HW05

202401833 신해솔

0. 수행 결과

- train_model.py

```
INFO: Start Training
Epoch 1, Loss: 0.3616
Epoch 2, Loss: 0.1377
Epoch 3, Loss: 0.0908
Epoch 4, Loss: 0.0636
Epoch 5, Loss: 0.0466
Epoch 6, Loss: 0.0327
Epoch 7, Loss: 0.0256
Epoch 8, Loss: 0.0190
Epoch 9, Loss: 0.0150
Epoch 10, Loss: 0.0124
INFO: Training Complete
```

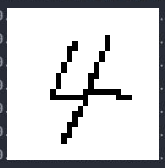
- eval_model.py

```
Total: 10000, Correct: 9790, Accuracy: 97.90%
```

- test_model.py

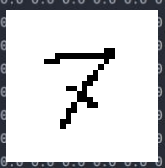
```
(.venv) root@74d84861763d:/home/AIDM/HW05# python test_model.py digit-3.png
Image shape: torch.Size([1, 28, 28])
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.
Model's prediction is 3
```

```
Image shape: torch.Size([1, 28, 28])
```



```
Model's prediction is 4
```

```
(.venv) root@74d84861763d:/home/AIDM/HW05# python test_model.py digit-7.png
Image shape: torch.Size([1, 28, 28])
```



```
Model's prediction is 7
```

1. 전체 소스 코드

- model.py

```python
import torch
import torch.nn as nn


class MLP(nn.Module):  6 usages  haesol1013
    def __init__(self):  haesol1013
        super().__init__()
        self.linear1 = nn.Linear( in_features: 784,  out_features: 512)
        self.linear2 = nn.Linear( in_features: 512, 512//2)
        self.out = nn.Linear(512//2,  out_features: 10)

    def forward(self, x):  haesol1013
        x = x.view(-1, 784)
        x = self.linear1(x)
        x = torch.sigmoid(x)
        x = self.linear2(x)
        x = torch.sigmoid(x)
        x = self.out(x)
        return x
```

- train_model.py

```python
import torch
import torch.nn as nn
from torch import optim
from torchvision import transforms, datasets
from torch.utils.data import DataLoader

from model import MLP


BATCH_SIZE = 32
train_dataset = datasets.MNIST(root="./data", train=True, download=True, transform=transforms.ToTensor())
train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)

model = MLP()

NUM_EPOCHS = 10
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

```
print("INFO: Start Training")
for epoch in range(NUM_EPOCHS):
    running_loss = 0.0

    for data in train_loader:
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

print("INFO: Training Complete")

out_path = "model.pt"
torch.save(model.state_dict(), f=out_path)
```

- eval_model.py

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

from model import MLP


BATCH_SIZE = 32
test_dataset = datasets.MNIST(root="./data", train=False, download=True, transform=transforms.ToTensor())
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False)

out_path = "model.pt"
model = MLP()
model.load_state_dict(torch.load(out_path))

with torch.no_grad():
    total_prediction = 0
    correct_prediction = 0

    for data in test_loader:
        inputs, labels = data
        outputs = model(inputs)
        _, predicted = torch.max(outputs, dim=1)
        total_prediction += labels.size(0)
        correct_prediction += (predicted == labels).sum().item()

print(f"Total: {total_prediction}, Correct: {correct_prediction}, Accuracy: {correct_prediction/total_prediction:.2%}")
```

- test_model.py

```
import argparse
import torch
from torchvision.io import read_image
import torchvision.transforms.functional as F

from model import MLP


parser = argparse.ArgumentParser()
parser.add_argument(*name_or_flags:"img_path", type=str)
args = parser.parse_args()

img_float32 = read_image(args.img_path)[:3, :, :].float() / 255.0
img_grayscale = F.rgb_to_grayscale(img_float32, num_output_channels=1)
inputs = 1 - img_grayscale

print(f"Image shape: {img_grayscale.shape}")
for width in inputs.squeeze(0):
    print(" ".join(f"{pixel.item():.1f}" for pixel in width))

out_path = "model.pt"
model = MLP()
model.load_state_dict(torch.load(out_path))

with torch.no_grad():
    outputs = model(inputs)
    _, pred = torch.max(outputs, dim=1)
print(f"Model's prediction is {pred[0]}")
```

2. model.py – 모델 정의

a. 필요 라이브러리 import

```
import torch
import torch.nn as nn
```

b. 모델 구조 정의

```
class MLP(nn.Module):  6 usages  haesol1013
    def __init__(self):  haesol1013
        super().__init__()
        self.linear1 = nn.Linear( in_features: 784,  out_features: 512)
        self.linear2 = nn.Linear( in_features: 512, 512//2)
        self.out = nn.Linear(512//2,  out_features: 10)
```

- 3개의 선형 레이어로 이루어진 모델 정의

c. 순전파 과정 정의

```
    def forward(self, x):  haesol1013
        x = x.view(-1, 784)
        x = self.linear1(x)
        x = torch.sigmoid(x)
        x = self.linear2(x)
        x = torch.sigmoid(x)
        x = self.out(x)
        return x
```

