Autoencoder &
Transposed Convolution

2018.02.28

최건호

INDEX

01

Autoencoder

- 정의
- 이유
- 활용

02

Convolution Transposed

- 필요성
- 연산과정
- 실제활용

03

Convolutional Autoencoder

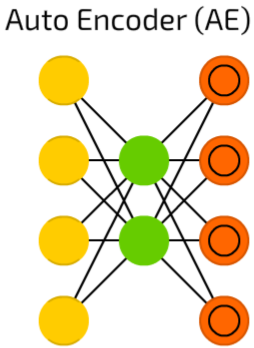
- 전체적 구조
- Denoising CAE

04

Variational Autoencoder

- Intuition
- Variational Inference

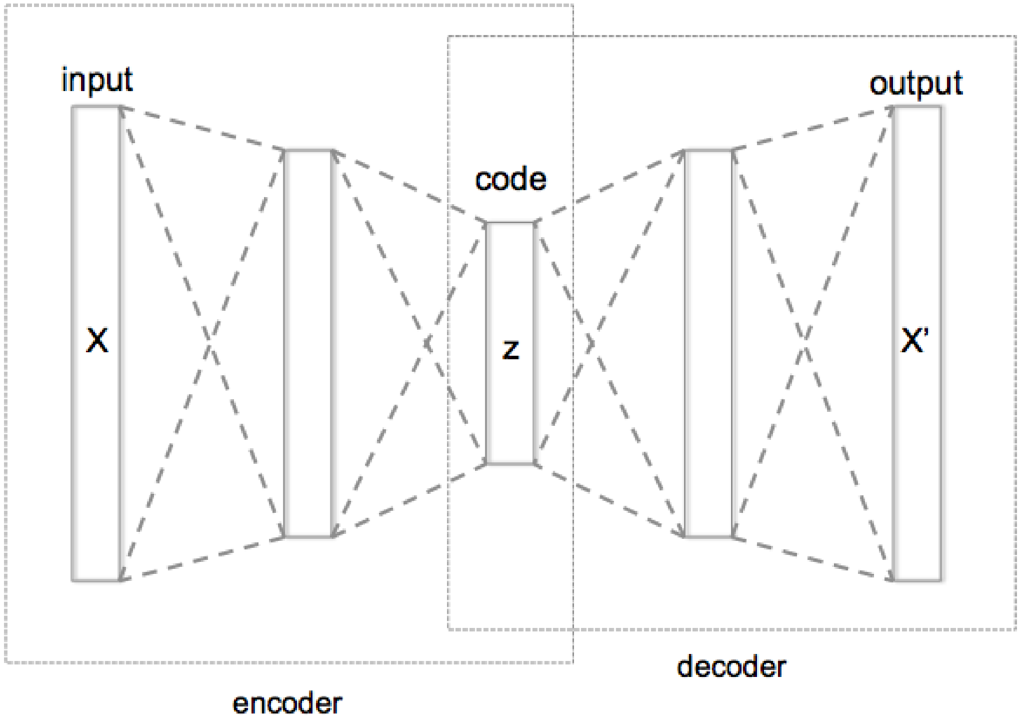
Autoencoder



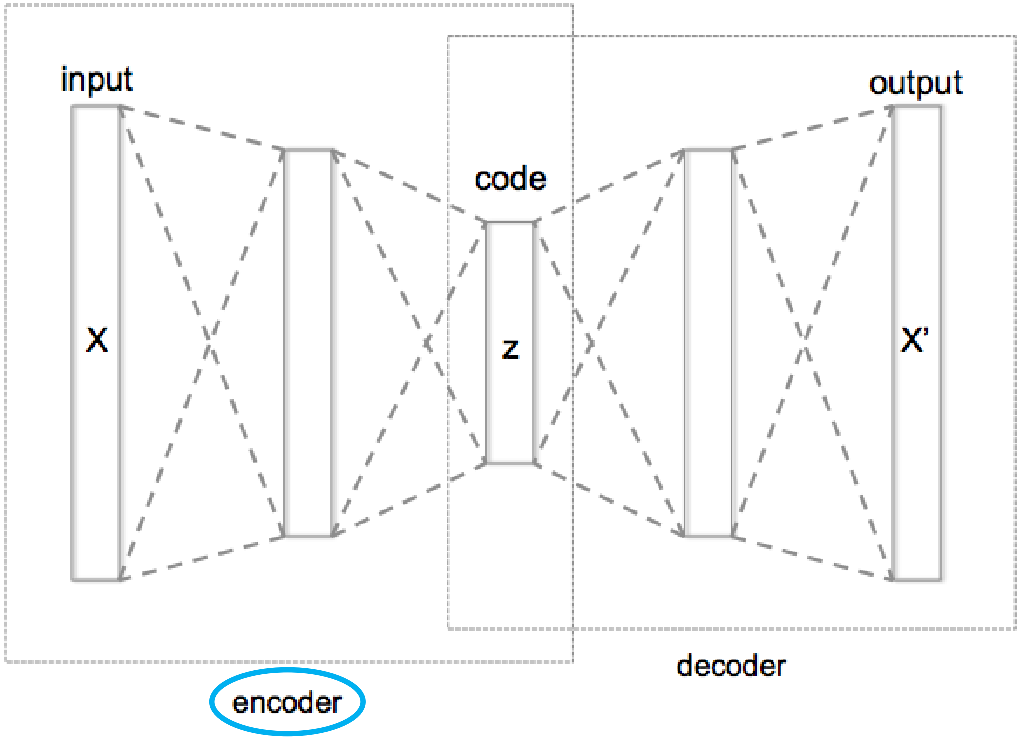
- Unsupervised Learning
- Feature Learning
- Representation Learning
- Efficient Coding
- Dimensionality Reduction

(출처: http://www.asimovinstitute.org/neural-network-zoo/)

