MNIST 관련 실습

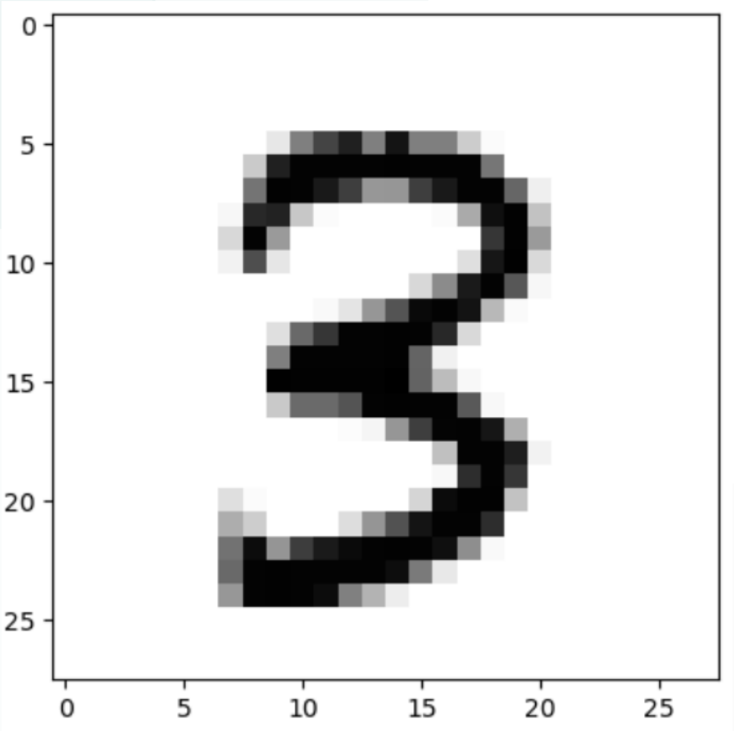
한신대학교 AINC Lab
노진산
rohjinsan02@hs.ac.kr
2024.12.17

목 차

- ▶ MNIST은 무엇인지
- ▶ MNIST_introduction 코드 실습
- ▶ MNIST_backprop 코드 실습
- ▶ Weight Initialization은 무엇인지
- ▶ Xabier / He initailization

MNIST

MNIST : handwritten digits data set



-28*28 image

- 1 channel gray image

-0~9 digits

training set : 60000 samples

test set ：10000 samples

MNIST_introduction 실습

```
In [1]: # Lab 7 Learning rate and Evaluation
        import torch
        import torchvision.datasets as dsets
        import torchvision.transforms as transforms
        import matplotlib.pyplot as plt
        import random
```

```
In [2]: device = 'cuda' if torch.cuda.is_available() else 'cpu'

        # for reproducibility
        random.seed(777)
        torch.manual_seed(777)
        if device == 'cuda':
            torch.cuda.manual_seed_all(777)
```

```
In [4]: # parameters
        training_epochs = 15
        batch_size = 100
```

```
In [5]: # MNIST dataset
        mnist_train = dsets.MNIST(root='MNIST_data/',
                                  train=True,
                                  transform=transforms.ToTensor(),
                                  download=True)

        mnist_test = dsets.MNIST(root='MNIST_data/',
                                 train=False,
                                 transform=transforms.ToTensor(),
                                 download=True)
```

torchvision패키지

-popular datasets
ex)MNIST,Fashion-MNIST,EMIST...

-model architectures
ex)Alexnet,Resnet

-common image transformation

-torchvision.utils

batch_size
-데이터를 몇개씩 불러올지

MNIST_introduction 실습

```
In [6]: # dataset loader
        data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                                  batch_size=batch_size,
                                                  shuffle=True,
                                                  drop_last=True)
```

```
In [7]: # MNIST data image of shape 28 * 28 = 784
        linear = torch.nn.Linear(784, 10, bias=True).to(device)
```

```
In [8]: # define cost/loss & optimizer
        criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
        optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)
```

