PyTorch로 배우는 딥러닝 입문 (2) - 실습

이영완

한국전자통신연구원

Contents

- Image classification on MNIST dataset
- Image classification on CIFAR-10 dataset

Contents

- Image classification on **MNIST** dataset
- Image classification on **CIFAR-10** dataset

지금도 ICML, NeurIPS 등 Major Machine Learning 학회에서 많이 쓰임

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

1x28x28

train-images-idx3-ubyte.gz: training set images (9912422 bytes; 60,000 samples)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
T10k-images-idx3-ubyte.gz : test set images (1648877 bytes; 10,000 samples)
T10k-labels-idx1-ubyte.gz : test set labels (4542 bytes)

http://yann.lecun.com/exdb/mnist/

https://www.youtube.com/watch?v=FwFduRA_L6Q

TORCHVISION.DATASETS

All datasets are subclasses of `torch.utils.data.Dataset` i.e, they have `__getitem__` and `__len__` methods implemented. Hence, they can all be passed to a `torch.utils.data.DataLoader` which can load multiple samples parallelly using `torch.multiprocessing` workers. For example:

```
imagenet_data = torchvision.datasets.ImageNet('path/to/imagenet_root/')
data_loader = torch.utils.data.DataLoader(imagenet_data,
                                          batch_size=4,
                                          shuffle=True,
                                          num_workers=args.nThreads)
```

Dataloader 중요합니다!! 기억합시다 Dataloader!!

Torchvision에서는 다양한 종류의 데이터셋을 다루는 Dataloader를 제공

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

NN Training terminology

- **Epoch** : one forward pass and one backward pass of **all** the training examples
- **Batch size** : The number of training samples in one forward/backward pass.
  The higher the batch size, the more memory space you'll need.
- **Iterations** : number of passes, each pass using [batch size] number of examples.
  one pass = one forward pass + one backward pass

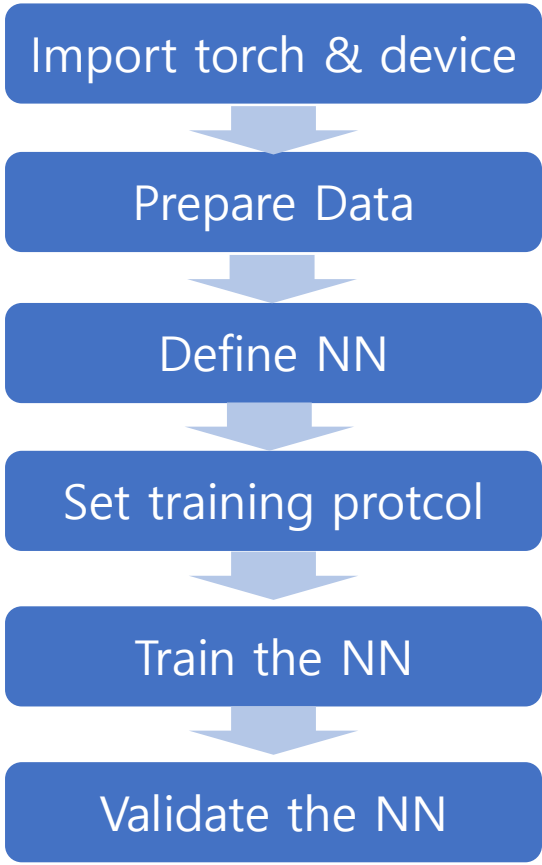
e.g., MNIST dataset

Total training samples : 60k images
Batch size : 10 samples
Iterations : 6k iters.