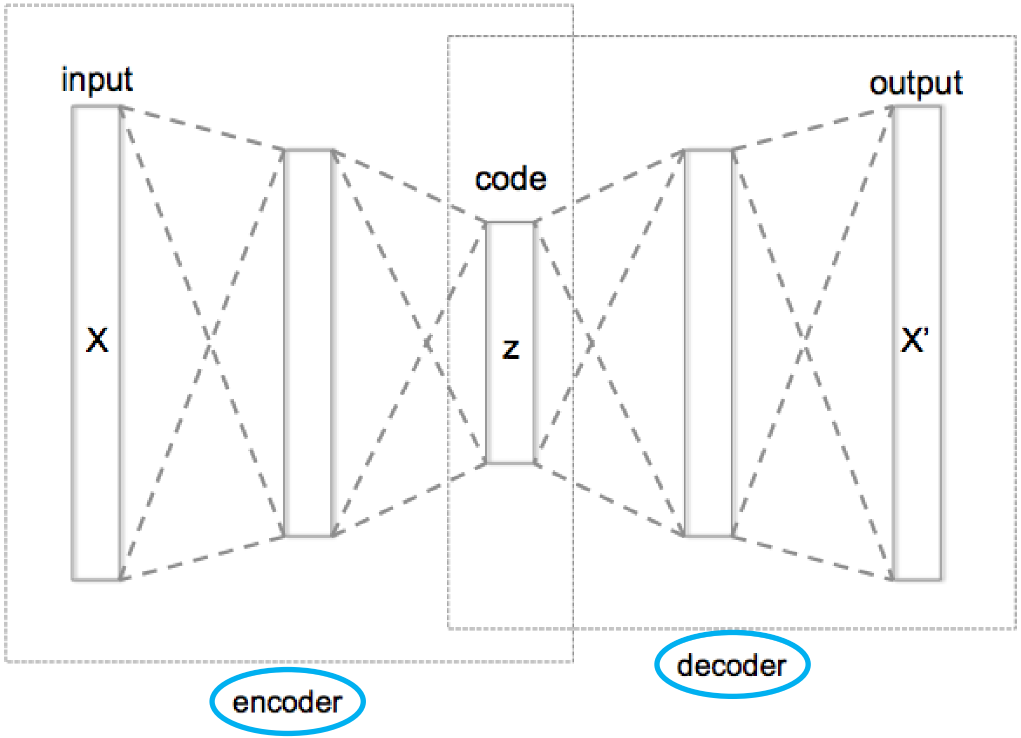
Autoencoder



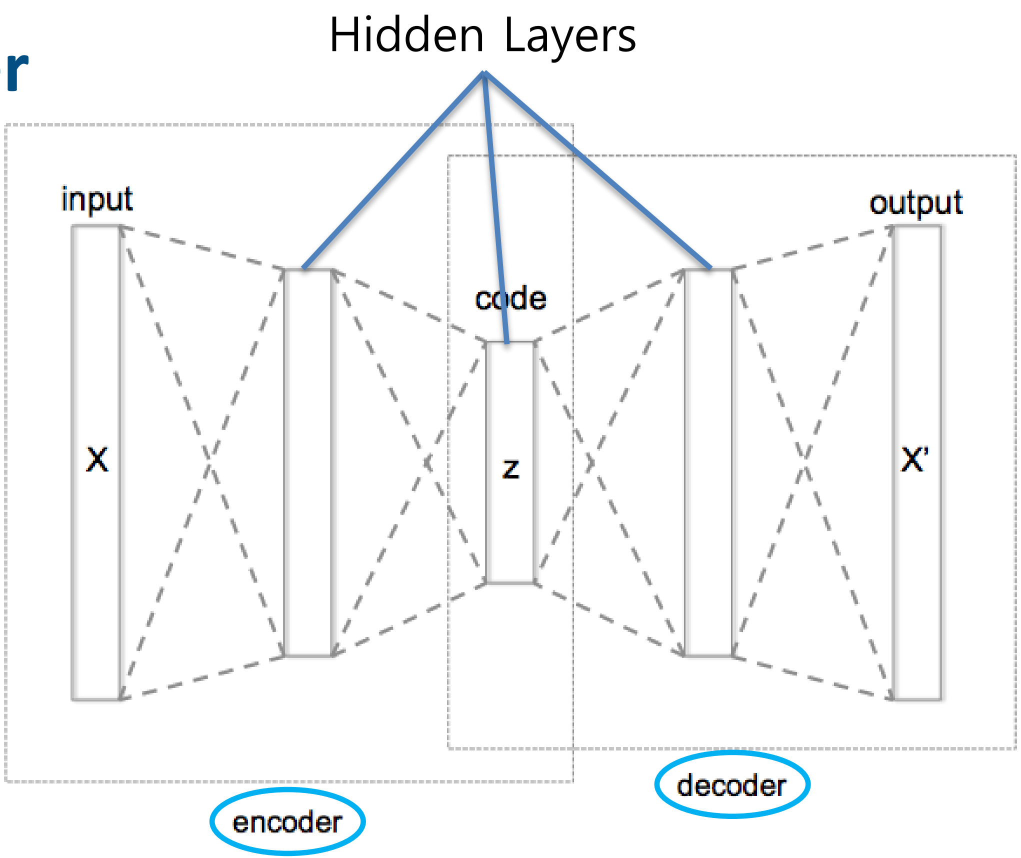
Autoencoder



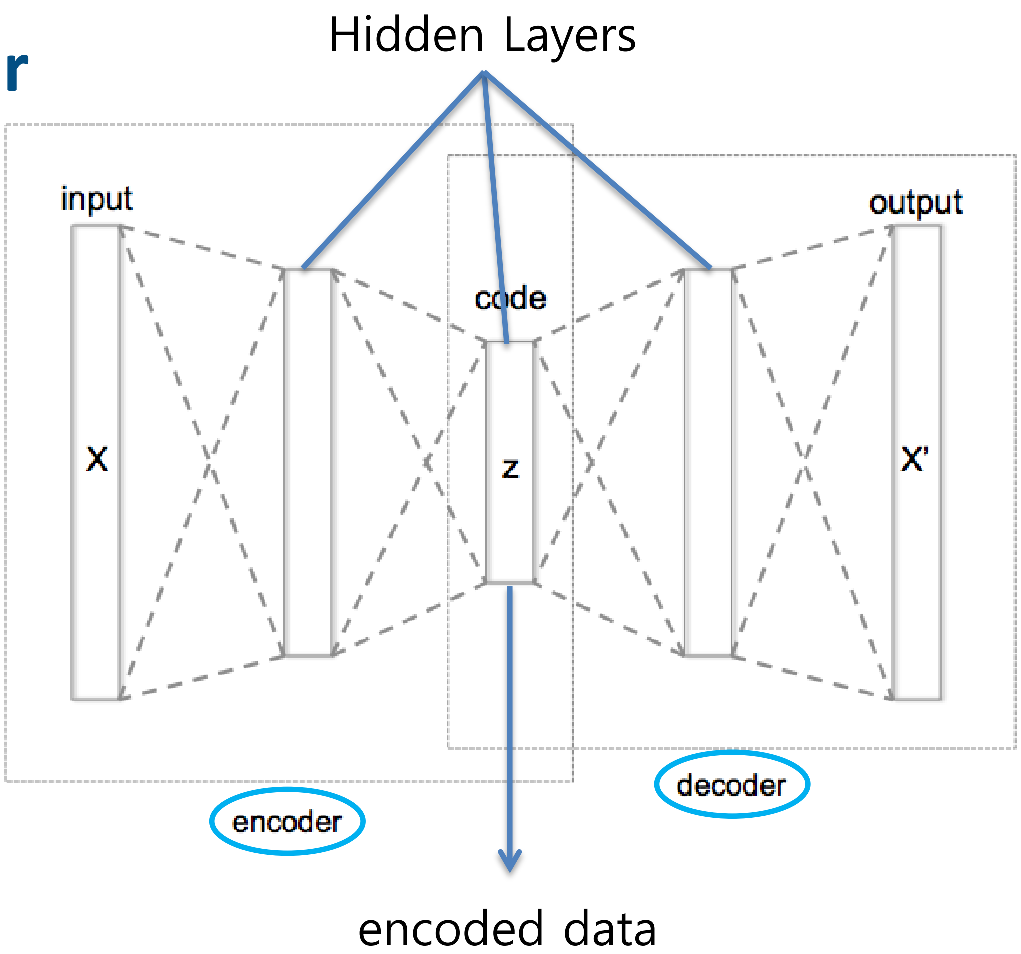
Autoencoder



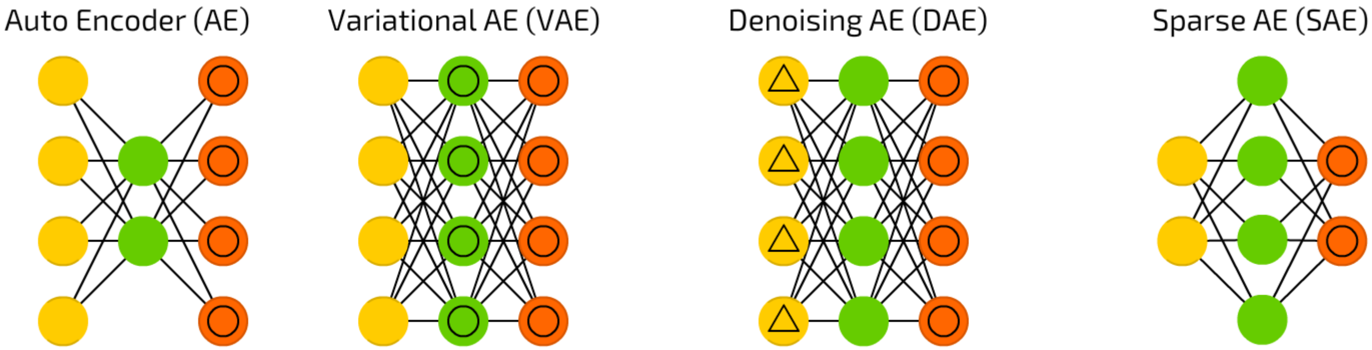
Autoencoder



Autoencoder



Autoencoder



다양한 형태가 있음

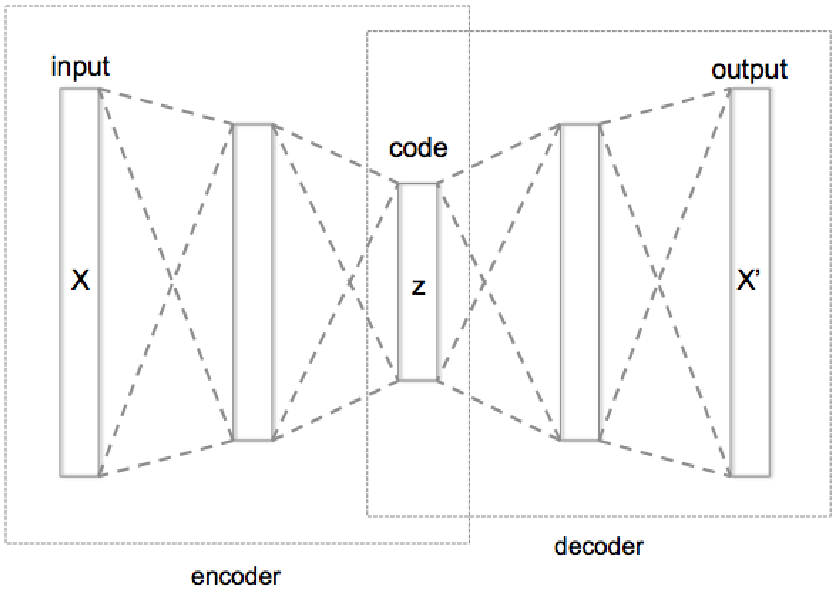
(출처: http://www.asimovinstitute.org/neural-network-zoo/)

Autoencoder

Loss는 어떻게 계산할까?

