实训的课程安排

一、项目中使用的开发工具

Pycharm

二．项目技术栈:

编程语言：python ， 深度学习框架：PyTorch (facebook /meta),

数据处理库：numpy , 图像处理库：PIL , Open CV

可视化库：Matplotlib等

三．项目内容:

1. 使用MNIST数据集实现数字识别的案例。

引入深度学习相关内容，全连接神经网络，卷积神经网络。

2. 讲一些线性回归的知识点。

线性回归的理解，进而更好地理解深度学习。手写线性回归，pytorch实现线性回归。集合元宇宙实验平台的案例对线性回归相关内容进行学习。

3. 实现水果分类的项目。

再次使用深度学习，训练出能够识别出不同种水果的模型。卷积神经网络的应用

4. 水果分拣（元宇宙3d项目的展示）

模型在生产线的实际应用的展示。

四．项目的开发步骤:

1.收集与准备

2.数据预处理模型

3.模型设计

5.模型训练

6.模型评估和调优

7.结果可视化等。

五．MNIST的开发

1.环境准备

```
需要使用到torchvision，matplotlib,如果没有就需要先安装
需要使用到pytorch，如果没有就需要先安装
pip install torchvision ，  pip install matplotlib
```

2.加载数据集

1.下载或者读取数据集

```
# 1 需要使用到torchvision，matplotlib,如果没有就需要先安装
# pip install torchvision ,  pip install matplotlib
import matplotlib.pyplot as plt
import torchvision

# 2.下载数据集(训练集，测试集)
# train=True 训练集，  train=False 测试集
# download=True 下载数据，  download=False ,加载本地数据集
train_data = torchvision.datasets.MNIST("",train=True , download=False)
test_data = torchvision.datasets.MNIST("",train=False , download=False)

# 3.读取其中一个数据值，显示这个数据
image,label = train_data[0] # image --train_data[0][0] , label -- train_data[0]
[1]
print(image , label)
print(train_data[0][1])
plt.imshow(image)
plt.title(label)
plt.show()
```

2.保存数据集中的图片

```
#1.加载数据集，然后读取数据集中的内容，存储为图片
# 迭代器的使用
import torchvision
import matplotlib.pyplot as plt

#2.加载数据
train_data = torchvision.datasets.MNIST("",train=True,download=False)

#3.迭代器
inter = iter(train_data)
#4.保存数据
for i in range(16):
    image ,label = next(inter)
    print(image , label)
    # 设置保存路径
    image.save(f"imgs//{i}_{label}.png")
    plt.subplot(2,8,i+1) # 2行8列
    plt.title(label)
    plt.imshow(image)
plt.show()
```

3.张量形式显示图片

```
# 张量的使用，  图片读取到程序中后，如何转换为方便使用的张量
from torchvision.transforms import ToTensor ,Resize ,Compose
from PIL import Image

# 1. 加载图片，得到一个文件类型
image = Image.open("./imgs/0_5.png")
```

```
# 2. 张量转换器
to_tensor = ToTensor()
# 3.文件类型转换为张量
image_tensor = to_tensor(image)
# 4. 测试类型
print(type(image_tensor)) # <class 'torch.Tensor'>
print(type(image)) # <class 'PIL.PngImagePlugin.PngImageFile'>
# 5. 张量的使用
print(image_tensor.shape)
print(image_tensor)
# 6. 改变张量的size
m = Resize((14,14))
image_tensor = m(image_tensor)
print(image_tensor.shape)
```

