HW-5-陈宸

Q1.实现 2-layer CNN 用于MNIST分类

1.1 可视化第一张图片：

在load_mnist_data()函数中load数据集后添加如下代码：

```
# display first image
first_image = train_images[0, :].reshape(28, 28)
img_pil = Image.fromarray(first_image)
img_pil.save("vis.png")
```

文件目录下可视化的vis.png如下，像素维数28*28：

1.2 手动构建两层CNN网络：

在本题中，计划构建一个两层卷积层，两层池化层的卷积神经网络，首先对于每一层输入与输出的tensor维数进行分析以确定全连接层的维数，进一步保证输出维数为[batch_size,10]。

第一层卷积层（conv1）:输入的灰度图像（单通道）tensor维数为：[Batch_size, 1, 28, 28]，经过32个3*3卷积核计算后，边界未进行填充，故输出维数减少2，输出的tensor维数为：[Batch_size, 1, 26, 26]。

最大池化层（pool1）:将前一层的输出通过2x2的最大池化层进行池化，输出的tensor维数减半为[Batch_size, 32, 13, 13]。

第二层卷积层（conv2）:使用64个3x3的卷积核对前一层的输出进行卷积运算，同理由于边界未进行填充，维数减少2，但由于卷积核翻倍，通道数也翻倍，输出的tensor维数为：[Batch_size, 64, 11, 11]

ReLU激活函数

池化层（pool2）:将前一层的输出通过2x2的最大池化层进行池化，输出的tensor维数减半为[Batch_size, 64, 5, 5]。

由于ReLU激活函数对维数不产生影响，故不做额外分析。

经过展平操作后，由于设定的第二全连接层维数为128*10以得到10个不同的分类结果，第一全连接层维数应与池化层2的输出与第二全连接层的输入对应，维数为64*5*5乘以128。故修改补全代码如下。

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.pool1 = nn.MaxPool2d(2)

        self.conv2 = nn.Conv2d(32, 64, 3, 1)
```

```
        self.pool2 = nn.MaxPool2d(2)
        self.dropout1 = nn.Dropout(0.3)
        self.dropout2 = nn.Dropout(0.3)
        self.fc1 = nn.Linear(5*5*64, 128)
        self.fc2 = nn.Linear(128, 10)
        # self.fc_dummy = nn.Linear(5408, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = torch.flatten(x, 1)
        x = self.dropout1(x)
        x = F.relu(self.fc1(x))
        x = self.dropout2(x)
        out = self.fc2(x)
        # out = self.fc_dummy(x)
        assert out.ndim == 2 and out.size(0) == x.size(0) and out.size(1) == 10

        return out
```

为检验其输出结果维数正确，编写main()函数如下：

```
def main():
    model = Net()
    input_tensor = torch.randn(1, 1, 28, 28)  # Batch_size=1,channel=1, 28x28 image size
    output = model(input_tensor)
    print("Output shape:", output.shape)
    assert output.shape == (1, 10), "Output shape error"
```

输出结果：

```
Output shape: torch.Size([1, 10])
```

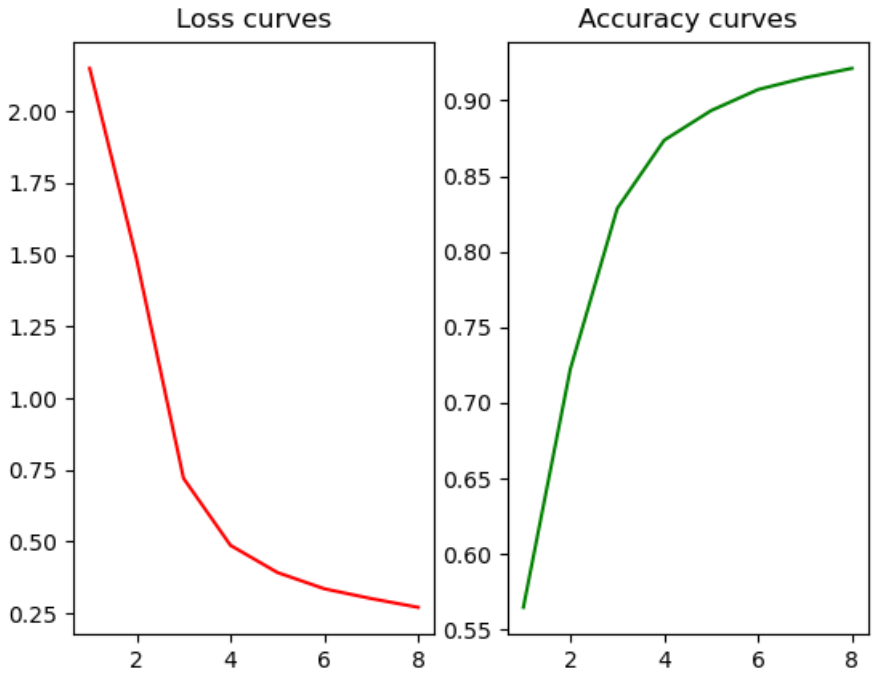
1.3 执行main.py结果并解释TODO代码

为test函数增加可视化功能。

