

Dong-A Univ. (ISPL)



동아대학교
DONG-A UNIVERSITY

[실습] Multi Layer Perceptron - Part 1

컴퓨터공학부 AI학과
2024년 1학기 인공지능

Contents

1. Perceptron을 이용한 분류 실습
2. Multi-layer Perceptron 실습



Contents

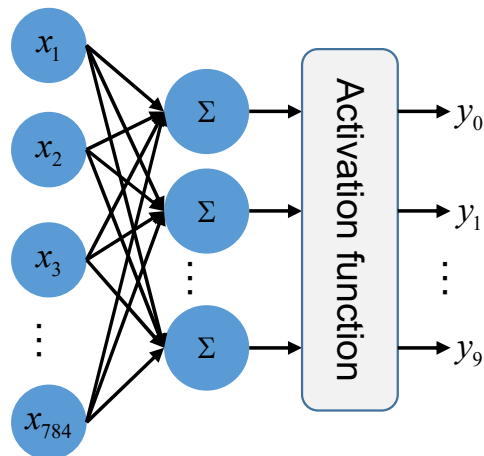
1. Perceptron을 이용한 분류 실습
2. Multi-layer Perceptron 실습



Perceptron을 이용한 분류 실습 - 실습 일정 공지

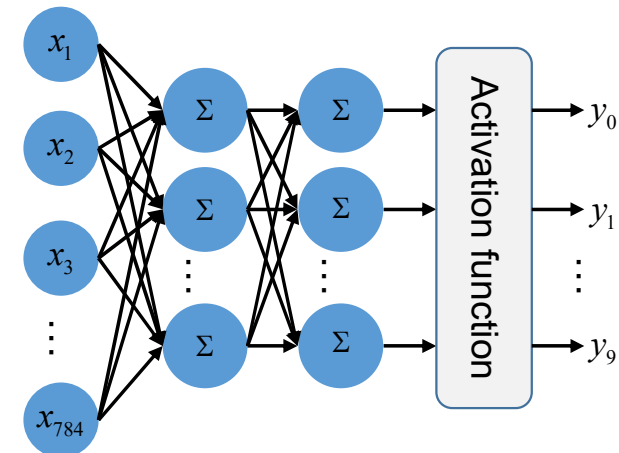
Perceptron을 이용한 MNIST 손글씨 데이터셋 분류

Single Layer Perceptron 실습



- ✓ Layer1 (Input: 784, Out: 10)
- ✓ Activation function: Softmax
- ✓ Loss function: Cross Entropy

Multi Layer Perceptron 실습



- ✓ Layer1 (Input: 784, Out: 100)
- ✓ Layer2 (Input: 100, Out: 10)
- ✓ Activation function: Softmax, Sigmoid
- ✓ Loss function: Cross Entropy

Perceptron을 이용한 분류 실습 - 데이터셋 소개

- **MNIST 데이터베이스 (Modified National Institute of Standards and Technology)**

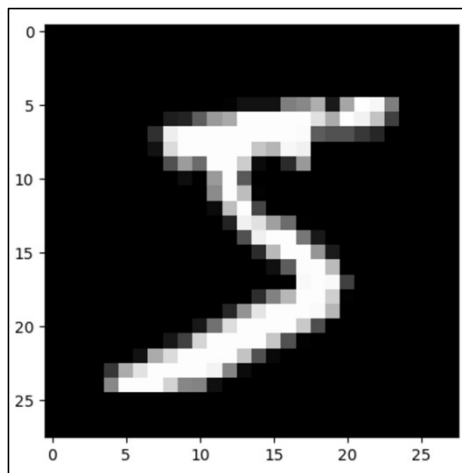
- 손으로 쓴 숫자들로 이루어진 대형 데이터베이스
- Train dataset 60,000개, Test dataset 10,000개로 구성됨



Perceptron을 이용한 분류 실습 - 실습 목표

- 실습 목표: MNIST 손글씨 데이터를 분류하는 단층 perceptron 모델 학습

- 입력: 손글씨 이미지 (28x28x1)
- 출력: 0~9까지 숫자들의 정답 확률



28 x 28 x 1

Perceptron

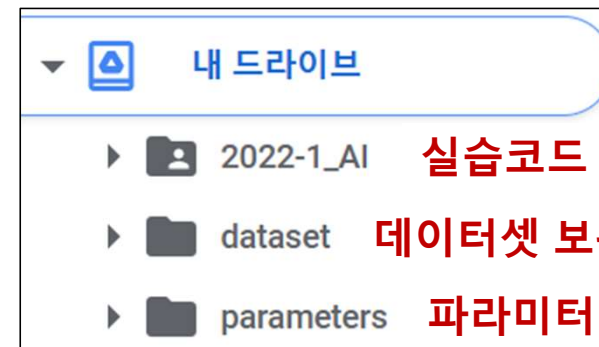
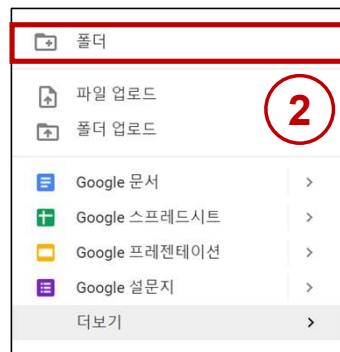
예측 결과

