <code>data</code> 的所有行，和最后一列，将提取出目标变量的数据。</p> <p>若修改为  <pre>X= data.iloc[:, :-2].values y= data.iloc[:, -1].values</pre> 则 <code>X</code> 将变成 <code>data</code> 中所有行，除最后两列外的所有数据  <code>y</code> 将变成 <code>data</code> 倒数第二列的值</p>
对标签值进行独热编码	<pre>y = y.reshape(len(y),1) from sklearn.preprocessing import OneHotEncoder ohe = OneHotEncoder() y = ohe.fit_transform(y).toarray()</pre>	<p>首先将 <code>y</code> 的 <code>shape</code> 调整为(样本数, 1)的二维数组形式。</p> <p>然后调用 <code>fit_transform</code> 方法，将 <code>y</code> 进行独热编码转换，且该方法会同时拟合编码器的参数并对 <code>y</code> 进行编码转换。</p> <p>最后将独热编码后的结果，通过 <code>toarray</code> 方法，将结果转换为密集矩阵。</p>
划分训练集	<pre>from sklearn.model_selection import train_test_split  x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=666)</pre>	<p><code>train_test_split</code> 函数用于将数据集分为数据集和测试集。本段代码将数据集划分为 80% 的训练集和 20% 的测试集。经过该部分代码后，<code>x_train</code> 为训练集的特征变量，<code>x_test</code> 为测试集的特征变量；</p>

和测试集		<p><code>y_train</code> 为训练集的目标变量，<code>y_test</code> 为测试集的目标变量；</p> <p>若 <code>test_size</code> 修改为 0.25，则将数据集划分为 75% 的训练集和 25% 的测试集。</p>
建立模型，并进行训练	<pre>bp=bpnetwork() bp.fit(x_train,y_train,100,200,0.01,2)</pre>	<p>第一行代码创建了一个名为 <code>bp</code> 的神经网络</p> <p>第二行调用了 <code>fit</code> 方法来训练神经网络模型，<code>x_train</code> 是训练集的特征变量，<code>y_train</code> 是训练集的目标变量，迭代次数设置为 100，每次选择训练集中的 200 个样本进行训练，0.01 是学习率，即更新权重时的步长大小，2 是隐藏层的数量。</p> <p>若将第二行代码修改为 <code>bp.fit(x_train,y_train,200,100,0.005,2)</code>，则迭代次数变为 200，单次选择样本量为 100，学习率为 0.005，隐藏层数目为 2</p>
用训练的模型进行预测	<pre>ypredict= bp.predict(x_test)</pre>	<p>调用 <code>predict</code> 方法，对输入数据进行预测</p>
将概率值转换为标签值	<pre>p = np.argmax(ypredict,axis =1)#找到概率值最大的那个位置 y_test = np.argmax(y_test,axis =1)</pre>	<p>第一行代码，会在预测结果的第一行上找到最大值的索引位置。因 <code>ypredict</code> 的每一行代表一个样本的预测结果，故此处目的是找到每个样本预测结果中概率值最大的类别所在的位置，并将结果存储在变量 <code>p</code> 中。</p> <p>第二行代码，第一行代码，会在预测结果的第一行上找到最大值的索引位置。因 <code>y_test</code> 的每一行代表一个样本的预测结果，故此处目的是找到每个</p>

		<p>样本预测结果中概率值最大的类别所在的位置，并将结果存储在变量 <code>y_test</code> 中。</p>
评估预测准确率	<pre>acc = np.mean(p==y_test)  print('准确率为%.4f'%acc)</pre>	<p>第一行代码，通过比较 <code>p</code> 和 <code>y_test</code> 中的每个元素，计算其相等的比例，然后 <code>mean</code> 方法将 <code>True</code> 的比例计算为准确率。</p> <p>第二行代码，将准确率以小数点后四位的格式打印输出。</p>

`__init__(self)`: 初始化函数, 用于初始化神经网络的权重和偏置。在这个函数中, `self.w0`、`self.w1`、`self.b0`、`self.b1` 被初始化为 `None`, 表示它们的值暂时为空。

