《人工智能导论》大作业

任务名称： Mnist条件生成器

完成组号： 4

小组人员： 刘灿，刘发中，汪何希，王伊婷，王梓睿

完成时间： 2023年6月17日

1 任务目标

1. 基于Mnist数据集，构建一个条件生成模型，当输入的条件为0~9的数字时，输出对应条件的生成图像。
2. 输出图像能够随机产生。
3. 在CPU上的运行时间合理。

2 具体内容

2.1 实施方案

本项目选用**ACGAN**模型作为条件生成模型。

ACGAN（Auxiliary Classifier Generative Adversarial Network）是**生成对抗网络（GAN）**的一种**扩展模型**。在传统的GAN中，生成器网络通过生成逼真的数据样本来欺骗判别器网络，而判别器网络则试图将生成的样本和真实样本区分开。ACGAN在此基础上引入了一个**辅助分类器**（auxiliary classifier），该分类器用于对生成的样本进行类别分类。通过辅助分类器引入监督信息，不仅可以生成逼真的数据样本，还可以对生成的样本进行分类，生成特定类别的样本。

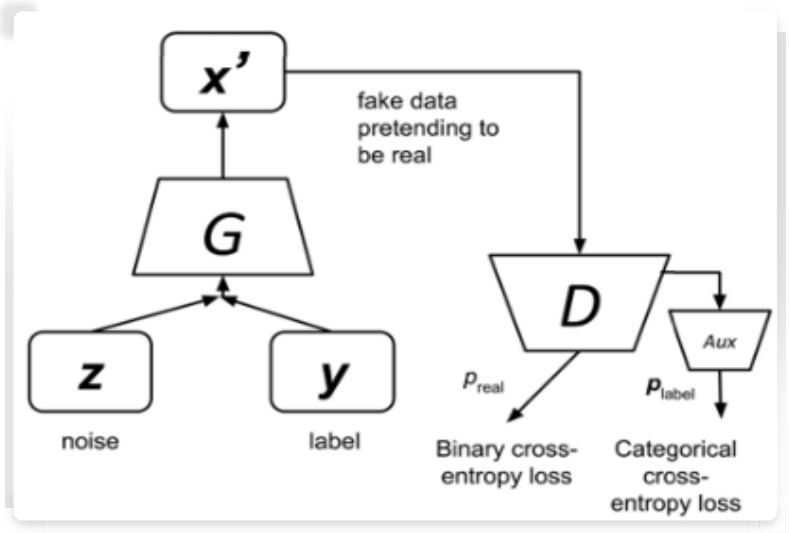

图1 ACGAN模型示意图

本项目主要包括四个部分：**数据导入(dataloader.py)，模型加载与初始化(aigc_mn.py)，模型训练与保存(ACGAN.py)，图片生成(utils.py)。**

- dataloader.py使用PyTorch创建MNIST数据集的数据加载器
- aigc_mn.py为接口文件，用函数`initialize_weights(net)`初始化网络的权重，实现了`generator`类作为ACGAN的生成器网络架构，实现`ACGAN`类作为辅助分类器生成对抗网络。
- ACGAN.py包含了生成器（Generator）和判别器（Discriminator）的定义，以及ACGAN的训练和生成方法，实现了ACGAN模型的网络结构。
- utils.py包括一些工具函数，主要用于加载和提取数据、生成并保存图像。

