MNIST with CNN
(with Pytorch)

김재광 교수 (소프트웨어융합대학 글로벌융합학부)

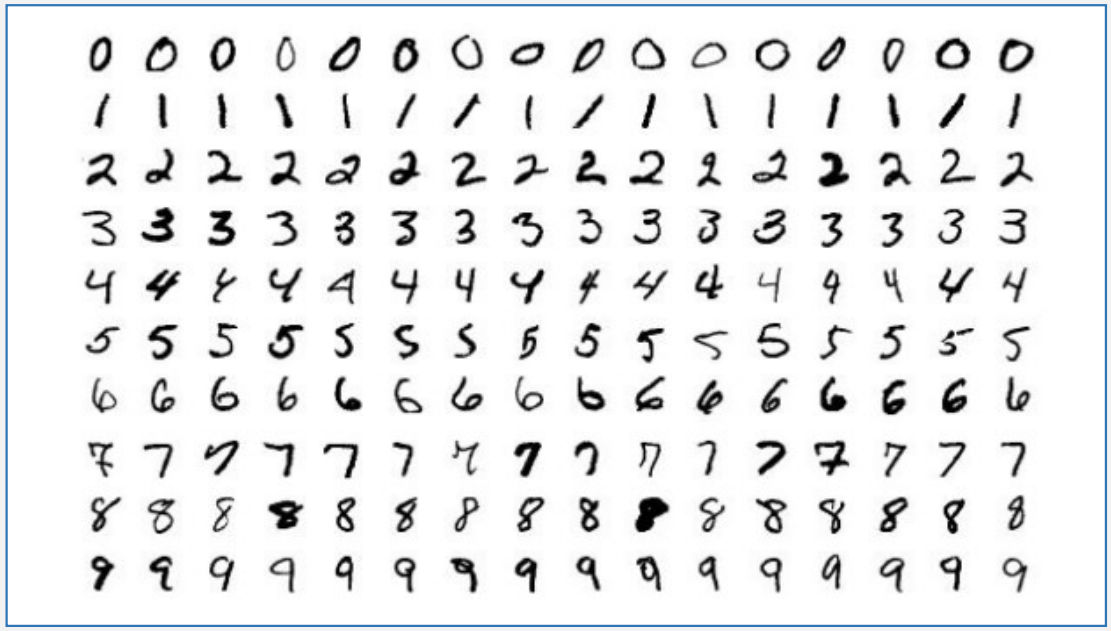
07 MNIST with CNN
- with Pytorch

Contents

- What is MNIST?
- MNIST Data Manipulation
- CNN 학습 단계
- MNIST CNN code

What is MNIST?

- MNIST: handwritten digits dataset

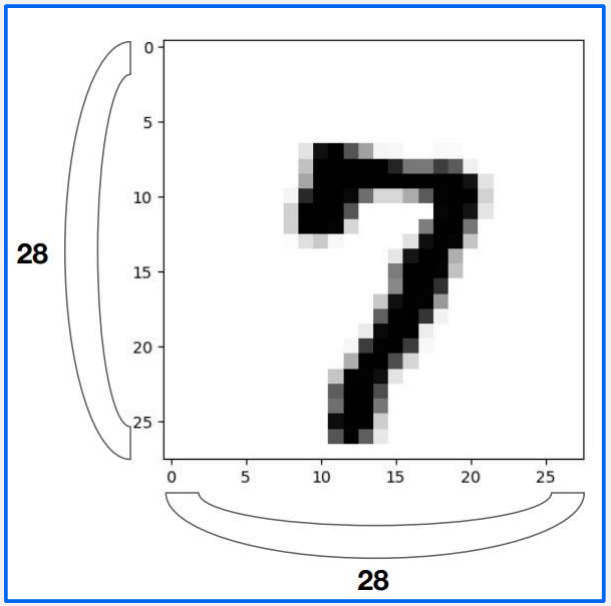


  - train-images-idx3-ubyte.gz: training set images (9912422 bytes)
  - train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
  - t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
  - t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

What is MNIST?(Cont'd)

- **Example of MNIST**
  - 28x28 image
  - 1 channel gray image
  - 0-9 digits

```
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)
```



What is MNIST?(Cont'd)

- **TORCHVISION**
  - The TORCHVISION package consists of popular datasets, model architectures, and common image transformations for computer vision.

- torchvision.datasets
  - MNIST
  - Fashion-MNIST
  - KMNIST
  - EMNIST
  - QMNIST
  - FakeData
  - COCO
  - LSUN
  - ImageFolder
  - DatasetFolder
  - ImageNet
  - CIFAR
  - STL10
  - SVHN
  - PhotoTour
  - SBU
  - Flickr
  - VOC
  - Cityscapes
  - SBD
  - USPS
  - Kinetics-400
  - HMDB51
  - UCF101
  - CelebA
- torchvision.io
  - Video
- torchvision.models
  - Classification
  - Semantic Segmentation
  - Object Detection, Instance Segmentation and Person Keypoint Detection
  - Video classification
- torchvision.ops
- torchvision.transforms
  - Transforms on PIL Image
  - Transforms on torch.*Tensor
  - Conversion Transforms
  - Generic Transforms
  - Functional Transforms
- torchvision.utils

https://pytorch.org/docs/stable/torchvision/index.html

MNIST Data Manipulation

Reading data

```
import torchvision.datasets as dsets
...
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)
data_loader = torch.utils.DataLoader(DataLoader=mnist_train, batch_size=batch_size,shuffle=True, drop_last=True)
...
for epoch in range(training_epochs):
...
    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)
```