클래스	정답 확률
0	0.1666
1	0.0889
...	...
5	0.750
...	...
8	0.0008
9	0.0113

Perceptron을 이용한 분류 실습

■ 데이터셋 다운로드

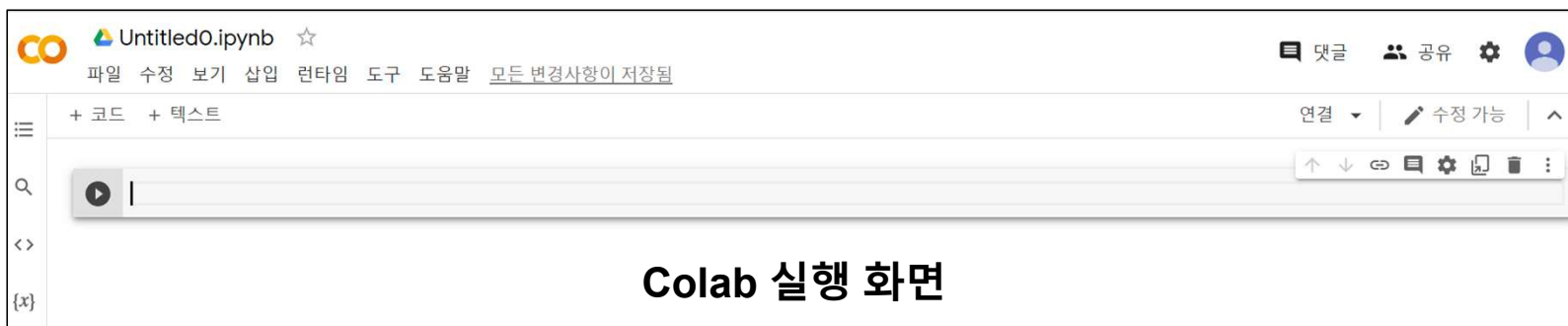
- (1) 구글 드라이브 실행
- (2) 데이터셋 및 파라미터 저장을 위한 폴더 생성 (Ex. dataset, parameters)
- (3) 새로운 노트북 파일 생성 및 실행 (Ex. MLP_Experiment_Part1.ipynb)



실습코드 보관용 폴더

데이터셋 보관용 폴더

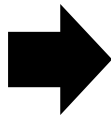
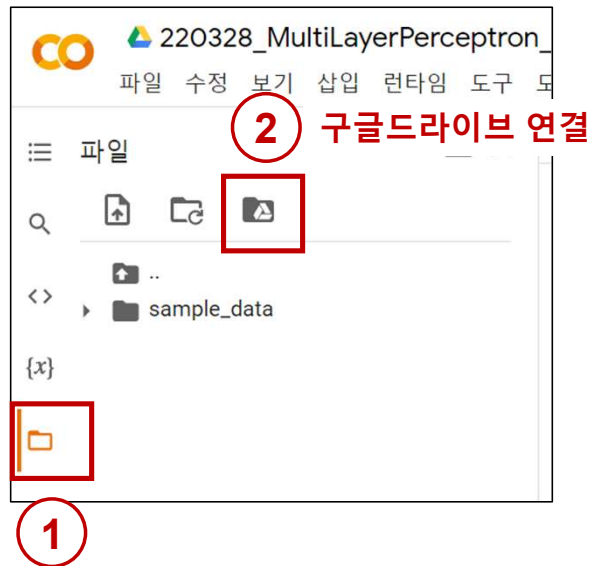
파라미터 보관용 폴더



Perceptron을 이용한 분류 실습

■ 데이터셋 다운로드

- (4) Colab에서 구글 드라이브 연결
- (5) 데이터셋을 저장 할 경로 가져오기 → 변수로 저장



Perceptron을 이용한 분류 실습

■ 데이터셋 다운로드

- (6) 패키지 선언 및 MNIST 데이터셋 저장

▼ 패키지 선언 ①

```
✓ [2] import torch
      import torch.nn as nn
      import torchvision.datasets as dataset
      import torchvision.transforms as transform
      from torch.utils.data import DataLoader
```

▼ Dataset 선언 ②

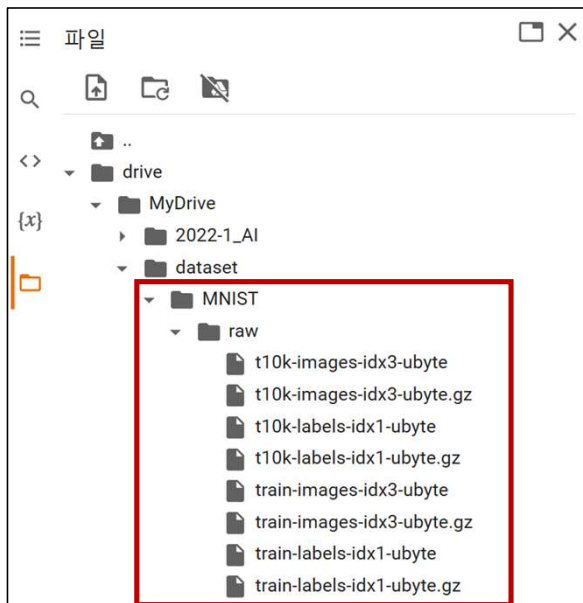
```
✓ [3] # Training dataset 다운로드
      mnist_train = dataset.MNIST(root = datasetPath, # 데이터셋을 저장할 위치
                                  train = True,
                                  transform = transform.ToTensor(),
                                  download = True)

      # Testing dataset 다운로드
      mnist_test = dataset.MNIST(root = datasetPath,
                                  train = False,
                                  transform = transform.ToTensor(),
                                  download = True)
```