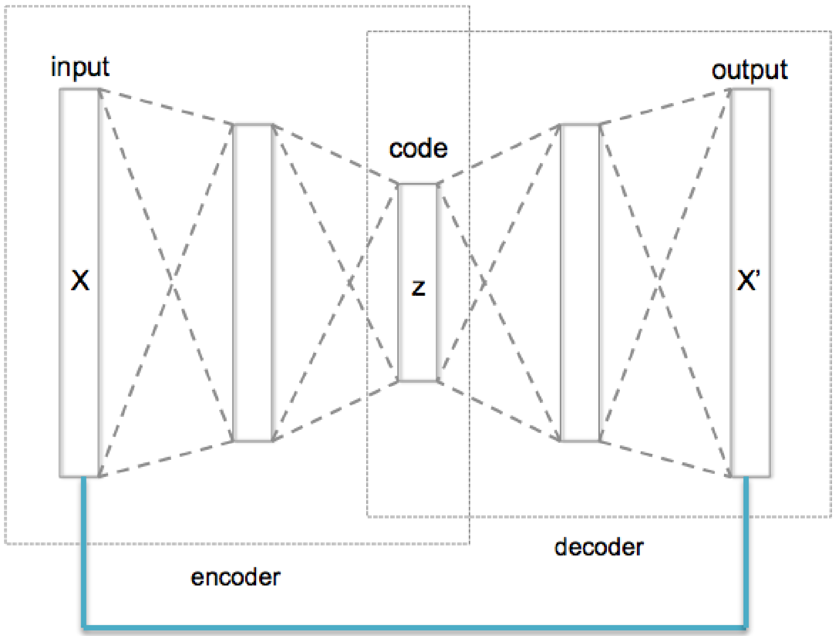
Autoencoder

Loss는 어떻게 계산할까?



Autoencoder

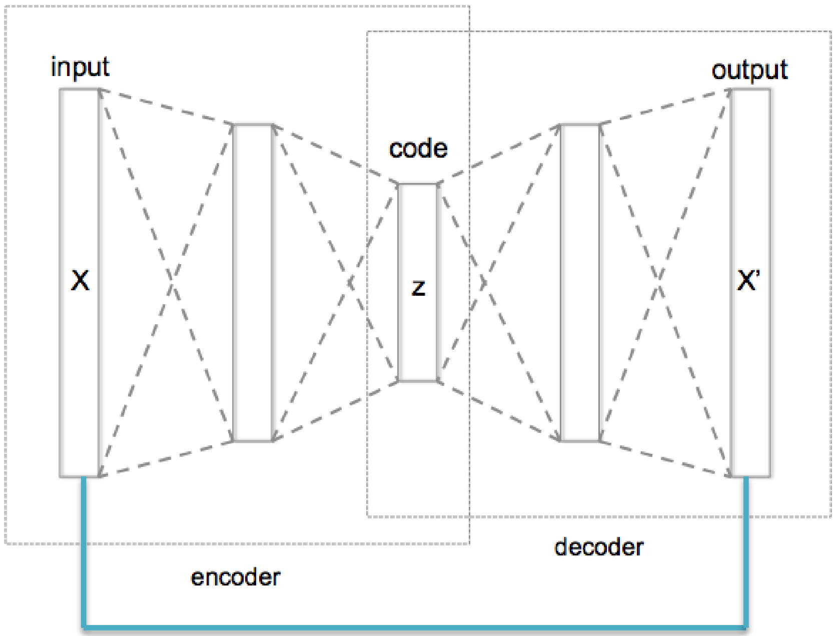
Loss는 어떻게 계산할까?



$$\|x - x'\|$$

Autoencoder

Loss는 어떻게 계산할까?

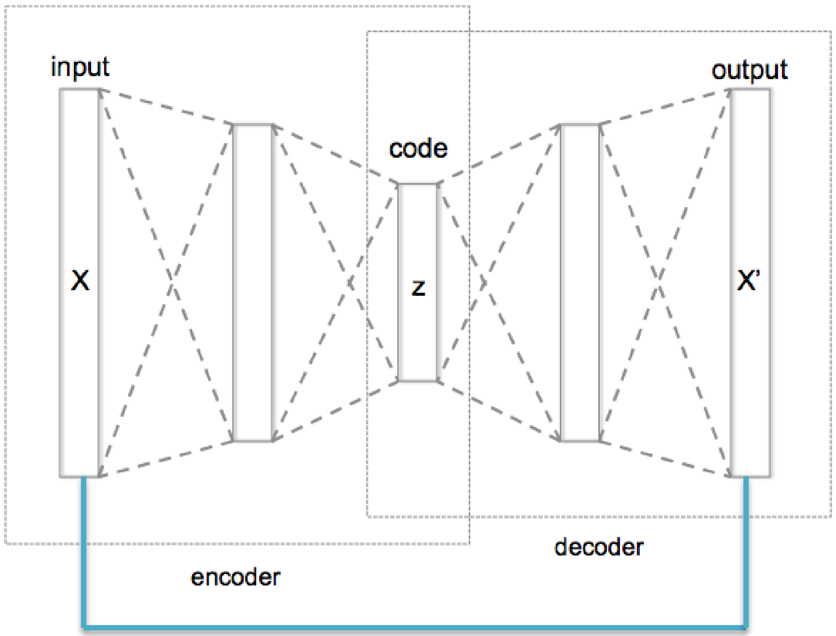


$$\|x - x'\|$$

원본 데이터 x 자체가 라벨의 역할을 하여 reconstructed 된 데이터 또는 decoded 데이터와의 차이로 loss를 계산.

Autoencoder

Loss는 어떻게 계산할까?



$$\|x - x'\|$$

원본 데이터 x 자체가 라벨의 역할을 하여 reconstructed 된 데이터 또는 decoded 데이터와의 차이로 loss를 계산.

L1 loss나 L2 loss를 많이 사용함

Autoencoder

MNIST data

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.encoder = nn.Linear(28*28,50)
        self.decoder = nn.Linear(50,28*28)

    def forward(self,x):
        x = x.view(batch_size,-1)
        encoded = self.encoder(x)
        out = self.decoder(encoded).view(batch_size,1,28,28)

        return out

model = Autoencoder().cuda()
```

Autoencoder

MNIST 데이터는 28x28 이기 때문에 784개의 숫자를 50짜리 latent feature로 encode

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.encoder = nn.Linear(28*28,50)
        self.decoder = nn.Linear(50,28*28)

    def forward(self,x):
        x = x.view(batch_size,-1)
        encoded = self.encoder(x)
        out = self.decoder(encoded).view(batch_size,1,28,28)

        return out

model = Autoencoder().cuda()
```

Autoencoder

MNIST data

MNIST 데이터는 28x28 이기 때문에 784개의 숫자를 50짜리 latent feature로 encode

50개의 latent feature에서 다시 784개로 decode

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.encoder = nn.Linear(28*28,50)
        self.decoder = nn.Linear(50,28*28)

    def forward(self,x):
        x = x.view(batch_size,-1)
        encoded = self.encoder(x)
        out = self.decoder(encoded).view(batch_size,1,28,28)

        return out

model = Autoencoder().cuda()
```

Autoencoder

MNIST data

MNIST 데이터는 28x28 이기 때문에 784개의 숫자를 50짜리 latent feature로 encode

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.encoder = nn.Linear(28*28,50)
        self.decoder = nn.Linear(50,28*28)

    def forward(self,x):
        x = x.view(batch_size,-1)
        encoded = self.encoder(x)
        out = self.decoder(encoded).view(batch_size,1,28,28)

        return out

model = Autoencoder().cuda()
```

50개의 latent feature에서 다시 784개로 decode

모델 구조에 알맞게 데이터를 reshape하여 전달

Autoencoder

```python
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

loss_arr =[]

for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = Variable(image).cuda()

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output,x)
        loss.backward()
        optimizer.step()

    if j % 1000 == 0:
        print(loss)
        loss_arr.append(loss.cpu().data.numpy()[0])
```

Autoencoder

Loss function은 Mean Squared Error

```python
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

loss_arr =[]

for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = Variable(image).cuda()

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output,x)
        loss.backward()
        optimizer.step()

    if j % 1000 == 0:
        print(loss)
        loss_arr.append(loss.cpu().data.numpy()[0])
```

Autoencoder

Loss function은 Mean Squared Error

원본 데이터 x와 모델의 output간의 차이를 통해 loss를 구하고 Back Prop.

```
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

loss_arr =[]

for i in range(num_epoch):
    for j,[image,label] in enumerate(train_loader):
        x = Variable(image).cuda()

        optimizer.zero_grad()
        output = model.forward(x)
        loss = loss_func(output,x)
        loss.backward()
        optimizer.step()

    if j % 1000 == 0:
        print(loss)
        loss_arr.append(loss.cpu().data.numpy()[0])
```

Convolution Transposed

이미지 데이터를 사용할 때는 Encoding, Decoding을 Fully connected layer 말고 Convolution으로 할 수는 없을까?

Convolution Transposed

이미지 데이터를 사용할 때는 Encoding, Decoding을 Fully connected layer 말고 Convolution으로 할 수는 없을까?

⇩

Encoding 부분은 기존의 convolution 연산으로 가능한데 Decoding 부분은 어떻게 하지?

Convolution Transposed

이미지 데이터를 사용할 때는 Encoding, Decoding을 Fully connected layer 말고 Convolution으로 할 수는 없을까?