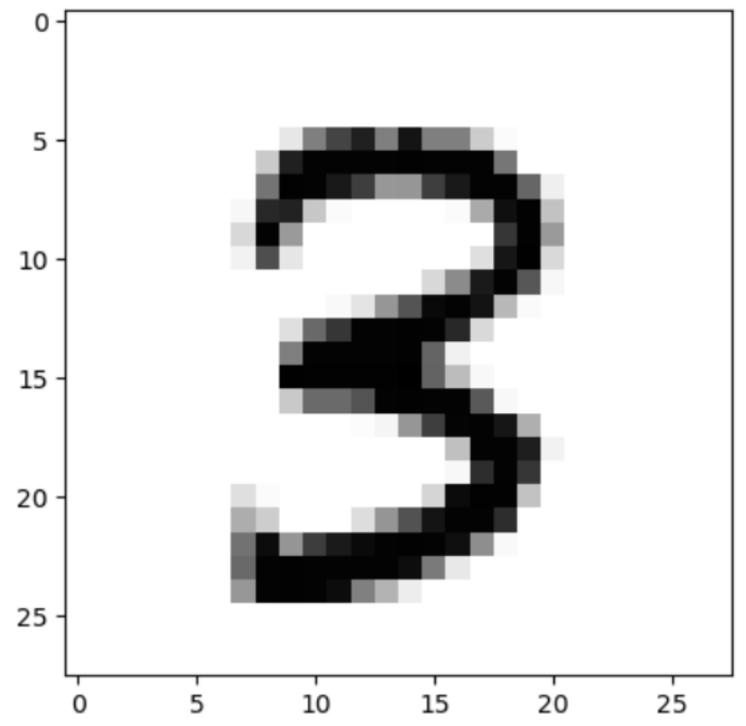
```
In [9]: for epoch in range(training_epochs):
            avg_cost = 0
            total_batch = len(data_loader)

            for X, Y in data_loader:
                # reshape input image into [batch_size by 784]
                # label is not one-hot encoded
                X = X.view(-1, 28 * 28).to(device)
                Y = Y.to(device)

                optimizer.zero_grad()
                hypothesis = linear(X)
                cost = criterion(hypothesis, Y)
                cost.backward()
                optimizer.step()

                avg_cost += cost / total_batch

            print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

        print('Learning finished')
```

```
Epoch: 0001 cost = 0.535150588
Epoch: 0002 cost = 0.359577745
Epoch: 0003 cost = 0.331264287
Epoch: 0004 cost = 0.316404700
Epoch: 0005 cost = 0.307106972
Epoch: 0006 cost = 0.300456554
Epoch: 0007 cost = 0.294933408
Epoch: 0008 cost = 0.290956199
Epoch: 0009 cost = 0.287074089
Epoch: 0010 cost = 0.284515619
Epoch: 0011 cost = 0.281914055
Epoch: 0012 cost = 0.279526860
Epoch: 0013 cost = 0.277636588
Epoch: 0014 cost = 0.275874794
Epoch: 0015 cost = 0.274422705
Learning finished
```

MNIST_introduction 실습

In [14]:
```python
# Test the model using test sets
with torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())

    # Get one and predict
    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device)
    Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

    print('Label: ', Y_single_data.item())
    single_prediction = linear(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())

    plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28), cmap='Greys', interpolation='nearest')
    plt.show()
```

```
Accuracy: 0.8883000016212463
Label:  3
Prediction:  3
```



MNIST_backprop 실습

In [1]:
```python
# Lab 10 MNIST and softmax
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
```

In [2]:
```python
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```python
# parameters
learning_rate = 0.5
batch_size = 10
```

In [4]:
```python
# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)
```

In [5]:
```python
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

MNIST_backprop 실습

```
In [6]: w1 = torch.nn.Parameter(torch.Tensor(784, 30)).to(device)
        b1 = torch.nn.Parameter(torch.Tensor(30)).to(device)
        w2 = torch.nn.Parameter(torch.Tensor(30, 10)).to(device)
        b2 = torch.nn.Parameter(torch.Tensor(10)).to(device)
```

입력 뉴런 : 784
은닉층 뉴런 :30
출력층 뉴런 : 10

```
In [7]: torch.nn.init.normal_(w1)
        torch.nn.init.normal_(b1)
        torch.nn.init.normal_(w2)
        torch.nn.init.normal_(b2)

Out[7]: Parameter containing:
        tensor([ 0.3078, -1.9857,  1.0512,  1.5122, -1.0199, -0.7402, -1.3111,  0.6142,
                -0.6474,  0.1758], requires_grad=True)
```

가중치와 편향 값을 정규 분포에서 샘플링하여 초기화

평균 0, 표준편차 1인 정규분포를 사용

```
In [8]: def sigmoid(x):
            #  sigmoid function
            return 1.0 / (1.0 + torch.exp(-x))
            # return torch.div(torch.tensor(1), torch.add(torch.tensor(1.0), torch.exp(-x)))
```

시그모이드 함수 정의
-출력층 활성화 함수로 사용

```
In [9]: def sigmoid_prime(x):
            # derivative of the sigmoid function
            return sigmoid(x) * (1 - sigmoid(x))
```

시그모이드_도함수 정의
-역전파를 통해 기울기 계산할 때 필요

In [10]:
```python
X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)[:1000]
Y_test = mnist_test.test_labels.to(device)[:1000]
i = 0
while not i == 10000:
    for X, Y in data_loader:
        i += 1

