AutoEncoder

AILAB
Hanyang Univ.

오늘 실습 내용

1. AutoEncoder 구현
2. Denoising AutoEncoder 구현
3. Stacked AutoEncoder 구현

오늘 실습 내용

1. AutoEncoder 구현
2. Denoising AutoEncoder 구현
3. Stacked AutoEncoder 구현

1 데이터 전처리

2 모델 및 optimizer 정의

3 학습

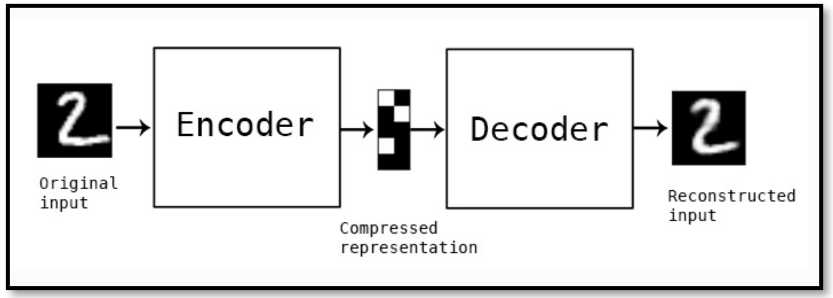
4 학습 결과 확인

AutoEncoder

- Data Compression
  - 데이터 압축
- Data Visualization
  - 데이터 가시화
- Curse of dimensionality
  - 차원의 저주 해결
- Discovering most important features
  - 가장 중요한 피쳐 찾기

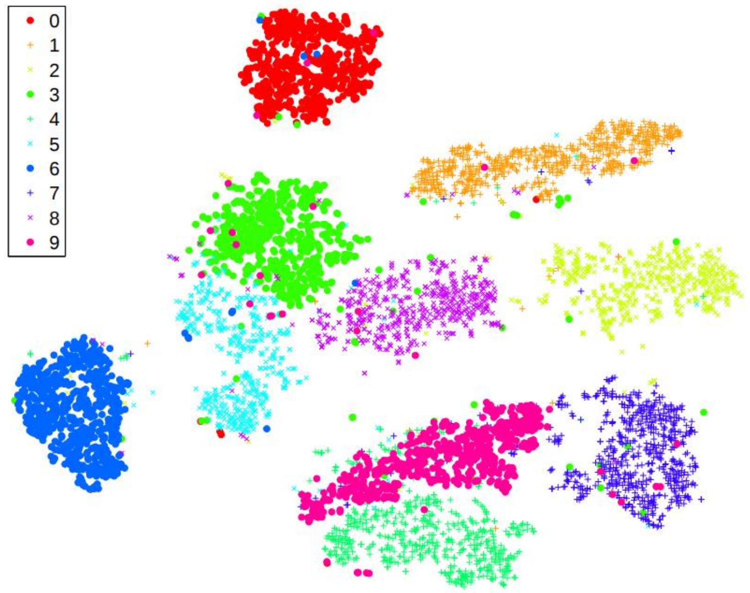
Why AutoEncoder?

- Data Compression
  - 데이터 압축
- Data Visualization
  - 데이터 가시화
- Curse of dimensionality
  - 차원의 저주 해결
- Discovering most important features
  - 가장 중요한 피쳐 찾기



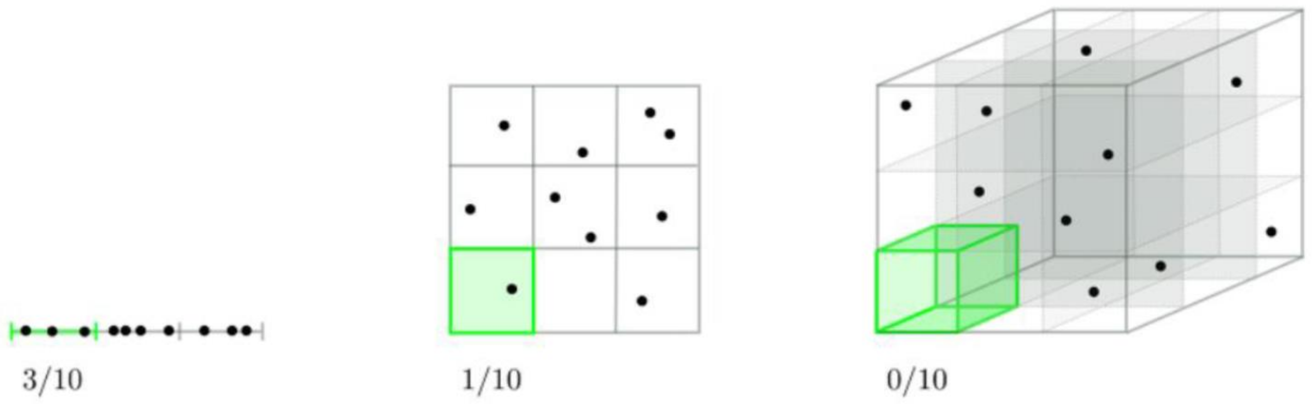
Why AutoEncoder?

- Data Compression
  - 데이터 압축
- Data Visualization
  - 데이터 가시화
- Curse of dimensionality
  - 차원의 저주 해결
- Discovering most important features
  - 가장 중요한 피쳐 찾기



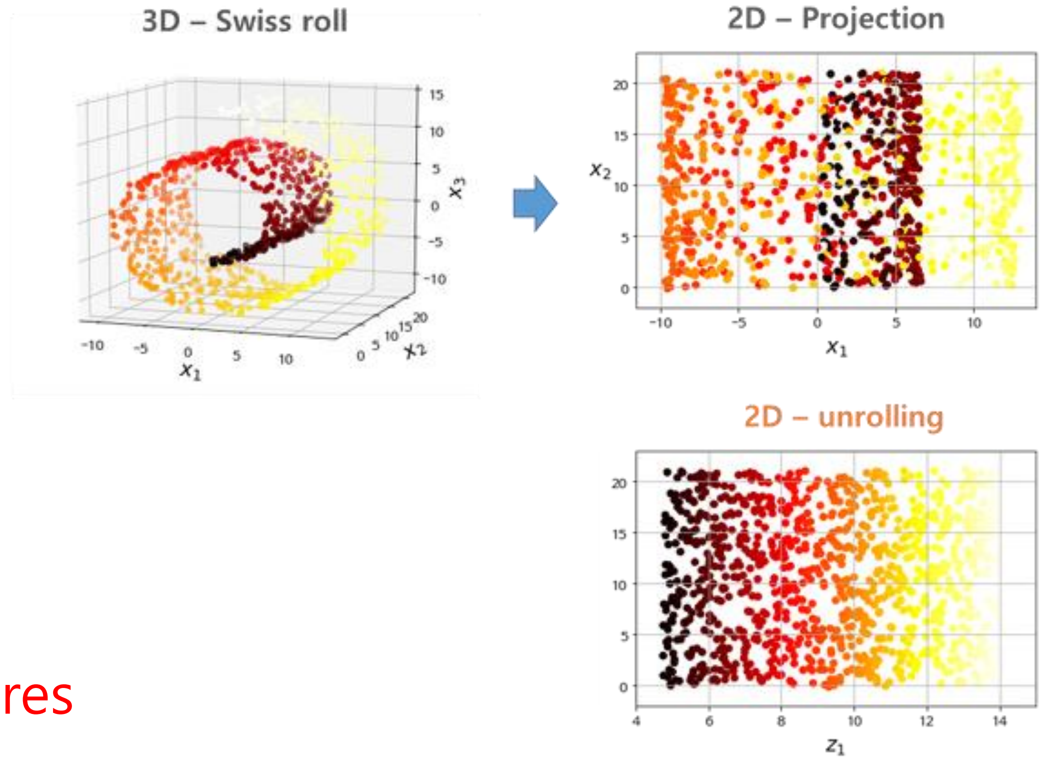
Why AutoEncoder?

- Data Compression
  - 데이터 압축
- Data Visualization
  - 데이터 가시화
- Curse of dimensionality
  - 차원의 저주 해결
- Discovering most important features
  - 가장 중요한 피쳐 찾기



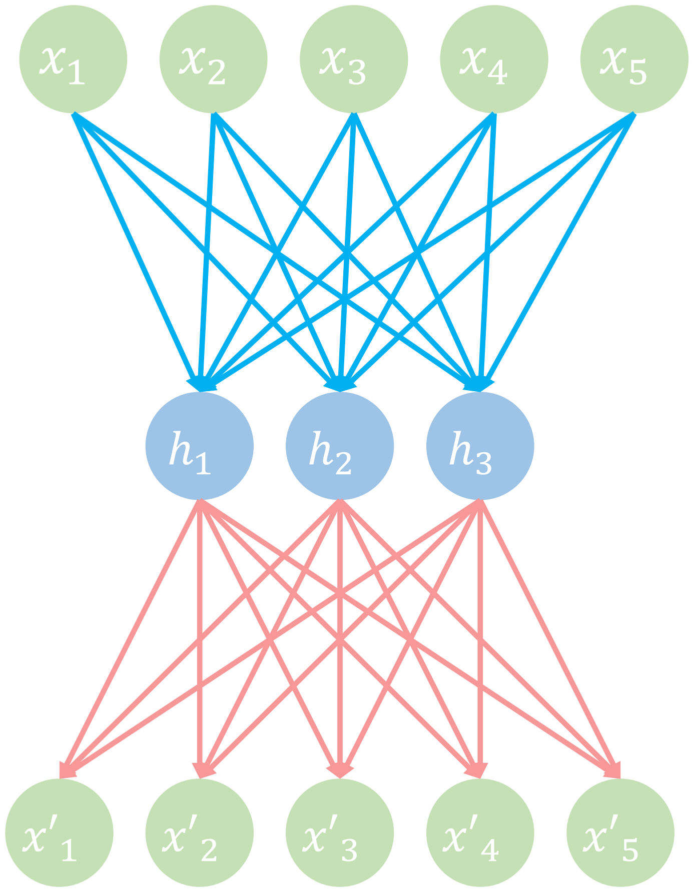
Why AutoEncoder?

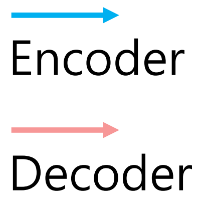
- Data Compression
  - 데이터 압축
- Data Visualization
  - 데이터 가시화
- Curse of dimensionality
  - 차원의 저주 해결
- Discovering most important features
  - 가장 중요한 feature 찾기



AutoEncoder

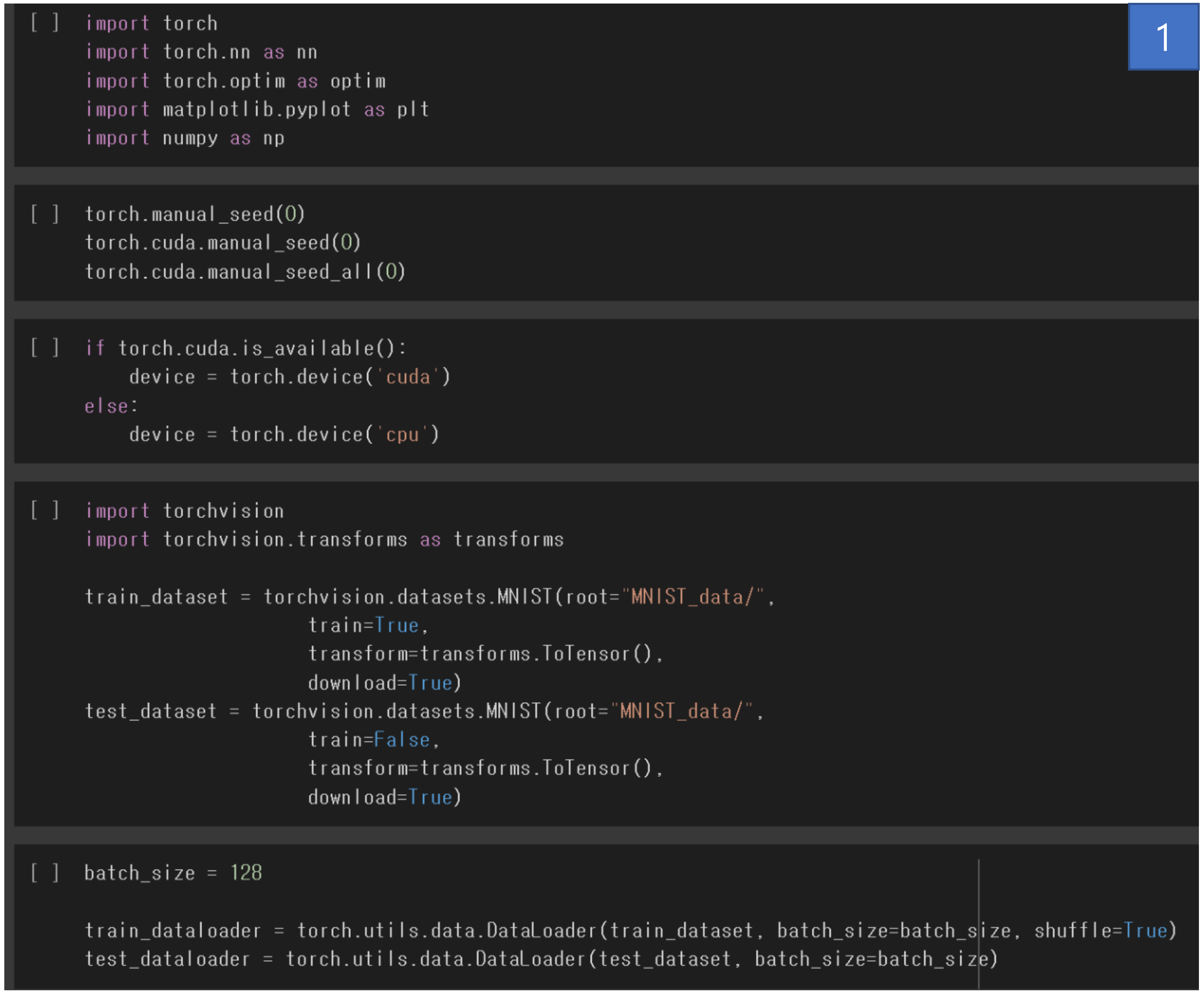
- 모델 아키텍쳐





AutoEncoder

- 모델 코드



```
[ ]  import torch
     import torch.nn as nn
     import torch.optim as optim
     import matplotlib.pyplot as plt
     import numpy as np
```

```
[ ]  torch.manual_seed(0)
     torch.cuda.manual_seed(0)
     torch.cuda.manual_seed_all(0)
```