经过多轮训练后，最终得到的输出结果如图2所示。


图2 训练产出

此外，我们还记录了每轮训练中的loss，并绘制了loss曲线，以评估模型的性能。曲线如图3所示。

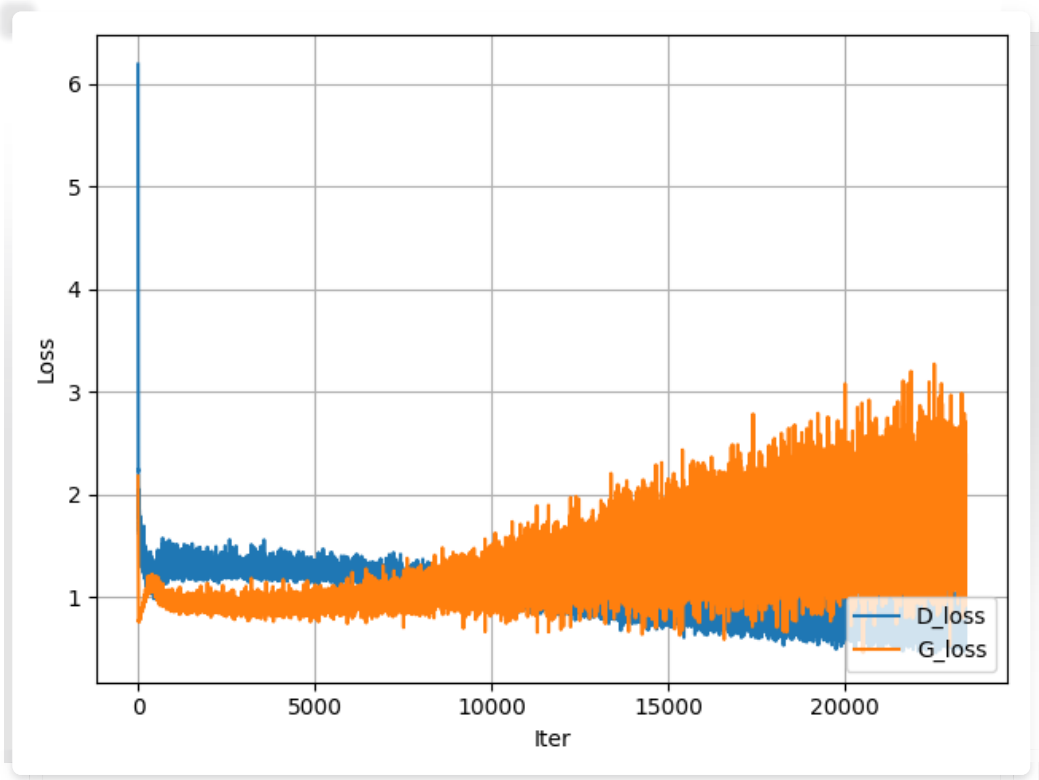

图3 loss变化曲线

可以发现在前5000次训练中，loss值经过一开始的减少，后续不再有较大的波动，说明模型参数已相对稳定。5000次后G的loss值波动逐渐增大，此时可能出现了过拟合等问题，可以减少训练轮数。

2.2 核心代码分析

2.2.1 ACGAN.py

该文件实现了基于ACGAN的生成对抗网络模型，主要定义了一个**生成器（generator）**和一个**判别器（discriminator）**，以及一个**ACGAN类**用于训练和生成图像。

生成器接收随机噪声向量和类别标签为输入，生成与给定标签相关的合成图像。判别器用于判别真实图像和合成图像的真实性，并预测图像的类别。训练过程中，通过最小化判别器和生成器的损失函数来优化网络参数，以实现更好的生成图像质量和类别判别能力。训练完成后，可以使用生成器来生成具有指定类别的图像样本。

1. class generator

`__init__()`**函数**用于在创建类的实例时进行初始化操作。

```
    def __init__(self, input_dim=100, output_dim=1, input_size=32, class_num=10):
        super(generator, self).__init__()
        # 设置输入维度、输出维度、输入尺寸和类别数量等参数
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.input_size = input_size
        self.class_num = class_num

        self.fc = nn.Sequential(
```

```
            nn.Linear(self.input_dim + self.class_num, 1024),
            nn.BatchNorm1d(1024),
            nn.ReLU(),
            nn.Linear(1024, 128 * (self.input_size // 4) * (self.input_size // 4)),
            nn.BatchNorm1d(128 * (self.input_size // 4) * (self.input_size // 4)),
            nn.ReLU(),
        )
        self.deconv = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 4, 2, 1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.ConvTranspose2d(64, self.output_dim, 4, 2, 1),
            nn.Tanh(),
        )
        utils.initialize_weights(self)
```

从上述代码可以看出：

- 生成器由全连接层和转置卷积层（反卷积层）组成，经过一系列线性变换和激活函数后生成合成图像。
- 全连接层，将输入的随机噪声向量与类别标签连接起来；反卷积层，将全连接层的输出转换为图像.
- 生成器的参数通过`utils.initialize_weights()`进行初始化。

`forward()`函数定义了模型的前向传播过程

```
    def forward(self, input, label):
            x = torch.cat([input, label], 1)
            x = self.fc(x)
            x = x.view(-1, 128, (self.input_size // 4), (self.input_size // 4))
            x = self.deconv(x)

            return x
```

`forward()`函数首先将输入和类别标签进行拼接，以便将它们作为条件信息传递给生成器网络。然后，将拼接后的输入传递给全连接层`self.fc`进行计算。接下来，通过`view`方法将计算得到的结果进行形状变换，以适应后续的反卷积操作。最后，将形状变换后的结果传递给反卷积层`self.deconv`，生成最终的输出图像。

2. class discriminator

与生成器类似，定义了初始化函数与`forward()`函数：

```
    def __init__(self, input_dim=1, output_dim=1, input_size=32, class_num=10):
        super(discriminator, self).__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.input_size = input_size
        self.class_num = class_num

        self.conv = nn.Sequential(
            nn.Conv2d(self.input_dim, 64, 4, 2, 1),
```

```
            nn.LeakyReLU(0.2),
            nn.Conv2d(64, 128, 4, 2, 1),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2),
        )
        self.fc1 = nn.Sequential(
            nn.Linear(128 * (self.input_size // 4) * (self.input_size // 4), 1024),
            nn.BatchNorm1d(1024),
            nn.LeakyReLU(0.2),
        )
        self.dc = nn.Sequential(
            nn.Linear(1024, self.output_dim),
            nn.Sigmoid(),
        )
        self.cl = nn.Sequential(
            nn.Linear(1024, self.class_num),
        )
        utils.initialize_weights(self)
```