4.激活函数的使用对比

```
# 不同的激活函数对预测值的影响
# 激活函数：
# 1.引入非线性，神经网络旨在处理各种复杂的非线性问题，而非线性模型的表达能力有限。
# 激活函数能为神经网络引入非线性因素，使网络可以学习和模拟各种复杂的非线性关系，通过激活函数的
# 使用，神经网络能拟合复杂的函数曲线，处理图像识别，语音识别等领域中的非线性问题。
# 2. 增加模型的表达能力 ， 激活函数使神经网络能够逼近任何复杂的函数，增加了模型的表示能力和灵活
性
# 提高模型对复杂任务的处理能力
# 3. 决定神经元的输出状态，激活函数根据神经元的输入来确定其输出。
# 4. 缓解梯度消失和爆炸问题，在神经网络训练中，梯度消失或爆炸（模型训练中的一种异常情况，反向
# 传播过程中，梯度值变得过大，呈指数级增长，导致模型无法正常训练）会导致训练困难。
# 5. 实现特征的稀疏性 ，像ReLU这样的激活函数，会使大量神经元的输出为0，从而使模型具有稀疏性，
这有助于
# 减少模型的参数数量，降低模型的复杂度，提高模型的泛化能力，还能加快模型的训练速度，减少计算量。
import matplotlib.pyplot as plt
import torch
import numpy as np
# 1. 产生100个数据，数据区间是-10~~10
x = np.linspace(-10,10,100)
x = torch.tensor(x) # 转换为张量
# 2. 使用RuLU激活函数
rulu = torch.nn.ReLU()
y = rulu(x)
plt.subplot(1,3,1)
plt.title("ReLU")
plt.plot(x,y,"b--")

# 3. 使用Sigmoid激活函数
sigmoid = torch.nn.Sigmoid()
y = sigmoid(x)
plt.subplot(1,3,2)
plt.title("Sigmoid")
plt.plot(x,y,"r--")

# 4.调用Tanh激活函数
tanh = torch.nn.Tanh()
y = tanh(x)
plt.subplot(1,3,3)
plt.title("Tanh")
```

```
plt.plot(x,y,"g--")

plt.show()
```

5.定义神经网络模型

```
# 使用pytorch框架， 自定义类，然后继承nn.Model ， 完成构造函数和forward函数
# 创建一个用于数字识别的模型：
# 1， 使用全连接神经网络模型 FNN_Model.py （输入数据比较小的时候，可以选择只用全连接神经网络）
# 2. 使用卷积神经网络模型 CNN_Model.py（输入数据比较大的时候，
#      需要先使用卷积神经网络，然后展平， 在使用全连接神经网络）

import torch
class MyModel(torch.nn.Module):
    # 构造方法
    def __init__(self):
        super(MyModel , self ).__init__()
        print(f"执行自定义的神经网络的构造方法，该方法中定义神经网络中的每一层")

    # 前向传播的方法 ，给神经网络模型传入参数之后， 会自动调用前向传播的方法。
    def forward(self , x):
        print(f"执行前向传播的方法，输入是x。")
        return  x ** 2

if __name__ == "__main__":
    # 创建神经网络模型 ，创建模型对象的时候， 会自动调用构造函数
    module = MyModel()
    y = module(3) # 给模型传入参数， 则会调用前向传播函数
    print(y)
```

全连接神经网络模型

```
import torch
import torch.nn as nn
class FNN_Model(nn.Module):
    #构造函数
    def __init__(self):
        super(FNN_Model , self).__init__()
        self.sequential = nn.Sequential(
            # a.张量展平： MNIST中的图片原结构为（1，28，28），展平后变为一个
            # 一维数组，张量为(1,28*28)
            nn.Flatten(),
            # b.经过全连接层
            nn.Linear(1*28*28,100),
            # c.激活函数
            nn.ReLU(),
            # d.经过全连接层
            nn.Linear(100,10),
            # e.归一化处理：使多分类问题的概率值和为1.
            nn.LogSoftmax(dim=1) # LogSoftmax函数，将上一步的输出特征，映射到（0，1）范围内
        )
```

```
    def forward(self , x):
        return self.sequential(x);
```