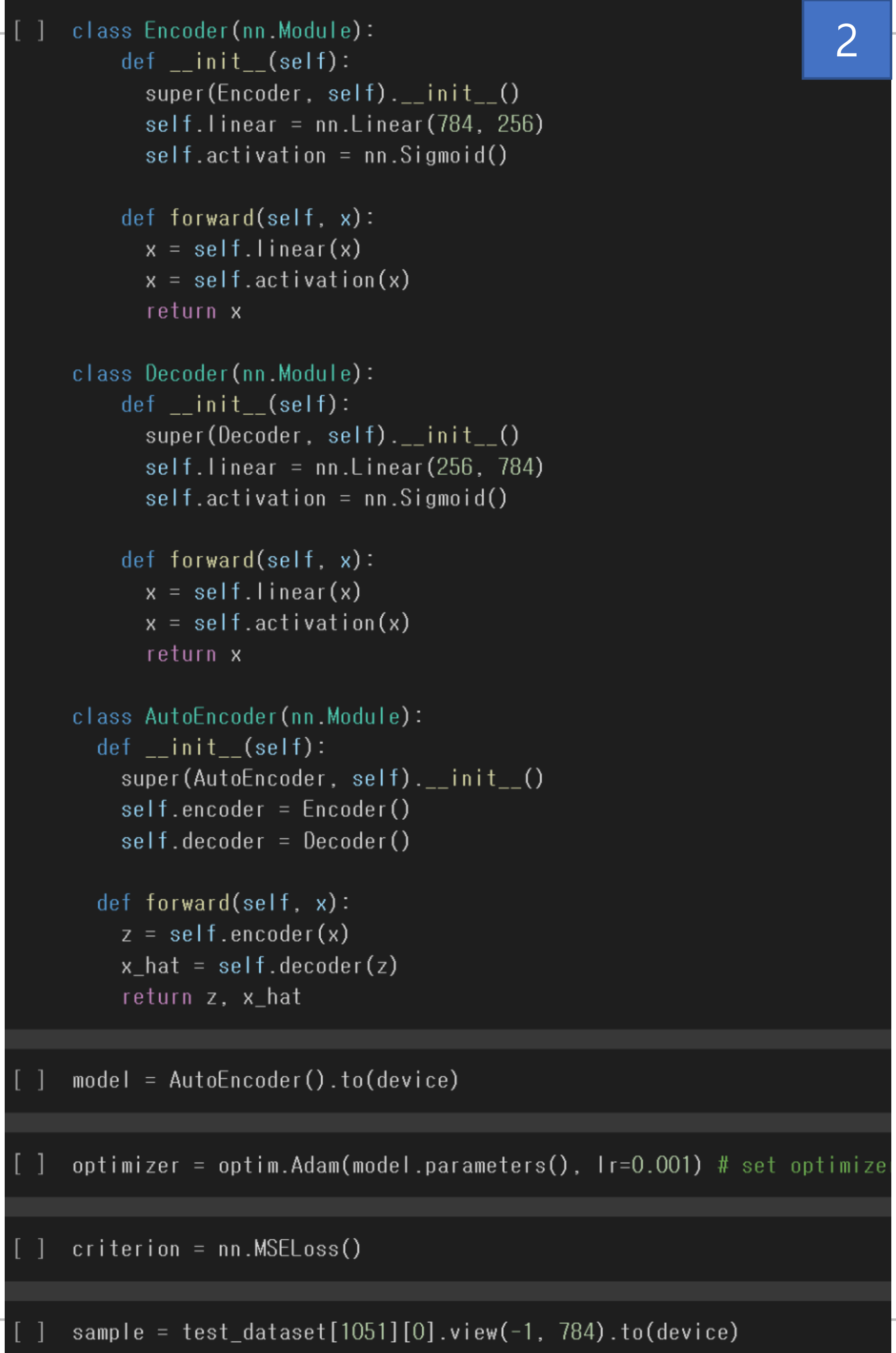
```
[ ]  if torch.cuda.is_available():
         device = torch.device('cuda')
     else:
         device = torch.device('cpu')
```

```
[ ]  import torchvision
     import torchvision.transforms as transforms

     train_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                                                train=True,
                                                transform=transforms.ToTensor(),
                                                download=True)
     test_dataset = torchvision.datasets.MNIST(root="MNIST_data/",
                                                train=False,
                                                transform=transforms.ToTensor(),
                                                download=True)
```

```
[ ]  batch_size = 128

     train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
     test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size)
```

AutoEncoder

- 모델 코드



```
[ ] class Encoder(nn.Module):
        def __init__(self):
          super(Encoder, self).__init__()
          self.linear = nn.Linear(784, 256)
          self.activation = nn.Sigmoid()

        def forward(self, x):
          x = self.linear(x)
          x = self.activation(x)
          return x

    class Decoder(nn.Module):
        def __init__(self):
          super(Decoder, self).__init__()
          self.linear = nn.Linear(256, 784)
          self.activation = nn.Sigmoid()

        def forward(self, x):
          x = self.linear(x)
          x = self.activation(x)
          return x

    class AutoEncoder(nn.Module):
      def __init__(self):
        super(AutoEncoder, self).__init__()
        self.encoder = Encoder()
        self.decoder = Decoder()

      def forward(self, x):
        z = self.encoder(x)
        x_hat = self.decoder(z)
        return z, x_hat
```

```
[ ] model = AutoEncoder().to(device)
```

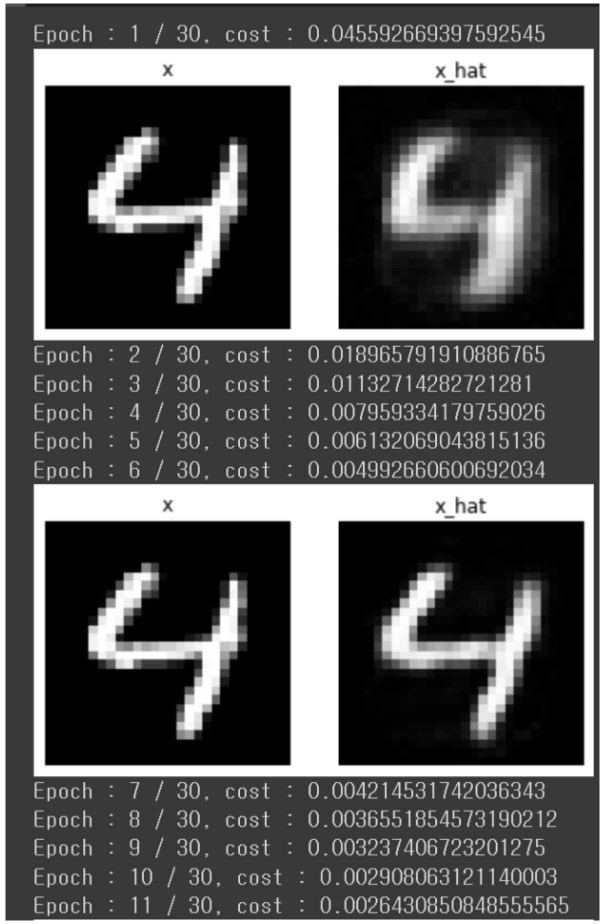
```
[ ] optimizer = optim.Adam(model.parameters(), lr=0.001) # set optimize
```

```
[ ] criterion = nn.MSELoss()
```

```
[ ] sample = test_dataset[1051][0].view(-1, 784).to(device)
```

AutoEncoder

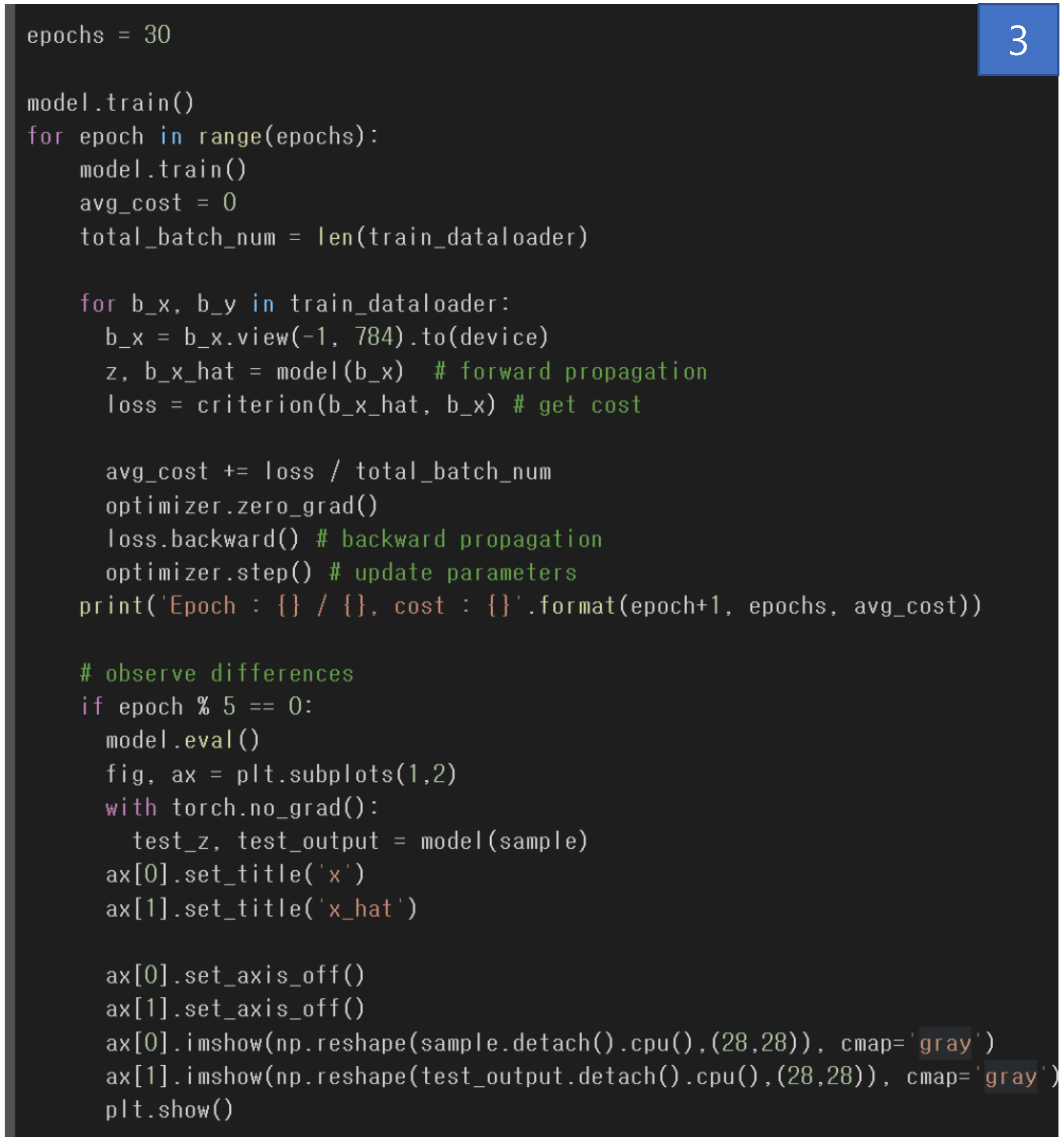
- 모델 코드



```
Epoch : 1 / 30, cost : 0.045592669397592545
```

```
Epoch : 2 / 30, cost : 0.018965791910886765
Epoch : 3 / 30, cost : 0.011327142827212181
Epoch : 4 / 30, cost : 0.007959334179759026
Epoch : 5 / 30, cost : 0.006132069043815136
Epoch : 6 / 30, cost : 0.004992660600692034
```