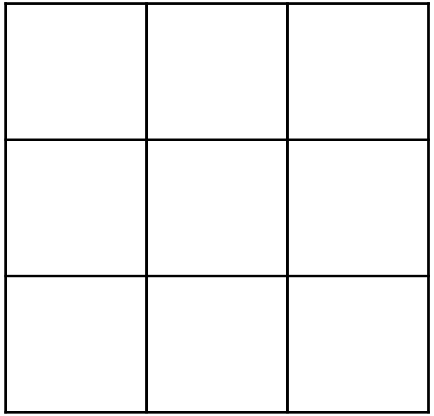
⇩

Encoding 부분은 기존의 convolution 연산으로 가능한데 Decoding 부분은 어떻게 하지?

⇩

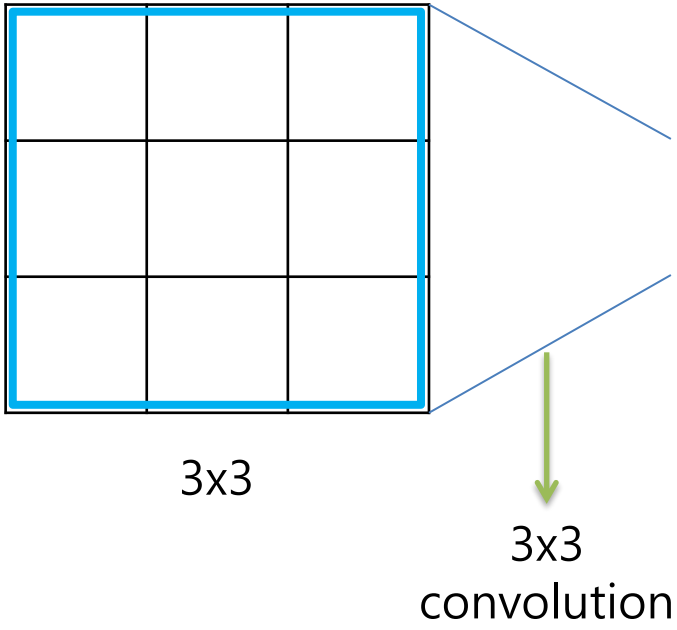
Transposed Convolution!!