由上述代码可知：

- 输入是真实图像或生成器输出的图像，经过卷积和线性变换后分别得到判别结果和类别预测结果。
- 判别器的架构与生成器相对应，由卷积层（`conv`）、全连接层（`fc1`）和分类层（`dc`和`cl`）组成。
- 卷积层用于提取图像特征，全连接层用于进一步处理特征，分类层用于输出判别结果和类别预测。

```
def forward(self, input):
        x = self.conv(input)
        x = x.view(-1, 128 * (self.input_size // 4) * (self.input_size // 4))
        x = self.fc1(x)
        d = self.dc(x)
        c = self.cl(x)

        return d, c
```

`forward`函数通过将输入数据（图像）传递到不同的网络层，以便对数据进行特征提取和判别预测。

3. class ACGAN

该类实现了ACGAN的训练、生成和保存功能。下面介绍几个重要的函数。

- `__init__`函数：
  - 初始化。
- `train`函数：
  - 训练 ACGAN 模型。
  - 迭代训练过程中，依次更新判别器和生成器的参数。
  - 计算判别器和生成器的损失，并记录损失值。
  - 每个 epoch 结束后，可视化生成器生成的样本并保存训练结果。
  - 最终保存训练好的生成器和判别器模型以及训练历史记录。
- `visualize_results`函数：

    - 根据是否使用固定噪声和条件信息来生成样本，并保存在指定路径。
- `generate` 函数：
    - 根据给定的类别编号生成样本。
- `save` 函数和 `load` 函数：
    - 保存训练好模型以及训练记录；加载已保存的模型。

2.2.2 aigc_mn.py

该文件为接口类文件。

1. 定义了函数 `initialize_weights(net)`，用于初始化网络的权重。此函数遍历网络的所有模块，对卷积层（`nn.Conv2d`）、反卷积层（`nn.ConvTranspose2d`）和全连接层（`nn.Linear`）进行权重初始化。
2. 实现 `generator` 类，作为ACGAN的生成器网络架构。生成器类继承自 `nn.Module`，包含了全连接层和反卷积层。
3. 实现 `ACGAN` 的类，作为辅助分类器生成对抗网络。该类包含了模型的参数、网络的初始化、图像生成和模型加载等功能。

   值得一提的是，为了提高模型和“带OOD检测的Mnist分类器”组的对抗性，我们在`generate` 函数中加入了额外噪声，让生成图像在肉眼难以察觉的前提下，更容易被对抗组判为OOD。

   ```
       def generate(self, num):
           # Directory to save results
           if not os.path.exists(self.result_dir):
               os.makedirs(self.result_dir)

           sample_y_ = torch.zeros(1, self.class_num)
           sample_y_[0][num] = 1.
           sample_z_ = torch.rand((1, self.z_dim))

           # 创建一个形状为 (1, 28, 28) 的均匀噪声张量，乘以 0.05 作为系数，可调
           noise = torch.rand(1, 28, 28) * 0.05

           if torch.cuda.is_available():
               sample_z_, sample_y_, noise = sample_z_.cuda(), sample_y_.cuda(),
   noise.cuda()

           samples = (self.G(sample_z_, sample_y_) + 1) / 2 + noise    # 加入噪声
           samples = samples / samples.max()       # 归一化

           return samples
   ```

4. 实现接口类`AiGcMn`，用于创建ACGAN模型的实例。
5. 最后定义了 `main` 函数，在该函数中创建了ACGAN模型的实例，生成一系列随机数字对应的图像张量，再通过 `save_images` 函数将张量转化为图像并保存到指定路径。

2.2.3 utils.py

utils.py文件中定义了一些工具函数，如图像生成相关函数等。

3 工作总结

3.1 收获、心得

- 对条件生成模型有了深入的理解，它在生成样本时考虑了额外的条件信息，能生成与条件相匹配的逼真图像。
- 大致掌握了ACGAN的网络结构，并且学会了如何将条件信息引入生成器和判别器中。这个项目让我们意识到ACGAN的强大能力，它可以生成高质量的图像，并且可以控制生成的结果，使其符合特定的条件。
- 学会了如何构建生成器网络和判别器网络。
- 这次实践加深了我们对课堂知识的理解，扩展了知识面，有助于我们将所学概念运用于实际场景，也增强了相关的实践技能与团队协作能力。

3.2 遇到问题及解决思路

- GAN的训练过程不稳定，生成器和判别器出现互相优化的困境。
    - 尝试调整学习率、更新策略、损失函数等来解决。
- GAN模型的训练需要耗费较长的时间。
    - 对于GAN模型的生成器与判别器，在保证效果的前提下尽量缩减网络大小，尝试优化模型结构、减小输入维度，并且将模型移植到GPU上，充分利用算力。

4 课程建议

本课程提供了对人工智能的综合性介绍，涵盖了不同应用领域的基本概念和技术，让我们对人工智能有了较为全面的了解。与之相对的是实践经验的不足，希望能提供更多实际的案例、项目或实验，鼓励创新和探索，帮助我们更好地理解、掌握人工智能的实际应用。