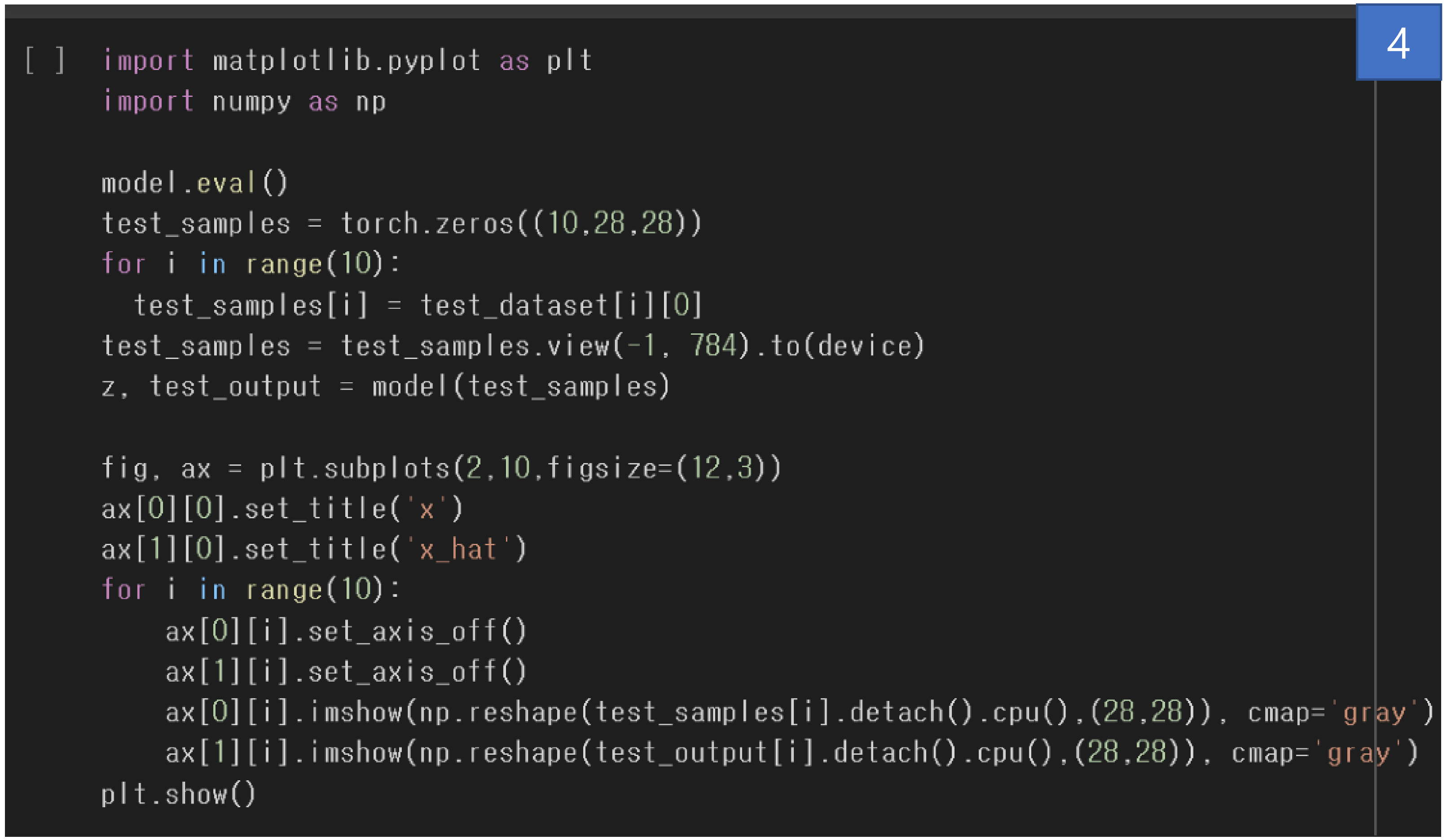
```
Epoch : 7 / 30, cost : 0.004214531742036343
Epoch : 8 / 30, cost : 0.0036551854573190212
Epoch : 9 / 30, cost : 0.003237406723201275
Epoch : 10 / 30, cost : 0.002908063121140003
Epoch : 11 / 30, cost : 0.002643085084855565
```



```
epochs = 30

model.train()
for epoch in range(epochs):
    model.train()
    avg_cost = 0
    total_batch_num = len(train_dataloader)

    for b_x, b_y in train_dataloader:
      b_x = b_x.view(-1, 784).to(device)
      z, b_x_hat = model(b_x)  # forward propagation
      loss = criterion(b_x_hat, b_x) # get cost

      avg_cost += loss / total_batch_num
      optimizer.zero_grad()
      loss.backward() # backward propagation
      optimizer.step() # update parameters
    print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))

    # observe differences
    if epoch % 5 == 0:
      model.eval()
      fig, ax = plt.subplots(1,2)
      with torch.no_grad():
        test_z, test_output = model(sample)
      ax[0].set_title('x')
      ax[1].set_title('x_hat')

      ax[0].set_axis_off()
      ax[1].set_axis_off()
      ax[0].imshow(np.reshape(sample.detach().cpu(),(28,28)), cmap='gray')
      ax[1].imshow(np.reshape(test_output.detach().cpu(),(28,28)), cmap='gray')
      plt.show()
```

AutoEncoder

- 학습 결과 확인

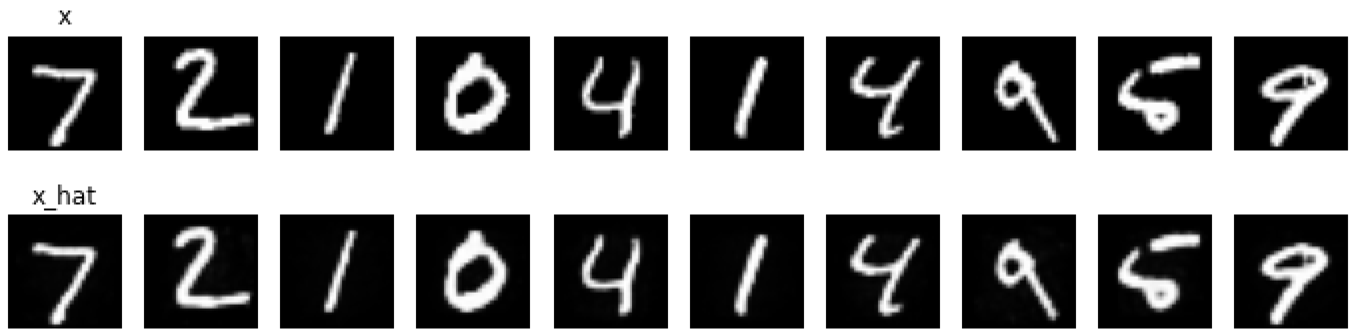


```
[ ]  import matplotlib.pyplot as plt
     import numpy as np

     model.eval()
     test_samples = torch.zeros((10,28,28))
     for i in range(10):
       test_samples[i] = test_dataset[i][0]
     test_samples = test_samples.view(-1, 784).to(device)
     z, test_output = model(test_samples)

     fig, ax = plt.subplots(2,10,figsize=(12,3))
     ax[0][0].set_title('x')
     ax[1][0].set_title('x_hat')
     for i in range(10):
         ax[0][i].set_axis_off()
         ax[1][i].set_axis_off()
         ax[0][i].imshow(np.reshape(test_samples[i].detach().cpu(),(28,28)), cmap='gray')
         ax[1][i].imshow(np.reshape(test_output[i].detach().cpu(),(28,28)), cmap='gray')
     plt.show()
```

4

AutoEncoder

- 학습 결과



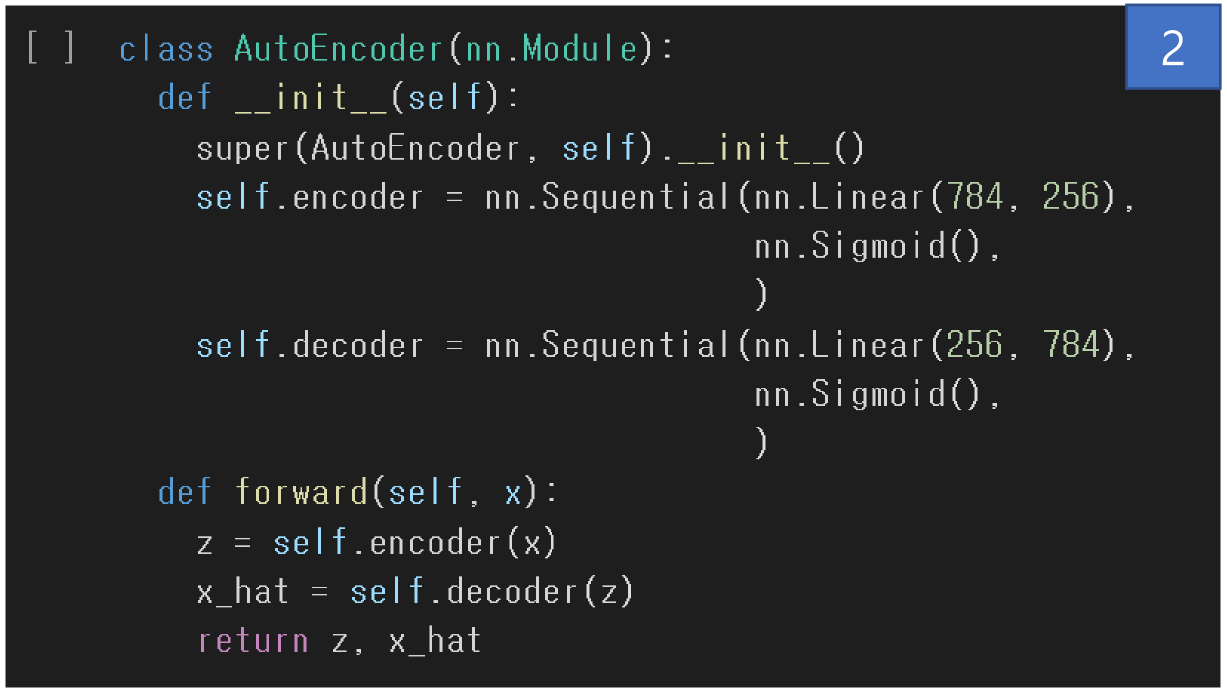
AutoEncoder

- 다양한 Encoder와 Decoder 학습 방법
  1. Encoder, Decoder class 각각 나눠서 생성 후 하나의 모델로 만들기 (이전 코드)
  2. nn.Sequential 을 사용해 한 모델에서 autoencoder 작성하기
  3. Encoder, Decoder class 각각 나눠서 하나씩 부르기

  등등

AutoEncoder

- 다양한 Encoder와 Decoder 학습 방법
  1. Encoder, Decoder class 각각 나눠서 생성 후 하나의 모델로 만들기 (이전 코드)
  2. **nn.Sequential 을 사용해 한 모델에서 autoencoder 작성하기**
  3. Encoder, Decoder class 각각 나눠서 하나씩 부르기

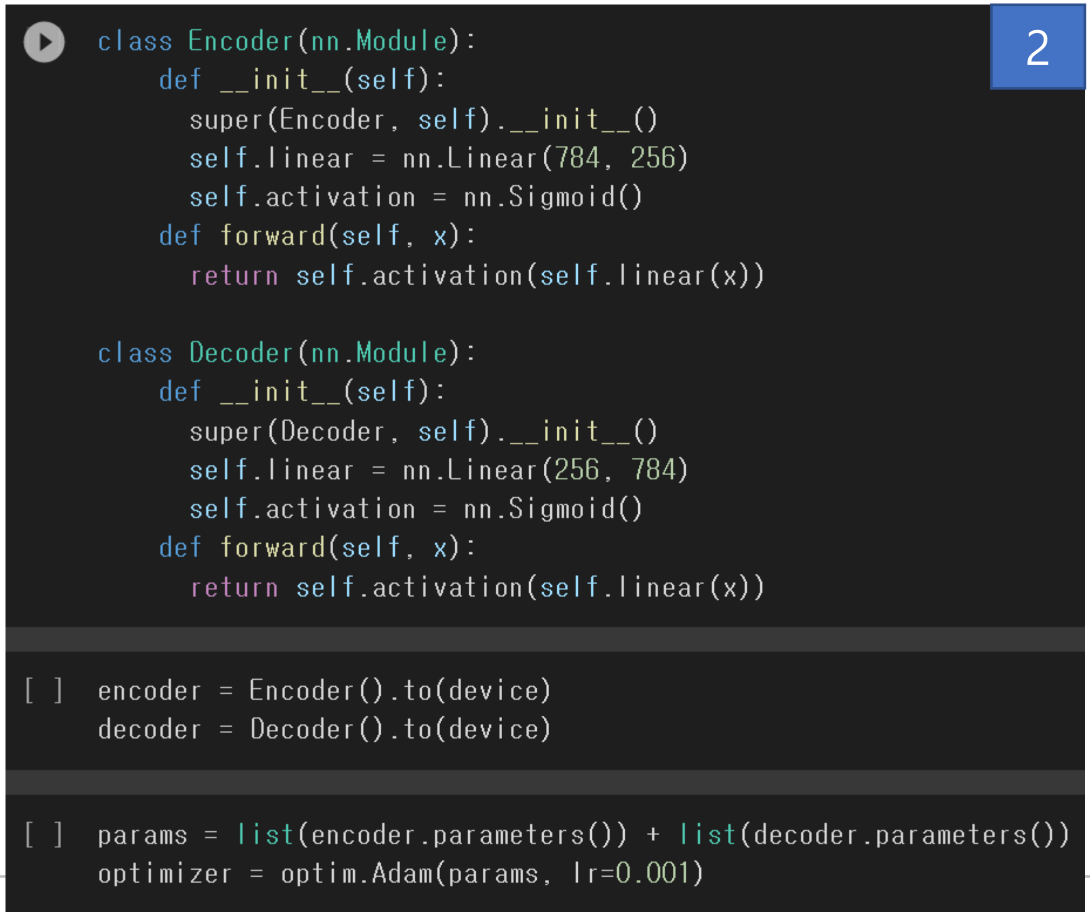


2

```
[ ] class AutoEncoder(nn.Module):
      def __init__(self):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Sequential(nn.Linear(784, 256),
                                     nn.Sigmoid(),
                                     )
        self.decoder = nn.Sequential(nn.Linear(256, 784),
                                     nn.Sigmoid(),
                                     )
      def forward(self, x):
        z = self.encoder(x)
        x_hat = self.decoder(z)
        return z, x_hat
```