6.训练FNN模型

```
import torch.optim
import torchvision
from FNN_Model import FNN_Model
from torch.utils.data import DataLoader
import numpy as np

# 训练模型
# 1.定义超参数
EPOCH = 3
BATCH_SIZE = 10
LEARNING_RATE = 0.001
# 2.加载数据集: 需要做张量的转换
to_tensor = torchvision.transforms.ToTensor()
train_data = torchvision.datasets.MNIST("",transform=to_tensor,
train=True,download=False)
test_data =
torchvision.datasets.MNIST("",transform=to_tensor,train=False,download=False)
train_loader = DataLoader(train_data , batch_size= BATCH_SIZE , shuffle=True)
test_loader = DataLoader(test_data , batch_size= BATCH_SIZE , shuffle=True)
# 3.创建自定义的神经网络模型，优化器，损失函数
model = FNN_Model()
opt = torch.optim.Adam(model.parameters() , lr = LEARNING_RATE)
loss_fun = torch.nn.NLLLoss()
# 4.评估函数
def eval(model,test_loader):
    # 1.开启评估
    model.eval()
    # 2.评估时候不要调整训练参数（不计算梯度）
    with torch.no_grad():
        # 测试样本总数，正确数
        total = 0
        correct = 0
        # 遍历测试集
        for images ,labels in test_loader:
            # 预测数据
            outputs = model(images)
            # 检查预测结果
            for i ,output in enumerate(outputs):
                # print(f"np.argmax(output) : {np.argmax(output)}")
                # print(f"torch.argmax(output):{torch.argmax(output)}")
                if np.argmax(output) == labels[i]:
                    correct += 1
                total += 1
    return correct / total
# 5.训练模型
for i  in range(EPOCH):
    print(f"训练纪元: {i+1}次")
    #开启训练
    model.train(True)
    #遍历训练集
```

```
    for i ,(images , labels) in enumerate(train_loader):
        #梯度清零（优化器的操作）
        opt.zero_grad()
        #前向传播
        outputs = model.forward(images)
        # 计算损失
        loss  = loss_fun(outputs , labels)
        # 反向传播
        loss.backward()
        # 优化参数
        opt.step()
    # 测试模型
    accurate =  eval(model,test_loader)
    print(f"{i+1}/{EPOCH}次测试结果的正确率是：{accurate}")
# 6.保存模型
torch.save(model.state_dict(),"FNN_Model.pt")
```

7.单张真实图片测试FNN模型

```
#  读取imgs中的一张图片，然后预测图片是数字多少
from PIL import Image,ImageOps
from torchvision.transforms import ToTensor ,Resize ,Compose
import torch
from FNN_Model import FNN_Model

# 1. 打开一张图片
image = Image.open("imgs/1_0.png")
#image = image.convert("L") # 转换为灰度图
#image = ImageOps.invert(image) # 转为黑底白字
to_tensor = Compose({
    ToTensor(),
    Resize((28,28))
})
#通道 高 宽
image_tensor = to_tensor(image)
print(image_tensor.shape)
# 2.加载模型， 预测
model = FNN_Model()
model.load_state_dict(torch.load("FNN_Model.pt"))
output = model(image_tensor)
# 3.得到预测结果
print(f"{torch.argmax(output)}")
```

8.多张真实图片批量测试FNN

```
# 批量读取准备好的真实图片，然后对这些图片进行转换，然后处理为一个数据集，最后批量预测，遍历预测结果
import torch
from PIL import Image , ImageOps
from torchvision.transforms import ToTensor , Resize , Compose
from FNN_Model import FNN_Model

#1.图片路径list
img_list =
["MNIST/trueImgs/0.png","MNIST/trueImgs/1.png","MNIST/trueImgs/2.png",
```

```
 "MNIST/trueImgs/3.png","MNIST/trueImgs/4.png","MNIST/trueImgs/5.png",

 "MNIST/trueImgs/6.png","MNIST/trueImgs/7.png","MNIST/trueImgs/8.png",
            "MNIST/trueImgs/9.png"]
# 2. 转换器
to_tengsor = Compose({ToTensor(),
                      Resize((28,28))})
# 3.图片转换成的张量集合(list类型)
img_tensor_list=[]
for img in img_list:
    # 转变为灰度图，  转换背景色，  转换为张量
    image = Image.open(img)
    image = image.convert("L")
    image = ImageOps.invert(image)
    image_tensor = to_tengsor(image)
    img_tensor_list.append(image_tensor)
print(img_tensor_list[0].shape)

#4.图片张量集合转换为一个张量对象
img_tensor_obj = torch.stack(img_tensor_list)
#5. 加载模型，预测
model = FNN_Model()
model.load_state_dict(torch.load("FNN_Model.pt"))
outputs = model(img_tensor_obj)
# 6. 输出预测结果
for i,output in enumerate(outputs):
    print(f"{i} ---- {torch.argmax(output)}")
```

