LeNet-5 在 MNIST 上的训练和测试
实验报告

姓名：于海煊　学号：123106222860　学院：计算机科学与工程学院

一、数据集及预处理

MNIST 数据集是一个包含手写数字图片的经典数据集，包括 60,000 张训练图片和 10,000 张测试图片。每张图片为 28x28 像素的灰度图像，标签为 0 到 9 的数字。由于 MNIST 数据集图片尺寸是 28x28 单通道的，而 LeNet-5 网络输入 Input 图片尺寸是 32x32，因此使用 transforms.Resize 将输入图片尺寸调整为 32x32。

```
pipline_train = transforms.Compose([
    #随机旋转图片
    transforms.RandomHorizontalFlip(),
    #将图片尺寸resize到32x32
    transforms.Resize((32,32)),
    #将图片转化为Tensor格式
    transforms.ToTensor(),
    #正则化(当模型出现过拟合的情况时，用来降低模型的复杂度)
    transforms.Normalize( mean: (0.1307,), std: (0.3081,))
])
pipline_test = transforms.Compose([
    #将图片尺寸resize到32x32
    transforms.Resize((32,32)),
    transforms.ToTensor(),
    transforms.Normalize( mean: (0.1307,), std: (0.3081,))
])
#下载数据集
train_set = datasets.MNIST(root="./data", train=True, download=True, transform=pipline_train)
test_set = datasets.MNIST(root="./data", train=False, download=True, transform=pipline_test)
#加载数据集
trainloader = torch.utils.data.DataLoader(train_set, batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(test_set, batch_size=32, shuffle=False)
```

二、搭建 LeNet-5 神经网络及前向传播

```
2 用法
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d( in_channels: 1,  out_channels: 6,  kernel_size: 5)
        self.relu = nn.ReLU()
        self.maxpool1 = nn.MaxPool2d( kernel_size: 2,  stride: 2)
        self.conv2 = nn.Conv2d( in_channels: 6,  out_channels: 16,  kernel_size: 5)
        self.maxpool2 = nn.MaxPool2d( kernel_size: 2,  stride: 2)
        self.fc1 = nn.Linear(16 * 5 * 5,  out_features: 120)
        self.fc2 = nn.Linear( in_features: 120,  out_features: 84)
        self.fc3 = nn.Linear( in_features: 84,  out_features: 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = self.maxpool1(x)
        x = self.conv2(x)
        x = self.maxpool2(x)
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

三、部署 GPU 及定义优化器

```
#创建模型，部署gpu
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LeNet().to(device)
#定义优化器
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

四、定义训练过程

```python
def train_runner(model, device, trainloader, optimizer, epoch):
    # 训练模型, 启用 BatchNormalization 和 Dropout, 将BatchNormalization和Dropout置为True
    model.train()
    total = 0
    correct = 0.0

    # enumerate迭代已加载的数据集,同时获取数据和数据下标
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        # 把模型部署到device上
        inputs, labels = inputs.to(device), labels.to(device)
        # 初始化梯度
        optimizer.zero_grad()
        # 保存训练结果
        outputs = model(inputs)
        # 计算损失和
        # 多分类情况通常使用cross_entropy(交叉熵损失函数), 而对于二分类问题, 通常使用sigmod
        loss = F.cross_entropy(outputs, labels)
        # 获取最大概率的预测结果
        # dim=1表示返回每一行的最大值对应的列下标
        predict = outputs.argmax(dim=1)
        total += labels.size(0)
        correct += (predict == labels).sum().item()
```

```python
        # 反向传播
        loss.backward()
        # 更新参数
        optimizer.step()
        if i % 1000 == 0:
            # loss.item()表示当前loss的数值
            print(
                "Train Epoch{} \t Loss: {:.6f}, accuracy: {:.6f}%".format( *args: epoch, loss.item(), 100 * (correct / total)))
            Loss.append(loss.item())
            Accuracy.append(correct / total)
    return loss.item(), correct / total
```

五、模型验证过程

```python
1 个用法
def test_runner(model, device, testloader):
    #模型验证, 必须要写, 否则只要有输入数据, 即使不训练, 它也会改变权值
    #因为调用eval()将不启用 BatchNormalization 和 Dropout, BatchNormalization和Dropout置为False
    model.eval()
    #统计模型正确率, 设置初始值
    correct = 0.0
    test_loss = 0.0
    total = 0
    #torch.no_grad将不会计算梯度, 也不会进行反向传播
    with torch.no_grad():
        for data, label in testloader:
            data, label = data.to(device), label.to(device)
            output = model(data)
            test_loss += F.cross_entropy(output, label).item()
            predict = output.argmax(dim=1)
            #计算正确数量
            total += label.size(0)
            correct += (predict == label).sum().item()
        #计算损失值
        print("test_avarage_loss: {:.6f}, accuracy: {:.6f}%".format( *args: test_loss/total, 100*(correct/total)))
```