AutoEncoder

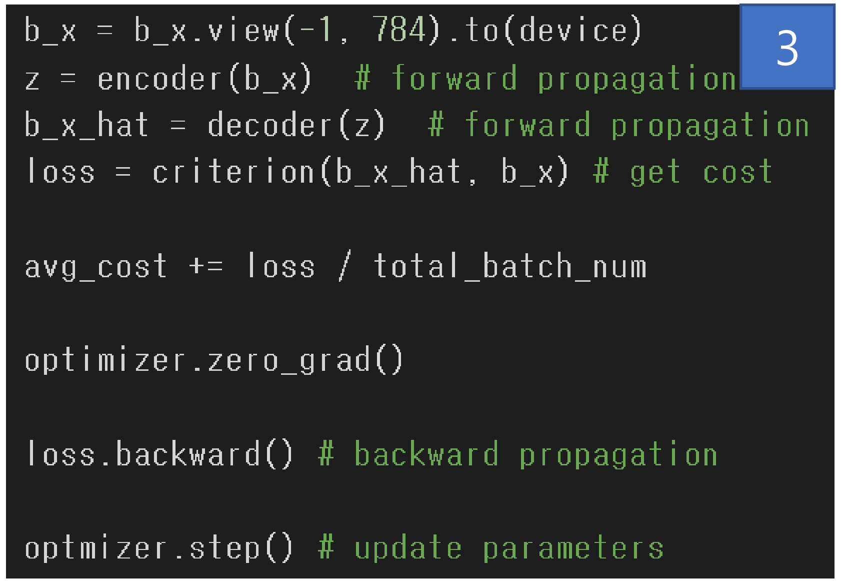
- 다양한 Encoder와 Decoder 학습 방법
  1. Encoder, Decoder class 각각 나눠서 생성 후 하나의 모델로 만들기 (이전 코드)
  2. nn.Sequential 을 사용해 한 모델에서 autoencoder 작성하기
  3. **Encoder, Decoder class 각각 나눠서 하나씩 부르기 1) Parameter List** 2) Parameter group



```
class Encoder(nn.Module):
    def __init__(self):
      super(Encoder, self).__init__()
      self.linear = nn.Linear(784, 256)
      self.activation = nn.Sigmoid()
    def forward(self, x):
      return self.activation(self.linear(x))

class Decoder(nn.Module):
    def __init__(self):
      super(Decoder, self).__init__()
      self.linear = nn.Linear(256, 784)
      self.activation = nn.Sigmoid()
    def forward(self, x):
      return self.activation(self.linear(x))
```

```
[ ]  encoder = Encoder().to(device)
     decoder = Decoder().to(device)
```

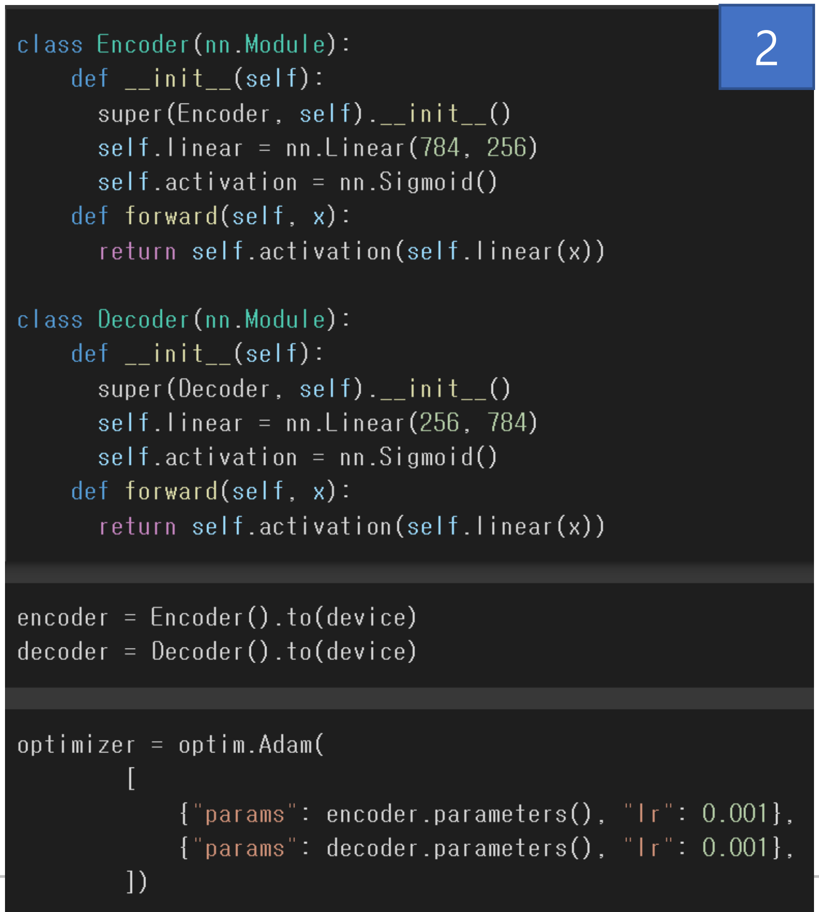
```
[ ]  params = list(encoder.parameters()) + list(decoder.parameters())
     optimizer = optim.Adam(params, lr=0.001)
```



```
b_x = b_x.view(-1, 784).to(device)
z = encoder(b_x)  # forward propagation
b_x_hat = decoder(z)  # forward propagation
loss = criterion(b_x_hat, b_x) # get cost

avg_cost += loss / total_batch_num

optimizer.zero_grad()

loss.backward() # backward propagation

optmizer.step() # update parameters
```

AutoEncoder

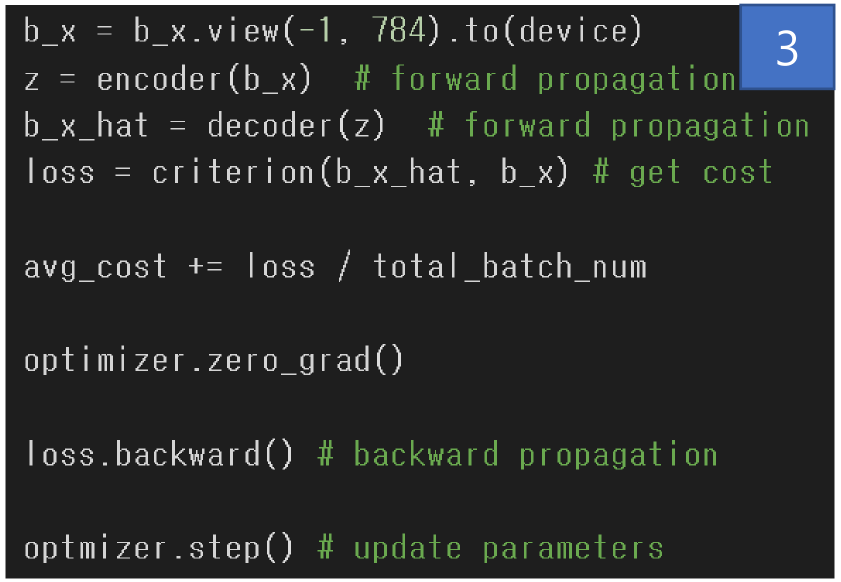
- 다양한 Encoder와 Decoder 학습 방법
  1. Encoder, Decoder class 각각 나눠서 생성 후 하나의 모델로 만들기 (이전 코드)
  2. nn.Sequential 을 사용해 한 모델에서 autoencoder 작성하기
  3. **Encoder, Decoder class 각각 나눠서 하나씩 부르기** 1) Parameter List **2) 두개의 optimizer**


2

```
class Encoder(nn.Module):
    def __init__(self):
      super(Encoder, self).__init__()
      self.linear = nn.Linear(784, 256)
      self.activation = nn.Sigmoid()
    def forward(self, x):
      return self.activation(self.linear(x))

class Decoder(nn.Module):
    def __init__(self):
      super(Decoder, self).__init__()
      self.linear = nn.Linear(256, 784)
      self.activation = nn.Sigmoid()
    def forward(self, x):
      return self.activation(self.linear(x))
```

```
encoder = Encoder().to(device)
decoder = Decoder().to(device)
```

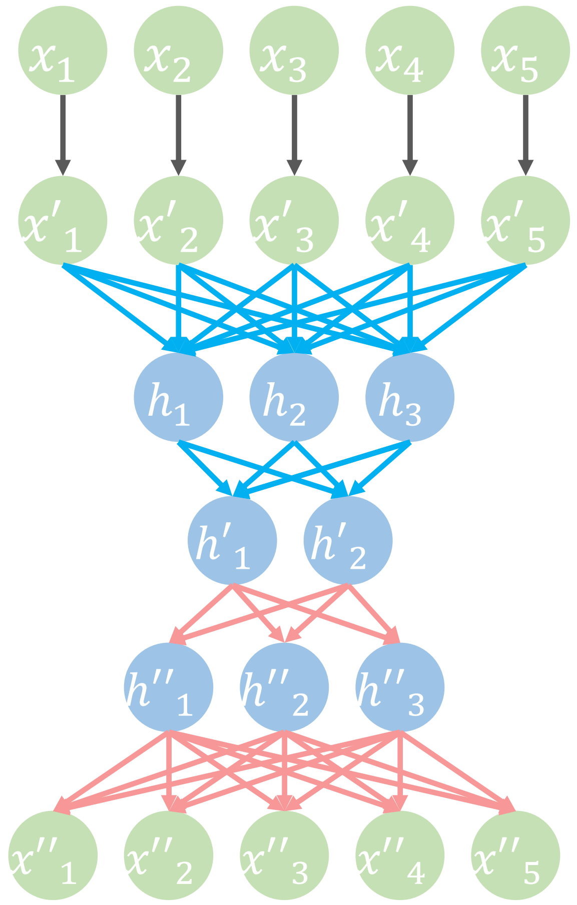
```
optimizer = optim.Adam(
        [
            {"params": encoder.parameters(), "lr": 0.001},
            {"params": decoder.parameters(), "lr": 0.001},
        ])
```


3

```
b_x = b_x.view(-1, 784).to(device)
z = encoder(b_x)  # forward propagation
b_x_hat = decoder(z)  # forward propagation
loss = criterion(b_x_hat, b_x) # get cost

avg_cost += loss / total_batch_num

optimizer.zero_grad()

loss.backward() # backward propagation

optmizer.step() # update parameters
```