`hidden_in(self, feature, w0, b0)`: 计算隐藏层的输入函数。输入参数包括特征数据 `feature`、输入层到隐藏层之间的权重 `w0` 和偏置 `b0`。首先, 将特征数据转换为矩阵形式, 并获取特征的行数 `m`。然后, 通过矩阵乘法计算隐藏层的输入 `hidden_in = feature * w0`。接下来, 对每一行的隐藏层输入加上对应的偏置 `b0`。最后, 返回计算得到的隐藏层输入。

`sig(self, x)`: Sigmoid 函数, 将输入值 `x` 转换为概率值。这里使用了 Sigmoid 函数的常见表达式  $1.0 / (1 + \text{np.exp}(-x))$ , 通过 `np.exp` 函数计算指数值, 并将其应用于逐元素计算。

`hidden_out(self, hidden_in)`: 计算隐藏层的输出函数。输入参数为隐藏层的输入 `hidden_in`。该函数通过调用 `self.sig` 方法, 将隐藏层的输入应用于 Sigmoid 函数, 从而得到隐藏层的输出 `hidden_output`。

`predict_in(self, hidden_out, w1, b1)`: 计算输出层的输入函数。输入参数包括隐藏层的输出 `hidden_out`、隐藏层到输出层之间的权重 `w1` 和偏置 `b1`。通过矩阵乘法计算输出层的输入 `predict_in = hidden_out * w1`, 然后对每一行的输出层输入加上对应的偏置 `b1`。最后, 返回计算得到的输出层输入。

`predict_out(self, predict_in)`: 计算输出层的输出函数。输入参数为输出层的输入 `predict_in`。该函数通过调用 `self.sig` 方法, 将输出层的输入应用于 Sigmoid 函数, 从而得到输出层的输出 `result`。

`partial_sig(self, x)`: 计算 Sigmoid 函数的偏导数。输入参数为矩阵 `x`, 函数根据偏导数的计算公式  $\text{sig}(x) * (1 - \text{sig}(x))$ , 对输入矩阵中的每个元素逐个计算并返回。

`fit(self, feature, label, n_hidden, maxcycle, alpha, n_output)`: BP 神经网络的训练方法。输入参数包括特征数据 `feature`、标签数据 `label`、隐藏层的节点个数 `n_hidden`、最大迭代次数 `maxcycle`、学习率 `alpha` 和输出层的节点个数 `n_output`。在该方法

中，首先初始化网络的权重和偏置，使用随机数生成并经过一定的缩放。然后进行训练过程，根据最大迭代次数循环执行以下步骤：

fit 函数中，分为初始化和训练两部分。

初始化部分：

```
self.w0 = np.mat(np.random.rand(n,n_hidden))
```

首先，使用 `np.random.rand(n,n_hidden)` 生成一个大小为 `(n, n_hidden)` 的随机矩阵 `w0`。`np.random.rand()` 函数会生成一个由 0 到 1 之间的随机数填充的数组。`n` 表示输入层的节点数，`n_hidden` 表示隐藏层的节点数。

```
self.w0 = self.w0*(8.0*sqrt(6)/sqrt(n+n_hidden)) - np.mat(np.ones((n,n_hidden))) * (4.0*sqrt(6)/sqrt(n+n_hidden))
```

接下来，对刚才生成的随机矩阵 `w0` 进行缩放和平移操作。

`(8.0*sqrt(6)/sqrt(n+n_hidden))` 表示缩放因子，可以确保权重值适合神经网络的训练。

`np.mat(np.ones((n,n_hidden)))` 创建一个大小为 `(n, n_hidden)` 的全 1 矩阵，表示平移量。

最后，通过将缩放和平移应用于随机矩阵 `w0`，更新 `self.w0` 的值。

```
self.b0 = np.mat(np.random.rand(1,n_hidden))
```

类似地，使用 `np.random.rand(1,n_hidden)` 生成一个大小为 `(1, n_hidden)` 的随机矩阵 `b0`，表示隐藏层的偏置。

```
self.b0 = self.b0*(8.0*sqrt(6)/sqrt(n+n_hidden)) - np.mat(np.ones((1,n_hidden))) * (4.0*sqrt(6)/sqrt(n+n_hidden))
```

