实验六：多层感知机分类

一、实验介绍

1.1 实验内容

多层感知机（MLP，Multi-Layer Perceptron）是一种经典的人工神经网络架构。本次实验将带领了解多层感知机的基本原理，并学习使用 **pytorch** 来构建一个简单的多层感知机分类模型，最后使用此模型预测鸢尾花的种类。

1.2 实验知识点

多层感知机的基本原理。

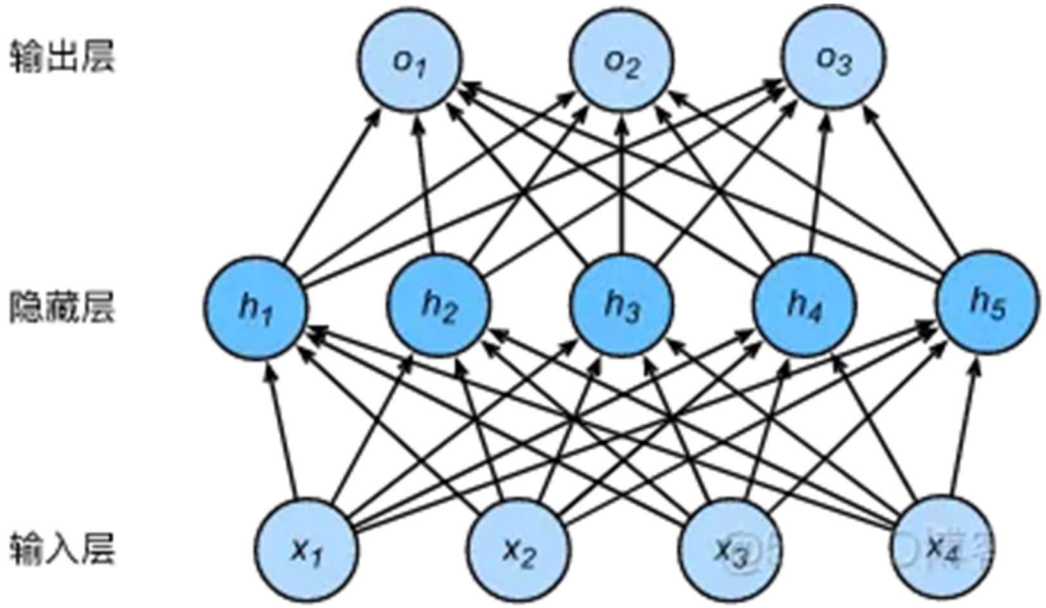
使用 **pytorch** 搭建简单的多层感知机分类模型。

使用鸢尾花数据集对多层感知机模型进行实例验证。

二、多层感知机基本原理

2.1 多层感知机简介

多层感知机（MLP）是一种常用的前馈神经网络模型，由输入层、多个隐藏层和输出层组成。每层的神经元通过加权连接传递数据，并使用激活函数引入非线性，以提高模型的表达能力。MLP 的主要特点是全连接结构，即每个神经元与前一层的所有神经元连接。通过反向传播算法调整权重，MLP 可以学习输入数据与输出目标之间的映射关系。MLP 在图像识别、语音处理等任务中有广泛应用，其结构简单，通常与其他深度学习模型结合使用。



2.2 多层感知机学习

多层感知机（MLP）的学习过程是基于反向传播算法进行的，其目的是通过训练数据逐步优化网络中的权重参数，以使模型的预测结果尽可能接近实际输出。具体来说，MLP 的学习步骤如下：

前向传播：输入数据经过网络的各层处理，并通过激活函数逐层计算输出，最终得到预测结果。

计算损失：将预测结果与真实值进行比较，计算损失函数（如均方误差或交叉熵），衡量当前网络的预测误差。

反向传播：从输出层开始，逐层计算损失相对于各权重的梯度，利用链式法则进行误差传播，将误差信息向输入层传递。

权重更新：通过梯度下降算法（或其变种，如 Adam 优化器），利用计算出的梯度更新网络中的权重和偏置参数，使模型逐步降低损失值。

三、鸢尾花分类实验

使用 **pytorch** 构建一个多层感知机模型，实现鸢尾花分类。

3.1 数据集简介

鸢尾花数据集是机器学习领域一个非常经典的分类数据集。接下来，我们就

用这个训练集为基础，一步一步地训练一个机器学习模型。首先，我们来看一下该数据集的基本构成。数据集名称的准确名称为 Iris Data Set，总共包含 150 行数据。每一行数据由 4 个特征值及一个目标值组成。其中 4 个特征值分别为：萼片长度、萼片宽度、花瓣长度、花瓣宽度。而目标值及为三种不同类别的鸢尾花，分别为：Iris Setosa，Iris Versicolour，Iris Virginica。