Denoising Auto-Encoder model

- Denoising AutoEncoder model



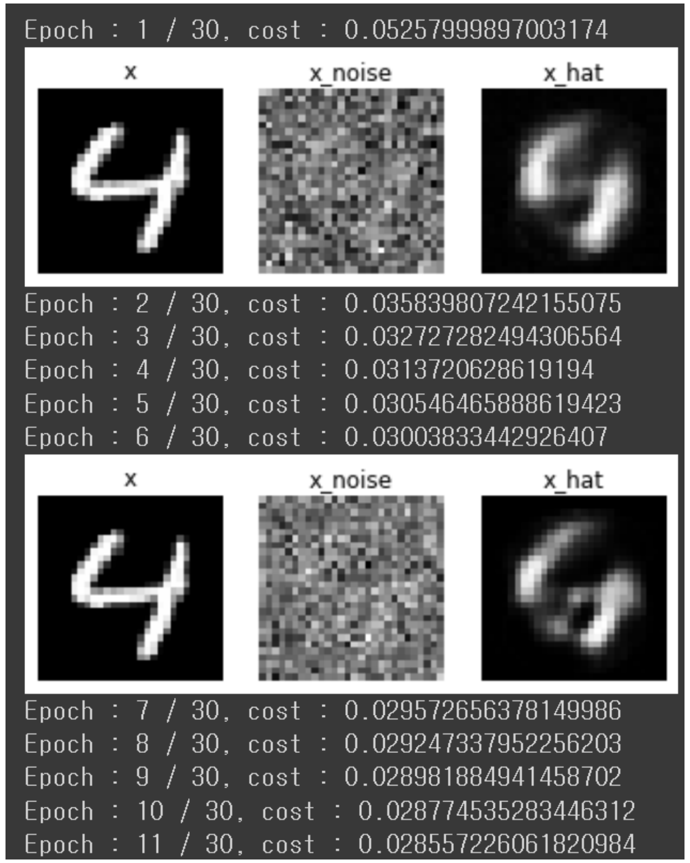
$\longrightarrow$ Random Noise Adder

$\longrightarrow$ Encoder

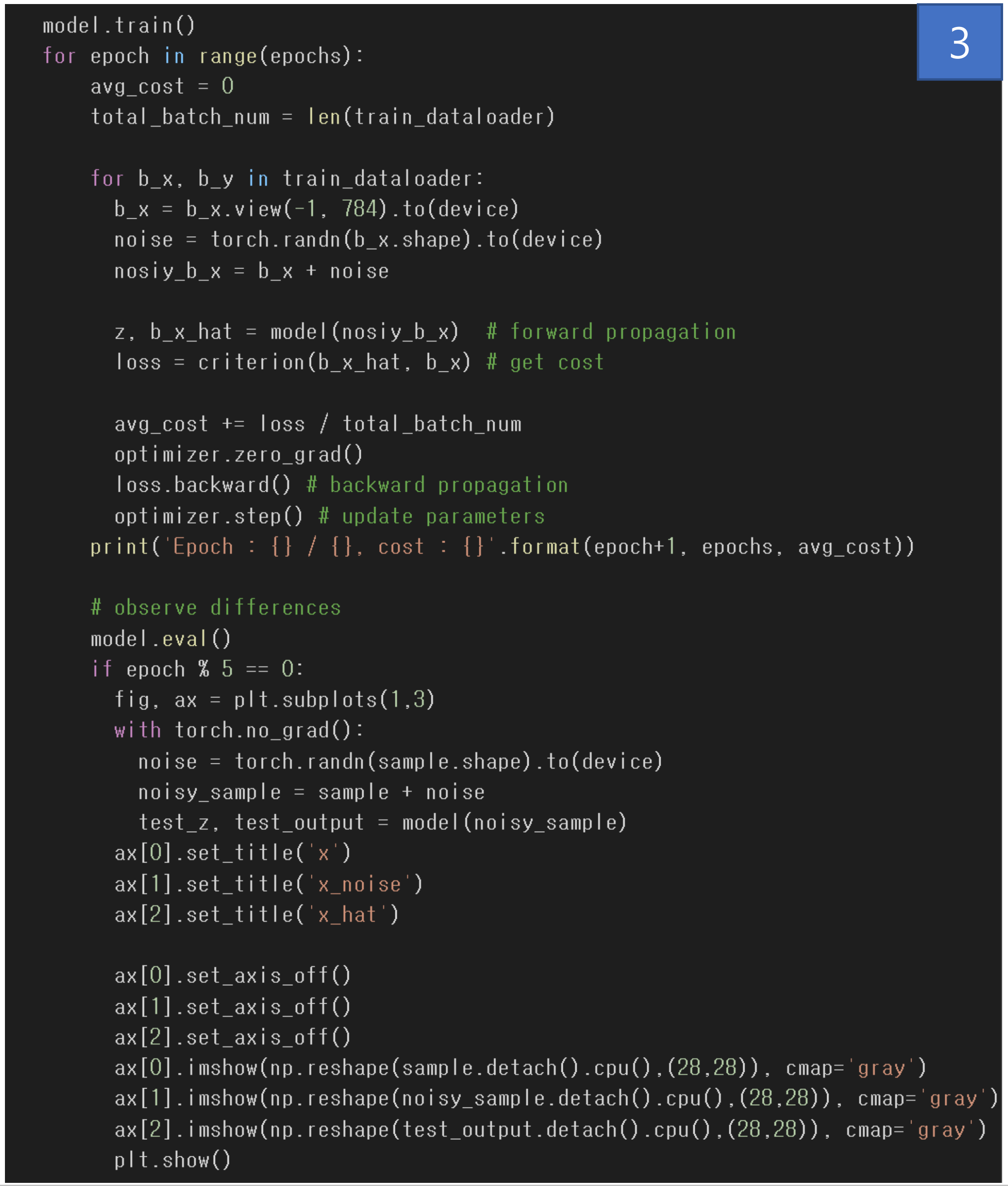
$\longrightarrow$ Decoder

Denoising Auto-Encoder model

- 학습 코드
  - 앞 코드는 이전 autoencoder 코드와 같음



```
Epoch : 1 / 30, cost : 0.05257999897003174
Epoch : 2 / 30, cost : 0.035839807242155075
Epoch : 3 / 30, cost : 0.032727282494306564
Epoch : 4 / 30, cost : 0.0313720628619194
Epoch : 5 / 30, cost : 0.030546465888619423
Epoch : 6 / 30, cost : 0.03003833442926407
Epoch : 7 / 30, cost : 0.029572656378149986
Epoch : 8 / 30, cost : 0.029247337952256203
Epoch : 9 / 30, cost : 0.028981884941458702
Epoch : 10 / 30, cost : 0.028774535283446312
Epoch : 11 / 30, cost : 0.028557226061820984
```

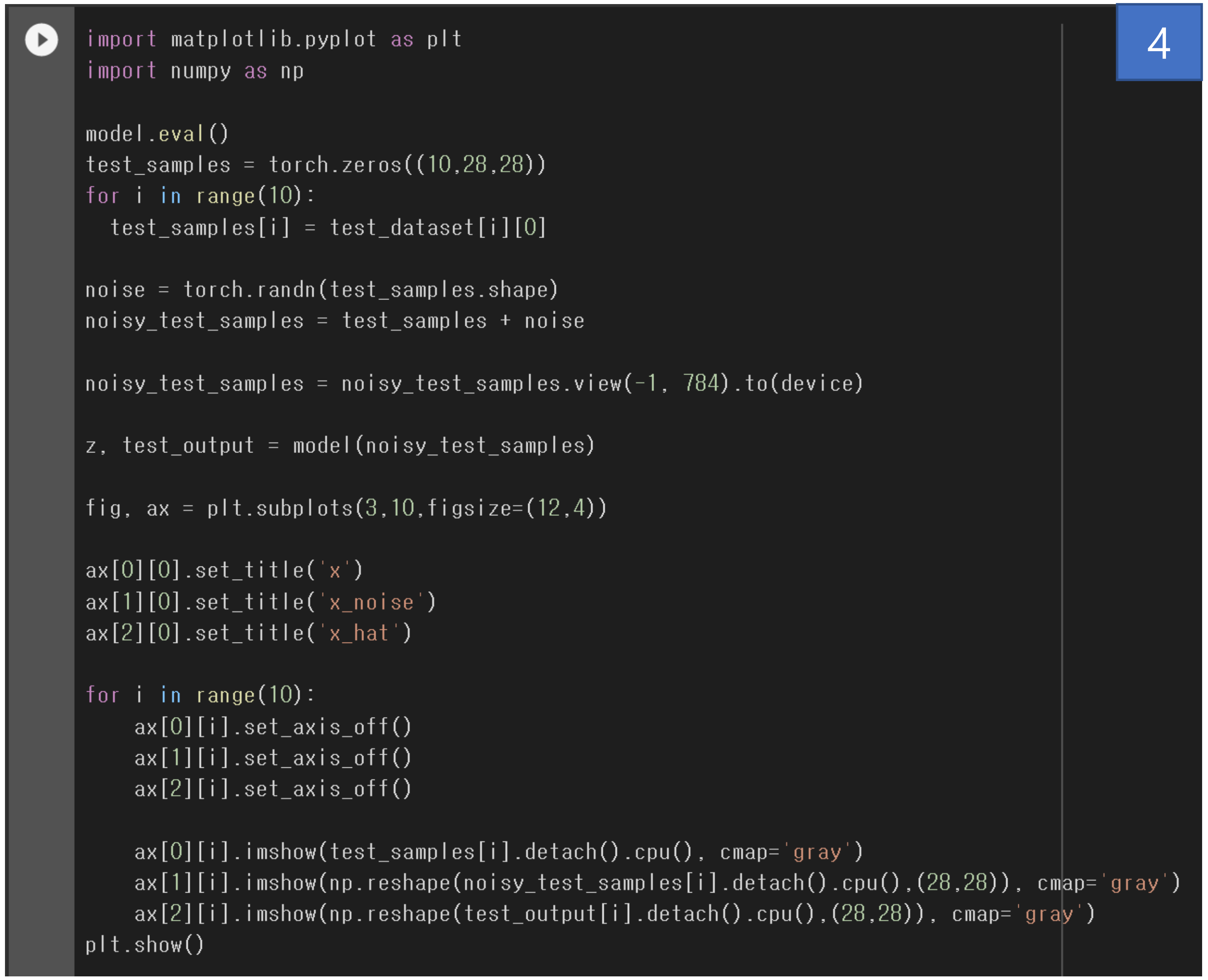


3

```
model.train()
for epoch in range(epochs):
    avg_cost = 0
    total_batch_num = len(train_dataloader)

    for b_x, b_y in train_dataloader:
      b_x = b_x.view(-1, 784).to(device)
      noise = torch.randn(b_x.shape).to(device)
      nosiy_b_x = b_x + noise

      z, b_x_hat = model(nosiy_b_x)  # forward propagation
      loss = criterion(b_x_hat, b_x) # get cost

      avg_cost += loss / total_batch_num
      optimizer.zero_grad()
      loss.backward() # backward propagation
      optimizer.step() # update parameters
    print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))

    # observe differences
    model.eval()
    if epoch % 5 == 0:
      fig, ax = plt.subplots(1,3)
      with torch.no_grad():
        noise = torch.randn(sample.shape).to(device)
        noisy_sample = sample + noise
        test_z, test_output = model(noisy_sample)
      ax[0].set_title('x')
      ax[1].set_title('x_noise')
      ax[2].set_title('x_hat')

      ax[0].set_axis_off()
      ax[1].set_axis_off()
      ax[2].set_axis_off()
      ax[0].imshow(np.reshape(sample.detach().cpu(),(28,28)), cmap='gray')
      ax[1].imshow(np.reshape(noisy_sample.detach().cpu(),(28,28)), cmap='gray')
      ax[2].imshow(np.reshape(test_output.detach().cpu(),(28,28)), cmap='gray')
      plt.show()
```

Denoising Auto-Encoder model

- 학습 결과 확인

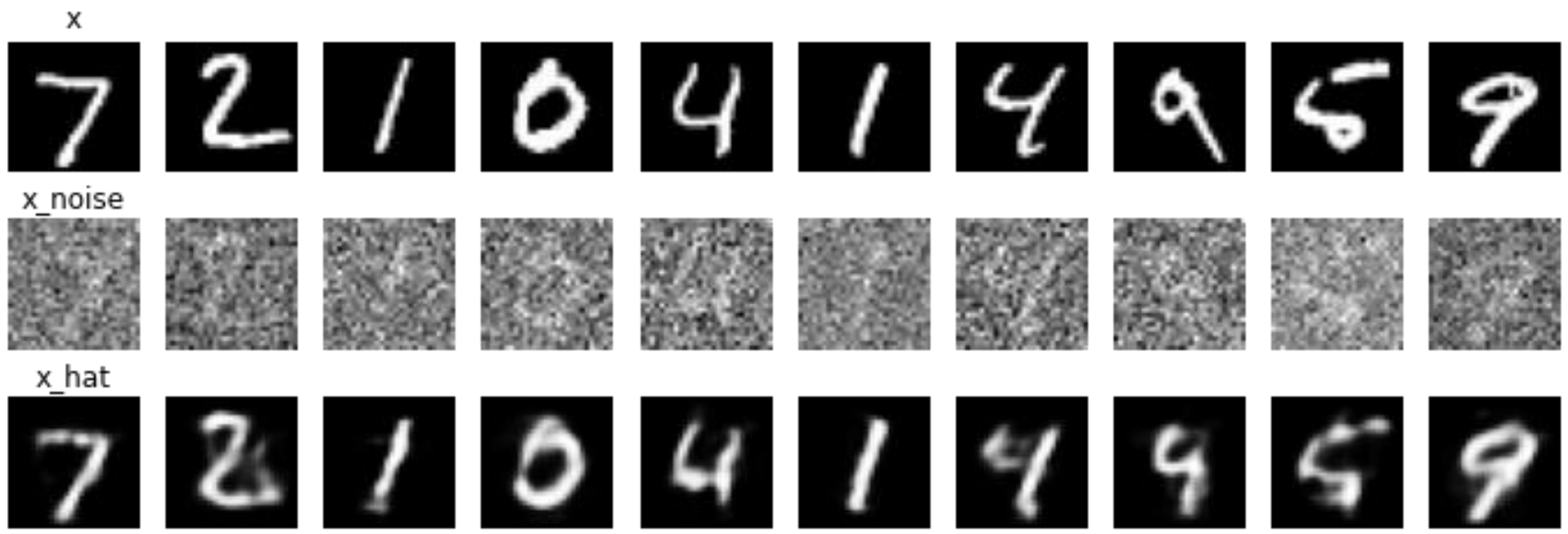


4

```
import matplotlib.pyplot as plt
import numpy as np

model.eval()
test_samples = torch.zeros((10,28,28))
for i in range(10):
  test_samples[i] = test_dataset[i][0]

noise = torch.randn(test_samples.shape)
noisy_test_samples = test_samples + noise

noisy_test_samples = noisy_test_samples.view(-1, 784).to(device)

z, test_output = model(noisy_test_samples)

fig, ax = plt.subplots(3,10,figsize=(12,4))

ax[0][0].set_title('x')
ax[1][0].set_title('x_noise')
ax[2][0].set_title('x_hat')

for i in range(10):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[2][i].set_axis_off()

    ax[0][i].imshow(test_samples[i].detach().cpu(), cmap='gray')
    ax[1][i].imshow(np.reshape(noisy_test_samples[i].detach().cpu(),(28,28)), cmap='gray')
    ax[2][i].imshow(np.reshape(test_output[i].detach().cpu(),(28,28)), cmap='gray')
plt.show()
```

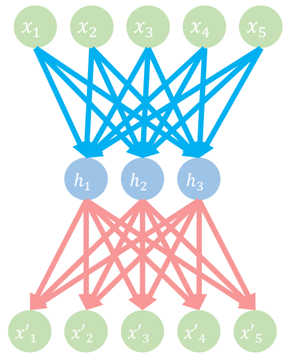
Stacked AutoEncoder and Semi-Supervised Learning