Perceptron을 이용한 분류 실습

■ 데이터셋 다운로드

- (7) MNIST 데이터셋 저장 여부 확인
- (8) MNIST 데이터셋 형상 확인



데이터셋 저장 여부 확인

▼ MNIST 데이터셋 형상 확인

```
[5] import matplotlib.pyplot as plt
    print(len(mnist_train)) # training dataset 개수 확인

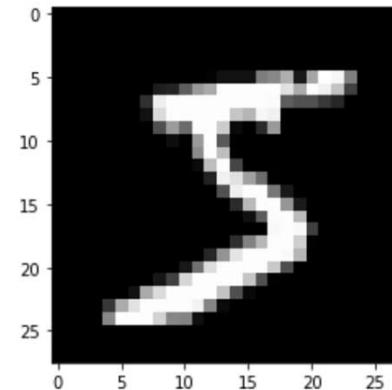
    first_data = mnist_train[0]
    print(first_data[0].shape) # 첫번째 data의 형상 확인
    print(first_data[1]) # 첫번째 data의 정답 확인

    plt.imshow(first_data[0][0,:,:], cmap='gray')
    plt.show()
```

60000

torch.Size([1, 28, 28])

5



→ 60000

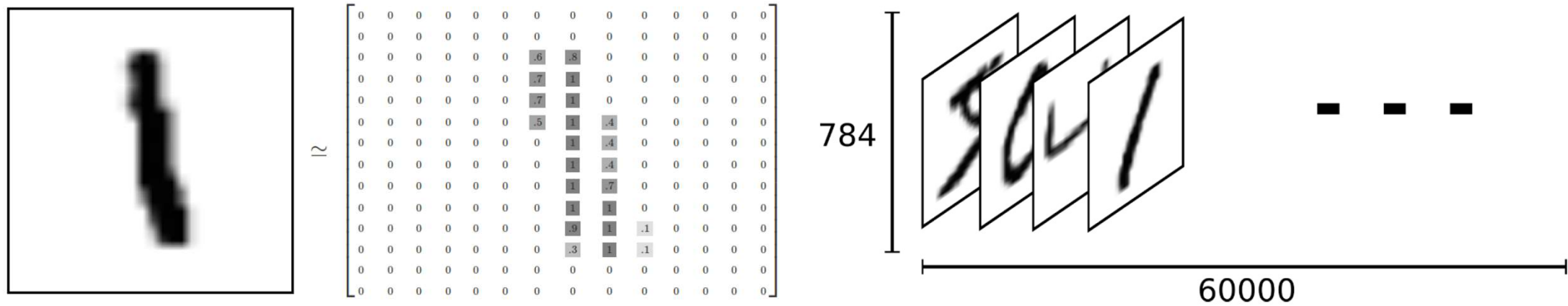
→ 1 x 28 x 28

→ 5

Perceptron을 이용한 분류 실습

- **[참고] MNIST 데이터셋 형상**




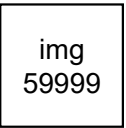
- Image: 28-by-28 (pixels)



Perceptron을 이용한 분류 실습

- [참고] MNIST 데이터셋 형상

MNIST train dataset

Image				...	
Label	5	0	4	...	8

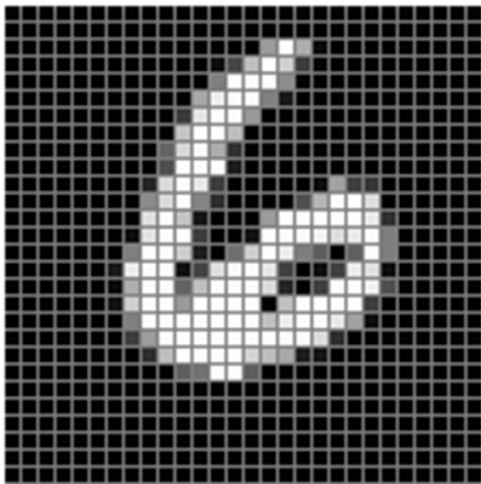
MNIST test dataset

Image				...	
Label	7	2	1	...	6

Perceptron을 이용한 분류 실습

■ MNIST 데이터셋의 perceptron 입력 방법

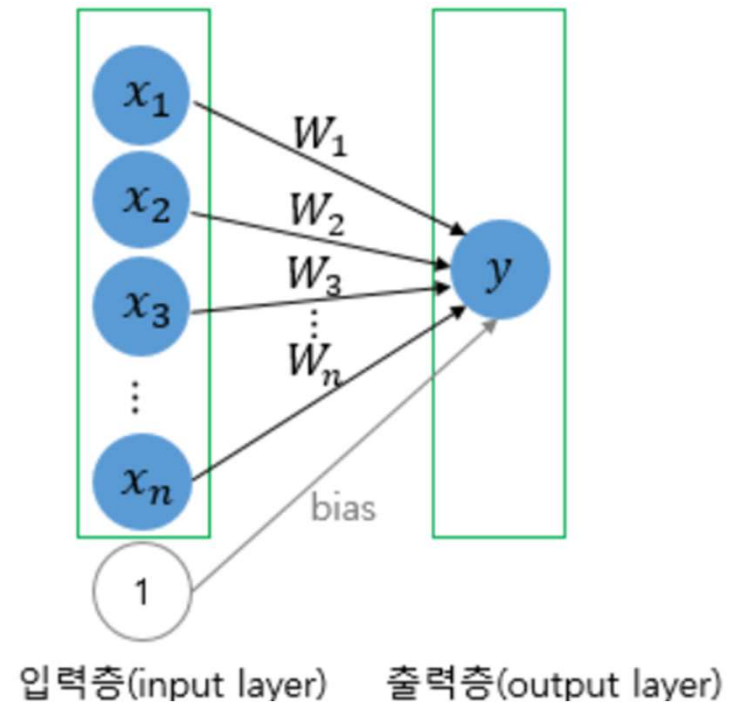
- Perceptron의 각 노드는 한번에 1개의 값을 입력 받을 수 있음
- 따라서 2D 형태 이미지의 전처리가 필요함