对刚才生成的随机矩阵 `b0` 进行缩放和平移操作，方式与权重 `w0` 相似。

`(8.0*sqrt(6)/sqrt(n+n_hidden))` 表示缩放因子，`(4.0*sqrt(6)/sqrt(n+n_hidden))` 表示平移量。

最后，通过将缩放和平移应用于随机矩阵 `b0`，更新 `self.b0` 的值。

类似地，通过上述步骤，初始化输出层的权重 `self.w1` 和偏置 `self.b1`，使用与输入层到隐藏层相似的方法进行初始化。

前向传播：

计算隐藏层的输入 `hidden_input`，调用 `self.hidden_in` 方法传入特征数据、输入层到隐藏层之间的权重和偏置。

计算隐藏层的输出 `hidden_output`，调用 `self.hidden_out` 方法传入隐藏层的输入。

计算输出层的输入 `output_in`，调用 `self.predict_in` 方法传入隐藏层的输出、隐藏层到输出层之间的权重和偏置。

计算输出层的输出 `output_out`，调用 `self.predict_out` 方法传入输出层的输入。

反向传播：

计算输出层到隐藏层之间的残差 `delta_output`，根据误差和偏导数计算得到。

计算输入层到隐藏层之间的残差 `delta_hidden`，根据输出层残差和偏导数计算得到。

更新权重和偏置，即进行梯度下降：

更新隐藏层到输出层之间的权重 `self.w1`，根据隐藏层的输出和输出层残差计算得到。

更新隐藏层到输出层之间的偏置 `self.b1`，根据输出层残差的列求和并乘以学习率。

更新输入层到隐藏层之间的权重 `self.w0`，根据特征数据和隐藏层残差计算得到。

更新输入层到隐藏层之间的偏置 `self.b0`，根据隐藏层残差的列求和并乘以学习率。

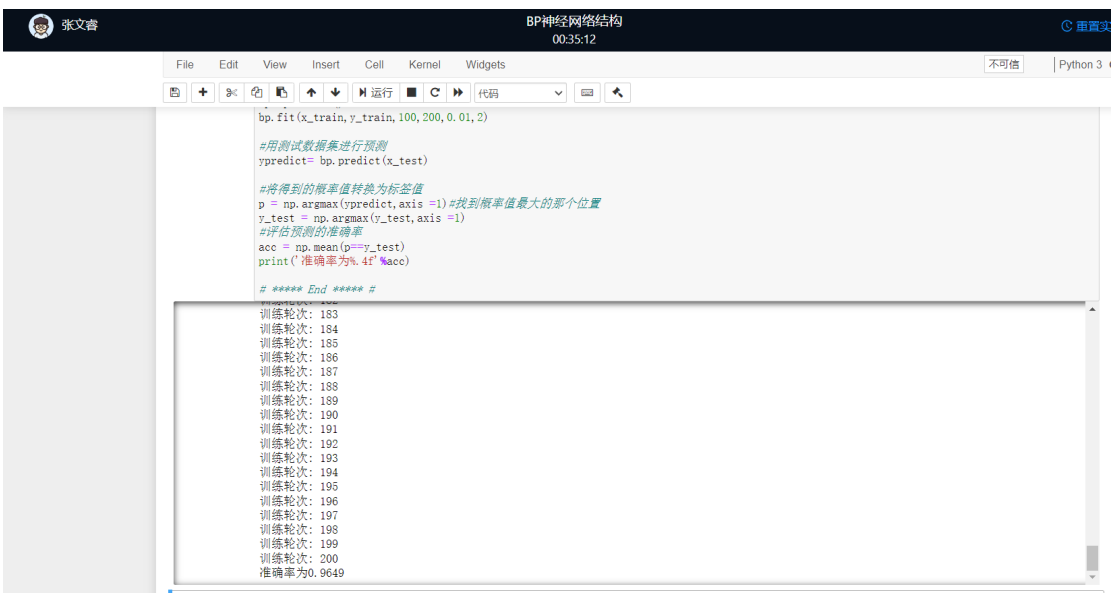
`predict(self, x_test)`: 对给定的测试样本进行预测。输入参数为测试样本 `x_test`，这里假设为一个 NumPy 数组。在该方法中，将测试样本转换为矩阵形式，并依次调用 `self.hidden_in`、`self.hidden_out`、`self.predict_in` 和 `self.predict_out` 方法，最终将预测结果转换为 NumPy 数组并返回。