→ 1 epoch : training(forward/backward) with batch size of 10 during 6k iterations

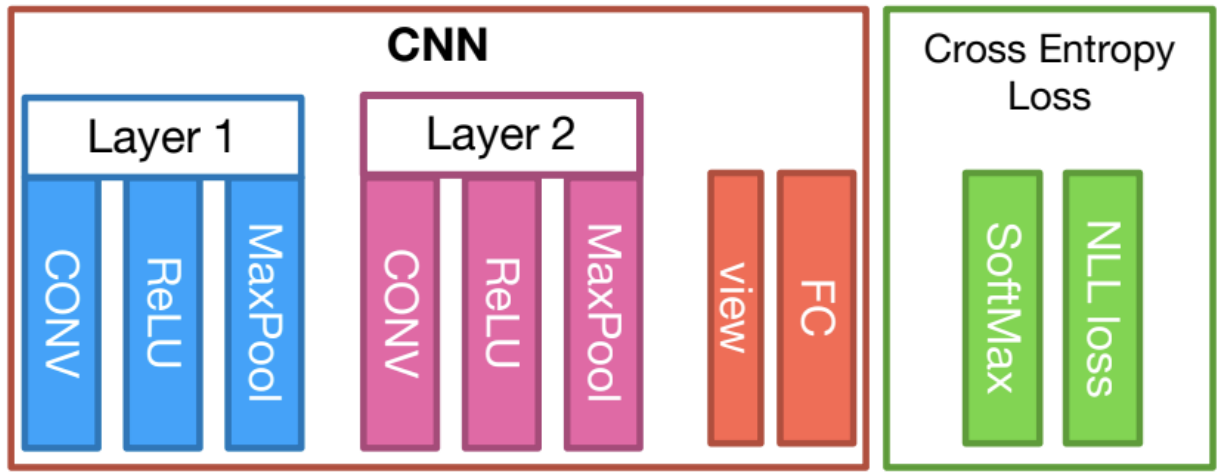
Training procedure

1. Prepare Data with Dataloader
2. Define my Neural Network
3. Set training protocol
   - Loss function,
   - Optimizer,
   - Learning_rate, Learning_rate scheduler
   - Batch_size, Epoch
4. Training my NN
5. Validation my NN



우리가 만들 CNN 구조 확인!

1x28x28



(Layer 1) Convolution layer = (in_c=1, out_c=32,kernel_size =3, stride=1,padding=1)
(Layer 1) MaxPool layer = (kernel_size=2, stride =2)

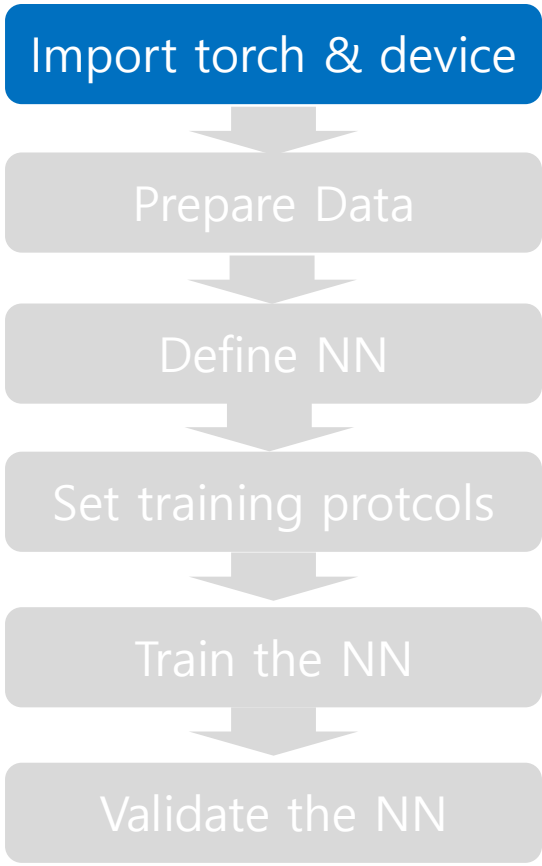
(Layer 2) Convolution layer = (in_c=32, out_c=64, kernel_size =3, stride=1,padding=1)
(Layer 2) MaxPool layer = (kernel_size=2, stride =2)

view => (batch_size x [7,7,64] => batch_size x [3136])
Fully_Connect layer => (input=3136, output = 10)