3.2 数据获取及划分

通过著名的 UCI 机器学习数据集网站下载该数据集。本实验中，为了更加便捷地实验。我们直接实验 scikit-learn 提供的方法导入该数据集即可。打开实验环境右下角的菜单 > 附件 > ipython，依次键入代码。

```
#基于多层感知机的鸢尾花分类
'''
- 程序开发环境：win10 64 位
- Python 版本：64 位 3.7
- 依赖库：torch、numpy、pandas、sklearn、matplotlib
- 程序输入：iris.cvs
- 程序输出：predicted(预测值)、accuracy（准确率）
# -*- coding: utf-8 -*-
'''
```

接下来，可以直接通过 print iris_target 查看一下花的分类数据。这里，scikit-learn 已经将花的原名称进行了转换，其中 0，1，2 分别代表 Iris Setosa，Iris Versicolour 和 Iris Virginica。

这些数据是按照鸢尾花类别的顺序排列的。所以，如果将其直接划分为训练集和数据集的话，就会造成数据的分布不均。详细来讲，直接划分容易造成某种类型的花在训练集中一次都未出现，训练的模型就永远不可能预测出这种花来。你可能会想到，将这些数据打乱后再划分训练集和数据集。当然，更方便地，scikit-learn 为我们提供了训练集和数据集的方法。

其中，x_train，x_test，y_train，y_test 分别代表训练集特征、测试集

特征、训练集目标值、验证集特征。test_size 参数代表划分到测试集数据占全部数据的百分比，你也可以用 train_size 来指定训练集所占全部数据的百分比。一般情况下，我们会将整个训练集划分为 70% 训练集和 30% 测试集。最后的 random_state 参数表示乱序程度。

导入模块

```
import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score

from torch.utils.data import DataLoader, TensorDataset
```

读取数据

```
data = load_iris()

X = data.data  # 特征数据

y = data.target  # 标签数据
```

数据可视化

#4 种特征维度分布情况

```
#直方图 histograms

data.hist()
```

```
plt.show()
```

数据标准化及数据格式转换

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# 数据标准化

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

# 将数据转换为 PyTorch 的 Tensor 格式

X_train_tensor = torch.tensor(X_train, dtype=torch.float32)

y_train_tensor = torch.tensor(y_train, dtype=torch.long)

X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

y_test_tensor = torch.tensor(y_test, dtype=torch.long)
```

```
# 创建 DataLoader

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

3.3 模型构建及训练预测

首先是通过 pytorch 构建多层感知机分类模型。然后实验 fit 方法和 predict 方法对模型进行训练和预测。

MLP() 模型方法中也包含非常多的参数值。例如：

Input_size= X_train.shape[1]用来选择输入模型的数据形状。

hidden_size= 16 用来确定隐藏层的神经元数量，可根据数据大小自行调节。

num_classes = 3 用来确定数据类别数，鸢尾花数据分为 3 类，所以其值为 3。

可以将预测结果和测试集的真实值分别输出，对照比较。

激活函数选择

ReLu 函数是一个通用的激活函数，目前在大多数情况下使用。但是，ReLU 函数只能在隐藏层中使用。

用于分类器时，sigmoid 函数及其组合通常效果更好。由于梯度消失问题，有时要避免使用 sigmoid 和 tanh 函数。

在神经网络层数较多的时候，最好使用 ReLu 函数， ReLu 函数比较简单计算量少，而 sigmoid 和 tanh 函数计算 大很多。

在选择激活函数的时候可以先选用 ReLu 函数如果效果不理想可以尝试其他激活函数。

模型构建

```
# 定义多层感知机模型

class MLP(nn.Module):

    def __init__(self, input_size, hidden_size, num_classes):

        super(MLP, self).__init__()

        self.layer1 = nn.Linear(input_size, hidden_size)

        self.relu = nn.ReLU()

        self.layer2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):

        x = self.layer1(x)

        x = self.relu(x)

        x = self.layer2(x)

        return x

# 设置模型参数

input_size = X_train.shape[1]  # 输入特征数

hidden_size = 16  # 隐藏层神经元数量，可调

num_classes = 3  # 类别数（鸢尾花有三类）



# 实例化模型

model = MLP(input_size, hidden_size, num_classes)

# 定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)
```