Convolution Transposed

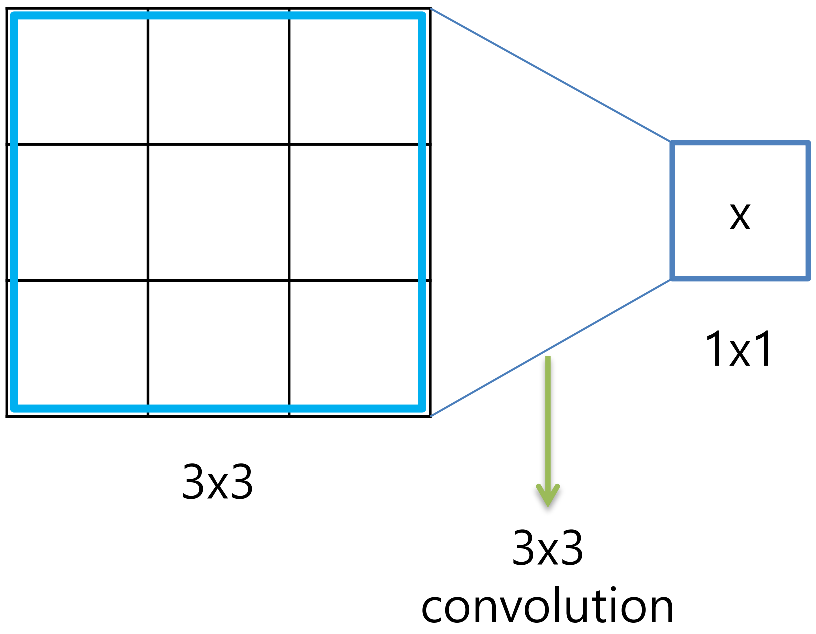


3x3

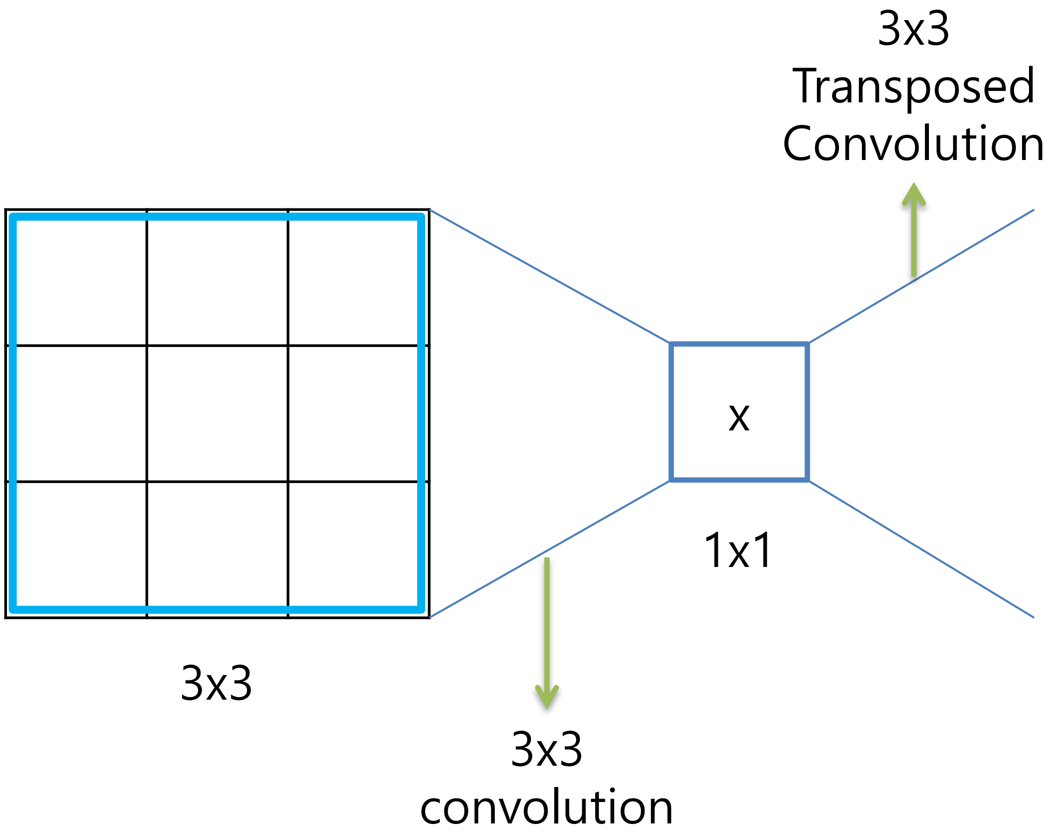
Convolution Transposed



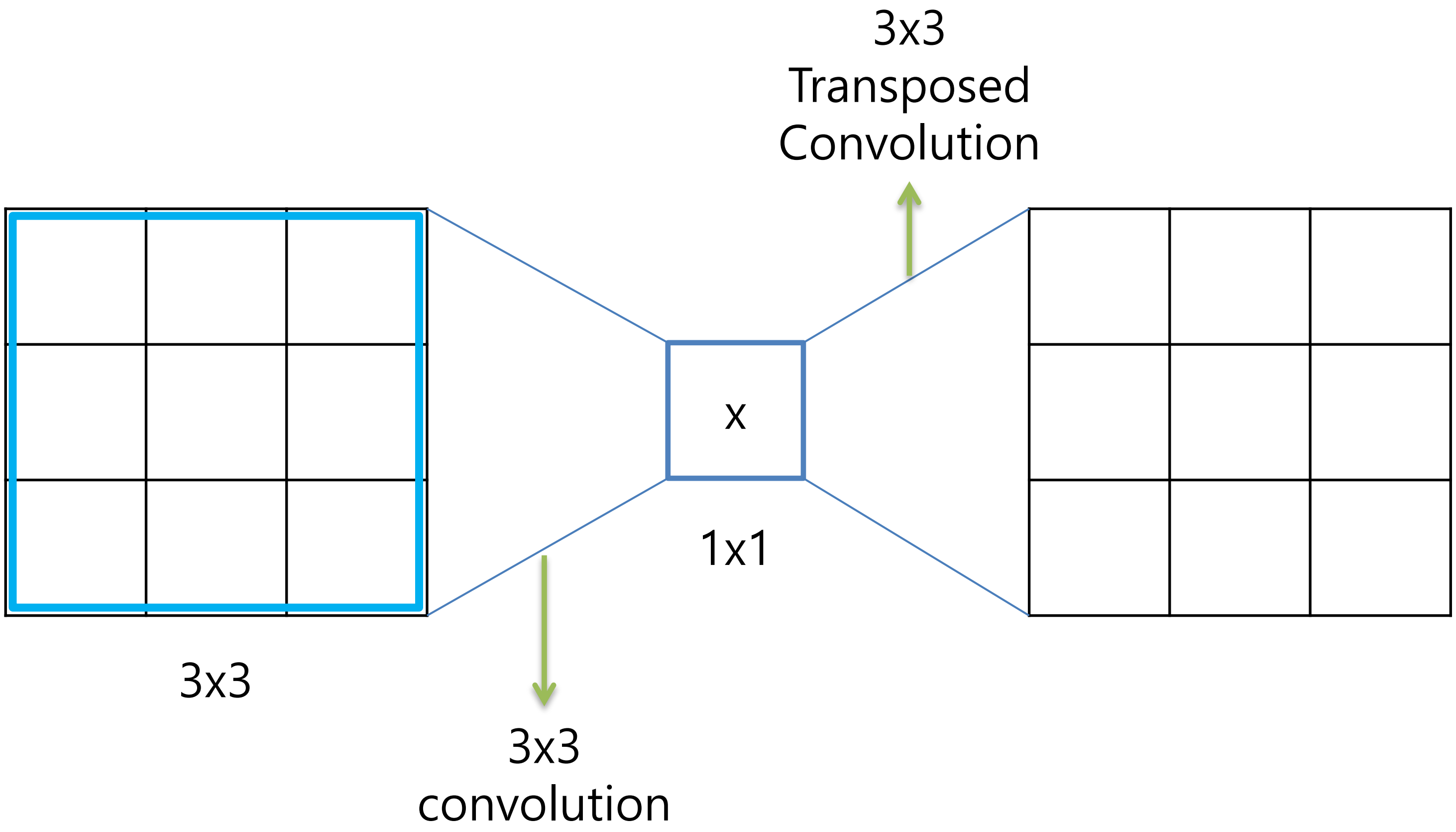
Convolution Transposed



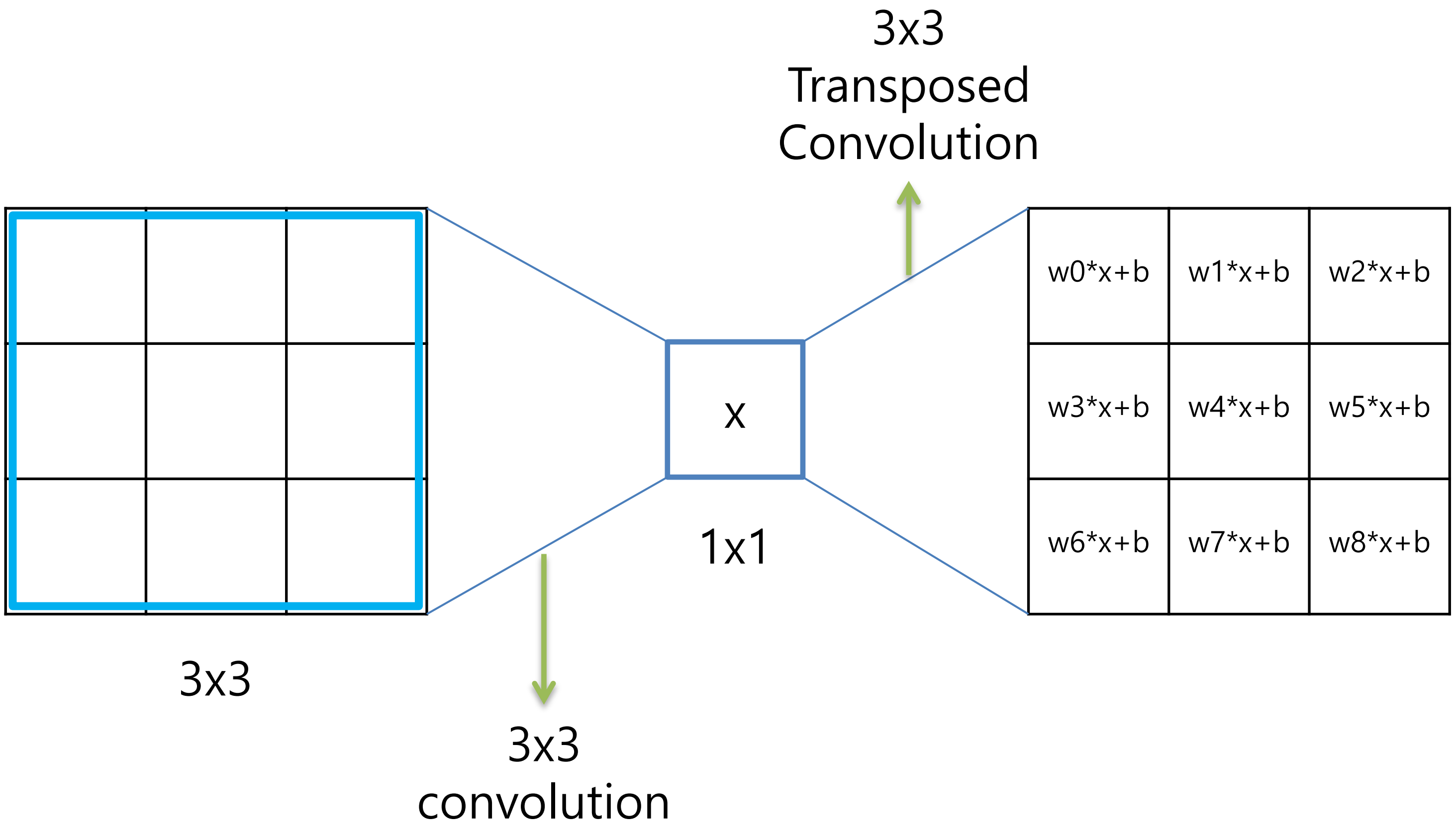
Convolution Transposed



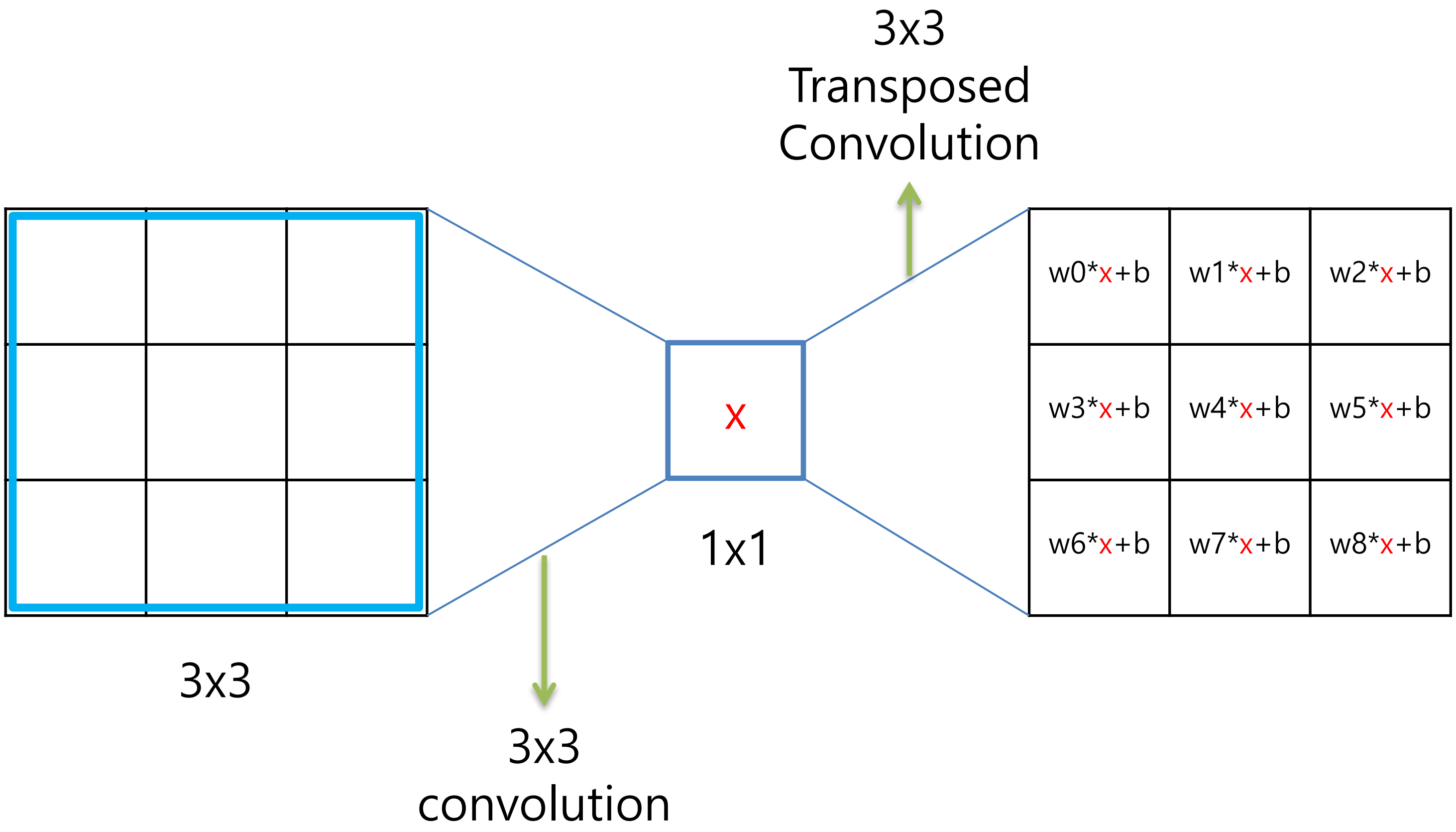
Convolution Transposed



Convolution Transposed

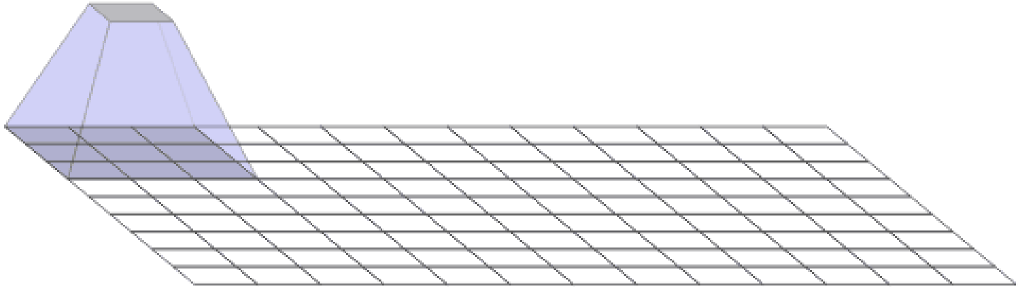


Convolution Transposed



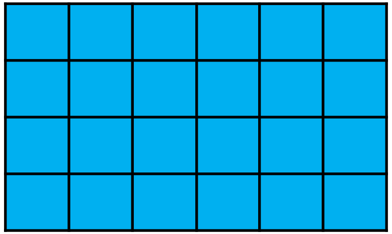
Convolution Transposed

Kernel size 3x3
stride 2



Convolution Transposed

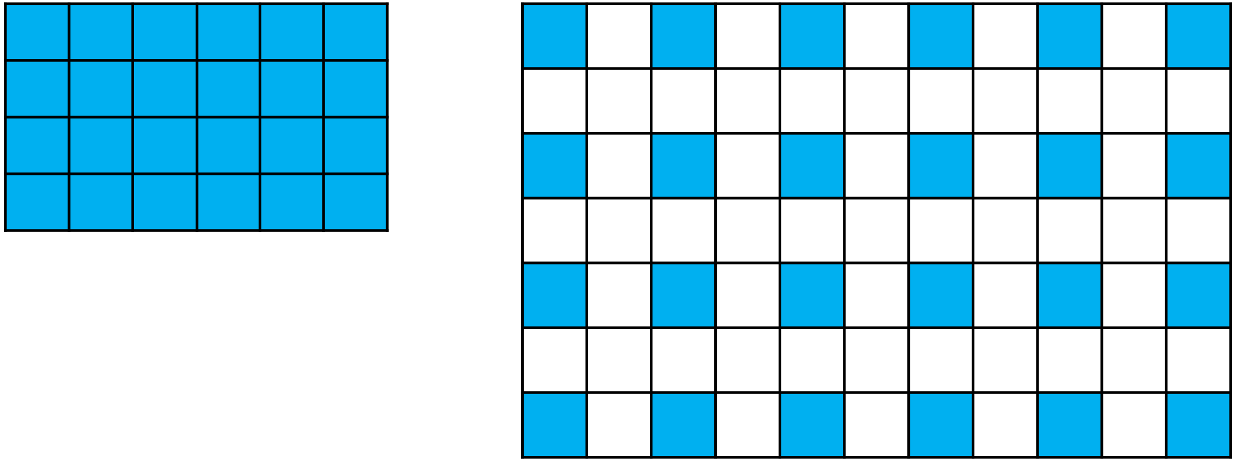