28 x 28 이미지



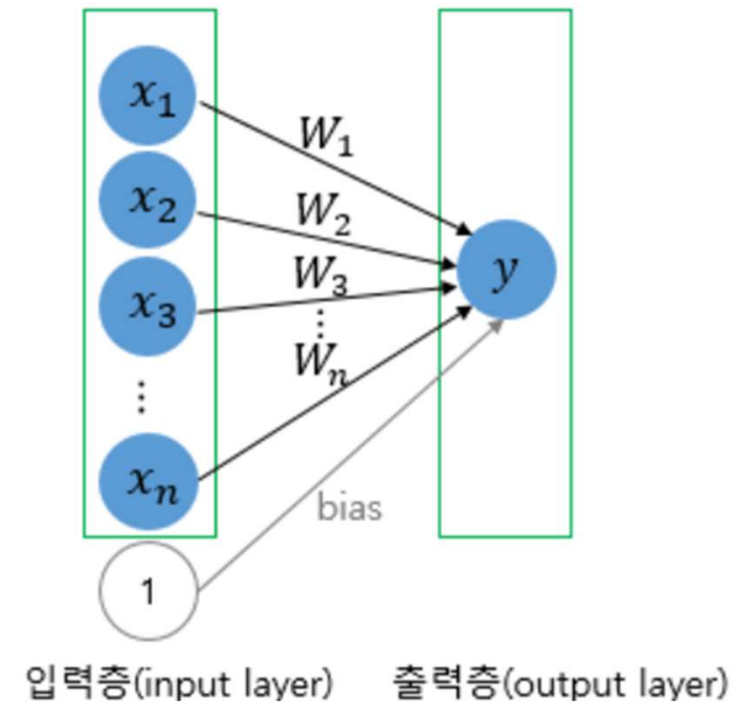
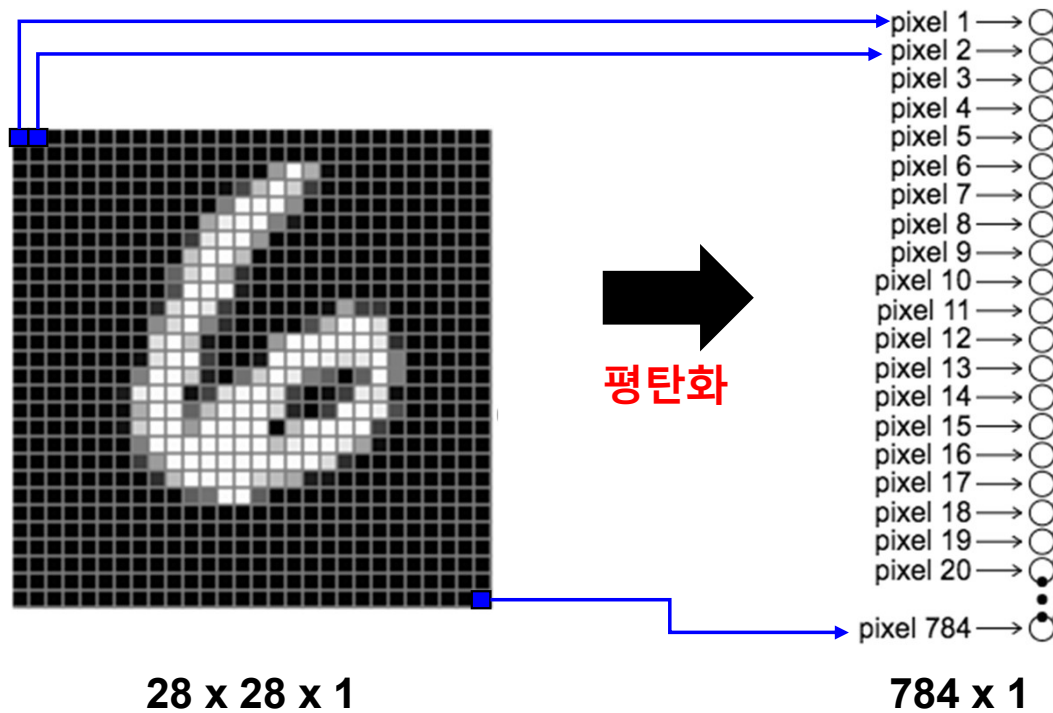
입력 불가!