```
    fig, ax = plt.subplots(1, 2)
    x = np.arange(1, args.epochs + 1)
    ax[0].plot(x, loss_list, color='r', label='Loss')
    ax[0].set_title("Loss curves")
    ax[1].plot(x, acc_list, color='g', label='Accuracy')
    ax[1].set_title("Accuracy curves")
    plt.show()
```

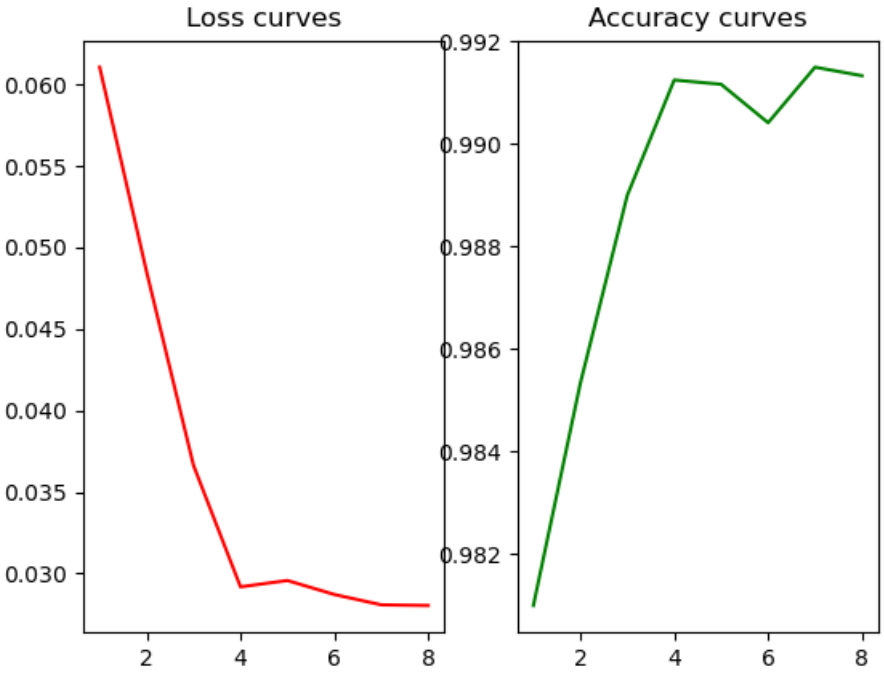
optimzer=SGD

```
========= Final Testing =============

 Test: Average loss: 0.2616, Accuracy: 9276/10000    (93%)
```



optimzer=ADAM

```
========= Final Testing =============

 Test: Average loss: 0.0241, Accuracy: 9921/10000    (99%)
```



经过比较，ADAM优化器的收敛速度更快，整体分类准确率更高。

TODO1 in train():explain the function of train()

首先将模型切换至训练模式，启用dropout，并将传入的标签与数据移动到指定设备（如GPU），训练过程中首先对优化器的梯度进行清零，接着前向传播得到预测输出，并计算交叉熵损失做后向传播计算梯度后更新模型参数。

TODO2 in test():explain the function of eval()

在测试过程中，eval()函数的作用是将模型切换到测试模式，禁用dropout等正则化策略。同时，测试的过程中还禁用了梯度的计算，猜想是为了减少内存的占用。

TODO3 in test():explain the function of argmax

查看文档，此函数可返回输入张量最大值的索引，在测试过程中，此处是为了获取最大概率对应的类别。

TODO4 in getitem():explain each line of reading in a gray image

首先将导入的image的ndarray数组转为PIL图像，接着利用transform函数进行预处理的转换，最后调整图像的维度，将其转换为（1，28，28）

TODO5 in load_mnist_data(): explain transforms of images

在Compose()函数内实现数据类型的转换功能与标准化处理，ToTensor()函数将像素值区间为[0, 255]的ndarray转为取值区间为[0, 1]的浮点数tensor格式，以便后续的训练处理。

Normalize函数采用了MNIST数据集的全局均值(0.1307)与标准差(0.3081)进行标准化。使每个像素值减去均值并除以标准差。

Q2.运行Stable Diffusion

根据huggingface官网示例代码修改了pipe后运行代码如下

```
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

init_image =
Image.open('Everest_North_Face_toward_Base_Camp_Tibet_Luca_Galuzzi_2006.jpg').convert("RGB")
init_image = init_image.resize((768, 512))

model_id = "sd-legacy/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id, torch_dtype=torch.float16)
pipe = pipe.to("cuda")

prompt = "A fantasy landscape, trending on artstation"
image = pipe(prompt=prompt, image=init_image, strength=0.75, guidance_scale=7.5).images[0]

image.save("fantasy_landscape.png")
```

采用colab中T4GPU进行训练

输出的fantasy_landscape.png

查看diffuser包的Unet_2d_condition.py中的UNet2DConditionModel class：

- downblock class中，存在三个CrossAttnDownBlock2D
- upblock class中，存在三个CrossAttnUpBlock2D

Bonus.

根据unet_2d_condition.py， UNetMidBlock2DCrossAttn充当了UNet模型的中间块，但其作用需要进入unet_2d_blocks.py查看分析才能够得知

根据unet_2d_blocks.py中UNetMidBlock2DCrossAttn的类定义，发现其构造函数中定义了一个 CrossAttention2D层、两个卷积层conv1和conv2，两个归一化层norm1与norm2，还有一个激活函数act_fn。

结合Microsoft Copilot （prompt="Please review the class UNetMidBlock2DCrossAttn of the diffusers1.5 model and analyze its role in UNet"）与https://blog.csdn.net/xd_wjc/article/details/134441396解释，**概括出以下作用**：

UNetMidBlock2DCrossAttn 作为 UNet 模型的中间层，通过注意力机制来增强特征的表示能力。具体来说，它通过 CrossAttention2D层对不同层次的特征进行交叉注意力操作，然后通过卷积层和归一化层进一步处理特征，最终输出经过激活函数处理的特征。