9.卷积神经网络模型：

CNN_Model.py

```
# 卷积神经网络
import torch.nn as nn

class CNN_Model(nn.Module):
    # 构造函数
    def __init__(self):
        super(CNN_Model, self).__init__()
        # 通过神经网络序列来定义神经网络中的每一层
        self.sequential = nn.Sequential(
            # 第一次卷积，将灰度图按3*3的卷积核进行卷积，输出10个特征。
            # 28*28通过3*3的卷积核，大小变为(28-3+1),即26*26
            nn.Conv2d(1, 10, 3),  # nn.Conv2d(输入通道数，输出通道数，卷积核大小)二维卷积
            # 激活函数
            nn.ReLU(),
            # 最大值池化。将26*26的图片池化，获取每2*2窗口中的最大值，每次跳过2个长度，最终长和宽都小一半，
            # 得到13*13的特征图
            nn.MaxPool2d(2, 2),  # nn.MaxPool2d(池化窗口大小，步长)
            # 第二次卷积，将13*13的特征图，按5*5的卷积核卷积，输出20个特征
            # 13*13通过5*5的卷积核，大小变为(13-5+1),即9*9
```

```
            nn.Conv2d(10, 20, 5),
            # 激活函数
            nn.ReLU(),
            # 展平
            nn.Flatten(),
            # 经过全连接层，将9*9*20(上一步卷积后的特征数)， 输出100个新特征
            nn.Linear(9 * 9 * 20, 100),
            # 激活函数
            nn.ReLU(),
            # 全连接层，得到10个最终的输出，即0-9的概率
            nn.Linear(100, 10),
            nn.LogSoftmax(dim=1)  # 归一化
        )

    # 前向传播的函数
    def forward(self, x):
        return self.sequential(x)
```

训练CNN模型

```
import torch
import torchvision
from CNN_Model import CNN_Model
#1.定义超参数
EPOCH = 3
BATCH_SIZE=10
LEARNING_BATE=0.0001 # 0.001的时候，准确率很低， 0.0001的时候准确率提高了很多。
#2.加载数据集
to_tensor = torchvision.transforms.ToTensor()
train_set =
torchvision.datasets.MNIST("",transform=to_tensor,train=True,download=False)
test_set =
torchvision.datasets.MNIST("",transform=to_tensor,train=False,download=False)
train_loader =
torch.utils.data.DataLoader(train_set,batch_size=BATCH_SIZE,shuffle=True)
test_loader =
torch.utils.data.DataLoader(test_set,batch_size=BATCH_SIZE,shuffle=True)
#3.定义模型，优化器，损失函数
model = CNN_Model()
loss_fun = torch.nn.NLLLoss()
opt = torch.optim.Adam(model.parameters(),lr=LEARNING_BATE)
#4.评估函数
def eval(model,test_loader):
    #1.开启评估
    model.eval()
    #2.计算正确率
    if torch.no_grad():
        total = 0
        correct = 0
        for images , labels in test_loader:
            outputs = model(images)
            for i , output in enumerate(outputs):
                if torch.argmax(output) == labels[i]:
                    correct += 1
                total += 1
    return  correct / total
```

```
#5.训练模型
for i in range(EPOCH):
    print(f"第{i+1}次训练纪元：")
    #1.开启训练
    model.train(True)
    #2.循环训练
    for i,(images , labels) in enumerate(train_loader):
        #梯度清零
        '''
        遇到的错误： 代码写成了torch.no_grad() ， 这里应该是对优化器的梯度清零
        由于没有对优化器梯度清零，导致训练的模型，准去率较低，并且需要把学习率
        设置得很小，否则模型的准去率很低。
        '''
        opt.zero_grad()
        #前向传播
        outputs = model(images)
        #计算损失
        loss = loss_fun(outputs , labels)
        #反向传播
        loss.backward()
        #优化参数
        opt.step()
    #3.测试模型
    accurate = eval(model , test_loader)
    print(f"{i+1}次训练的正确率是：{accurate}")
#6.保存模型
# ***这里加个判断，读取之前的训练结果，保存准确率最高的模型
torch.save(model.state_dict(), "CNN_Model.pt")
```