这些函数共同组成了一个简单的 BP 神经网络类，用于训练和预测任务



# 运行结果展示：

按照默认参数运行程序，得到结果如图 1 所示，准确率为 0.9649



```
bp.fit(x_train,y_train,100,200,0.01,2)

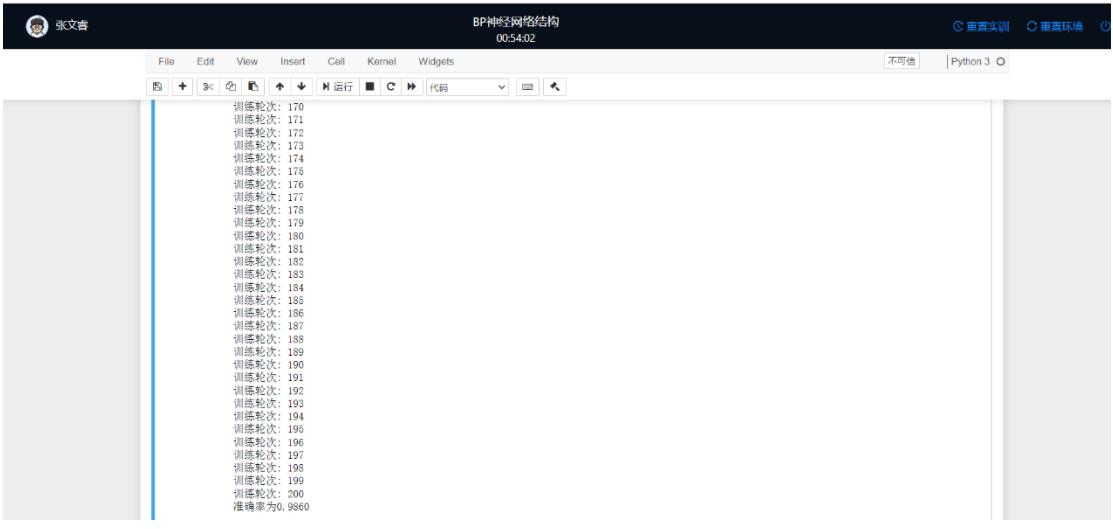
#用测试数据集进行预测
ypredict= bp.predict(x_test)

#将得到的概率值转换为标签值
p = np.argmax(ypredict,axis =1)#找到概率值最大的那个位置
y_test = np.argmax(y_test,axis =1)
#评估预测的准确率
acc = np.mean(p==y_test)
print(' 准确率为%.4f'%acc)

# ***** End ***** #
训练轮次: 183
训练轮次: 184
训练轮次: 185
训练轮次: 186
训练轮次: 187
训练轮次: 188
训练轮次: 189
训练轮次: 190
训练轮次: 191
训练轮次: 192
训练轮次: 193
训练轮次: 194
训练轮次: 195
训练轮次: 196
训练轮次: 197
训练轮次: 198
训练轮次: 199
训练轮次: 200
准确率为0.9649
```

图 1：默认参数准确率

仅修改训练集和测试集的比例，得到结果如图 2 所示，准确率为 0.9860



```
训练轮次: 170
训练轮次: 171
训练轮次: 172
训练轮次: 173
训练轮次: 174
训练轮次: 175
训练轮次: 176
训练轮次: 177
训练轮次: 178
训练轮次: 179
训练轮次: 180
训练轮次: 181
训练轮次: 182
训练轮次: 183
训练轮次: 184
训练轮次: 185
训练轮次: 186
训练轮次: 187
训练轮次: 188
训练轮次: 189
训练轮次: 190
训练轮次: 191
训练轮次: 192
训练轮次: 193
训练轮次: 194
训练轮次: 195
训练轮次: 196
训练轮次: 197
训练轮次: 198
训练轮次: 199
训练轮次: 200
准确率为0.9860
```