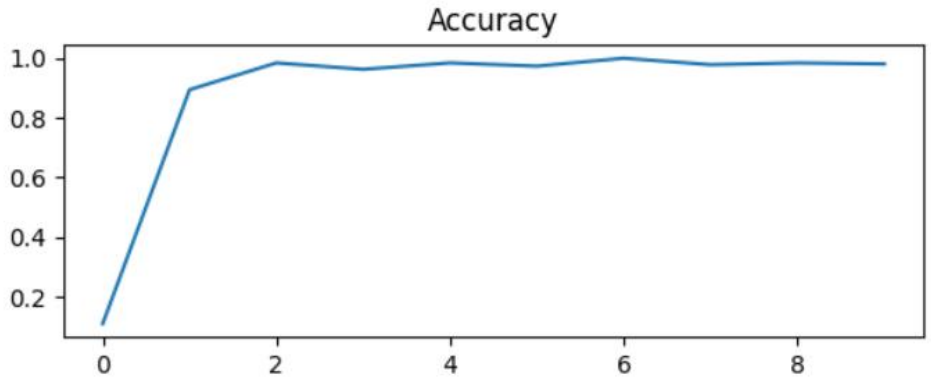
六、运行及保存模型

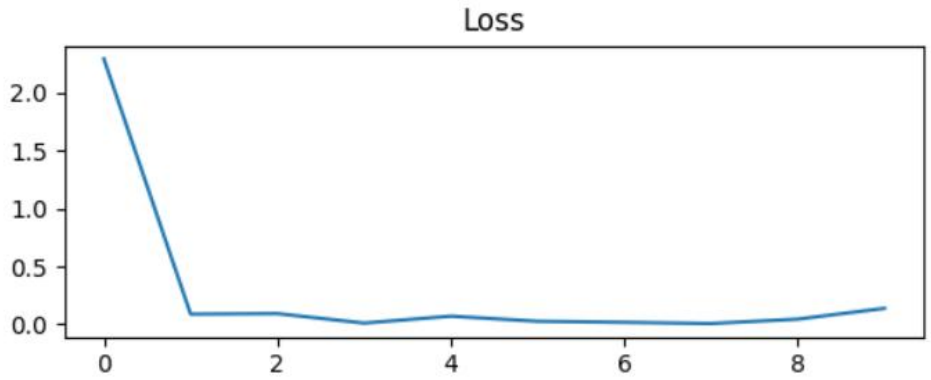
```python
# 调用
epoch = 5
Loss = []
Accuracy = []
for epoch in range(1, epoch + 1):
    print("start_time", time.strftime( format: '%Y-%m-%d %H:%M:%S', time.localtime(time.time())))
    loss, acc = train_runner(model, device, trainloader, optimizer, epoch)
    Loss.append(loss)
    Accuracy.append(acc)
    test_runner(model, device, testloader)
    print("end_time: ", time.strftime( format: '%Y-%m-%d %H:%M:%S', time.localtime(time.time())), '\n')

print('Finished Training')
plt.subplot( *args: 2, 1, 1)
plt.plot(Loss)
plt.title('Loss')
plt.show()
plt.subplot( *args: 2, 1, 2)
plt.plot(Accuracy)
plt.title('Accuracy')
plt.show()

print(model)
torch.save(model,  f: './models/model-mnist.pth') #保存模型
```

七、运行结果

以下为运行结果截图，以及 epoch=5 的 Loss 和 Accuracy 的折线图

```
运行:  main

start_time 2024-05-06 08:30:09
Train Epoch3     Loss: 0.070869, accuracy: 98.437500%
test_avarage_loss: 0.002376, accuracy: 97.330000%
end_time:  2024-05-06 08:30:29

start_time 2024-05-06 08:30:29
Train Epoch4     Loss: 0.017920, accuracy: 100.000000%
test_avarage_loss: 0.002388, accuracy: 97.330000%
end_time:  2024-05-06 08:30:50

start_time 2024-05-06 08:30:50
Train Epoch5     Loss: 0.045298, accuracy: 98.437500%
test_avarage_loss: 0.002265, accuracy: 97.620000%
end_time:  2024-05-06 08:31:11

Finished Training
LeNet(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (relu): ReLU()
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```





八、定义测试过程

利用训练好的 LeNet-5 模型进行手写数字图片的测试

train.py dataset.py Loss.png test.py test_mnist.jpg

```python
if __name__ == '__main__':
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = torch.load('models/model-mnist.pth')  # 加载模型
    model = model.to(device)
    model.eval()  # 把模型转为test模式

    # 读取要预测的图片
    img = cv2.imread("./images/test_mnist.jpg")
    img = cv2.resize(img, dsize=(32, 32), interpolation=cv2.INTER_NEAREST)
    plt.imshow(img, cmap="gray")  # 显示图片
    plt.axis('off')  # 不显示坐标轴
    plt.show()

    # 导入图片，图片扩展后为[1, 1, 32, 32]
    trans = transforms.Compose(
        [
            transforms.ToTensor(),
            transforms.Normalize( mean: (0.1307,), std: (0.3081,))
        ])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # 图片转为灰度图，因为mnist数据集都是灰度图
    img = trans(img)
    img = img.to(device)
    img = img.unsqueeze(0)  # 图片扩展多一维,因为输入到保存的模型中是4维的[batch_size,通道,长，宽]，而普通图片只有三维，[通道,长，宽]

    # 预测
    output = model(img)
    prob = F.softmax(output, dim=1)  # prob是10个分类的概率
    print("概率：", prob)
    value, predicted = torch.max(output.data, 1)
```

九、测试结果

测试结果较为准确


```
概率：tensor([[2.0888e-07, 1.1599e-07, 6.1852e-05, 1.5797e-04, 1.4975e-09, 9.9977e-01,1.9271e-06, 3.1589e-06, 1.2186e-07, 4.3405e-07]],grad_fn= < SoftmaxBackward >)
预测类别：5

进程已结束，退出代码为 0
```

十、总结

LeNet-5 在 MNIST 数据集上表现优秀，通常可以达到超过 98%的准确率。通过对模型架构的调整、超参数的调优等方法，展现了其在图像分类任务中的有效性和可靠性，还可以进一步提升模型性能。