Kernel size 3x3
stride 2



6x4 image

Convolution Transposed

Kernel size 3x3
stride 2



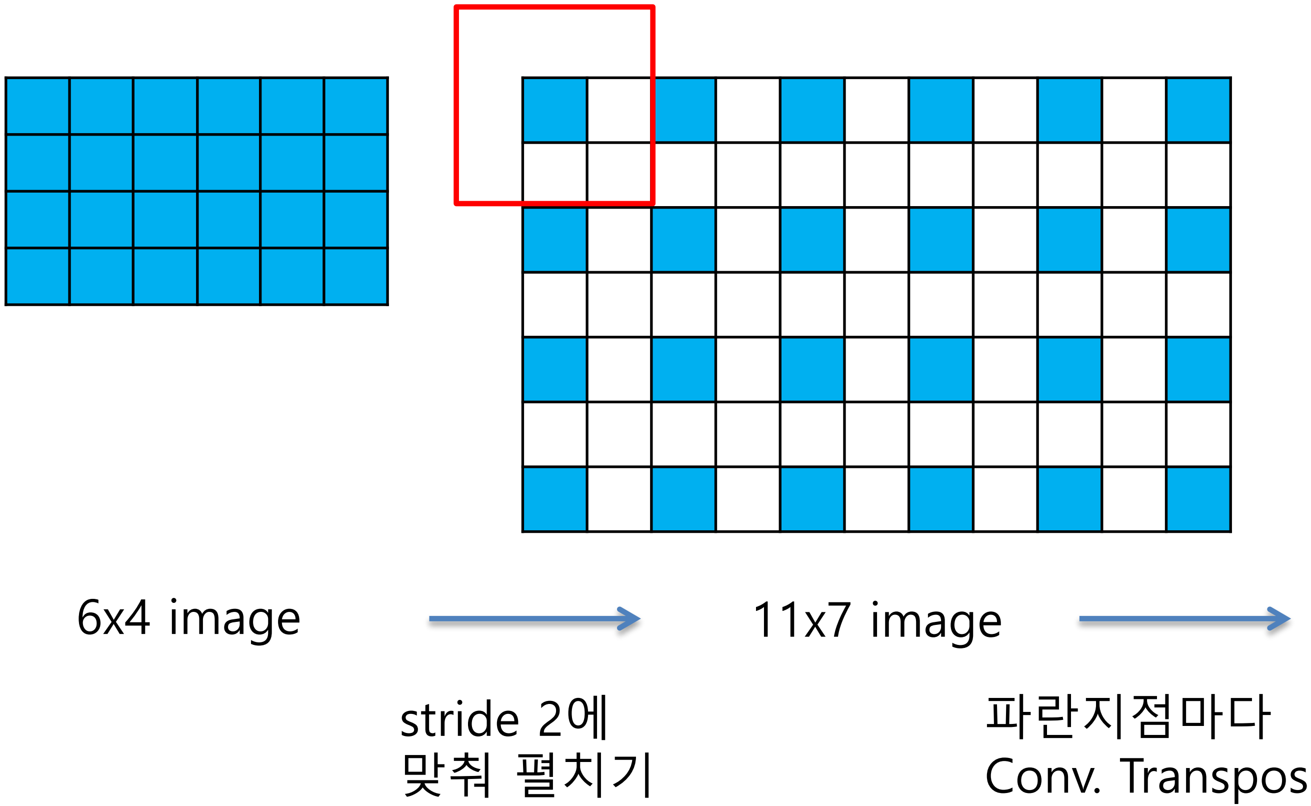
6x4 image

11x7 image

stride 2에
맞춰 펼치기

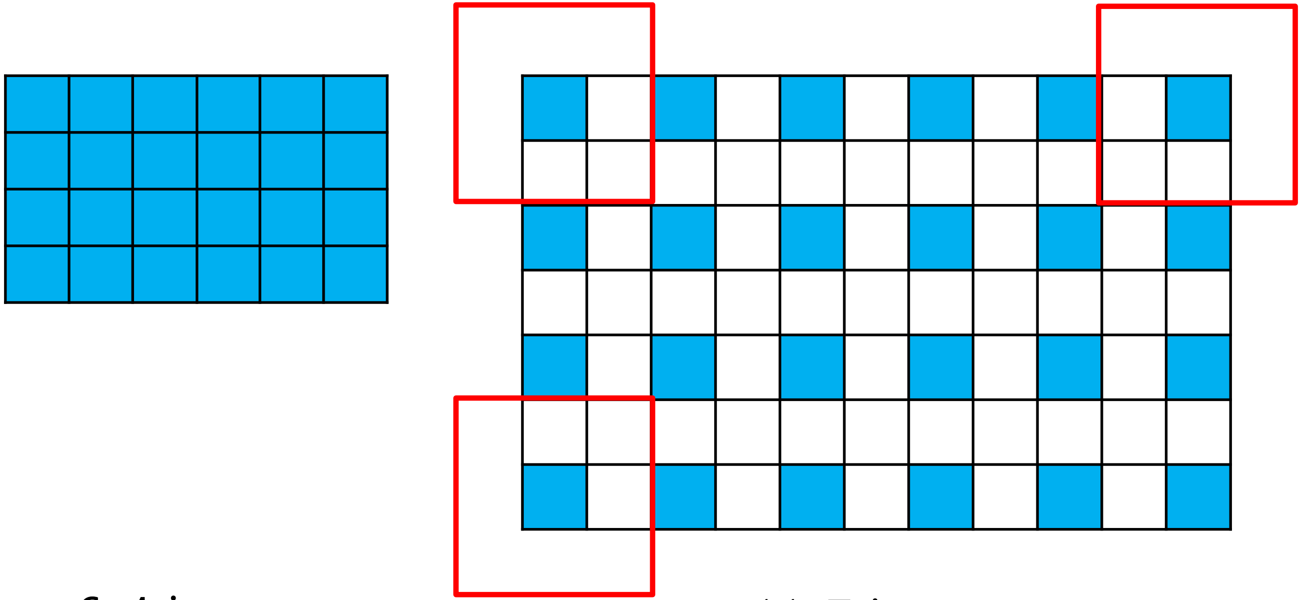
Convolution Transposed

Kernel size 3x3
stride 2



Convolution Transposed

Kernel size 3x3
stride 2



6x4 image → 11x7 image → 13x9 image

stride 2에
맞춰 펼치기

파란지점마다
Conv. Transposed

Convolution Transposed

class `torch.nn.ConvTranspose2d`(*in_channels, out_channels, kernel_size, stride=1, padding=0, output_padding=0, groups=1, bias=True, dilation=1*) [source]