图 2：测试集比例 25%对应准确率

仅修改训练轮次，得到结果如图 3 所示，准确率为 0.9737

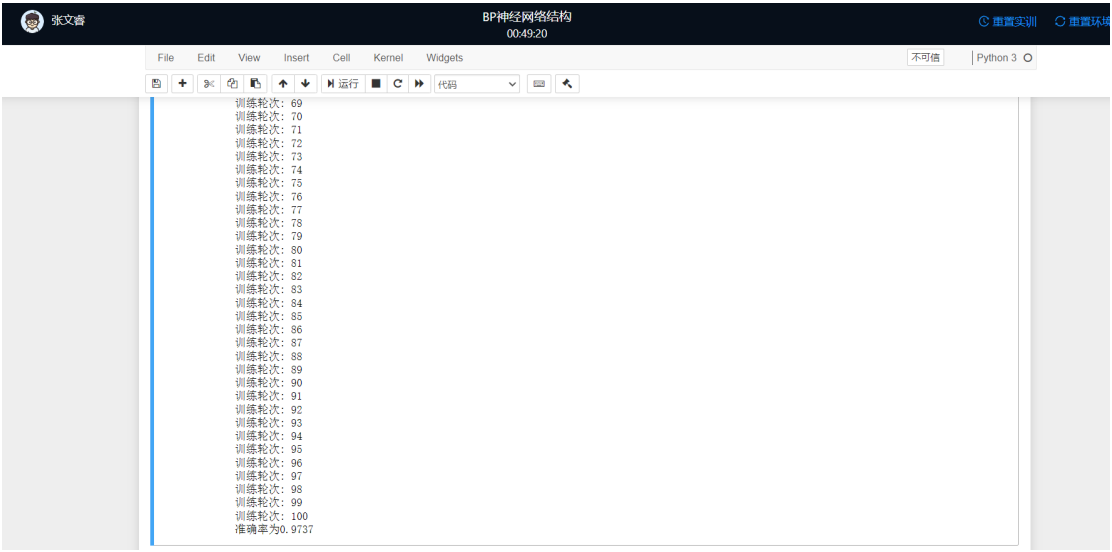


图 3：训练轮次 100 次对应准确率

仅修改学习率，得到结果如图 4 所示，准确率为 0.9649

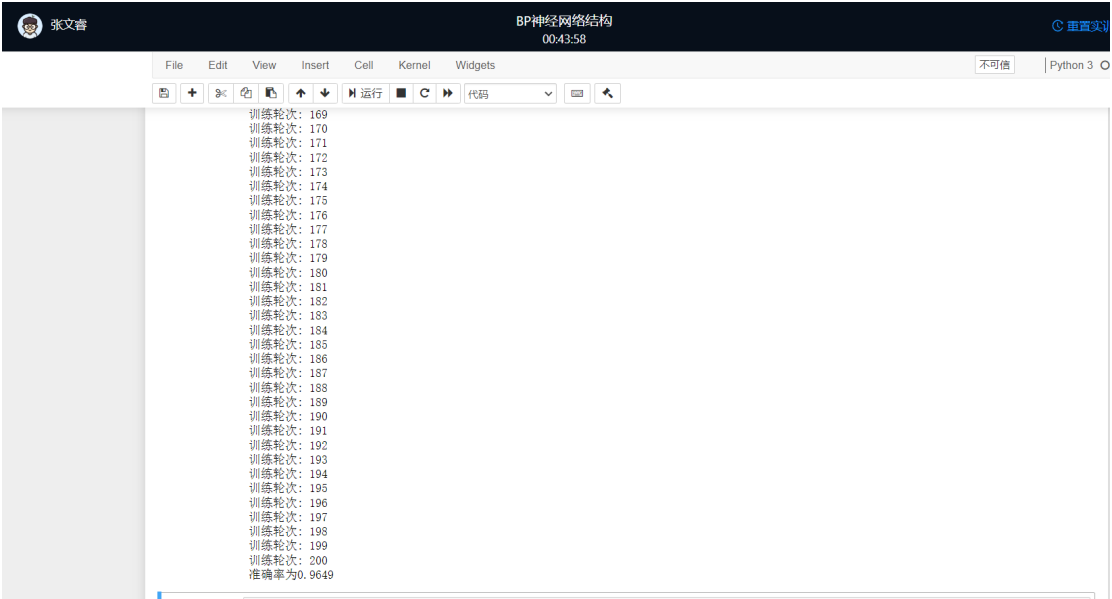


图 4：学习率 0.02 对应准确率

根据实验结果可知，学习率、测试集训练集的比例、训练轮次和隐层数目是影响 BP 神经网络的预测准确率的关键参数。它们的设置会对神经网络的训练过程和最终的预测性能产生影响。