- 학습 결과

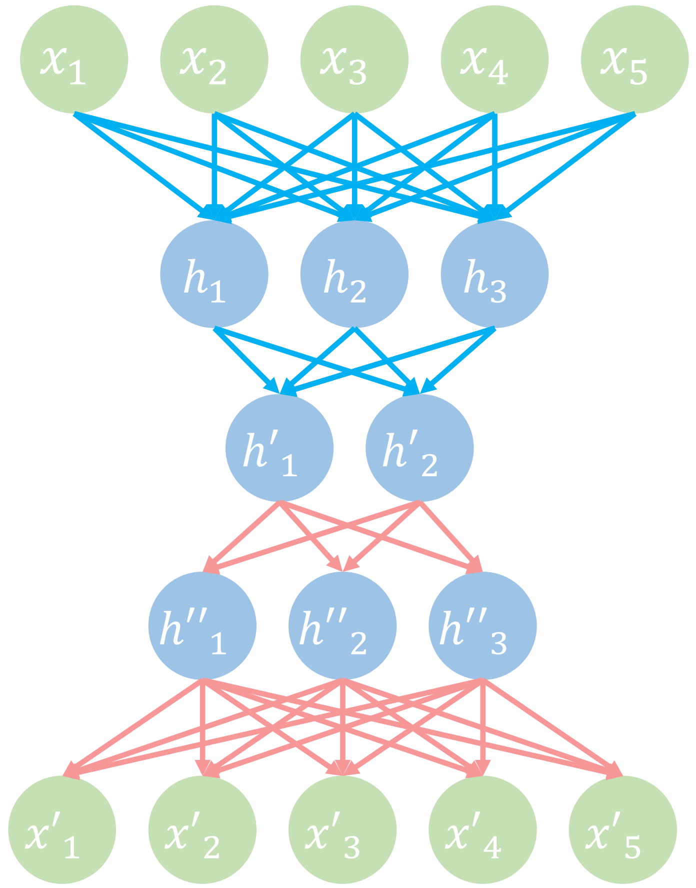


Stacked AutoEncoder

- Stacked AutoEncoder



Autoencoder

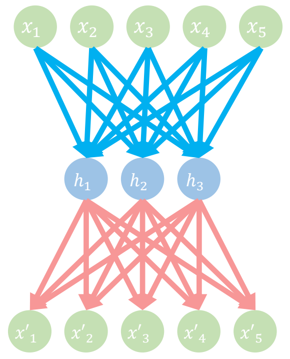


$\longrightarrow$ Encoder

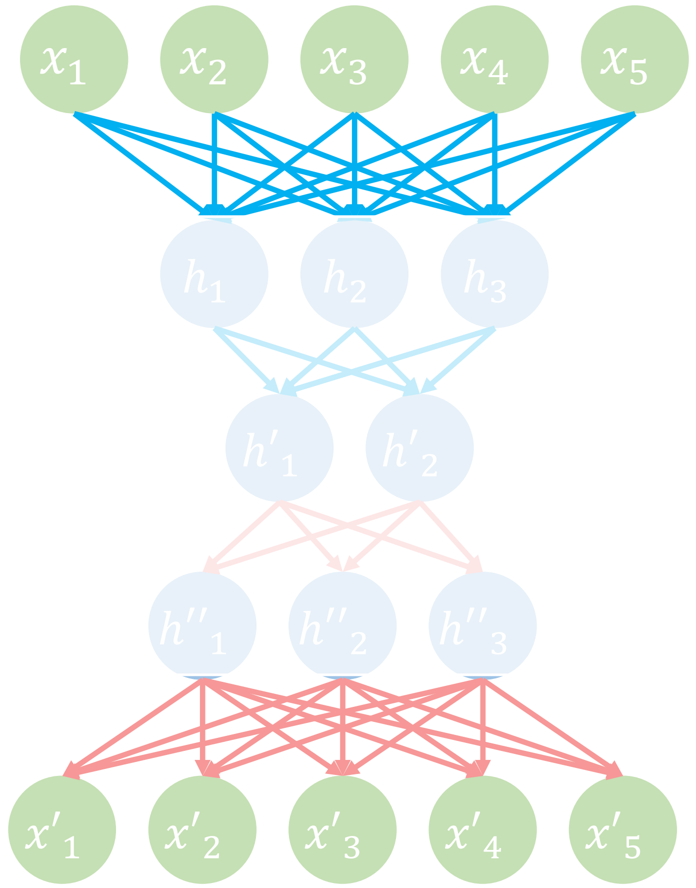
$\longrightarrow$ Decoder

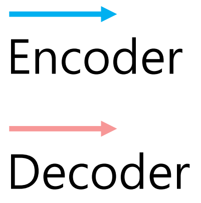
Stacked AutoEncoder

- Stacked AutoEncoder



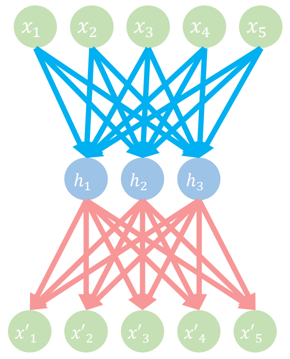
Autoencoder



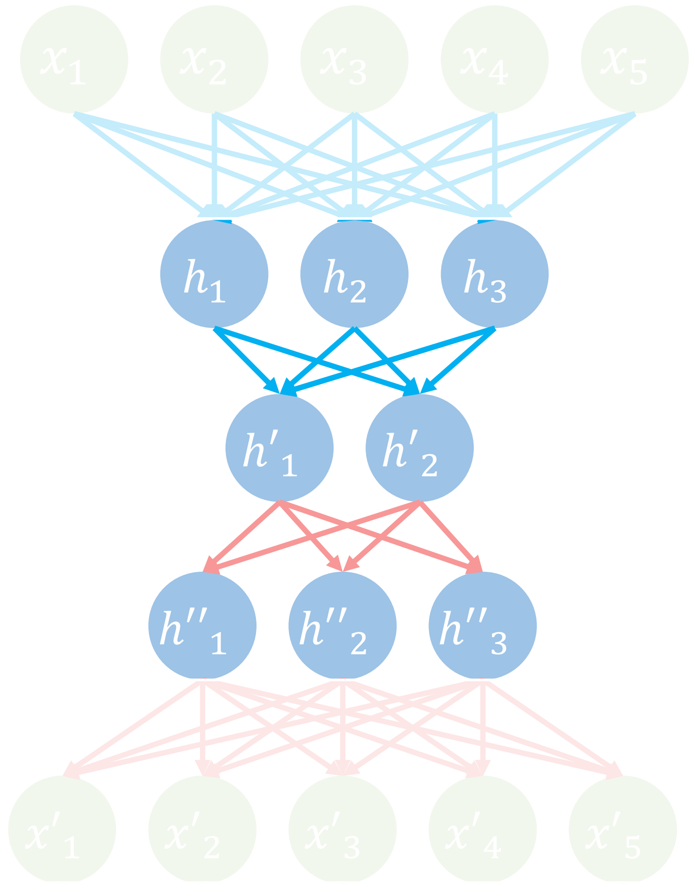


Stacked AutoEncoder

- Stacked AutoEncoder



Autoencoder

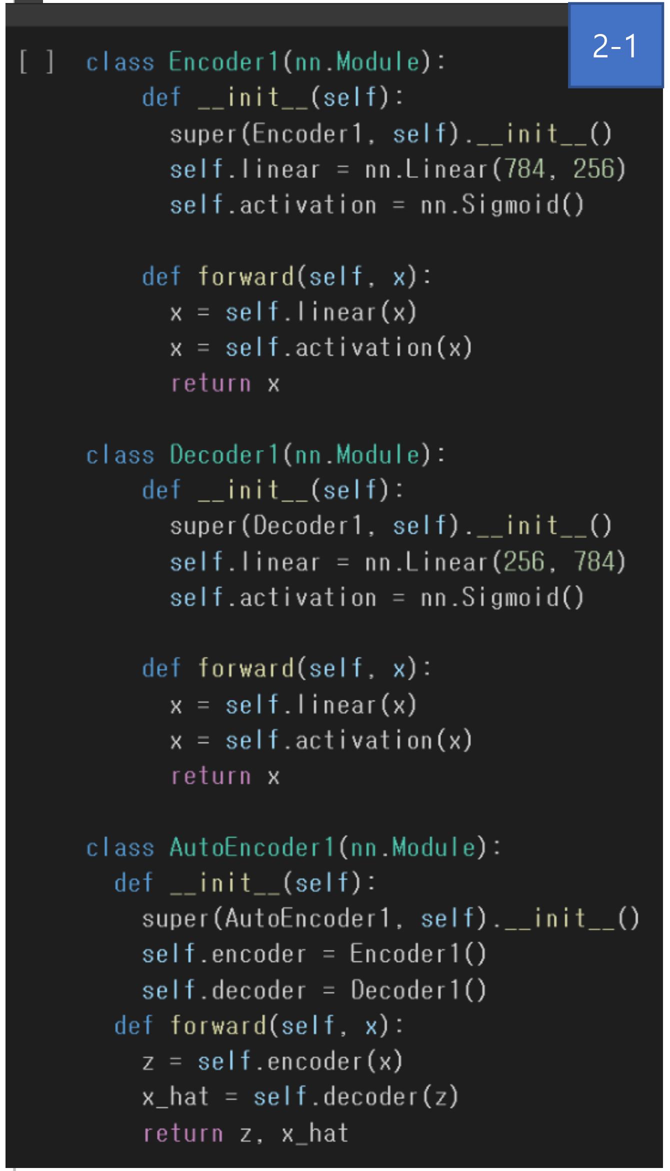


Encoder

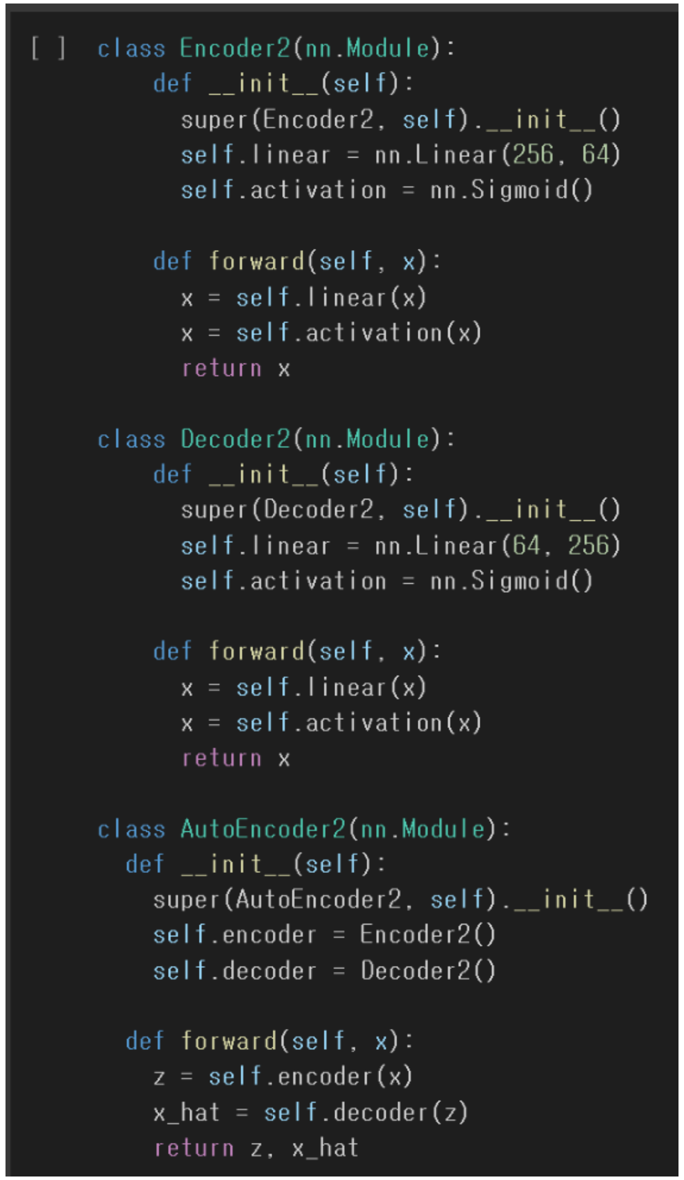
Decoder

Stacked AutoEncoder

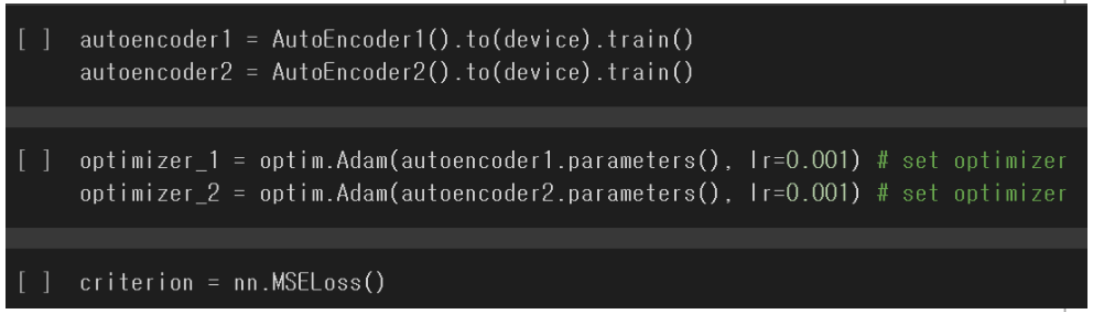
2-1



```
[ ]  class Encoder1(nn.Module):
         def __init__(self):
           super(Encoder1, self).__init__()
           self.linear = nn.Linear(784, 256)
           self.activation = nn.Sigmoid()

         def forward(self, x):
           x = self.linear(x)
           x = self.activation(x)
           return x

     class Decoder1(nn.Module):
         def __init__(self):
           super(Decoder1, self).__init__()
           self.linear = nn.Linear(256, 784)
           self.activation = nn.Sigmoid()

         def forward(self, x):
           x = self.linear(x)
           x = self.activation(x)
           return x

     class AutoEncoder1(nn.Module):
       def __init__(self):
         super(AutoEncoder1, self).__init__()
         self.encoder = Encoder1()
         self.decoder = Decoder1()
       def forward(self, x):
         z = self.encoder(x)
         x_hat = self.decoder(z)
         return z, x_hat
```