Import torch & Device check

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

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print(f'avaiable device : %s' % device)
# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

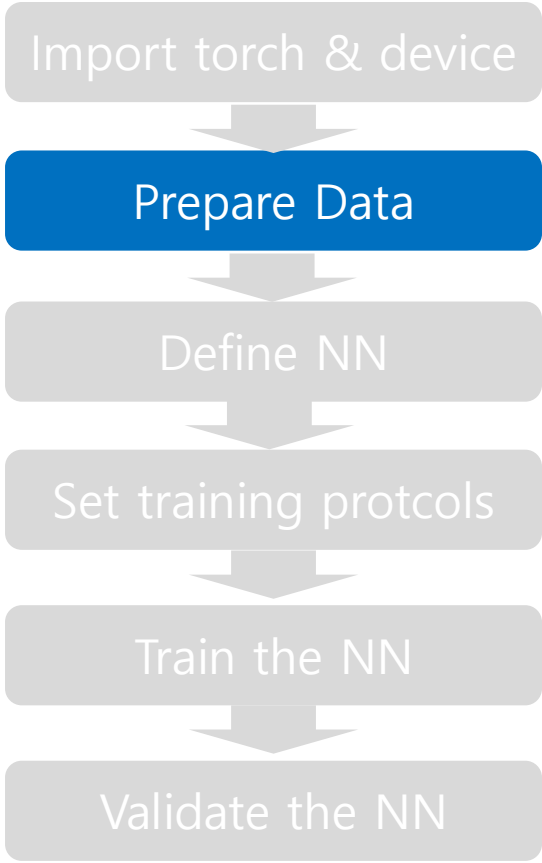


Data 준비 with Dataloader

```
# MNIST dataset
mnist_train = torchvision.datasets.MNIST(root='MNIST_data/',
                                         train=True,
                                         transform=transforms.ToTensor(),
                                         download=True)
mnist_test = torchvision.datasets.MNIST(root='MNIST_data/',
                                        train=False,
                                        transform=transforms.ToTensor(),
                                        download=True)
```

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

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



Define my Neural Network

- Using torch.nn.Module

Define your NN

상속

```
class myNN(torch.nn.Module):

    def __init__(self):
        super(myNN, self).__init__()
        ...   네트워크 정의

    def forward(self, x):
            네트워크 실행
        ...

        return out
```

Define my Neural Network

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

    def __init__(self):
        super(myNN, self).__init__()
        # Torch tensor dim. (bath_size, C, H, W)
        # L1 ImgIn shape=(batch_size, 1,  28, 28)
        #    Conv     -> (batch_size, 32, 28, 28)
        #    Relu     -> (batch_size, 32, 28, 28)
        #    Pool     -> (batch_size, 32, 14, 14)
        self.conv1 = torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        self.relu1 = torch.nn.ReLU()
        self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2)

        # L2 ImgIn shape=(batch_size, 32, 14, 14)
        #    Conv      ->(batch_size, 64, 14, 14)
        #    Relu      ->(batch_size, 64, 14, 14)
        #    Pool      ->(batch_size, 64,  7,  7)
        self.conv2 = torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.relu2 = torch.nn.ReLU()
        self.pool2 = 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)
```

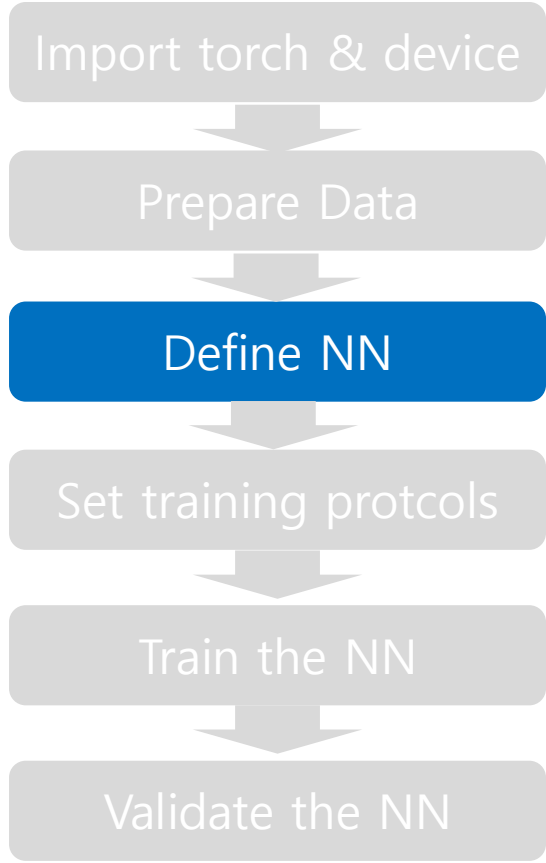
네트워크 정의
(__init__(...))

- Import torch & device
- Prepare Data
- **Define NN**
- Set training protcols
- Train the NN
- Validate the NN

Define my Neural Network