**学习率（Learning Rate）：**学习率决定了在每次权重更新时调整权重的步长大小。过大的学习率可能导致权重更新过大，训练不稳定或发散；而过小的学习率可能导致收敛缓慢，需要更多的训练轮次才能达到较好的性能。因此，适当选择学习率可以帮助网络更快地收敛并取得更好的预测准确率。

**测试集训练集的比例（Train-Test Split Ratio）：**将数据集划分为训练集和测试集是为了评估模型在未见过的数据上的泛化能力。测试集的比例选择过小可能导致测试集样本不足以准确评估模型的性能，而选择过大可能会导致训练集样本不足以训练出具有良好泛化能力的模型。因此，选择适当的训练集和测试集比例可以帮助避免过拟合或欠拟合，从而提高预测准确率。

**训练轮次（Number of Training Epochs）：**训练轮次表示模型在整个训练集上迭代的次数。过少的训练轮次可能导致模型无法充分学习训练集的特征，而过多的训练轮次可能导致模型过拟合训练集的特征，泛化能力下降。因此，选择适当的训练轮次可以帮助在训练集上充分学习特征并在测试集上获得良好的预测准确率。

**隐层数目（Number of Hidden Layers）：**神经网络的隐层数目决定了网络的复杂性和表示能力。较少的隐层可能导致网络无法学习复杂的非线性关系，而过多的隐层可能导致网络过于复杂，难以训练和泛化。因此，选择适当的隐层数目可以平衡网络的复杂性和训练的有效性，从而影响预测准确率。

# 正则化优化神经网络

## 一、实验目的

本关任务：编写一个能使用 L2 正则化优化神经网络的小程序

## 二、实验内容

任务需要补全文件中 **Begin-End** 中间的代码，实现 L2 正则化技术来解决过拟合问题，提高验证集上的准确率，具体分为如下几个部分：

导入训练集与测试集数据；

设置添加 L2 正则化的 adam 优化器；

搭建神经网络并进行训练。

## 三、实验环境

中南大学 AI 能力支撑平台

## 四、实验步骤

- ① 导入训练集与测试集数据
- ② 设置添加 L2 正则化的 adam 优化器
- ③ 搭建神经网络并进行训练
- ④ 会根据搭建的模型精度是否可以超越 90%进行判断，若超过 90%会说输出预期结果，否则会输出“模型仍需进行调整”

## 五、程序介绍

```
batch_size_train = 64、batch_size_test = 1000、learning_rate = 0.01、  
log_interval = 10、random_seed = 1
```

这些变量定义了批处理大小、学习率、日志间隔和随机种子等超参数。

```
torch.manual_seed(random_seed)
```

这行代码设置随机种子，用于使实验可复现。

`train_loader` 和 `test_loader` 定义了训练数据集和测试数据集的数据加载器。

`torchvision.datasets.MNIST` 用于加载 MNIST 数据集，`train=True` 表示加载训练集，`train=False` 表示加载测试集。

`torchvision.transforms.Compose` 用于定义数据预处理的组合，例如将图像转换为张量和归一化等。

`batch_size` 表示每个批次的样本数，`shuffle=True` 表示在每个迭代中随机打乱数据。

```
class Net(nn.Module):  
    def __init__(self):
```

这部分代码定义了一个名为 `Net` 的神经网络模型类，并在 `__init__` 方法中定义了神经网络的结构。

神经网络有 7 个全连接层(`fc1` 到 `fc7`)，其中前 6 层的输出大小为 700，最后一层的输出大小为 10，对应 10 个类别。

```
    def forward(self, x):
```