Perceptron을 이용한 분류 실습

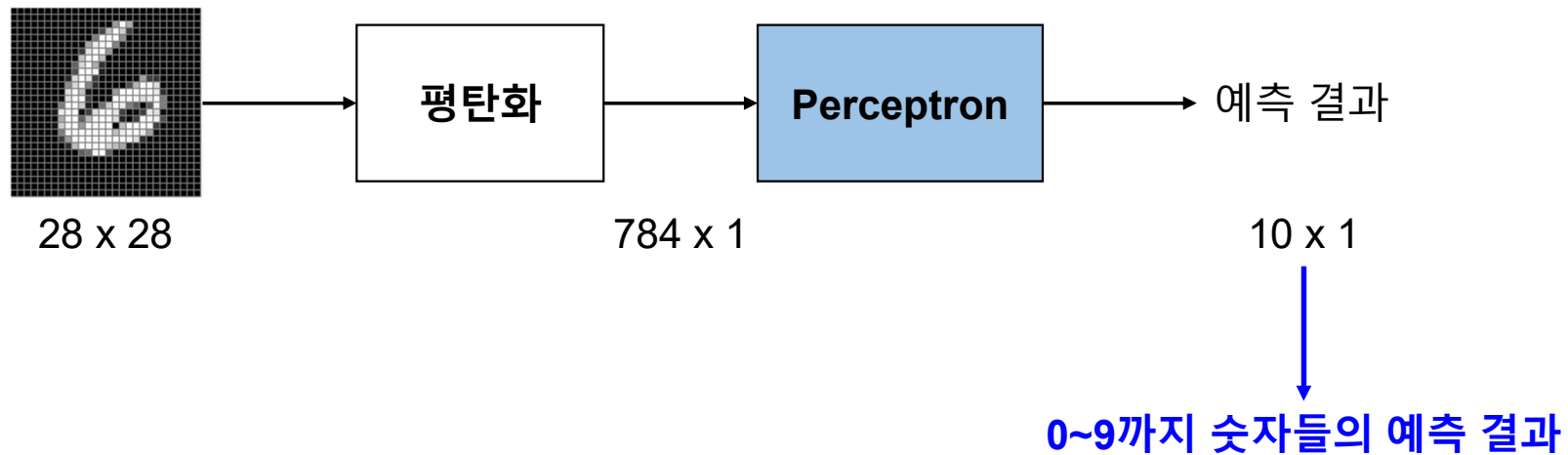
■ MNIST 데이터셋의 perceptron 입력 방법

- Perceptron의 각 노드는 한번에 1개의 값을 입력 받을 수 있음
- 따라서 2D 형태 이미지의 전처리가 필요함 → 평탄화



Perceptron을 이용한 분류 실습

- MNIST 데이터셋의 perceptron 입력 방법
 - Perceptron의 각 노드는 한번에 1개의 값을 입력 받을 수 있음
 - 따라서 2D 형태 이미지의 전처리가 필요함 → 평탄화



Perceptron을 이용한 분류 실습

▪ MNIST 데이터셋의 perceptron 입력 방법

- Perceptron의 각 노드는 한번에 1개의 값을 입력 받을 수 있음
- 따라서 2D 형태 이미지의 전처리가 필요함 → 평탄화

```
[14] first_img = first_data[0]
      print(first_img.shape)

      first_img = first_img.view(-1, 28*28) # 이미지 평탄화 수행 2D -> 1D
      print(first_img.shape)

      torch.Size([1, 28, 28])
      torch.Size([1, 784])
```

- ❖ torch.Tensor.view 함수를 통해 tensor의 형태를 변경할 수 있음
(<https://pytorch.org/docs/stable/generated/torch.Tensor.view.html>)

Perceptron을 이용한 분류 실습

▪ MNIST 분류를 위한 단층 perceptron 모델 학습

- (1) 단층 perceptron 모델 정의
 - ✓ Pytorch에서 구현하는 모델은 반드시 2가지 함수를 선언해야 함: `__init__`, `forward`

▾ Single Layer Perceptron 모델 정의

```
[15] class SLP(nn.Module):  
  
    def __init__(self):  
        super(SLP, self).__init__()  
        # SLP의 입력은 784개, 출력은 10개  
        self.fc = nn.Linear(in_features=784, out_features=10)  
  
    def forward(self, x):  
        x = x.view(-1, 28*28) # 이미지 평탄화  
        y = self.fc(x)  
        return y
```

Perceptron을 이용한 분류 실습

▪ MNIST 분류를 위한 단층 perceptron 모델 학습

- (1) 단층 perceptron 모델 정의
 - ✓ Pytorch에서 구현하는 모델은 반드시 2가지 함수를 선언해야 함: `__init__`, `forward`

▾ Single Layer Perceptron 모델 정의

```
[15] class SLP(nn.Module):  
    def __init__(self):  
        super(SLP, self).__init__()  
        # SLP의 입력은 784개, 출력은 10개  
        self.fc = nn.Linear(in_features=784, out_features=10)  
  
    def forward(self, x):  
        x = x.view(-1, 28*28) # 이미지 평탄화  
        y = self.fc(x)  
        return y
```

❖ `__init__()` 함수에 포함해야 하는 정보

- ✓ Parameter를 가지는 layer 정보
(Fully connected layer, Convolutional layer 등)
- ✓ Activation function

❖ [참고] `torch.nn.Linear` = Fully connected layer

Perceptron을 이용한 분류 실습

▪ MNIST 분류를 위한 단층 perceptron 모델 학습

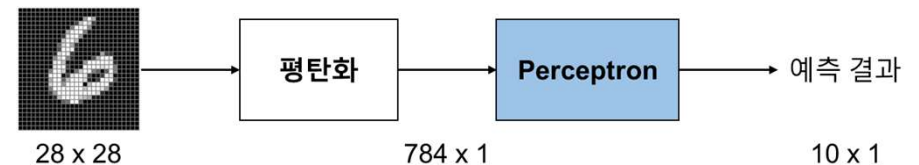
- (1) 단층 perceptron 모델 정의
 - ✓ Pytorch에서 구현하는 모델은 반드시 2가지 함수를 선언해야 함: `__init__`, `forward`