```
def forward(self, x):

    out = self.conv1(x)
    out = self.relu1(out)
    out = self.pool1(out)
    #out = self.pool1(self.relu1(self.conv1(x)))

    out = self.conv2(out)
    out = self.relu2(out)
    out = self.pool2(out)
    #out = self.pool2(self.relu2(self.conv2(x)))
    # dim : (batch_size, 64, 7, 7)

    # Flatten them for FC --> dim : (batch_size, 7*7*64)
    out = out.view(out.size(0), -1)
    out = self.fc(out) # dim : (batch_size, 10)
    return out
```

네트워크 실행
(forward(...))

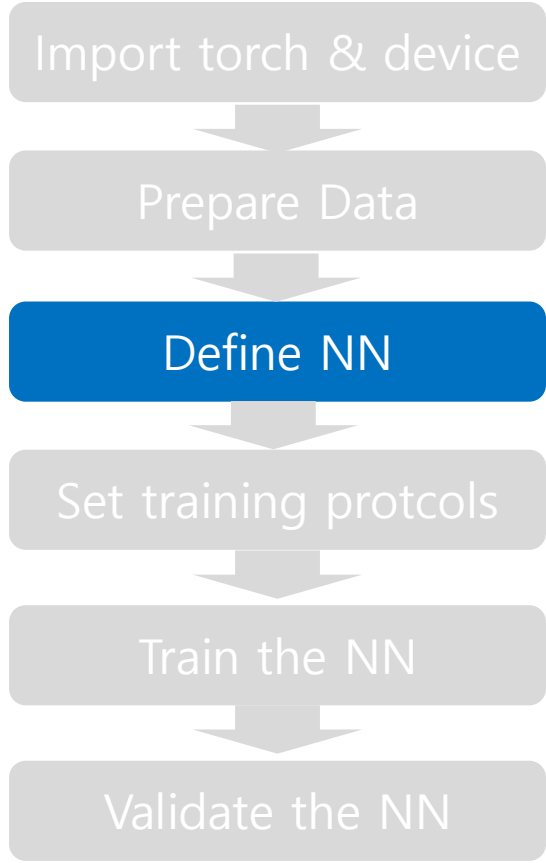


Define my Neural Network

```
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # Torch tensor dim. (bath_size, C, H, W)
        # L1 ImgIn shape=(batch_size, 1,  28, 28)
        #    Conv     -> (batch_size, 32, 28, 28)
        #    Relu     -> (batch_size, 32, 28, 28)
        #    Pool     -> (batch_size, 32, 14, 14)
        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=(batch_size, 32, 14, 14)
        #    Conv      ->(batch_size, 64, 14, 14)
        #    Relu      ->(batch_size, 64, 14, 14)
        #    Pool      ->(batch_size, 64,  7,  7)
        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)