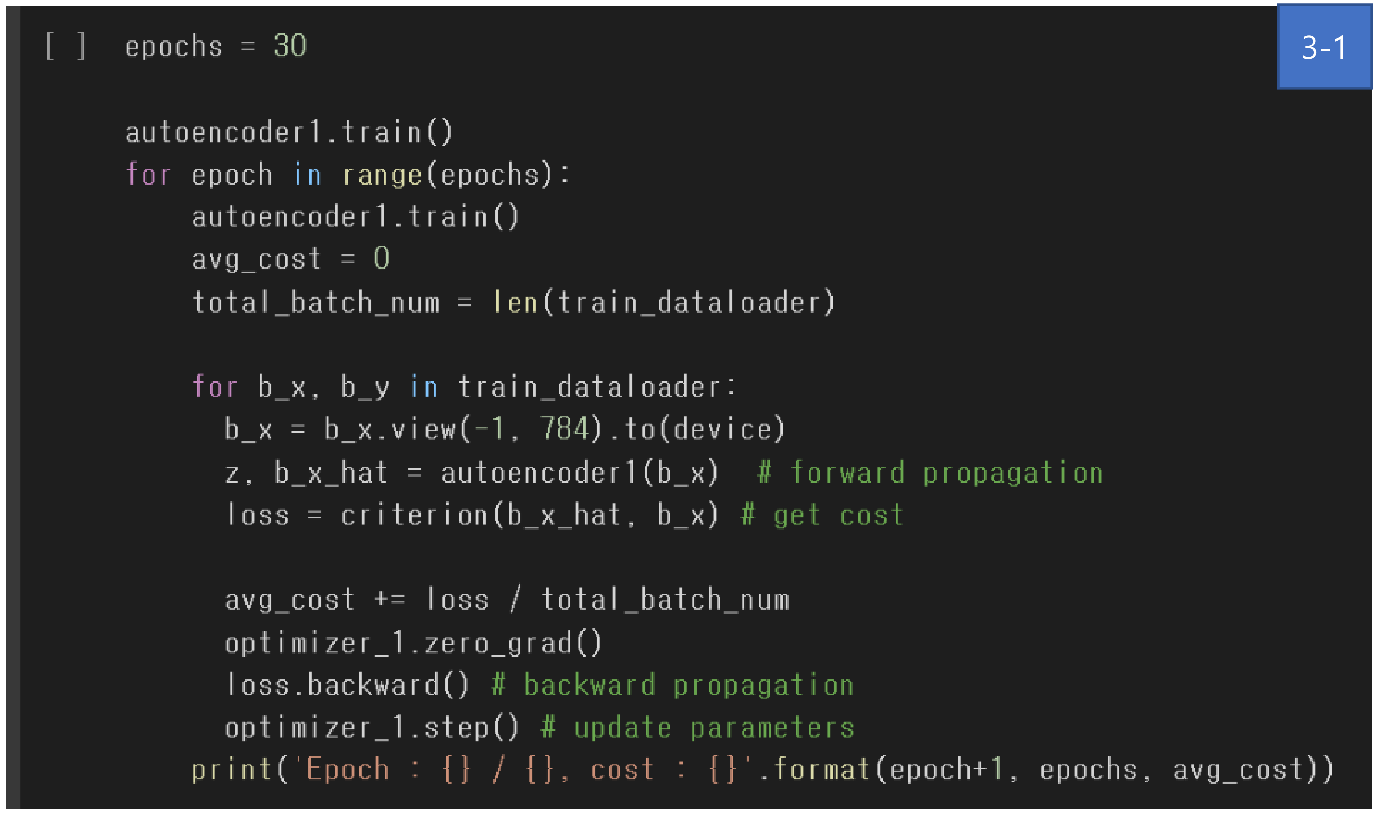
```
[ ]  class Encoder2(nn.Module):
         def __init__(self):
           super(Encoder2, self).__init__()
           self.linear = nn.Linear(256, 64)
           self.activation = nn.Sigmoid()

         def forward(self, x):
           x = self.linear(x)
           x = self.activation(x)
           return x

     class Decoder2(nn.Module):
         def __init__(self):
           super(Decoder2, self).__init__()
           self.linear = nn.Linear(64, 256)
           self.activation = nn.Sigmoid()

         def forward(self, x):
           x = self.linear(x)
           x = self.activation(x)
           return x

     class AutoEncoder2(nn.Module):
       def __init__(self):
         super(AutoEncoder2, self).__init__()
         self.encoder = Encoder2()
         self.decoder = Decoder2()

       def forward(self, x):
         z = self.encoder(x)
         x_hat = self.decoder(z)
         return z, x_hat
```



```
[ ]  autoencoder1 = AutoEncoder1().to(device).train()
     autoencoder2 = AutoEncoder2().to(device).train()

[ ]  optimizer_1 = optim.Adam(autoencoder1.parameters(), lr=0.001) # set optimizer
     optimizer_2 = optim.Adam(autoencoder2.parameters(), lr=0.001) # set optimizer

[ ]  criterion = nn.MSELoss()
```

Stacked AutoEncoder

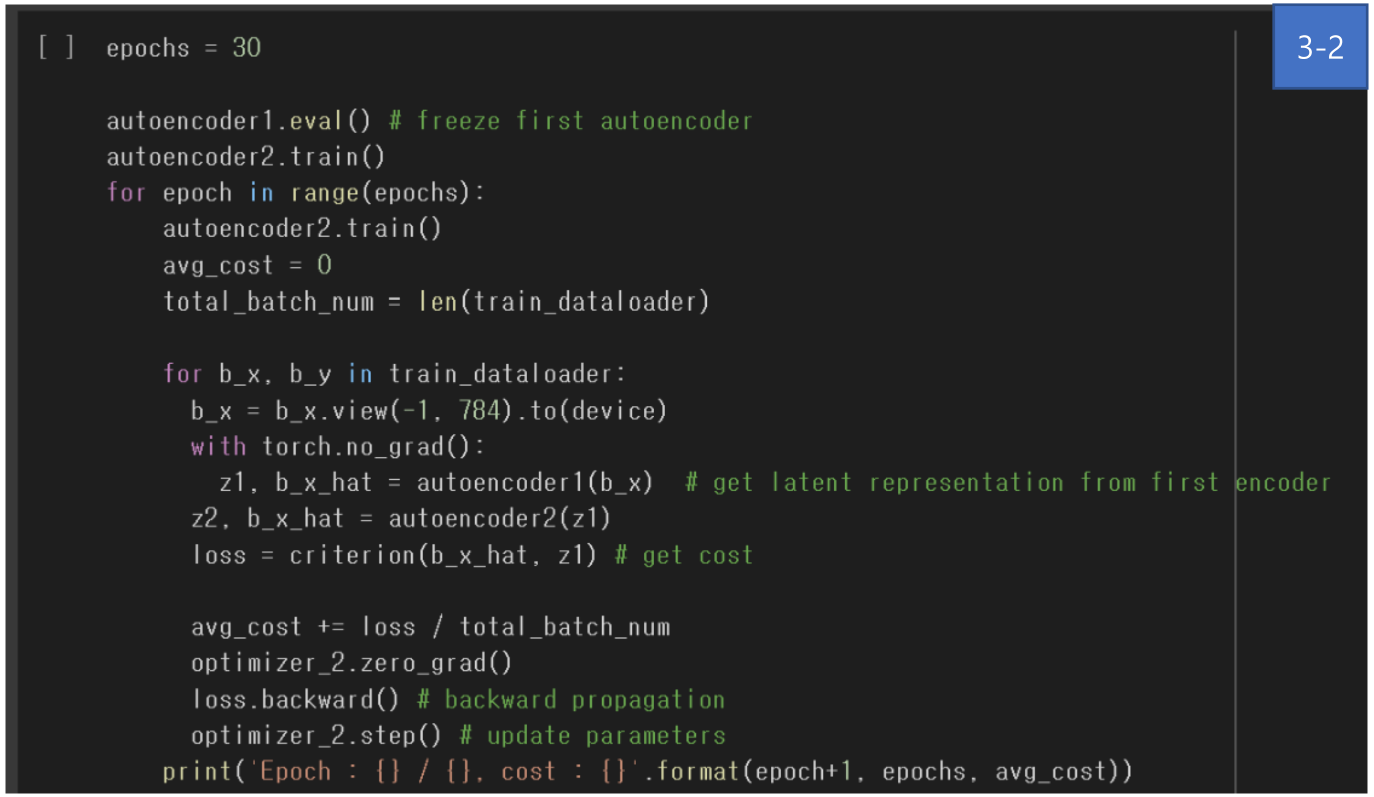
- 첫번째 AutoEncoder 학습


3-1

```
[ ]  epochs = 30

     autoencoder1.train()
     for epoch in range(epochs):
         autoencoder1.train()
         avg_cost = 0
         total_batch_num = len(train_dataloader)

         for b_x, b_y in train_dataloader:
           b_x = b_x.view(-1, 784).to(device)
           z, b_x_hat = autoencoder1(b_x)  # forward propagation
           loss = criterion(b_x_hat, b_x) # get cost

           avg_cost += loss / total_batch_num
           optimizer_1.zero_grad()
           loss.backward() # backward propagation
           optimizer_1.step() # update parameters
         print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

Stacked AutoEncoder

- 두번째 AutoEncoder 학습

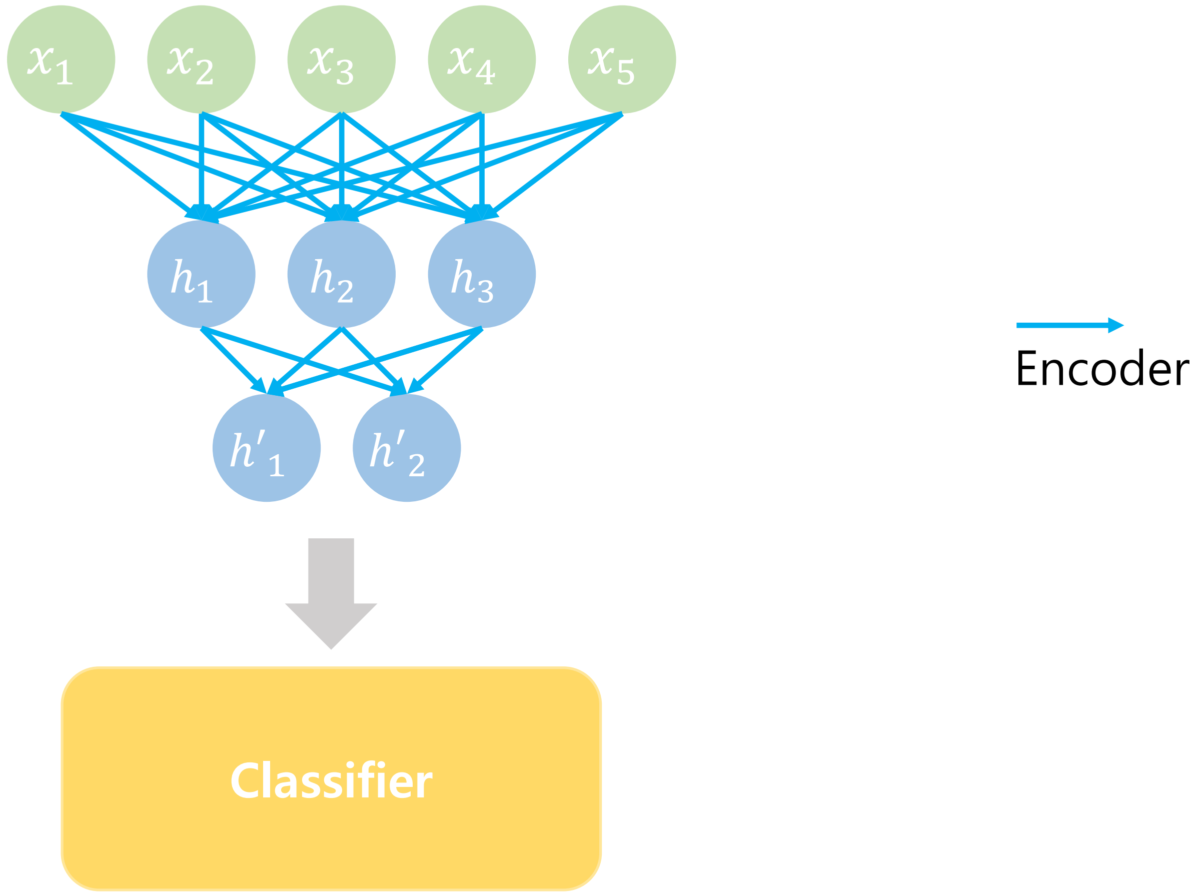


3-2

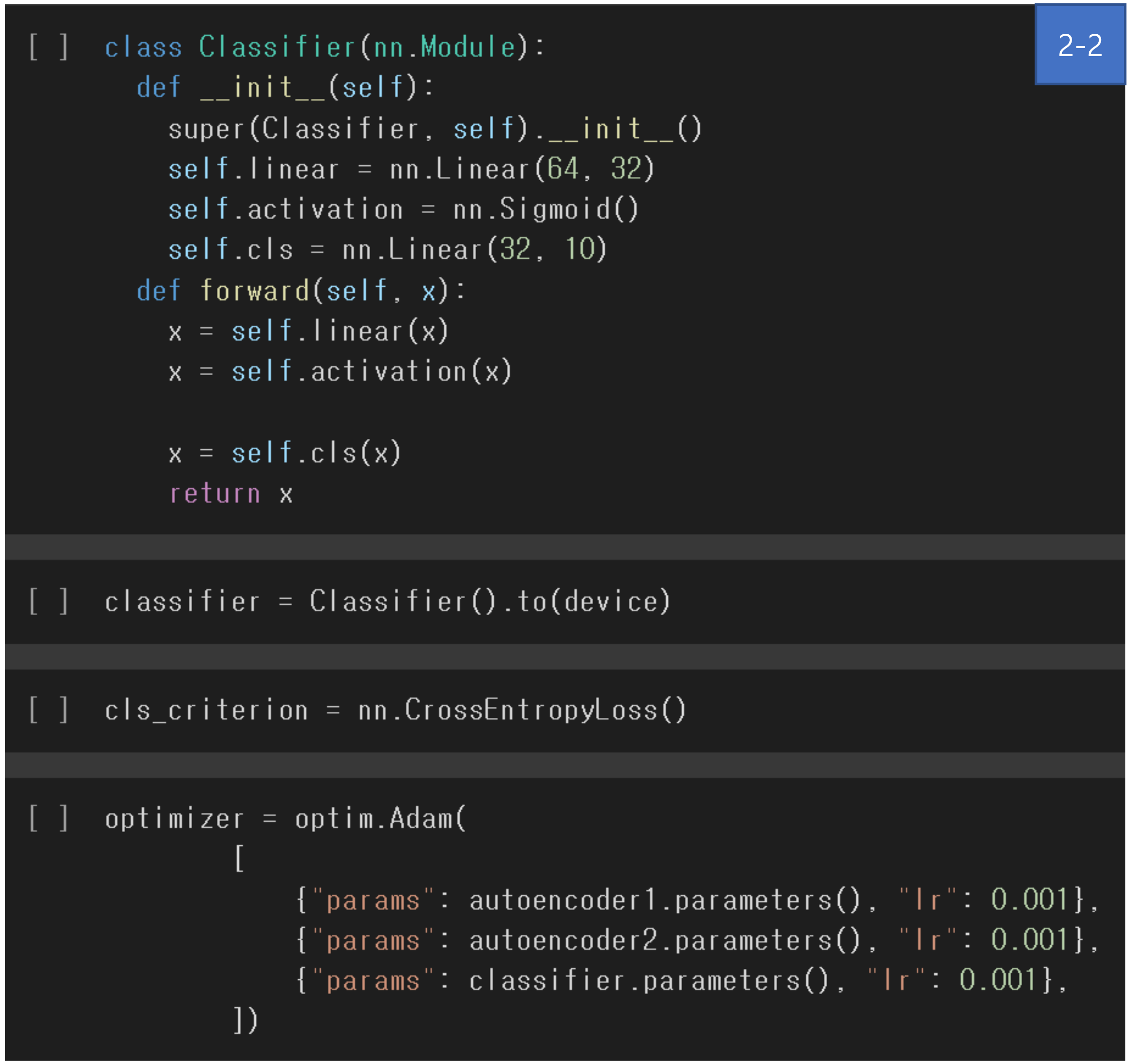
```
[ ]  epochs = 30