MNIST Data Manipulation (Cont'd)

- **Epoch / Batch size / Iteration**
  - One epoch
    - ✓ One forward / backward pass of all the training examples
  - Batch size
    - ✓ # of training examples in one forward / backward pass
    - ✓ The higher the batch size, the more memory space is required
  - # of Iteration
    - ✓ # of passes, each pass using [batch size] # of examples To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes)

ex

❖ if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch

MNIST Data Manipulation (Cont'd)

- **Softmax**

```
# MNIST data image of shape 28 * 28 = 784 Softmax
linear = torch.nn.Linear(784, 10, bias=True).to(device)
# initialization
torch.nn.init.normal_(linear.weight)
# parameters
training_epochs = 15
batch_size = 100
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device) # Softmax is internally computed.
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)
```

MNIST Data Manipulation (Cont'd)

- Softmax (cont'd)

```
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)
        optimier.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        avg_cost += cost / total_batch

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

MNIST Data Manipulation (Cont'd)

- **Softmax (cont'd)**

```
Epoch: 0001 cost = 2.511683702
Epoch: 0002 cost = 0.977319956
Epoch: 0003 cost = 0.797017217
Epoch: 0004 cost = 0.710427940
Epoch: 0005 cost = 0.655205429
Epoch: 0006 cost = 0.615207732
Epoch: 0007 cost = 0.584421575
Epoch: 0008 cost = 0.559486568
Epoch: 0009 cost = 0.538655698
Epoch: 0010 cost = 0.520880997
Epoch: 0011 cost = 0.505315244
Epoch: 0012 cost = 0.491431117
Epoch: 0013 cost = 0.479477882
Epoch: 0014 cost = 0.468681127
Epoch: 0015 cost = 0.458788306
Learning finished
Accuracy: 0.8718999624252319
```

MNIST Data Manipulation (Cont'd)

- **Test**

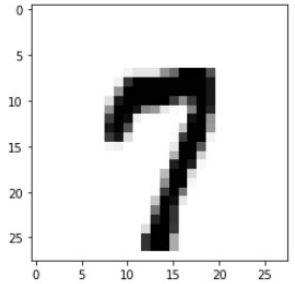
```
# Test the model using test sets
With torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)
    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print("Accuracy: ", accuracy.item())
```

MNIST Data Manipulation (Cont'd)

- Visualization

```
import matplotlib.pyplot as plt
import random
...
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: 0.8748999834060669
Label:  7
Prediction:  7
/usr/local/lib/python3.6/dist-packages/t
  warnings.warn("test_data has been renar
/usr/local/lib/python3.6/dist-packages/t
  warnings.warn("test_labels has been re
```



CNN 학습 단계

- 라이브러리를 가져오기
- GPU 사용 설정 및 random seed 설정
- 학습에 필요한 parameter 설정
  - lr, training epochs, batch_size, …
- 데이터 셋 로드
- 학습 모델 생성
- Loss function 및 optimizer 설정
- Loss 확인을 통해 모델 학습
- 성능 확인

MNIST CNN Code

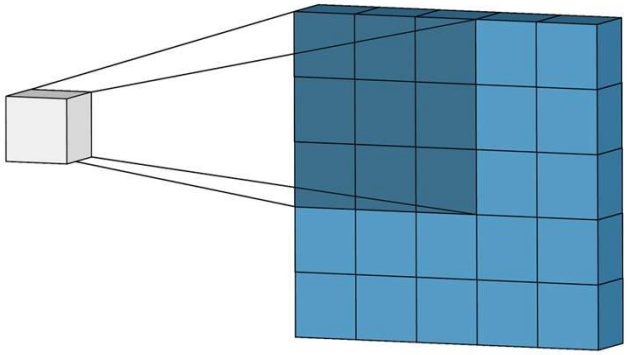
```
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn.init

device = 'cuda' if torch.cuda.is_available() else 'cpu'

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

MNIST CNN Code (Cont'd)

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# 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)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

MNIST CNN Code (Cont'd)

```
# CNN Model (2 conv layers)
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform_(self.fc.weight)
```



MNIST CNN Code (Cont'd)

```
def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)   # Flatten them for FC
        out = self.fc(out)
        return out
```

```
# instantiate CNN model
model = CNN().to(device)

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

MNIST CNN Code (Cont'd)

```
# train my model
total_batch = len(data_loader)
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    for X, Y in data_loader:
        # image is already size of (28x28), no reshape
        # label is not one-hot encoded
        X = X.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: {:>4}] cost = {:>.9}'.format(epoch + 1, avg_cost))
print('Learning Finished!')
```

```
Learning started. It takes sometime.
[Epoch:    1] cost = 0.159776822
[Epoch:    2] cost = 0.0413324498
[Epoch:    3] cost = 0.0292899609
[Epoch:    4] cost = 0.02193336
[Epoch:    5] cost = 0.0172540527
[Epoch:    6] cost = 0.0135775162
[Epoch:    7] cost = 0.0111167822
[Epoch:    8] cost = 0.0123288101
[Epoch:    9] cost = 0.0087507898
[Epoch:   10] cost = 0.0086597465
[Epoch:   11] cost = 0.00723933568
[Epoch:   12] cost = 0.00641156221
[Epoch:   13] cost = 0.00667931139
[Epoch:   14] cost = 0.00576675031
[Epoch:   15] cost = 0.00654875953
Learning Finished!
```

MNIST CNN Code (Cont'd)

```
# Test model and check accuracy
with torch.no_grad():
    X_test = mnist_test.test_data.view(len(mnist_test), 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())
```

```
Accuracy: 0.9869999885559082
```