    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
```

- torch.nn.Sequential()

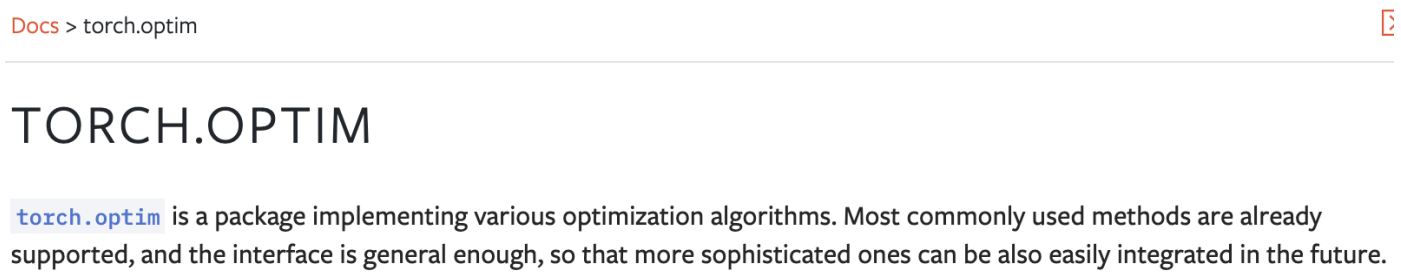
Forward 단계에서 피처맵을 조작하거나 피처맵의 정보를 꺼내올 필요없이 단순히 sequential한 작업을 하고 싶을 때 유용!!



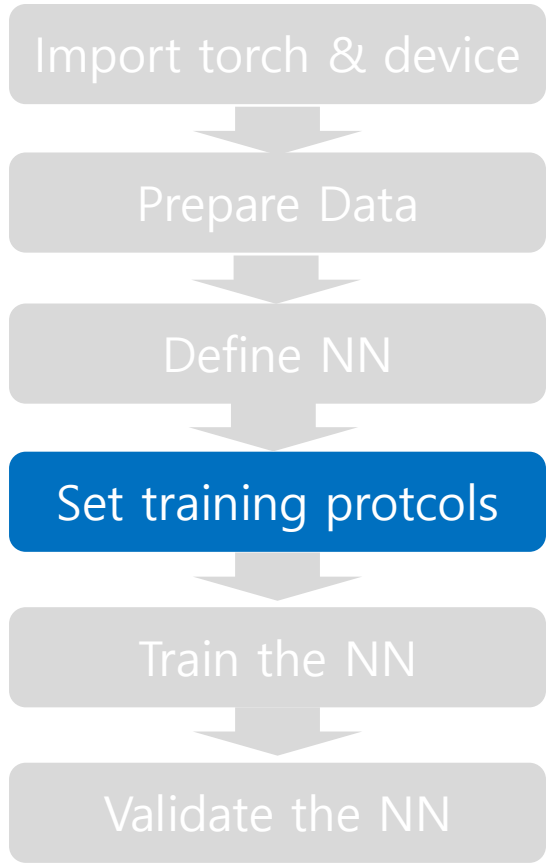
Set training protocols

- Define(select) loss / optimizer
  - Loss : Entropy, BCELoss, NLLLoss, SmoothL1Loss…

  - Optimizer : SGD, ADAM, RmsProp…



https://pytorch.org/docs/stable/_modules/torch/nn/modules/loss.html
https://pytorch.org/docs/stable/optim.html



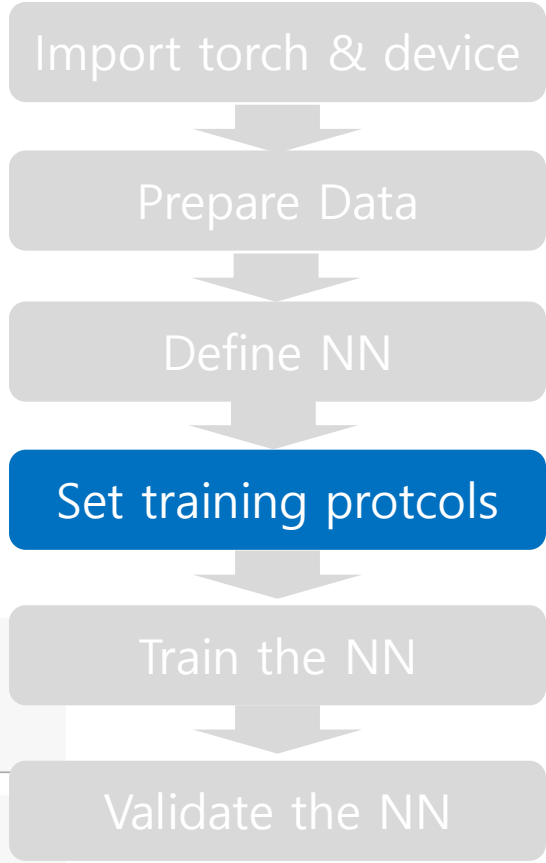
Set training protocols

- Define(select) loss / optimizer

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

myNN 모델 GPU에 올리기

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



Train myNN

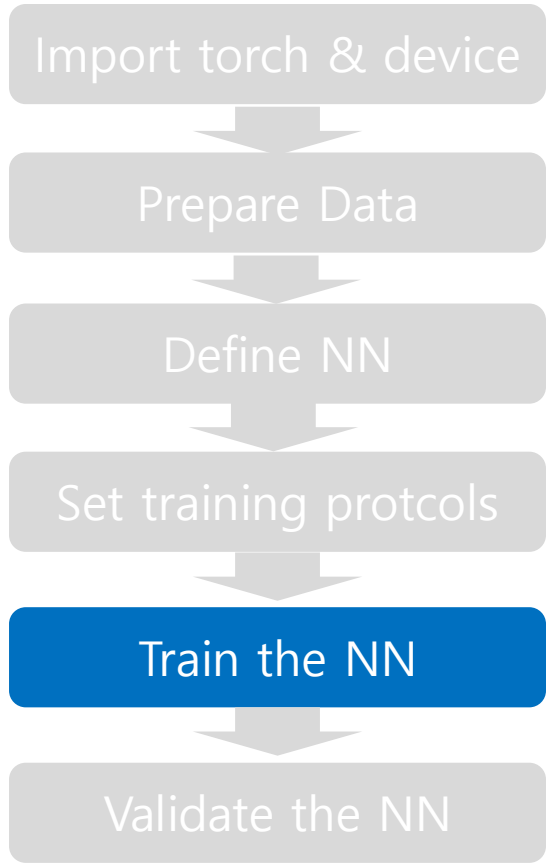
```
# paramters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