     autoencoder1.eval() # freeze first autoencoder
     autoencoder2.train()
     for epoch in range(epochs):
         autoencoder2.train()
         avg_cost = 0
         total_batch_num = len(train_dataloader)

         for b_x, b_y in train_dataloader:
           b_x = b_x.view(-1, 784).to(device)
           with torch.no_grad():
             z1, b_x_hat = autoencoder1(b_x)  # get latent representation from first encoder
           z2, b_x_hat = autoencoder2(z1)
           loss = criterion(b_x_hat, z1) # get cost

           avg_cost += loss / total_batch_num
           optimizer_2.zero_grad()
           loss.backward() # backward propagation
           optimizer_2.step() # update parameters
         print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

Stacked AutoEncoder and Semi-Supervised Learning

- Stacked AutoEncoder and Classifier



Stacked AutoEncoder and Semi-Supervised Learning



2-2

```
[ ]  class Classifier(nn.Module):
       def __init__(self):
         super(Classifier, self).__init__()
         self.linear = nn.Linear(64, 32)
         self.activation = nn.Sigmoid()
         self.cls = nn.Linear(32, 10)
       def forward(self, x):
         x = self.linear(x)
         x = self.activation(x)

         x = self.cls(x)
         return x
```

```
[ ]  classifier = Classifier().to(device)
```

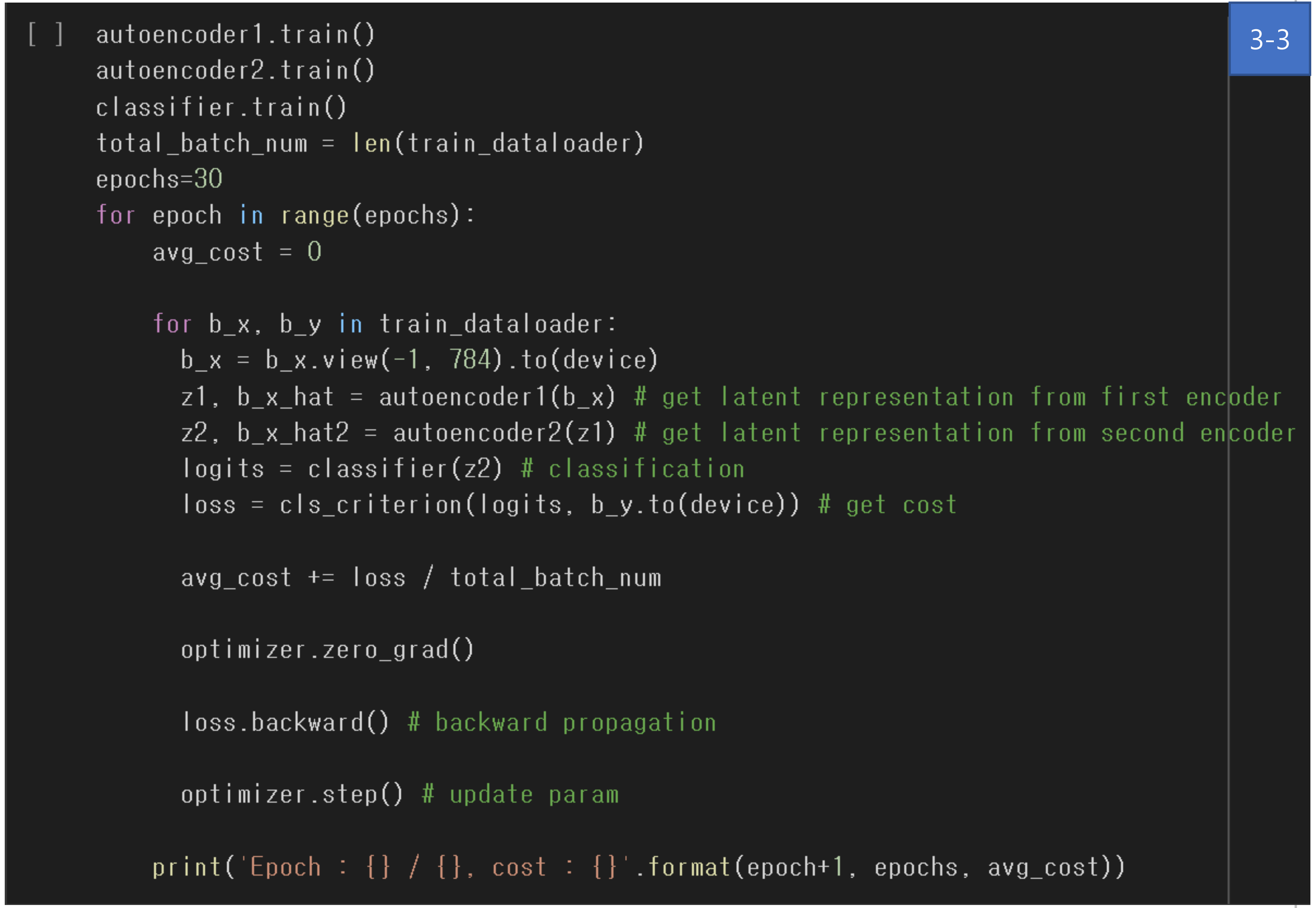
```
[ ]  cls_criterion = nn.CrossEntropyLoss()
```

```
[ ]  optimizer = optim.Adam(
             [
                 {"params": autoencoder1.parameters(), "lr": 0.001},
                 {"params": autoencoder2.parameters(), "lr": 0.001},
                 {"params": classifier.parameters(), "lr": 0.001},
             ])
```

Stacked AutoEncoder and Semi-Supervised Learning

- Classifier 학습
  - **Fine-tuning** autoencoder
  - Fine-tune
    - 미리 학습된 weight을 **task**에 맞게 학습하는 것

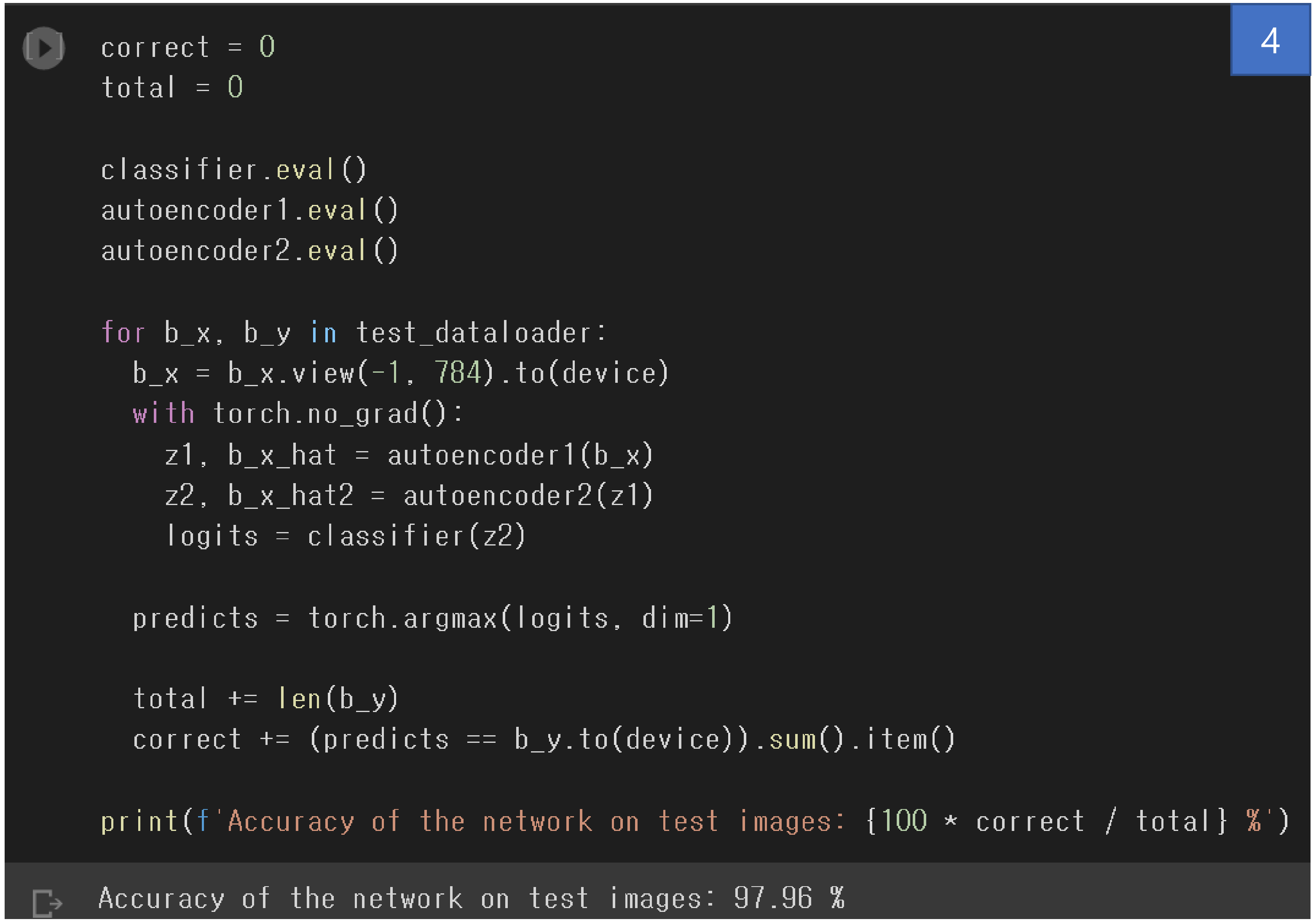
task : classification, object detection, sentence prediction …



3-3

```
[ ]  autoencoder1.train()
     autoencoder2.train()
     classifier.train()
     total_batch_num = len(train_dataloader)
     epochs=30
     for epoch in range(epochs):
         avg_cost = 0

         for b_x, b_y in train_dataloader:
           b_x = b_x.view(-1, 784).to(device)
           z1, b_x_hat = autoencoder1(b_x) # get latent representation from first encoder
           z2, b_x_hat2 = autoencoder2(z1) # get latent representation from second encoder
           logits = classifier(z2) # classification
           loss = cls_criterion(logits, b_y.to(device)) # get cost

           avg_cost += loss / total_batch_num

           optimizer.zero_grad()

           loss.backward() # backward propagation

           optimizer.step() # update param

         print('Epoch : {} / {}, cost : {}'.format(epoch+1, epochs, avg_cost))
```

Stacked AutoEncoder and Semi-Supervised Learning


4

```
correct = 0
total = 0

classifier.eval()
autoencoder1.eval()
autoencoder2.eval()

for b_x, b_y in test_dataloader:
  b_x = b_x.view(-1, 784).to(device)
  with torch.no_grad():
    z1, b_x_hat = autoencoder1(b_x)
    z2, b_x_hat2 = autoencoder2(z1)
    logits = classifier(z2)

  predicts = torch.argmax(logits, dim=1)

  total += len(b_y)
  correct += (predicts == b_y.to(device)).sum().item()

print(f'Accuracy of the network on test images: {100 * correct / total} %')
```

```
Accuracy of the network on test images: 97.96 %
```

오늘 실습 내용

1. AutoEncoder 구현
2. Denoising AutoEncoder 구현
3. Stacked AutoEncoder 구현

- 각 모델의 Latent Representation에 classifier를 붙여 학습해보고 성능 비교해보기