单张图片测试CNN

```
#  读取imgs中的一张图片，然后预测图片是数字多少
#  ** 卷积神经网络不能处理单个样本数据 **
from PIL import Image,ImageOps
from torchvision.transforms import ToTensor ,Resize ,Compose
import torch
from CNN_Model import CNN_Model

# 1. 打开一张图片
image = Image.open("imgs/8_1.png")
#image = image.convert("L") # 转换为灰度图
#image = ImageOps.invert(image) # 转为黑底白字
to_tensor = Compose({
    ToTensor(),
    Resize((28,28))
})
# 批次 通道 高 宽 ， . 这里因为用到卷积神经网络，需要把单个样本数据转换为可以批量处理的形式
# unsqueeze(0) 是PyTorch中用于张量操作的函数， 作用是在指定位置插入一个维度为1的新维度，
# ‘0’代表在第0维度，即最左边维度插入。
image_tensor = to_tensor(image)
print(f"{image_tensor.shape}======") # [1, 28, 28]
image_tensor = image_tensor.unsqueeze(0)
print(image_tensor.shape,'======')# [1, 1, 28, 28]
# 2.加载模型， 预测
model = CNN_Model()
model.load_state_dict(torch.load("CNN_Model.pt"))
output = model(image_tensor)
```

```
# 3.得到预测结果
print(f"{torch.argmax(output)}")
```

使用CNN批量测试图片

```
# 批量读取准备好的真实图片，然后对这些图片进行转换，然后处理为一个数据集，最后批量预测，遍历预测结果
import torch
from PIL import Image , ImageOps
from torchvision.transforms import ToTensor , Resize , Compose
from CNN_Model import CNN_Model

#1.图片路径list
'''
这个问题找了很久的错误， 因为使用的{}是set类型， 导致遍历的时候获取到
数据值是无序的，导致一致识别率很低。 后面参考别人使用的[],是list类型
，遍历是有序的， 这样才可以用序号对应着图片上的数据值，来对比识别是否准确。
img_list = {"MNIST/trueImgs/0.png",
            "MNIST/trueImgs/1.png",
            "MNIST/trueImgs/2.png",
            "MNIST/trueImgs/3.png",
            "MNIST/trueImgs/4.png",
            "MNIST/trueImgs/5.png",
            "MNIST/trueImgs/6.png",
            "MNIST/trueImgs/7.png",
            "MNIST/trueImgs/8.png",
            "MNIST/trueImgs/9.png"}
'''
img_path_list = [
    "MNIST/trueImgs/0.png",
    "MNIST/trueImgs/1.png",
    "MNIST/trueImgs/2.png",
    "MNIST/trueImgs/3.png",
    "MNIST/trueImgs/4.png",
    "MNIST/trueImgs/5.png",
    "MNIST/trueImgs/6.png",
    "MNIST/trueImgs/7.png",
    "MNIST/trueImgs/8.png",
    "MNIST/trueImgs/9.png",
]

# 2. 转换器
to_tengsor = Compose([Resize((28,28)),ToTensor()])
# 3.图片转换成的张量集合(list类型)
img_tensor_list=[]
for img in img_path_list:
    print(img ,"==================")
    # 转变为灰度图， 转换背景色， 转换为张量
    image = Image.open(img)
    image = image.convert("L")
    image = ImageOps.invert(image)
    image_tensor = to_tengsor(image)
    img_tensor_list.append(image_tensor)

#4.图片张量集合转换为一个张量对象
img_tensor_obj = torch.stack(img_tensor_list)
```

```
print(img_tensor_obj.shape)

#5. 加载模型，预测
model = CNN_Model()
model.load_state_dict(torch.load("CNN_Model.pt"))
outputs = model(img_tensor_obj)

prediction = torch.argmax(outputs,dim=1)
print(prediction)

# 6. 输出预测结果
for i ,output in enumerate(outputs):
    print(i , torch.argmax(output))
```

