

이제는 각 파트별로 소스코드들이 무엇을 의미하는지를 알아보도록 하겠습니다.

DNN Model.

```
1 class Net(nn.Module):
2     def __init__(self):
3         super(Net, self).__init__()
4         self.fc1 = nn.Linear(784, 512)
5         self.fc2 = nn.Linear(512, 256)
6         self.fc3 = nn.Linear(256, 128)
7         self.fc4 = nn.Linear(128, 64)
8         self.fc5 = nn.Linear(64, 32)
9         self.fc6 = nn.Linear(32, 10)
10    def forward(self, x):
11        x = x.float()  $\Rightarrow$  x를 실수형으로 변환.
12        h1 = F.relu(self.fc1(x.view(-1, 784)))  $\rightarrow$  fc1 행렬곱하고 relu로 비선형성 추가.
13        h2 = F.relu(self.fc2(h1))
14        h3 = F.relu(self.fc3(h2))
15        h4 = F.relu(self.fc4(h3))
16        h5 = F.relu(self.fc5(h4))
17        h6 = self.fc6(h5)
18        return F.log_softmax(h6, dim=1)
```

Handwritten annotations:

- Line 1: '행렬'
- Line 2: 'input'
- Line 3: 'output'
- Line 10: 'Module에 학습과 관련된 함수'
- Line 12: 'x를 실수형으로 변환.'
- Line 12: 'fc1 행렬곱하고 relu로 비선형성 추가.'

[`x.view(-1, 784)`]

Tensor의 모양을 바꾸는데 사용. 여기서는 28x28니까 1x784로 변환. (-1이면 자동으로 바꿔줌)

Relu() - 노드에 입력된 값들을 비선형 함수에 통과시킨 후 다음 레이어로 전달하는 함수는 활성화 함수(Activation Function)라고 한다.

Log_softmax() - elements들의 00에서 1사이의 확률을 갖게되고, 그 합은 10이 된다. softmax함수에 log를 취한 것.

```
1 batch_size = 64
2 test_batch_size = 1000
3 epochs = 10
4 lr = 0.01
5 momentum = 0.5
6 no_cuda = True
7 seed = 1
8 log_interval = 200
9
10 use_cuda = not no_cuda and torch.cuda.is_available()
11
12 torch.manual_seed(seed)
13
14 device = torch.device("cuda" if use_cuda else "cpu")
15
16 kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}
17
18 print("set vars and device done")
```

위의 parameter들을보통 딥러닝 모델을 train할 때, 많이 쓰이는 변수들. 딥러닝 모델에 큰 영향을 주는 parameter는 hyper parameter라고한다.

batch size - cpu 혹은 gpu에 몇 개씩의 데이터를 넣어줄 것인지. 이 개수에 따라 딥러닝 모델의 성능이 크게 달라짐.
epochs - training data를 1번씩 모두 셨을 때까지를 1epochs가 지났다고 한다.

```
1 #Prepare Data Loader for Training and Validation
2
3 transform = transforms.Compose([
4     transforms.ToTensor(),
5     transforms.Normalize((0.1307,), (0.3081,))])
6
7 train_loader = torch.utils.data.DataLoader(
8     datasets.MNIST('../data', train=True, download=True,
9                     transform=transform),
10    batch_size=batch_size, shuffle=True, **kwargs) } training set
11
12 test_loader = torch.utils.data.DataLoader(
13     datasets.MNIST('../data', train=False, download=True,
14                     transform=transform),
15    batch_size=test_batch_size, shuffle=True, **kwargs) } test set
```

Transform을 통해서 데이터를 어떻게 처리해줄지 결정

이미지 데이터를 .ToTensor()을 통해서 tensor형태로 데이터를 변환해준 뒤 Normalize과정을 해주기 위해 standard deviation와 variation 값을 직접 입력(미리 계산해논거) torchvision 함수 자체가 'train'이라는 파라미터를 통해 training set과 test set을 쉽게 준비할 수 있도록 설계해놨기 때문에 그대로 사용하면 됨.

```
1 model = Net().to(device)
2 optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
```

optimizer를 통해 모델이 어떤 방식으로 training할지를 고른다. 위의 코드는 SGD와 momentum을 사용하는 optimizer를 설정해준 뒤, model내의 parameter들을 training하도록 하였음.

```
2
3 def train(log_interval, model, device, train_loader, optimizer, epoch):
4     model.train()
5     for batch_idx, (data, target) in enumerate(train_loader):
6         data, target = data.to(device), target.to(device) # send data to
7         optimizer.zero_grad() # set gradient zero.
8         output = model(data) # get output from model
9         loss = F.nll_loss(output, target) # calculate nll loss
10        loss.backward() # do backpropagation
11        optimizer.step() # update weight and biases
12        if batch_idx % log_interval == 0:
13            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
14                epoch, batch_idx * len(data), len(train_loader.dataset),
15                100. * batch_idx / len(train_loader), loss.item()))
16
17 def test(log_interval, model, device, test_loader):
18     model.eval()
19     test_loss = 0
20     correct = 0
21     with torch.no_grad():
22         for data, target in test_loader:
23             data, target = data.to(device), target.to(device)
24             output = model(data)
25             test_loss += F.nll_loss(output, target, reduction='sum').item()
26             pred = output.argmax(dim=1, keepdim=True)
27             correct += pred.eq(target.view_as(pred)).sum().item()
28
29     test_loss /= len(test_loader.dataset)
30
31     print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%) \n'
32          .format(test_loss, correct, len(test_loader.dataset),
33                  100. * correct / len(test_loader.dataset)))
```

train 함수를 통해서 train data들을 한번씩 살피고 test 함수를 통해서 test data를 통해 모델의 성능을 평가.

Make machine works

```
1 # Train and Test the model and save it.
2
3 for epoch in range(1, 11):
4     train(log_interval, model, device, train_loader, optimizer, epoch)
5     test(log_interval, model, device, test_loader)
6     torch.save(model, './model.pt')
```

train함수와 test함수를 통해 model을 학습시키고 모든 epoch이 끝나게 되면 torch.save를 통해 모델을 저장.