Parameters:

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution
- **padding** (*int or tuple, optional*) – Zero-padding added to both sides of the input
- **output_padding** (*int or tuple, optional*) – Zero-padding added to one side of the output
- **groups** (*int, optional*) – Number of blocked connections from input channels to output channels
- **bias** (*bool, optional*) – If True, adds a learnable bias to the output
- **dilation** (*int or tuple, optional*) – Spacing between kernel elements

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$$
$$W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$$

Convolutional Autoencoder

```python
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.Conv2d(1,16,3,padding=1),    # batch x 16 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.Conv2d(16,32,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(32),
                        nn.Conv2d(32,64,3,padding=1),  # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
                        nn.MaxPool2d(2,2)   # batch x 64 x 14 x 14
        )
        self.layer2 = nn.Sequential(
                        nn.Conv2d(64,128,3,padding=1),  # batch x 64 x 14 x 14
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.MaxPool2d(2,2),
                        nn.Conv2d(128,256,3,padding=1),  # batch x 64 x 7 x 7
                        nn.ReLU()
        )


    def forward(self,x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(batch_size, -1)
        return out

encoder = Encoder().cuda()
```

Convolutional Autoencoder

일반적인 CNN model

```
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.Conv2d(1,16,3,padding=1),    # batch x 16 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.Conv2d(16,32,3,padding=1),   # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(32),
                        nn.Conv2d(32,64,3,padding=1),   # batch x 32 x 28 x 28
                        nn.ReLU(),
                        nn.BatchNorm2d(64),
                        nn.MaxPool2d(2,2)   # batch x 64 x 14 x 14
        )
        self.layer2 = nn.Sequential(
                        nn.Conv2d(64,128,3,padding=1),  # batch x 64 x 14 x 14
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.MaxPool2d(2,2),
                        nn.Conv2d(128,256,3,padding=1), # batch x 64 x 7 x 7
                        nn.ReLU()
        )


    def forward(self,x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(batch_size, -1)
        return out

encoder = Encoder().cuda()
```

Convolutional Autoencoder

```
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Convolutional Autoencoder

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where
  $H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$
  $W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$

Convolutional Autoencoder

[batch,256,7,7] -> [batch,128,14,14]

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where
  $H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$
  $W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$

Convolutional Autoencoder

[batch,256,7,7] -> [batch,128,14,14]

[batch,128,14,14] -> [batch,64,14,14]

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where
  $H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$
  $W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$

Convolutional Autoencoder

[batch,256,7,7] -> [batch,128,14,14]

[batch,128,14,14] -> [batch,64,14,14]

[batch,64,14,14] -> [batch,16,14,14]

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                nn.ConvTranspose2d(256,128,3,2,1,1),
                nn.ReLU(),
                nn.BatchNorm2d(128),
                nn.ConvTranspose2d(128,64,3,1,1),
                nn.ReLU(),
                nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                nn.ConvTranspose2d(64,16,3,1,1),
                nn.ReLU(),
                nn.BatchNorm2d(16),
                nn.ConvTranspose2d(16,1,3,2,1,1),
                nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where
  $H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$
  $W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$

Convolutional Autoencoder

[batch,256,7,7] -> [batch,128,14,14]

[batch,128,14,14] -> [batch,64,14,14]

[batch,64,14,14] -> [batch,16,14,14]

[batch,16,14,14] -> [batch,1,28,28]

```python
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder,self).__init__()
        self.layer1 = nn.Sequential(
                        nn.ConvTranspose2d(256,128,3,2,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128),
                        nn.ConvTranspose2d(128,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64)
        )
        self.layer2 = nn.Sequential(
                        nn.ConvTranspose2d(64,16,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(16),
                        nn.ConvTranspose2d(16,1,3,2,1,1),
                        nn.ReLU()
        )

    def forward(self,x):
        out = x.view(batch_size,256,7,7)
        out = self.layer1(out)
        out = self.layer2(out)
        return out

decoder = Decoder().cuda()
```

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where
  $H_{out} = (H_{in} - 1) * stride[0] - 2 * padding[0] + kernel\_size[0] + output\_padding[0]$
  $W_{out} = (W_{in} - 1) * stride[1] - 2 * padding[1] + kernel\_size[1] + output\_padding[1]$

Convolutional Autoencoder

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

encoder, decoder의 파라미터를 list로 묶어서 학습하도록 전달. loss는 Mean Squared Loss

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

encoder, decoder의 파라미터를 list로 묶어서 학습하도록 전달.
loss는 Mean Squared Loss

기존에 학습시켜 놓은 모델이 있다면 불러오고 아니면 pass

```python
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

encoder, decoder의 파라미터를
list로 묶어서 학습하도록 전달.
loss는 Mean Squared Loss

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)
```

기존에 학습시켜 놓은 모델이 있
다면 불러오고 아니면 pass

```
# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass
```

원본 이미지를 encoding,
decoding하여 결과값과
원본의 차이를 계산

```
for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

encoder, decoder의 파라미터를 list로 묶어서 학습하도록 전달. loss는 Mean Squared Loss

기존에 학습시켜 놓은 모델이 있다면 불러오고 아니면 pass

원본 이미지를 encoding, decoding하여 결과값과 원본의 차이를 계산

일정 기간마다 모델을 저장

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

```python
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

Denoising Autoencoder 같은 경우에는 원본 이미지에 noise를 추가하여 autoencoder를 통과하면 노이즈가 제거된 원본 이미지가 나오도록 학습함.

```
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

Denoising Autoencoder 같은 경우에는 원본 이미지에 noise를 추가하여 autoencoder를 통과하면 노이즈가 제거된 원본 이미지가 나오도록 학습함.

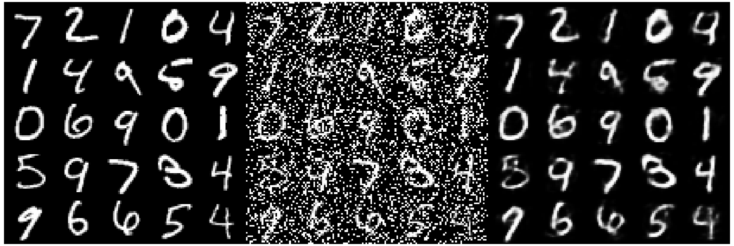
이렇게 되면 학습 이후 좀 지저분한 데이터가 들어오더라도 정제할 수 있음

```python
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

Denoising Autoencoder 같은 경우에는 원본 이미지에 noise를 추가하여 autoencoder를 통과하면 노이즈가 제거된 원본 이미지가 나오도록 학습함.

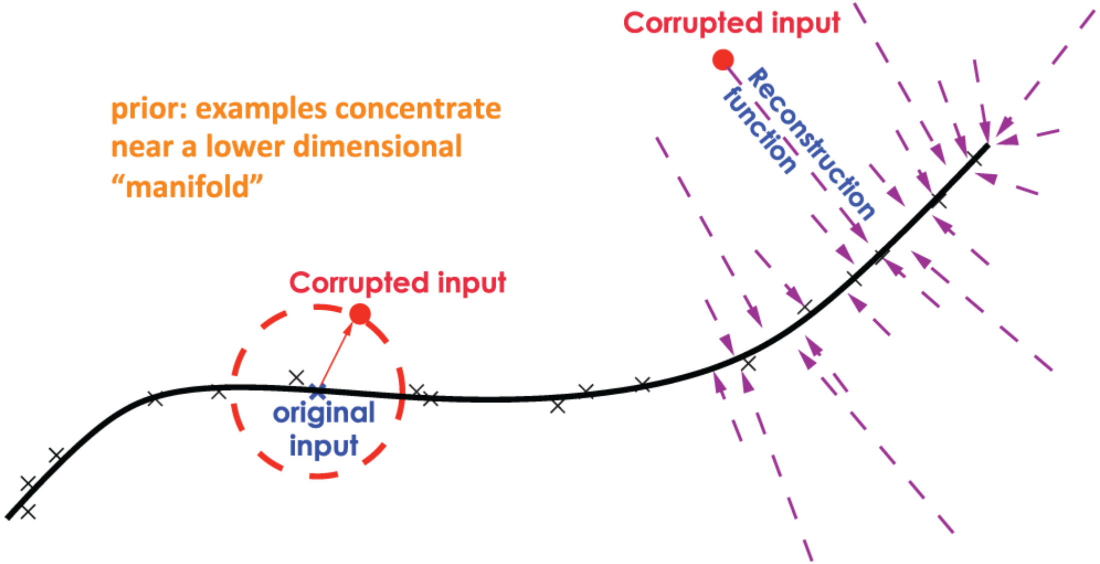
이렇게 되면 학습 이후 좀 지저분한 데이터가 들어오더라도 정제할 수 있음



