📌Q1. 가장 먼저 학습 데이터를 준비해보도록 하겠습니다. MNIST 데이터셋을 직접 Load해 봅시다. 데이터셋을 로드하고 DataLoader를 구현해보세요.

핵심: 학습데이터를 준비하는 단계입니다. **MNIST 데이터셋을 로드**하고 이를 활용해 **DataLoader을 구현**하는 것이 목표입니다.

1. **MNIST 데이터셋 로드**: MNIST 데이터셋의 train/test 데이터셋 객체를 만듭니다. 각 파라미터가 의미하는 바는 다음과 같습니다.

- train: 테스트용 데이터를 가져올지 학습용 데이터를 가져올지 표시한다
- transform: transforms.ToTensor()를 넣어서 일반 이미지를 pytorch에 적합한 tensor로 변환한다
- download: True로 설정하면, root에 MNIST 데이터가 없을시 다운을 받는다.

```
root = './data' # 데이터셋의 경로
mnist_train = dset.MNIST(root=root, train=True,
                         transform=transforms.ToTensor(), download=True)
mnist_test = dset.MNIST(root=root, train=False,
                        transform=transforms.ToTensor(), download=True)
```

2. DataLoader 구현: 파이토치의 DataLoader 메소드를 활용해 MNIST데이터셋을 학습할 준비를 합니다. train_loader 및 test_loader 객체를 만드는데, 파라미터 값으로 들어가는 값은 다음과 같다.

- dataset: 사용할 데이터셋
- batch_size: 배치 크기
- shuffle: 학습할 때 데이터를 섞을지의 유무

```
train_loader = DataLoader(mnist_train, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(mnist_test, batch_size=batch_size, shuffle=False)
```

※Batch_size는 앞서 100으로 할당했음.

소스코드:

```
training_epochs = 15  # training 반복 횟수 / 에폭
batch_size = 100 # 몇 개의 샘플로 가중치를 갱신할 것인지 설정 / 배치크기

root = './data'
mnist_train = dset.MNIST(root=root, train=True,
                         transform=transforms.ToTensor(), download=True)
mnist_test = dset.MNIST(root=root, train=False,
                        transform=transforms.ToTensor(), download=True)

train_loader = DataLoader(mnist_train, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(mnist_test, batch_size=batch_size, shuffle=False)
```

📌Q2.데이터가 준비가 되었다면, 이제 그 데이터를 학습할 모델을 구현할 차례입니다. 그후 모델 안의 가중치를 초기화시켜보세요. 입력 데이터 형태에 맞도록 linear한 모델을 구성해보세요.

핵심: 데이터를 학습할 모델을 구현하는 단계입니다. **선형(linear)모델을 구성** 및 내부의 **가중치를 초기화** 하는 것이 목표입니다.

1. **선형(linear) 모델 구성:** 파이토치의 Linear메소드를 활용해서 선형 모델을 구성합니다. 그 전에 모델이 활용할 디바이스(GPU/CPU)를 결정합니다. 파라미터 들어가는 조건문에서 cuda(GPU)를 활용할 수 있으면 활용하되, 할 수 없으면 CPU를 활용하도록 합니다.

   ※기계학습에 있어서 GPU를 활용하는 것이 성능이 월등히 좋습니다.

   선형 모델을 구성할 때 사용되는 파라미터 값은 다음과 같습니다.
   - 입력 차원: 모델의 입력으로 들어가는 데이터의 차원을 결정합니다. MNIST 이미지 입력의 크기는 **28x28=784**입니다.
   - 출력: MNIST 데이터셋의 라벨을 0-9까지 총 10개입니다.
   - bias: True를 할당하게 되면 가중치 외에도 편향값을 사용할 수 있습니다.

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
linear = torch.nn.Linear(784, 10, bias=True).to(device)
```

```
print(torch.cuda.is_available())
# False
```

※GPU사용가능 여부는 위와 같은 코드로 확인할 수 있습니다. 결과 값은 True/False로 나오게 됩니다.

2. **가중치 초기화:** 정의한 선형 모델의 가중치를 초기화합니다. 파이토치의 init메소드를 활용합니다. 파라미터로는 선형모델의 가중치를 넣어줍니다.

```
torch.nn.init.normal_(linear.weight)
```

소스코드:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
linear = torch.nn.Linear(784, 10, bias=True).to(device)

torch.nn.init.normal_(linear.weight)

print(torch.cuda.is_available())
```

📌Q3. 위에서 구현한 모델을 학습시키기 위해서는 loss 함수와 opitmizer가 필요합니다. 아래 제시된 loss 함수와 optimizer를 구현해보세요. Loss 함수와 optimizer는 모델 안의 가중치를 업데이트 할 때 사용됩니다.

핵심: **손실(loss) 함수**와 **optimizer**를 구현하는 것이 목표입니다.

1. **손실(loss) 함수:** 손실함수는 인공지능 모델이 예측한 답과 원래 정답의 오차를 표현, 판단하는 함수를 말합니다. 손실함수는 CrossEntropyLoss를 활용해 정의합니다.

2. 옵티마이저(optimizer): 가중치를 업데이트하기 위한 옵티마이저는 SGD를 활용합니다. 파라미터값으로 다음이 들어갑니다.

