Homework 3
DNN(Deep Neural Network)

2021050300
김재민

1. Source code of DNN

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import matplotlib.pyplot as plt

# DATA Loading
batch_size =12
train_dataset = datasets.MNIST(root='./data', train=True, transform=transforms.ToTensor(), download=True)
test_dataset = datasets.MNIST(root='./data', train=False, transform=transforms.ToTensor(), download=True)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)


# Model
class MLP(nn.Module):
    def __init__(self, num_hidden_layers):
        super().__init__()
        self.in_dim =28 *28
        self.out_dim =10
        dims = [self.in_dim, 512, 256, 128, 64, self.out_dim]
        self.linears = nn.ModuleList([
            nn.Linear(dims[i], dims[i +1]) for i in range(len(dims) -1)
        ])
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.view(-1, self.in_dim)
        for layer in self.linears[:-1]:
            x =self.relu(layer(x))
        return self.linears[-1](x)
```

2. Train and Test Code

```
# Evaluate
def evaluate(model, loader):
    model.eval()
    correct, total =0, 0
    with torch.no_grad():
        for images, labels in loader:
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
    return correct / total

# Train
accuracies = []
layers = [2, 3, 4, 5]

for num_layers in layers:
    print(f"₩nTraining {num_layers} number of layers")
    model = MLP(num_layers)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(3):
        epoch_loss = train(model, train_loader, optimizer, criterion)
        print(f"Epoch {epoch+1:2d} | Loss: {epoch_loss:.4f}")
    acc = evaluate(model, test_loader)
    accuracies.append(acc)
    print(f"Layers: {num_layers}, Accuracy: {acc:.4f}")

# Image Visualization Function
def imshow(img):
    npimg = img.numpy()
    plt.figure(figsize=(10, 2))
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.axis('off')
    plt.show()

dataiter = iter(test_loader)
images, labels =next(dataiter)
images = images[:10]
imshow(torchvision.utils.make_grid(images, nrow=10))
print('Ground Truth :', ' '.join(f'{label.item()}'for label in labels[:10]))
model.eval()

with torch.no_grad():
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)

print('Predicted    :', ' '.join(f'{pred.item()}'for pred in predicted))

# Accuracy Comparison Graph
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
plt.plot(layers, accuracies, marker='o')
plt.xlabel('Number of Hidden Layers')
plt.ylabel('Test Accuracy')
plt.title('Accuracy vs Number of Layers on MNIST')
plt.xticks(layers)
plt.grid(True)
plt.show()
```

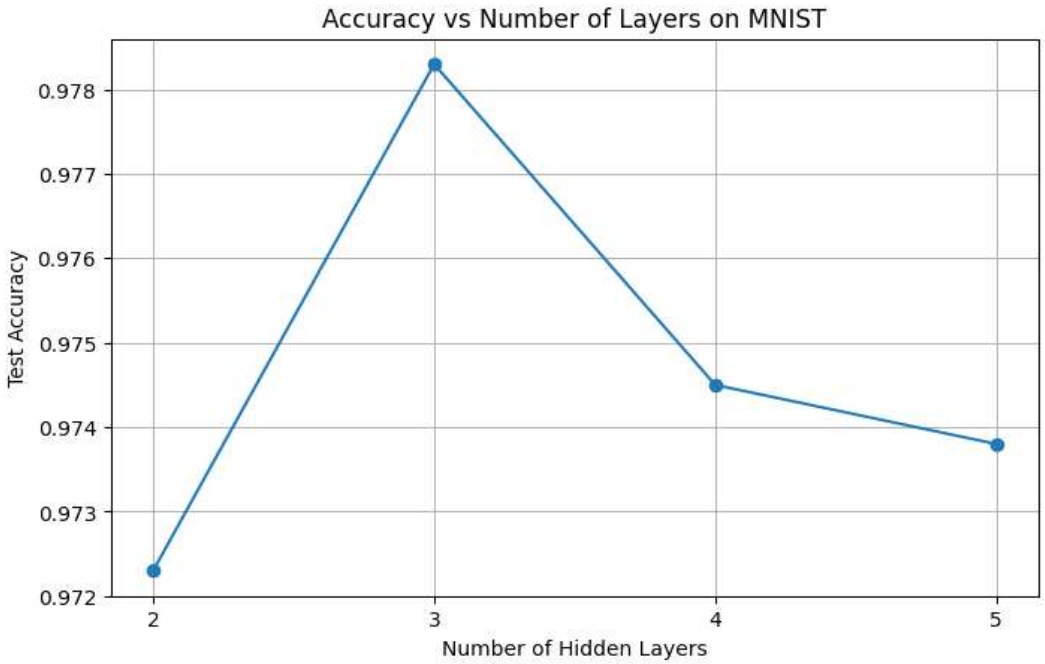
3. Conclusion

Layer Acuuracy

```
Training 2 number of layers
Epoch  1 | Loss: 0.2407
Epoch  2 | Loss: 0.1090
Epoch  3 | Loss: 0.0806
Layers: 2, Accuracy: 0.9723

Training 3 number of layers
Epoch  1 | Loss: 0.2484
Epoch  2 | Loss: 0.1099
Epoch  3 | Loss: 0.0793
Layers: 3, Accuracy: 0.9783

Training 4 number of layers
Epoch  1 | Loss: 0.2362
Epoch  2 | Loss: 0.1079
Epoch  3 | Loss: 0.0805
Layers: 4, Accuracy: 0.9745

Training 5 number of layers
Epoch  1 | Loss: 0.2315
Epoch  2 | Loss: 0.1080
Epoch  3 | Loss: 0.0801
Layers: 5, Accuracy: 0.9738
```

Image Visualization Function

Accuracy Comparison Graph



- 위 그래프는 은닉층의 개수가 2 ~ 5까지 총 4개의 경우의 정확도를 비교하고자 만들었습니다.
- 은닉층 수가 증가함에 따라 성능이 향상되다가, 3개에서 최대 성능을 보인 후 4개부터는 오히려 정확도가 하락하는 모습을 보여줍니다. (3개 : 0.9783 → 4개 : 0.9745)
- 이를 통해 층이 많아질수록 과적합 혹은 학습 난이도 증가가 발생할 수 있다는 점을 알 수 있습니다.
- 단순히 은닉층 수를 늘리는 것이 항상 성능의 향상으로 이어지지 않는다는 것을 알게 되었으며, 적절한 은닉층의 깊이 및 Neural Network의 구조를 선택하는 것이 중요하다는 것을 알게 되었습니다.
- 또한 `nn.ModuleList`를 사용하여 기존의 작성한 코드보다 간결하고 재사용 가능하게 구현이 가능하였습니다.