```python
parameters = list(encoder.parameters())+ list(decoder.parameters())
loss_func = nn.MSELoss()
optimizer = torch.optim.Adam(parameters, lr=learning_rate)

# train encoder and decoder

try:
    encoder, decoder = torch.load('./model/autoencoder.pkl')
    print("\n--------model restored--------\n")
except:
    pass

for i in range(epoch):
    for image,label in train_loader:
        image = Variable(image).cuda()
        optimizer.zero_grad()
        output = encoder(image)
        output = decoder(output)
        loss = loss_func(output,image)
        loss.backward()
        optimizer.step()

    if i % 2 == 0:
        torch.save([encoder,decoder],'./model/autoencoder.pkl')
        print(loss)
```

Convolutional Autoencoder

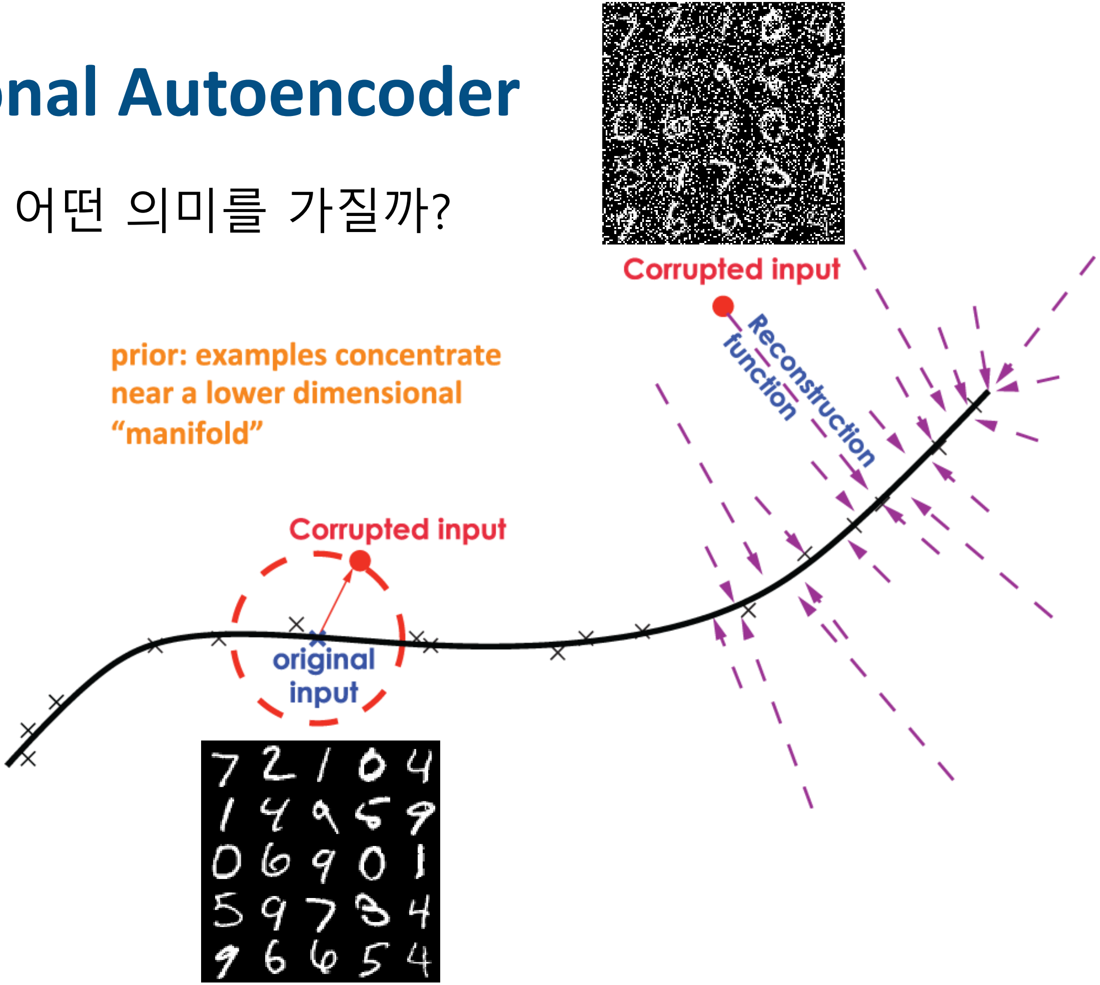
Denoising은 어떤 의미를 가질까?



https://www.slideshare.net/zukun/icml2012-tutorial-representationlearning

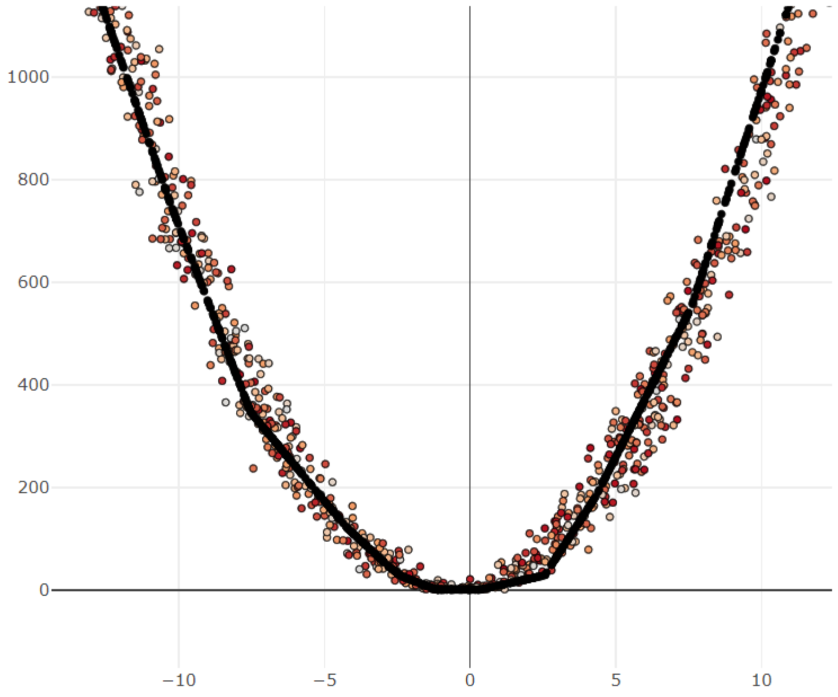
Convolutional Autoencoder

Denoising은 어떤 의미를 가질까?



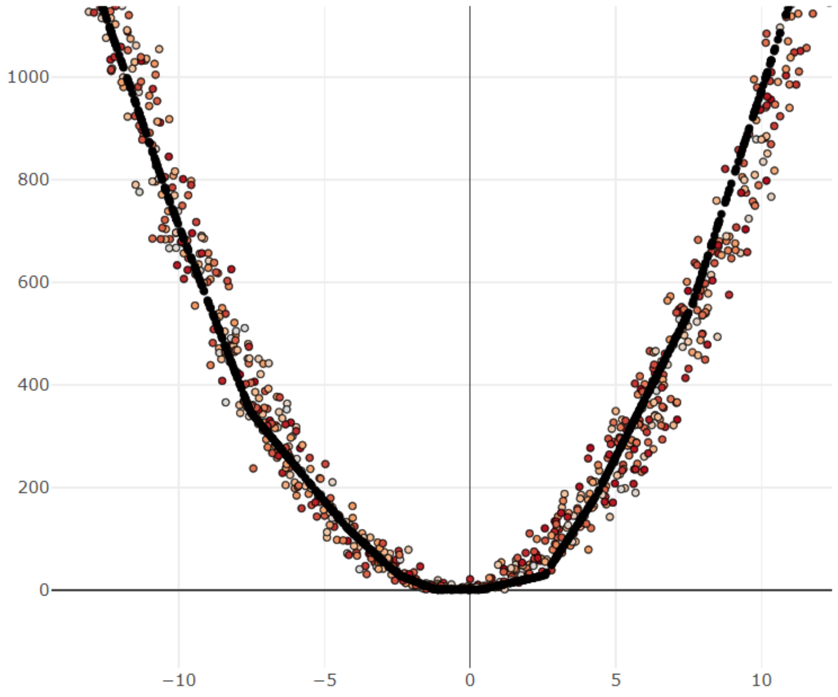
https://www.slideshare.net/zukun/icml2012-tutorial-representationlearning

Convolutional Autoencoder



Convolutional Autoencoder

Noise에 강하게(robust) 학습됨
Filter들도 Clean 데이터보다 더 선명하게 생성됨



https://www.slideshare.net/zukun/icml2012-tutorial-representationlearning

Q&A