训练分类

```
# 训练模型
```

```
num_epochs = 50

for epoch in range(num_epochs):

    for inputs, labels in train_loader:

        outputs = model(inputs)

        loss = criterion(outputs, labels)



        optimizer.zero_grad()

        loss.backward()

        optimizer.step()

    if (epoch + 1) % 20 == 0:

        print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}")
```

测试集预测

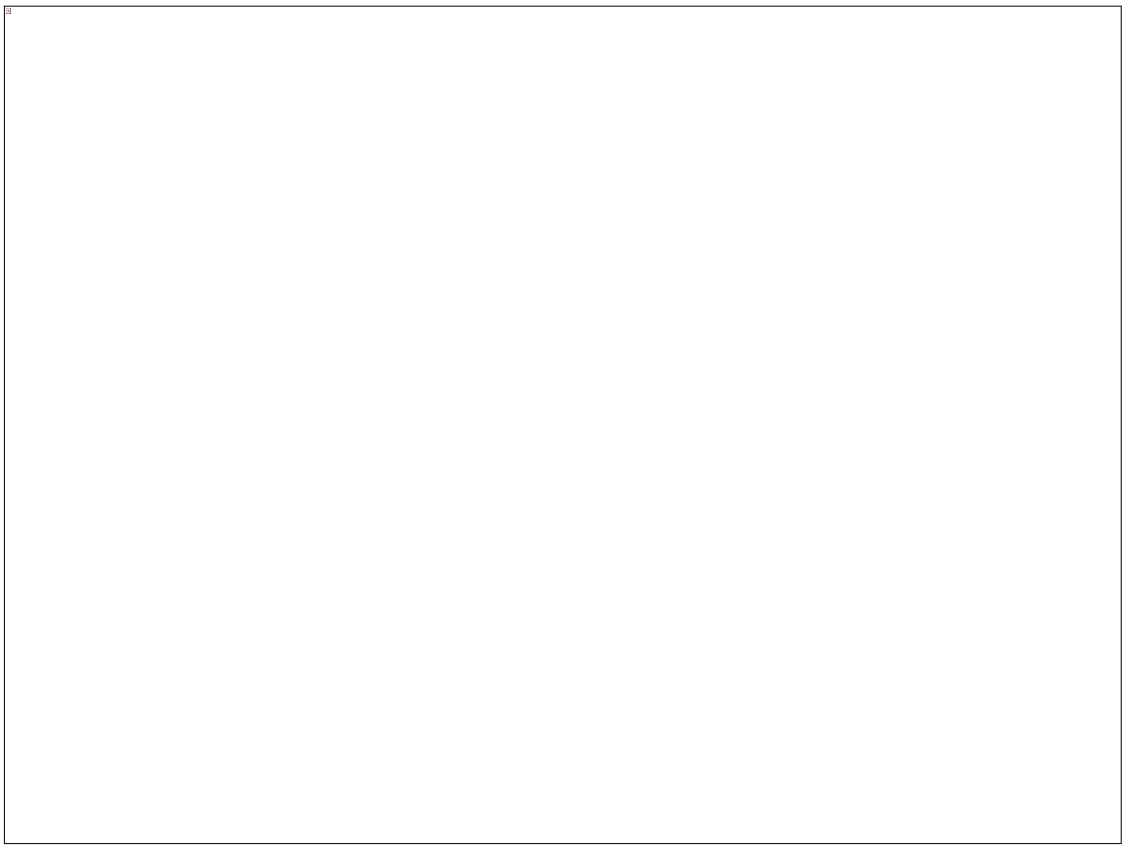
```
# 在测试集上评估模型
```

```
model.eval()

with torch.no_grad():

    test_outputs = model(X_test_tensor)

    _, predicted = torch.max(test_outputs, 1)

    accuracy = accuracy_score(y_test, predicted)

    print(f'Accuracy on test set: {accuracy * 100:.2f}%')
```

绘图

```
plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')

plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versicolor')
```

```
plt.scatter(X[100:, 0], X[100:, 1], color='green', marker='s', label='Virginica')



#中文乱码解决

plt.rcParams['font.sans-serif']=['SimHei']

plt.rcParams['axes.unicode_minus']=False



plt.title("DTC 基于决策数的鸢尾花分类")#标题

plt.xlabel('Sepal length')

plt.ylabel('Sepal width')

plt.xticks(())

plt.yticks(())

plt.legend(loc=2)

plt.show()
```



四、课后习题

尝试通过修改 MLP（）方法里面的参数值，查看模型参数对实验结果带来的影响。

尝试载入 scikit-learn 中提供的另一个著名的 digits 数据集，同样实验多层感知机分类器实现手写字体识别实验。