- parameters: 옵티마이저가 학습을 통해 업데이트할 모델의 가중치 파라미터를 의미합니다.
- lr(learning rate): learing rate는 옵티바이저의 기울기에 곱해지는 스칼라 값으로, 이값을 통해 다음 학습 지점을 결정하게 된다.

소스코드:

```
criterion = torch.nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)
```

📌Q4. 3번 문제까지 해결하셨다면, 이제 학습을 위한 준비는 거의 끝났다고 볼 수 있습니다. 위 구현 함수들을 이용해 학습 Loop를 구현해보세요.

핵심: 학습의 마지막 단계입니다. 1-4번에서 정의한 모델을 옵티마지저를 활용해 **학습**하는 것이 목표입니다.

1. 모델 학습: 루프를 활용해 정해진 epoch동안, 일정 크기의 batch들을 이용해 학습합니다. 과정을 다음과 같이 정리할 수 있습니다.

A. 학습할 데이터/라벨 불러오기:

imgs, labels = imgs.to(device), labels.to(device)

B. 데이터(이미지)의 크기를 모델의 입력으로 들어갈 수 있도록 전처리하기:

imgs = imgs.view(-1, 28*28)

C. 모델을 활용한 예측값 구하기 및 loss값 계산하기

outputs = linear(imgs)
loss = criterion(outputs, labels)

D. 옵티마이저를 활용해 가중치 업데이트하기(gradient 값 초기화/ gradient 값 갱신/ parameters 값 갱신)

optimizer.zero_grad()
loss.backward()
optimizer.step()

E. 모델 정확도 확인하기

_, argmax = torch.max(outputs, 1) # 예측값(가장 큰 확률을 가진 값)
accuracy = (labels == argmax).float().mean() # 정확도 계산

F. 100번마다 epoch/스텝/loss/정확도 출력하기

if(i+1) % 100 == 0:
print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Accuracy: {:.2f}%'.format(
epoch+1, training_epochs, i+1, len(train_loader), loss.item(), accuracy.item()*100))

소스코드:

```
for epoch in range(training_epochs): #지정한 epoch동안 반복
    for i, (imgs, labels) in enumerate(train_loader): # 각 배치의 번호/데이터/라벨을 가져옴
        imgs, labels = imgs.to(device), labels.to(device)
        imgs = imgs.view(-1, 28*28) # 정의한 모델의 입력으로 들어갈 수 있도록 데이터의 모양을 전처리, 모델의 입력의 크기과 동일하게.

        outputs = linear(imgs)
        loss = criterion(outputs, labels) # 손실함수를 불러와서 loss값 계산

        #옵티마이저를 활용해 가중치 값 업데이트
        optimizer.zero_grad() #gradient 값 초기화
        loss.backward() # gradient 값 갱신
        optimizer.step() # parameters 값 갱신

        _, argmax = torch.max(outputs, 1) # 예측값(가장 큰 확률을 가진 값)
        accuracy = (labels == argmax).float().mean() # 정확도 계산

        if(i+1) % 100 == 0: # 100번마다 진행과정 출력
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Accuracy: {:.2f}%'.format(
                epoch+1, training_epochs, i+1, len(train_loader), loss.item(), accuracy.item()*100))
```

📌Q5. 학습이 완료되면, 모델이 잘 동작하는지 테스트가 필요합니다. 데이터로드 파트에서 준비했던 테스트 데이터를 이용해 테스트를 진행해봅시다. 아래 테스트 코드를 완성해보세요.

핵심: 마지막으로 모델이 학습이 잘 되었는지 확인하기 위해 테스트 데이터셋을 활용한 **테스트**를 하는 것이 목표입니다.

1. **모델의 성능을 테스트:** 모델의 성능을 확인하기 위해 이번에는 test_loader에 있는 데이터셋을 불러와 성능 테스트를 해봅니다. 해당 데이터셋에 있는 정답(레이블)과 모델의 예측값을 비교해서 만일, 정답(labels == argmax)이면 correct변수를 업데이트 한 후, 마지막에 평균울 내서 모델의 정확도를 판단합니다.

- linear.eval() / with torch.no_grad():

with torch.no_grad()는 파이토치의 autograd 엔진을 꺼버리는 기능을 합니다. 그 말은 즉, gradient를 트래킹하지 않는다는 의미인데, 학습을 하지 않으므로 gradient를 업데이트할 필요가 없으므로 포함시켜주는게 연산 속도의 증가에 좋습니다. 또한, linear.eval()는 테스트 과정에서 사용하지 않아야하는 layer들을 꺼저리도록 하는 함수입니다.

※테스트를 할 때 연산 성능향상을 위해서, 통상적으로 많이 포함시켜주는 메소드들이라고 합니다.

소스코드:

```
linear.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for i, (imgs, labels) in enumerate(test_loader):
        imgs, labels = imgs.to(device), labels.to(device)
        imgs = imgs.view(-1, 28*28)
        outputs = linear(imgs)
        _, argmax = torch.max(outputs, 1)
        total += imgs.size(0)
        correct += (labels == argmax).sum().item()

    print('Test accuracy for {} images: {:.2f}%'.format(total, correct/total*100))
```

마지막 미션입니다. 모두 고생하셨습니다.