- view함수를 통해 (배치사이즈, 1, 28, 28)의 shape를 가지는 텐서를 (배치사이즈, 784)의 텐서로 변환

- 첫 번째 레이어와 두 번째 레이어 뒤에 시그모이드 적용

- 마지막 레이어를 거쳐 최종적으로 (32, 10)의 텐서 반환

3. train_model.py – 모델 학습

a. 필요 라이브러리 import

```
import torch
import torch.nn as nn
from torch import optim
from torchvision import transforms, datasets
from torch.utils.data import DataLoader

from model import MLP
```

b. 학습 데이터 로드 및 모델 로드

```
BATCH_SIZE = 32
train_dataset = datasets.MNIST(root="./data", train=True, download=True, transform=transforms.ToTensor())
train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)

model = MLP()
```

- MNIST 데이터셋 로드 및 배치사이즈를 32로 하는 데이터로더 생성(학습용)

- 정의한 모델 객체 생성

c. 하이퍼 파라미터 설정

```
BATCH_SIZE = 32
train_dataset = datasets.MNIST(root="./data", train=True, download=True, transform=transforms.ToTensor())
train_loader = DataLoader(dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True)

model = MLP()

NUM_EPOCHS = 10
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

- epoch는 10, 손실 함수는 Cross Entropy, optimizer는 Adam으로 설정

d. 모델 학습 및 저장

```
print("INFO: Start Training")
for epoch in range(NUM_EPOCHS):
    running_loss = 0.0

    for data in train_loader:
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss

    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader):.4f}")

print("INFO: Training Complete")

out_path = "model.pt"
torch.save(model.state_dict(), f=out_path)
```

- 매 배치마다 모델의 파라미터에 저장된 gradient 초기화 (optimizer.zero_grad())

- 모델의 ouput을 바탕으로 loss를 구한 뒤 역전파 수행 및 gradient 누적 (loss.backward())

- 모델의 파라미터 업데이트 (optimizer.step())

- 학습 결과 출력 및 학습 종료 출력

- 모델 가중치 저장

4. 모델 검증 – eval_model.py

a. 필요 라이브러리 import

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

from model import MLP
```

b. 테스트 데이터 로드 및 모델 로드

```
BATCH_SIZE = 32
test_dataset = datasets.MNIST(root="./data", train=False, download=True, transform=transforms.ToTensor())
test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=False)

out_path = "model.pt"
model = MLP()
model.load_state_dict(torch.load(out_path))
```

- 테스트 데이터 로드 및 모델 로드

- 저장된 가중치 로드

c. 모델 평가

```
with torch.no_grad():
    total_prediction = 0
    correct_prediction = 0

    for data in test_loader:
        inputs, labels = data
        outputs = model(inputs)
        _, predicted = torch.max(outputs, dim=1)
        total_prediction += labels.size(0)
        correct_prediction += (predicted == labels).sum().item()

print(f"Total: {total_prediction}, Correct: {correct_prediction}, Accuracy: {correct_prediction/total_prediction:.2%}")
```

- gradient 계산 비활성화 (torch.no_grad())

- (배치사이즈, 10) 텐서에 대해 dim=1에 대한 가장 큰 값의 인덱스 저장(predicted)

- label과 비교 후 맞을 경우 correct_prediction에 1 더하기

- 맞은 개수와 정확도 출력

5. 추론 – test_model.py

a. 필요 라이브러리 import

```
import argparse
import torch
from torchvision.io import read_image
import torchvision.transforms.functional as F

from model import MLP
```

- 코드 실행 시 추가 인자를 받기 위해 argparse 패키지 import

- png 파일을 읽고 텐서로 변환하기 위해 torchvision.io.read_image 함수 import

- 이미지를 grayscale로 변환하기 위해 torchvision.transforms.functional 모듈 import

b. 이미지 로드 및 변환

```
parser = argparse.ArgumentParser()
parser.add_argument(*name_or_flags: "img_path", type=str)
args = parser.parse_args()

img_float32 = read_image(args.img_path)[:3, :, :].float() / 255.0
img_grayscale = F.rgb_to_grayscale(img_float32, num_output_channels=1)
inputs = 1 - img_grayscale

print(f"Image shape: {img_grayscale.shape}")
for width in inputs.squeeze(0):
    print(" ".join(f"{pixel.item():.1f}" for pixel in width))
```

- argparse 패키지를 사용해 이미지 경로 저장

- read_image로 png 파일을 읽고, rgb부분만 슬라이싱 한 뒤, float32로 변환하고 정규화

- rgb_to_grayscale 함수를 사용하여 grayscale로 변환

- img_grayscale은 흰색은 1.0, 검은색은 0.0으로 표현되어 있으므로, MINIST 데이터셋과 맞추기 위해 색상 반전 (input = 1 – img_grayscale)

- shape와 이미지 텐서 출력

c. 추론

```
with torch.no_grad():
    outputs = model(inputs)
    _, pred = torch.max(outputs, dim=1)
print(f"Model's prediction is {pred[0]}")
```

- gradient 계산 비활성화 (torch.no_grad())

- 가장 큰 값의 인덱스 출력