▾ Single Layer Perceptron 모델 정의

```
[15] class SLP(nn.Module):  
  
    def __init__(self):  
        super(SLP, self).__init__()  
        # SLP의 입력은 784개, 출력은 10개  
        self.fc = nn.Linear(in_features=784, out_features=10)  
  
    def forward(self, x):  
        x = x.view(-1, 28*28) # 이미지 평탄화  
        y = self.fc(x)  
        return y
```

❖ `forward()` 함수에 포함해야 하는 정보

- ✓ 모델의 동작 순서
- ✓ 각 layer, 함수의 입출력 관계



Perceptron을 이용한 분류 실습

- MNIST 분류를 위한 단층 perceptron 모델 학습
 - (2) Hyper-parameter 지정

▾ Hyper-parameters 지정

```
✓ [16] batch_size = 100  
0초 learning_rate = 0.1  
training_epochs = 15  
loss_function = nn.CrossEntropyLoss()  
network = SLP()  
optimizer = torch.optim.SGD(network.parameters(), lr = learning_rate)  
  
data_loader = DataLoader(dataset=mnist_train,  
                           batch_size=batch_size,  
                           shuffle=True,  
                           drop_last=True)
```

❖ 실습에 사용되는 hyper-parameter

- ✓ Batch size: 100
- ✓ Learning rate: 0.1
- ✓ Epoch: 15회 학습
- ✓ Loss function: Cross entropy error
- ✓ Optimizer: SGD

Perceptron을 이용한 분류 실습

- MNIST 분류를 위한 단층 perceptron 모델 학습
 - (2) Hyper-parameter 지정

▾ Hyper-parameters 지정

```
✓ [16] batch_size = 100  
0초 learning_rate = 0.1  
training_epochs = 15  
loss_function = nn.CrossEntropyLoss()  
network = SLP()  
optimizer = torch.optim.SGD(network.parameters(), lr = learning_rate)  
  
data_loader = DataLoader(dataset=mnist_train,  
                           batch_size=batch_size,  
                           shuffle=True,  
                           drop_last=True)
```

❖ [참고사항] torch.nn.CrossEntropyLoss() 함수

- ① 예측 값들에 대해 자동으로 softmax 적용
- ② 정답 값과 예측 값을 이용해 cross entropy loss 측정

Perceptron을 이용한 분류 실습

- MNIST 분류를 위한 단층 perceptron 모델 학습
 - (2) Hyper-parameter 지정

▼ Hyper-parameters 지정

```
✓ [16] batch_size = 100  
0초 learning_rate = 0.1  
training_epochs = 15  
loss_function = nn.CrossEntropyLoss()  
network = SLP()  
optimizer = torch.optim.SGD(network.parameters(), lr = learning_rate)
```

```
data_loader = DataLoader(dataset=mnist_train,  
                          batch_size=batch_size,  
                          shuffle=True,  
                          drop_last=True)
```

Batch 단위 학습을 위해 DataLoader 함수 사용

Perceptron을 이용한 분류 실습

▪ MNIST 분류를 위한 단층 perceptron 모델 학습

- (3) Perceptron 학습을 위한 반복문 선언

```
[17] for epoch in range(training_epochs): → 전체 데이터에 대한 반복: epoch
```

```
    avg_cost = 0  
    total_batch = len(data_loader)
```

```
    for img, label in data_loader: → 1 epoch 내의 배치에 대한 반복: iteration
```

```
        pred = network(img) (1)
```

```
        loss = loss_function(pred, label) (2)
```

```
        optimizer.zero_grad() # gradient 초기화
```

```
        loss.backward() (3)
```

```
        optimizer.step() (4)
```

```
    avg_cost += loss / total_batch → 모든 배치에 대한 평균 loss 값 계산
```

```
    print('Epoch: %d Loss = %f'%(epoch+1, avg_cost))
```

```
    print('Learning finished')
```

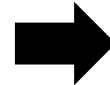
- 1) 입력 이미지에 대해 forward pass
- 2) 예측 값, 정답을 이용해 loss 계산
- 3) 모든 weight에 대해 편미분 값 계산
- 4) 파라미터 업데이트

Perceptron을 이용한 분류 실습

▪ MNIST 분류를 위한 단층 perceptron 모델 학습

- (3) Perceptron 학습을 위한 반복문 선언

```
[17] for epoch in range(training_epochs):  
    avg_cost = 0  
    total_batch = len(data_loader)  
  
    for img, label in data_loader:  
  
        pred = network(img)  
  
        loss = loss_function(pred, label)  
        optimizer.zero_grad() # gradient 초기화  
        loss.backward()  
        optimizer.step()  
  
        avg_cost += loss / total_batch  
  
    print('Epoch: %d Loss = %f'%(epoch+1, avg_cost))  
  
    print('Learning finished')
```



```
Epoch: 1 Loss = 0.535653  
Epoch: 2 Loss = 0.359482  
Epoch: 3 Loss = 0.331297  
Epoch: 4 Loss = 0.316829  
Epoch: 5 Loss = 0.307059  
Epoch: 6 Loss = 0.300268  
Epoch: 7 Loss = 0.295030  
Epoch: 8 Loss = 0.290915  
Epoch: 9 Loss = 0.287282  
Epoch: 10 Loss = 0.284525  
Epoch: 11 Loss = 0.282014  
Epoch: 12 Loss = 0.279782  
Epoch: 13 Loss = 0.277851  
Epoch: 14 Loss = 0.276001  
Epoch: 15 Loss = 0.274432  
Learning finished
```