9,使用Tkinter实现手写数字的测试

```
import tkinter as tk
from tkinter import Canvas , Label ,Button
from torchvision.transforms import ToTensor
from PIL import Image , ImageDraw , ImageOps
import torch
from CNN_Model import CNN_Model

class HandwritingApp:
    #构造函数
    def __init__(self,root):
        self.root = root
        self.root.title = "MNIST数字识别"
        # 创建Canvas
        self.canvas = Canvas(root,width=256,height=256,bg='white')
        # 使用pack布局管理器放置canvas
        # padx 水平方向的边距（组件左右两侧的空白空间）， pady垂直方向的边距
        self.canvas.pack(padx=10,pady=10)
        #绑定鼠标事件
        self.canvas.bind("<Button-1>",self.on_draw_start)
        self.canvas.bind("<B1-Motion>",self.on_draw_move)

        #创建一个空的PIL图像用于保存绘制结果
        self.drawing = Image.new("RGB",(256,256),
                                 'white')
        self.draw = ImageDraw.Draw(self.drawing)
        self.img_tensor = None

        # 识别按钮
        self.save_button = Button(root,text="识别",
                                  command=self.on_save_button_clicked)
        self.save_button.pack(pady=20)

        # 清楚发按钮
        self.clear_button = Button(root , text="清除"
                                   ,command=self.on_clear_button_clicked)
        self.clear_button.pack(pady=10)
        # 显示预测结果的标签
        self.prediction_label = Label(root,text="" , width=20)
```

```
        self.prediction_label.pack(pady=20)

        # 加载模型参数
        self.model = CNN_Model()
        state_dict = torch.load("CNN_Model.pt")
        self.model.load_state_dict(state_dict)


    def on_draw_start(self,event):
        self.lastx , self.lasty = event.x , event.y
    def on_draw_move(self,event):
        x,y = event.x ,event.y
        # 自己单独创建的一张特征图片上的内容修改
        self.draw.line((self.lastx , self.lasty, x,y),
                       fill="black",width=10)
        # 这个是在画布上绘制的图片，让用户可以看见的内容
        self.canvas.create_line(self.lastx , self.lasty , x, y,
                                fill="black" , width=10 , capstyle=tk.ROUND
                                ,smooth=tk.TRUE , splinesteps=36)
        # 点的位置在不断变化
        self.lastx , self.lasty = x, y


    def save_as_tensor(self):
        img_gray = self.drawing.convert("L")
        img_gray = ImageOps.invert(img_gray)
        img_gray = img_gray.resize((28,28))
        to_tensor = ToTensor()
        img_tensor = to_tensor(img_gray).float()
        self.img_tensor = img_tensor.unsqueeze(0) # 升维
    def on_save_button_clicked(self):
        '''
        function:这个函数识别按钮对应的函数
        '''
        self.save_as_tensor()
        outputs = self.model.forward(self.img_tensor)
        prediction = torch.argmax(outputs)
        self.prediction_label.config(text=f"识别结果: {int(prediction)}")


    def on_clear_button_clicked(self):
        self.canvas.delete("all")
        self.drawing = Image.new("RGB",(256,256),'white')
        self.draw = ImageDraw.Draw(self.drawing)
        self.img_tensor = None
        self.prediction_label.config(text="")
root = tk.Tk()
app = HandwritingApp(root)
root.mainloop()
```