实验一：深度学习框架熟悉

120L021716 蔡泽栋

一、实验目的:

使用 PyTorch 实现 MLP 并在 MNIST 数据集上验证。

二、实验环境:

Python 3.9.13; torch =2.0.0+cu117; torchvision=0.15.0+cu117; GPU

```
python                    3.9.13              h6244533_2
python-dateutil           2.8.2                  pypi_0
pytorch-mutex             1.0                      cpu

pytz                      2023.3                 pypi_0
regex                     2023.3.23              pypi_0
requests                  2.28.1            py39haa95532_1
scikit-learn              1.2.2                  pypi_0
scipy                     1.10.1                 pypi_0
setuptools                65.6.3            py39haa95532_0
six                       1.16.0              pyhd3eb1b0_1
sqlite                    3.41.1              h2bbff1b_0
sympy                     1.11.1            py39haa95532_0
threadpoolctl             3.1.0                  pypi_0
tk                        8.6.12              h2bbff1b_0
torch                     2.0.0+cu117            pypi_0
torchvision               0.15.0+cu117           pypi_0
tqdm                      4.65.0                 pypi_0
```

三、实验内容:

1. 初始化模型超参数并构建一个 MLP:

```python
# Hyperparameters
input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
```

MLP 含有两个线性层和一个隐藏层，隐藏层使用 ReLU 作为激活函数。

```python
# MLP Model
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

2. 数据读取：

```python
# MNIST dataset
train_dataset = dsets.MNIST(root='./data',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)

test_dataset = dsets.MNIST(root='./data',
                           train=False,
                           transform=transforms.ToTensor())

# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

读取 MNIST 数据集，分为训练集和测试集。如果文件中没有设定为自行下载转为张量形式返回。然后创建一个 DataLoader 对象，一次取 100 张图片，将其中数据加载到模型中。

3. 定义优化器并训练

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.to(device)
        labels = labels.to(device)

        # Reshape images to (batch_size, input_size)
        images = images.reshape(-1, 28 * 28)

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

使用交叉熵作为损失函数，Adam 作为优化器，训练模型：

Images：模型的输入数据。

optimizer.zero_grad()：清除优化器中之前保存的梯度。在反向传播计算梯度之前，需要先将梯度清零，否则之前计算的梯度会累加到当前的梯度中，导致计算出错。

outputs = model(images)：使用模型进行前向传播，生成输出结果。

loss = criterion(outputs, labels)：计算模型预测结果和真实结果之间的损失。

loss.backward()：计算损失函数关于模型参数的梯度。梯度的计算是通过自动微分机制实现的。

optimizer.step()：更新模型参数。根据优化算法的规则，以梯度下降的方式更新模型的参数，以尽可能减小损失函数的值。

每次执行一次可以让模型的参数更新一次，从而逐渐提升模型的准确性。反复执行这个训练步骤，直到模型收敛或达到指定的训练轮数。

四、实验结果分析:

```python
# Test the model
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        # Move tensors to the configured device
        images = images.to(device)
        labels = labels.to(device)

        # Reshape images to (batch_size, input_size)
        images = images.reshape(-1, 28 * 28)

        # Forward pass
        outputs = model(images)

        # Get predictions from the maximum value
        _, predicted = torch.max(outputs.data, 1)

        # Total number of labels
        total += labels.size(0)

        # Total correct predictions
        correct += (predicted == labels).sum().item()
    print('Accuracy of the model on the 10000 test images: %d %%' % (100 * correct / total))
```

参数：batch_size=10/50/100; epochs=5; optimizier : Adam (lr=0.001)

```
Epoch [5/5], Step [6000/6000], Loss: 0.0276
Accuracy of the model on the 10000 test images: 98.02 %
```

```
Epoch [5/5], Step [1200/1200], Loss: 0.0431
Accuracy of the model on the 10000 test images: 97.98 %
```

```
Epoch [5/5], Step [600/600], Loss: 0.1118
Accuracy of the model on the 10000 test images: 98.05 %
```

参数：batch_size=100; epochs=3/10/20; optimizier : Adam (lr=0.001)

```
Epoch [3/3], Step [600/600], Loss: 0.1401
Accuracy of the model on the 10000 test images: 97.63 %
```

```
Epoch [10/10], Step [600/600], Loss: 0.0154
Accuracy of the model on the 10000 test images: 98.10 %
```

```
Epoch [20/20], Step [600/600], Loss: 0.0017
Accuracy of the model on the 10000 test images: 98.02 %
```

参数：batch_size=100; epochs=5; optimizier : Adam (lr=0.01/0.001/0.0001)

```
Epoch [5/5], Step [600/600], Loss: 0.0626
Accuracy of the model on the 10000 test images: 96.77 %
```

```
Epoch [5/5], Step [600/600], Loss: 0.1118
Accuracy of the model on the 10000 test images: 98.05 %
```

```
Epoch [5/5], Step [600/600], Loss: 0.0502
Accuracy of the model on the 10000 test images: 94.61 %
```

参数：batch_size=100; epochs=20; optimizier : Adam (lr=0.0001)

```
Epoch [20/20], Step [600/600], Loss: 0.0260
Accuracy of the model on the 10000 test images: 97.63 %
```

对比每组的实验结果可以得出：

1. batch_size 的提升不能明显提升模型的准确率，但是可以减少迭代轮次并减少训练时间。
2. epochs 的提升对模型准确率提升也不大，尤其是 10 轮以上时效果很微小并且有过拟合的倾向。
3. 过大的学习率(0.01)很容易出现已经达到了最低点但是跨出去的情况，过小的学习率(0.0001)在训练轮次不够时明显还未收敛，需要提升训练轮次达到收敛效果。