这个方法定义了神经网络的前向传播过程，即将输入 `x` 传递到网络中，生成输出。

首先，将输入展平为一维向量，然后通过 `ReLU` 激活函数依次通过每个全连接层。

最后，通过 `F.log_softmax` 函数对输出进行 `log_softmax` 操作，得到概率分布。

```
network = Net() 和 optimizer = optim.Adam(network.parameters(),  
lr=learning_rate, weight_decay=0.0018)
```

这部分代码创建了一个 `Net` 类的实例作为神经网络模型。

`optim.Adam` 是一个优化器，用于更新神经网络的参数。

`network.parameters()` 获取神经网络模型的参数，用于优化器的更新操作。

`lr=learning_rate` 表示学习率，`weight_decay` 表示 L2 正则化的权重衰减系

数。

`def train(epoch):`和 `def test():`

这两个函数分别定义了训练过程和测试过程。

`train(epoch)`函数用于进行一轮训练，它遍历训练数据集的每个批次，计算损失并更新网络参数。

`test()`函数用于在测试数据集上评估模型的性能，计算损失和准确率。

`train(1)`

这行代码调用 `train` 函数开始进行一轮训练。

`epoch` 表示训练的轮数，这里设置为 1。

这段代码的作用是定义了一个具有 7 个全连接层的神经网络模型，并在 MNIST 数据集上进行训练和测试。它使用 Adam 优化器来更新模型参数，通过交叉熵损失函数计算损失，并使用 ReLU 激活函数进行非线性变换。最后，打印训练过程中的损失信息和在测试集上的准确率

## 六、 程序结果展示

程序运行结果如图 5 所示：



图 5：正则化优化神经网络结果

## 七、 代码附录

```
import warnings
warnings.filterwarnings("ignore")
import torch
import torchvision
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

batch_size_train = 64

batch_size_test = 1000
learning_rate = 0.01
log_interval = 10
random_seed = 1
torch.manual_seed(random_seed)

#####begin#####
# 导入数据
# 数据路径: /data/workspace/myshixun/data/
train_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('/data/workspace/myshixun/data/',
                               train=True,
                               download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_train, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('/data/workspace/myshixun/data/',
                               train=False,
                               download=True,
                               transform=torchvision.transforms.Compose([
                                   torchvision.transforms.ToTensor(),
                                   torchvision.transforms.Normalize(
                                       (0.1307,), (0.3081,))
                               ])),
    batch_size=batch_size_test, shuffle=True)

# 搭建神经网络
class Net(nn.Module):
```

```

def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(784, 700)
    self.fc2 = nn.Linear(700, 700)
    self.fc3 = nn.Linear(700, 700)
    self.fc4 = nn.Linear(700, 700)
    self.fc5 = nn.Linear(700, 700)
    self.fc6 = nn.Linear(700, 700)
    self.fc7 = nn.Linear(700, 10)

def forward(self, x):
    x = x.view(-1, 784)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
    x = F.relu(self.fc4(x))
    x = F.relu(self.fc5(x))
    x = F.relu(self.fc6(x))
    x = self.fc7(x)
    return F.log_softmax(x)

```

# 创建网络与优化器设置

#####end#####

```

network = Net()
optimizer = optim.Adam(network.parameters(), lr=learning_rate,
weight_decay=0.0018)

```

# 训练函数

```

def train(epoch):
    network.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data = data
        target = target
        optimizer.zero_grad()
        output = network(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                    100. * batch_idx / len(train_loader), loss.item()))

```



# 测试函数

def test():

network.eval()

test\_loss = 0

correct = 0

with torch.no\_grad():

for data, target in test\_loader:

data = data

target = target

output = network(data)

test\_loss += F.nll\_loss(output, target, size\_average=False).item()

pred = output.data.max(1, keepdim=True)[1]

correct += pred.eq(target.data.view\_as(pred)).sum()

test\_loss /= len(test\_loader.dataset)

print("\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%) \n".format(

test\_loss, correct, len(test\_loader.dataset),

100. \* correct / len(test\_loader.dataset)))

return 100 \* float(correct) / float(len(test\_loader.dataset))

train(1)