R E P O R T

[Homework#1]

수 강 과 목	: 딥러닝 심화
담 당 교 수	: 황영배교수님
학 과	: 산업인공지능학과
학 번	: 2024254015
이 름	: 박길순
제 출 일	: 2025.4.12

1. MNIST 데이터를 읽고, 70%는 학습데이터, 30%는 테스트데이터로 구분(dataloader)하시오

1.1 코드

```
import torch
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, random_split

# 1. MNIST 데이터를 불러오기 위한 전처리 정의 (이미지를 Tensor로 변환)
transform = transforms.ToTensor()

# 2. MNIST 전체 데이터셋 다운로드 (train=True가 전체 데이터를 가져옴)
mnist_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

# 3. 전체 데이터셋 크기 확인
total_size = len(mnist_dataset)
train_size = int(0.7 * total_size)  # 70%
test_size = total_size - train_size  # 30%

# 4. 학습용과 테스트용 데이터셋으로 나누기
train_dataset, test_dataset = random_split(mnist_dataset, [train_size, test_size])

# 5. 데이터 확인
print(f"MNIST 데이터셋 크기: {len(mnist_dataset)}")
print(f"학습 데이터셋 크기: {len(train_dataset)}")
print(f"테스트 데이터셋 크기: {len(test_dataset)}")
```

1.2 실행결과

```
MNIST 데이터셋 크기: 60000
학습 데이터셋 크기: 42000
테스트 데이터셋 크기: 18000
```

2. Hidden layer가 2개이고, 각 node는 20개인 MLP 모델(model)을 구현하시오.

2.1 코드

```
# MLP 모델 정의 (2개의 히든 레이어, 각 20개 노드)
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()  # 28x28 이미지를 784 벡터로 펼침
        self.model = nn.Sequential(
            nn.Linear(28*28, 20),
            nn.ReLU(),
            nn.Linear(20, 20),
            nn.ReLU(),
            nn.Linear(20, 10)  # 숫자 0~9 총 10개의 클래스
        )

    def forward(self, x):
        x = self.flatten(x)
        out = self.model(x)
        return out
```

3. 활성화함수를 ReLU와 Sigmoid로 바꿔서 테스트결과를 출력하시오.

3.1 활성함수 : ReLU

```
Epoch [1/5], Loss: 0.7230
Epoch [2/5], Loss: 0.3072
Epoch [3/5], Loss: 0.2552
Epoch [4/5], Loss: 0.2252
Epoch [5/5], Loss: 0.1999
Test Accuracy: 93.84%
```

3.2 활성함수 : Sigmoid

```
# MLP 모델 정의 (2개의 히든 레이어, 각 20개 노드)
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.flatten = nn.Flatten()  # 28x28 이미지를 784 벡터로 펼침
        self.model = nn.Sequential(
            nn.Linear(28*28, 20),
            nn.Sigmoid(),
            nn.Linear(20, 20),
            nn.Sigmoid(),
            nn.Linear(20, 10)  # 숫자 0~9 총 10개의 클래스
        )

    def forward(self, x):
        x = self.flatten(x)
        out = self.model
```

```
Epoch [1/5], Loss: 1.6392
Epoch [2/5], Loss: 0.7626
Epoch [3/5], Loss: 0.4924
Epoch [4/5], Loss: 0.3716
Epoch [5/5], Loss: 0.3096
Test Accuracy: 91.52%
```

4. Batch normalization을 적용했을 때와 하지 않았을 때의 결과를 출력하시오.

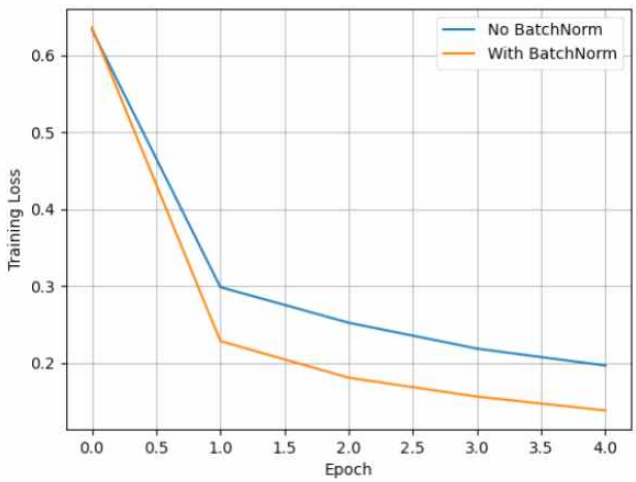
4.1 Batch normalization 적용

- 위의 ReLU 적용된 모델에 Batch normalization 적용

```
class MLP_BN(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.seq = nn.Sequential(
            nn.Linear(784, 20),
            nn.BatchNorm1d(20),
            nn.ReLU(),
            nn.Linear(20, 20),
            nn.BatchNorm1d(20),
            nn.ReLU(),
            nn.Linear(20, 10)
        )

    def forward(self, x):
        return self.seq(self.flatten(x))
```

4.2 Batch normalization 적용 Vs. 미적용



5. 카이밍 초기화, 제이비어 초기화, 정규분포 초기화를 적용했을 때의 결과를 출력하시오.

5.1 카이밍 초기화

```
=== Initializing with NORMAL ===
[NORMAL] Epoch 1, Loss: 0.9287
[NORMAL] Epoch 2, Loss: 0.4965
[NORMAL] Epoch 3, Loss: 0.3628
[NORMAL] Epoch 4, Loss: 0.3084
[NORMAL] Epoch 5, Loss: 0.2764
[NORMAL] Test Accuracy: 91.21%

=== Initializing with XAVIER ===
[XAVIER] Epoch 1, Loss: 0.5770
[XAVIER] Epoch 2, Loss: 0.2401
[XAVIER] Epoch 3, Loss: 0.1983
[XAVIER] Epoch 4, Loss: 0.1721
[XAVIER] Epoch 5, Loss: 0.1541
[XAVIER] Test Accuracy: 94.88%

=== Initializing with KAIMING ===
[KAIMING] Epoch 1, Loss: 0.5737
[KAIMING] Epoch 2, Loss: 0.2662
[KAIMING] Epoch 3, Loss: 0.2242
[KAIMING] Epoch 4, Loss: 0.1998
[KAIMING] Epoch 5, Loss: 0.1834
[KAIMING] Test Accuracy: 93.79%

NORMAL Final Test Accuracy: 91.21%
XAVIER Final Test Accuracy: 94.88%
KAIMING Final Test Accuracy: 93.79%
```