Perceptron을 이용한 분류 실습

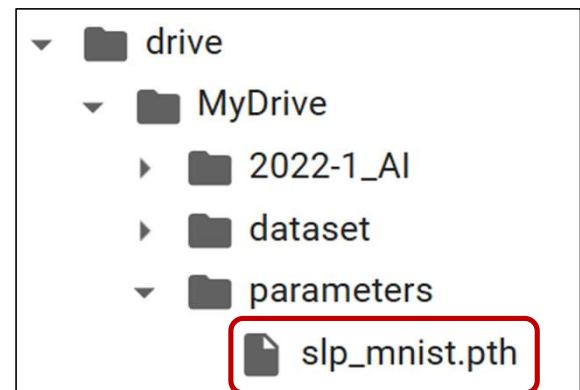
- MNIST 분류를 위한 단층 perceptron 모델 학습
 - (4) 학습이 완료된 weight parameter 저장 및 확인

❖ Weight parameter 저장

```
[23] torch.save(network.state_dict(), parameterPath+"slp_mnist.pth")
```

❖ 저장된 weight parameter 불러오기 (예시)

```
[24] new_network = SLP()  
      new_network.load_state_dict(torch.load(parameterPath+"slp_mnist.pth"))  
  
<All keys matched successfully>
```



Perceptron을 이용한 분류 실습

- MNIST 분류를 위한 단층 perceptron 모델 학습

- (5) MNIST test dataset 분류 성능 확인

```
[25] with torch.no_grad(): # test에서는 기울기 계산 제외

    img_test = mnist_test.data.float()
    label_test = mnist_test.targets

    prediction = network(img_test) # 전체 test data를 한번에 계산

    correct_prediction = torch.argmax(prediction, 1) == label_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())
```

Accuracy: 0.8895999789237976

정답률: 88.9%

예측 값이 가장 높은 숫자(0~9)와
정답데이터가 일치한 지 확인

Contents

1. Perceptron을 이용한 분류 실습
2. Multi-layer Perceptron 실습



Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron 모델 학습: 2-layer

- (1) 패키지 선언 및 MNIST 데이터셋 다운 (이전 실습 자료와 동일)

▼ 패키지 선언 ①

```
✓ [2] import torch
import torch.nn as nn
import torchvision.datasets as dataset
import torchvision.transforms as transform
from torch.utils.data import DataLoader
```

▼ 폴더 경로 저장 ②

```
✓ [2] 1 datasetPath = "./drive/MyDrive/dataset/"
1조 2 parameterPath = "./drive/MyDrive/parameters/"
```

▼ Dataset 선언 ③

```
✓ [3] # Training dataset 다운로드
3초 mnist_train = dataset.MNIST(root = datasetPath, # 데이터셋을 저장할 위치
                                train = True,
                                transform = transform.ToTensor(),
                                download = True)

# Testing dataset 다운로드
mnist_test = dataset.MNIST(root = datasetPath,
                            train = False,
                            transform = transform.ToTensor(),
                            download = True)
```

Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

- (2) MLP 모델 정의
 - ✓ fc1의 출력 노드와 fc2의 입력 노드가 반드시 동일하여야 함
 - ▾ Multi Layer Perceptron 모델 정의: 2-Layer

```
[4] 1 class MLP(nn.Module): # 2-layer
    2     def __init__(self):
    3         super(MLP, self).__init__()
    4         self.fc1 = nn.Linear(in_features=784, out_features=100)
    5         self.fc2 = nn.Linear(in_features=100, out_features=10)
    6         self.sigmoid = nn.Sigmoid()
    7
    8     def forward(self, x):
    9         x = x.view(-1, 28*28)
   10         y = self.sigmoid(self.fc1(x))
   11         y = self.fc2(y)
   12         return y
```

Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

- (3) Hyper-parameter 지정

▾ Hyper-parameter 지정

```
[5] 1 batch_size = 100
    2 learning_rate = 0.1
    3 training_epochs = 15
    4 loss_function = nn.CrossEntropyLoss()
    5 network = MLP()
    6 optimizer = torch.optim.SGD(network.parameters(), lr = learning_rate)
    7
    8 data_loader = DataLoader(dataset = mnist_train,
    9                          batch_size = batch_size,
   10                          shuffle = True,
   11                          drop_last = True)
```

❖ 실습에 사용되는 hyper-parameter

- ✓ Batch size: 100
- ✓ Learning rate: 0.1
- ✓ Epoch: 15회 학습
- ✓ Loss function: Cross entropy error
- ✓ Optimizer: SGD

Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

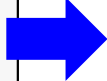
- (3) Network training 을 위한 반복문 선언

▼ Network Training

```
[6] 1 for epoch in range(training_epochs):
    2     avg_cost = 0
    3     total_batch = len(data_loader)
    4
    5     for img, label in data_loader:
    6         pred = network(img)
    7
    8         loss = loss_function(pred, label)
    9         optimizer.zero_grad()
   10         loss.backward()
   11         optimizer.step()
   12
   13     avg_cost += loss / total_batch
   14
   15     print('Epoch: %d Loss = %f' %(epoch+1, avg_cost))
   16 print('Learning finished')
```