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

```
# train myNN
total_batch = len(train_loader)
print('Learning started.')

for epoch in range(training_epochs):
    running_loss = 0.0
    for data, labels in train_loader: 지정된 batch size만큼 iterative하게 data를 내보내줌
        # image is already size of (1x28x28), no reshape
        # label is not one-hot encoded

        #load the data into the device(GPU)
        data = data.to(device)
        labels = labels.to(device)

        optimizer.zero_grad() # set gradients to zero
        outputs = model(data) # inference the myNN
        loss = criterion(outputs, labels) #compute the loss
        loss.backward() #back prop
        optimizer.step() #weight update

        running_loss += loss.item() / total_batch

    print('[Epoch: {:>4}] loss = {:>.9}'.format(epoch + 1, running_loss))

print('Learning Fishied.')
```

output

```
Learning started.
[Epoch:    1] loss = 2.1168668
[Epoch:    2] loss = 1.5206627
[Epoch:    3] loss = 0.855583688
[Epoch:    4] loss = 0.580259988
[Epoch:    5] loss = 0.472125877
Learning Fishied.
```



Validate myNN

```
correct = 0
total = 0
model.eval()
with torch.no_grad():
    for images, labels in test_loader:

        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of myNN on the test set: %d %% ' % (100 * correct/total))
```

test mode일 때 grad 추적하지 않도록

(max_value, indice)

정답 맞춘 개수 카운트

https://pytorch.org/docs/stable/torch.html#torch.max