        # forward
        X = X.view(-1, 28 * 28).to(device)
        Y = torch.zeros((batch_size, 10)).scatter_(1, Y.unsqueeze(1), 1).to(device)    # one-hot
        l1 = torch.add(torch.matmul(X, w1), b1)
        a1 = sigmoid(l1)
        l2 = torch.add(torch.matmul(a1, w2), b2)
        y_pred = sigmoid(l2)

        diff = y_pred - Y

        # Back prop (chain rule)
        d_l2 = diff * sigmoid_prime(l2)
        d_b2 = d_l2
        d_w2 = torch.matmul(torch.transpose(a1, 0, 1), d_l2)

        d_a1 = torch.matmul(d_l2, torch.transpose(w2, 0, 1))
        d_l1 = d_a1 * sigmoid_prime(l1)
        d_b1 = d_l1
        d_w1 = torch.matmul(torch.transpose(X, 0, 1), d_l1)

        w1 = w1 - learning_rate * d_w1
        b1 = b1 - learning_rate * torch.mean(d_b1, 0)
        w2 = w2 - learning_rate * d_w2
        b2 = b2 - learning_rate * torch.mean(d_b2, 0)

        if i % 1000 == 0:
            l1 = torch.add(torch.matmul(X_test, w1), b1)
            a1 = sigmoid(l1)
            l2 = torch.add(torch.matmul(a1, w2), b2)
            y_pred = sigmoid(l2)
            acct_mat = torch.argmax(y_pred, 1) == Y_test
            acct_res = acct_mat.sum()
            print(acct_res.item())

        if i == 10000:
            break
```

체인 룰에 따라 활성화 함수의 미분이 곱해짐

체인 룰은 Loss 함수의 Gradient를 전달하기 위해 사용됨

1000번째 반복마다 테스트의 정확도 출력값

```
806
854
885
891
888
889
901
908
887
906
```

Weight Initialization

신경망의 학습 과정에서 가중치를 처음 설정하는 방법

Xavier Initialization : 입력 뉴런의 수와 출력 뉴런의 수를 고려해 가중치를 초기화
가중치를 정규분포 또는 균등분포에서 샘플링하여 초기화하는 방법

$$W \sim U\left(-\sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}} + n_{\text{out}}}}\right)$$

균등 분포

$$W \sim N\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right)$$

정규 분포

sigmoid에서 사용

He Initialization : ReLU 및 그 변종(Leaky ReLU 등)을 사용할 때 적합한 초기화 방법

$$W \sim N\left(0, \frac{2}{n_{\text{in}}}\right)$$

정규 분포 초기화

$$W \sim U\left(-\sqrt{\frac{6}{n_{\text{in}}}}, \sqrt{\frac{6}{n_{\text{in}}}}\right)$$

균등 분포 초기화

MNIST_nn_xabier 실습

In [1]:
```python
# Lab 10 MNIST and softmax
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import random
```

In [2]:
```python
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
random.seed(777)
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```python
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

In [4]:
```python
# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)
```

MNIST_nn_xabier 실습

```
In [5]:
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

```
In [6]:
# nn layers
linear1 = torch.nn.Linear(784, 256, bias=True)
linear2 = torch.nn.Linear(256, 256, bias=True)
linear3 = torch.nn.Linear(256, 10, bias=True)
relu = torch.nn.ReLU()
```

```
In [7]:
# xavier initialization
torch.nn.init.xavier_uniform_(linear1.weight)
torch.nn.init.xavier_uniform_(linear2.weight)
torch.nn.init.xavier_uniform_(linear3.weight)
```

```
Out[7]: Parameter containing:
tensor([[-0.0215, -0.0894,  0.0598,  ...,  0.0200,  0.0203,  0.1212],
        [ 0.0078,  0.1378,  0.0920,  ...,  0.0975,  0.1458, -0.0302],
        [ 0.1270, -0.1296,  0.1049,  ...,  0.0124,  0.1173, -0.0901],
        ...,
        [ 0.0661, -0.1025,  0.1437,  ...,  0.0784,  0.0977, -0.0396],
        [ 0.0430, -0.1274, -0.0134,  ..., -0.0582,  0.1201,  0.1479],
        [-0.1433,  0.0200, -0.0568,  ...,  0.0787,  0.0428, -0.0036]],
       requires_grad=True)
```

입력 뉴런 : 784
히든층 뉴런 : 256
히든층 뉴런 : 256
출력층 뉴런 : 10

torch.nn.init.xavier_uniform_() : Xabier 초기화 / 균등 분포를 사용

MNIST_nn_xabier 실습

In [8]:
```python
# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3).to(device)
```

In [9]:
```python
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

In [10]:
```python
total_batch = len(data_loader)
for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning finished')
```

```
Epoch: 0001 cost = 0.249897048
Epoch: 0002 cost = 0.094330102
Epoch: 0003 cost = 0.061055195
Epoch: 0004 cost = 0.042816643
Epoch: 0005 cost = 0.032796543
Epoch: 0006 cost = 0.024419624
Epoch: 0007 cost = 0.020511184
Epoch: 0008 cost = 0.018132176
Epoch: 0009 cost = 0.015536907
Epoch: 0010 cost = 0.016846467
Epoch: 0011 cost = 0.012203062
Epoch: 0012 cost = 0.012871196
Epoch: 0013 cost = 0.011348661
Epoch: 0014 cost = 0.010990168
Epoch: 0015 cost = 0.006201488
Learning finished
```

MNIST_nn_xabier 실습

In [11]:

```
# Test the model using test sets
with torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = model(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())

    # Get one and predict
    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device)
    Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

    print('Label: ', Y_single_data.item())
    single_prediction = model(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())
```

```
Accuracy: 0.9804999828338623
Label:  8
Prediction:  8
```