→ 전체 데이터에 대한 반복: epoch

→ 1 epoch 내의 배치에 대한 반복: iteration

- 
- 1) 입력 이미지에 대해 forward pass
 - 2) 예측 값, 정답을 이용해 loss 계산
 - 3) 모든 weight에 대해 편미분 값 계산
 - 4) 파라미터 업데이트

→ 모든 배치에 대한 평균 loss 값 계산

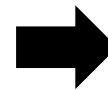
Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

- (3) Network training 을 위한 반복문 선언 → 결과 확인

▼ Network Training

```
✓ 1분 [6] 1 for epoch in range(training_epochs):
      2     avg_cost = 0
      3     total_batch = len(data_loader)
      4
      5     for img, label in data_loader:
      6         pred = network(img)
      7
      8         loss = loss_function(pred, label)
      9         optimizer.zero_grad()
     10         loss.backward()
     11         optimizer.step()
     12
     13         avg_cost += loss / total_batch
     14
     15     print('Epoch: %d Loss = %f' %(epoch+1, avg_cost))
     16 print('Learning finished')
```



```
Epoch: 1 Loss = 1.153683
Epoch: 2 Loss = 0.451747
Epoch: 3 Loss = 0.361733
Epoch: 4 Loss = 0.324818
Epoch: 5 Loss = 0.302182
Epoch: 6 Loss = 0.285703
Epoch: 7 Loss = 0.272042
Epoch: 8 Loss = 0.259743
Epoch: 9 Loss = 0.248782
Epoch: 10 Loss = 0.238609
Epoch: 11 Loss = 0.229102
Epoch: 12 Loss = 0.220335
Epoch: 13 Loss = 0.212063
Epoch: 14 Loss = 0.204168
Epoch: 15 Loss = 0.196959
Learning finished
```

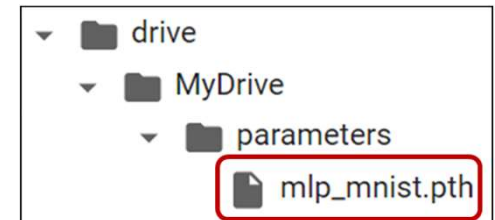
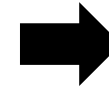
Multi-layer Perceptron 실습

- MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

- (4) 학습이 완료된 Network의 Weight parameter 저장 및 확인

▼ Weight parameter 저장

```
✓ [7] 1 torch.save(network.state_dict(), parameterPath+'mlp_mnist.pth')  
0초
```



▼ 저장된 Weight parameter 불러오기

```
✓ [8] 1 new_network = MLP()  
0초    2 new_network.load_state_dict(torch.load(parameterPath+'mlp_mnist.pth'))
```

Multi-layer Perceptron 실습

■ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

- (5) MNIST Test dataset 분류 성능 확인
 - ✓ Single layer의 성능보다 약 5% 높은 성능

▼ Training된 Network 성능 확인

```
✓ [9] 1 with torch.no_grad():  
0초 2     img_test = mnist_test.data.float()  
3     label_test = mnist_test.targets  
4  
5     prediction = network(img_test) # 전체 test data를 한번에 계산  
6  
7     correct_prediction = torch.argmax(prediction, 1) == label_test  
8     accuracy = correct_prediction.float().mean()  
9     print("Accuracy:", accuracy.item())
```

Accuracy: 0.9448000192642212

정답률: 94.4%

Multi-layer Perceptron 실습

▪ MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer

• (6) 예측 결과 값 확인

```
[10] 1 first_data = mnist_test.data[0] # Test dataset 중 첫번째 Image 추출
      2 with torch.no_grad():
      3     prediction = network(first_data.view(-1, 784).float())
      4 print(prediction)

tensor([[ -0.9629,  -7.2236,   1.5862,   3.5161,  -2.7709,   0.3299, -11.9995,
          14.1315,   0.3709,   3.6434]])
```

Test dataset 중 첫번째 Image에
대한 예측 값 확인

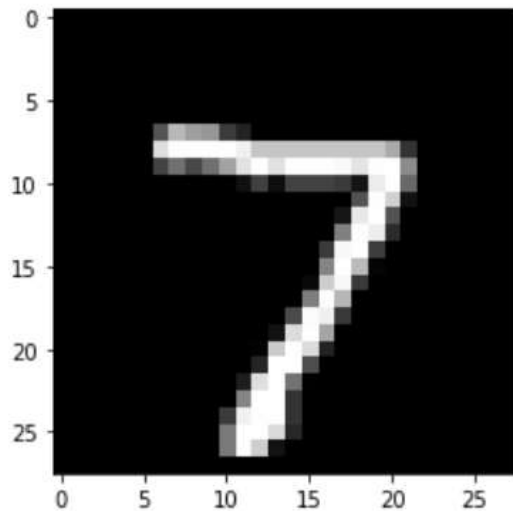
```
[11] 1 prediction_num = torch.argmax(prediction) # 예측 점수가 가장 높은 숫자 가져오기
      2 print("예측 값은 %d 입니다." %(prediction_num))
```

예측 값은 7 입니다.

Multi-layer Perceptron 실습

- MNIST 분류를 위한 Multi-layer Perceptron (MLP) 모델 학습: 2-layer
 - (7) 정답 이미지 확인

```
[12] 1 import matplotlib.pyplot as plt  
      2 plt.imshow(first_data, cmap='gray')  
      3 plt.show()
```





Questions & Answers

Dongsan Jun (dsjun@dau.ac.kr)

Image Signal Processing Laboratory (www.donga-ispl.kr)

Dept. of AI

Dong-A University, Busan, Rep. of Korea

