Optimization, VAE, and GAN

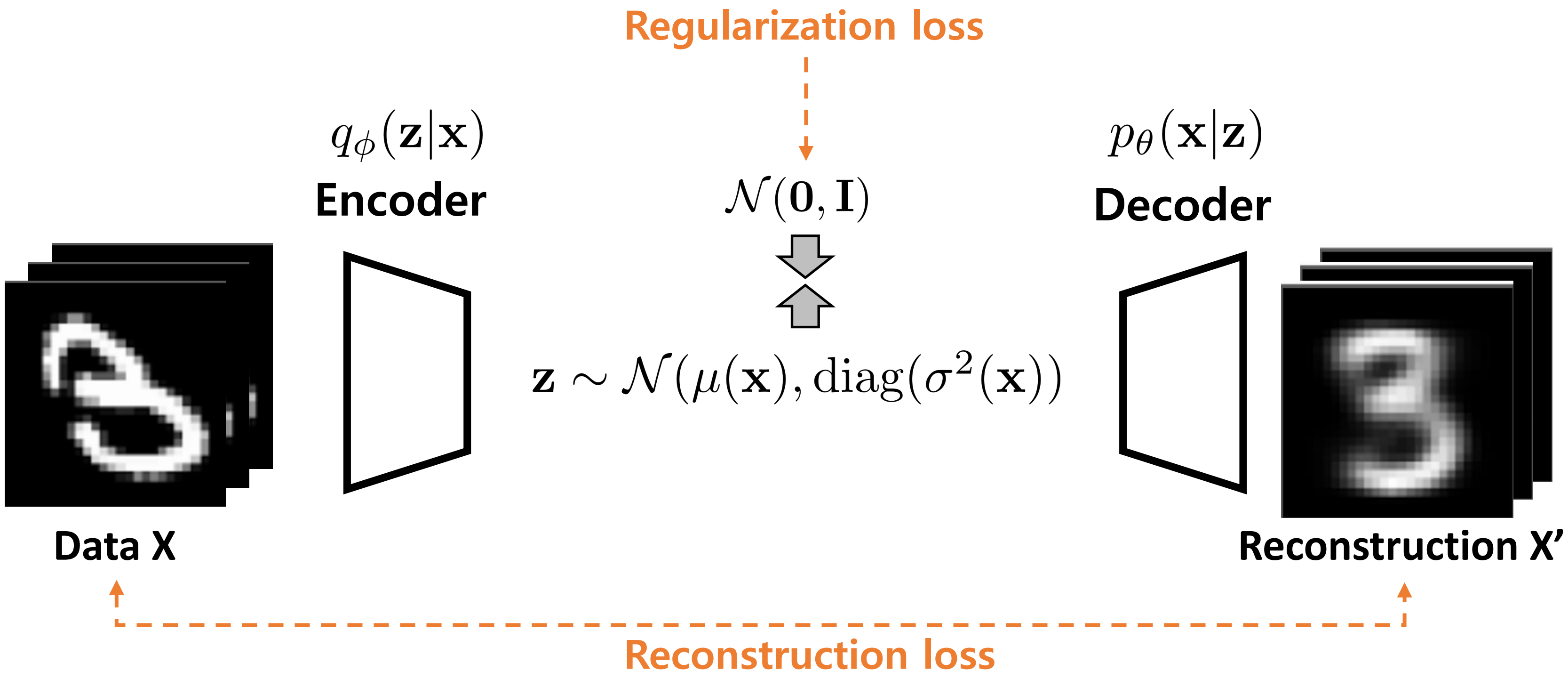
Seanie Lee
MLAI, KAIST
lsnfamily@kaist.ac.kr

In this tutorial,

- We do not cover optimization – Choose whatever optimizer you want.
- Train VAE and GAN to generate MNIST digits.
- Code is available at : https://github.com/seanie12/vae_gan

Variational Auto-Encoder (VAE)

VAE Overview



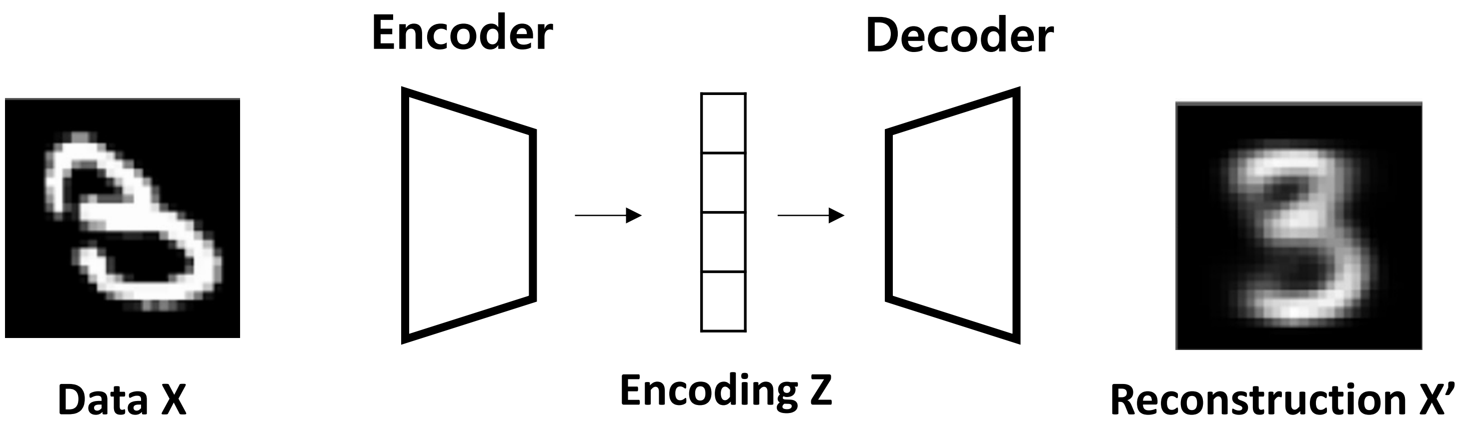
Implementation TODO

- Encoder & Decoder Network
- Loss function
- Training Loop
- Visualization

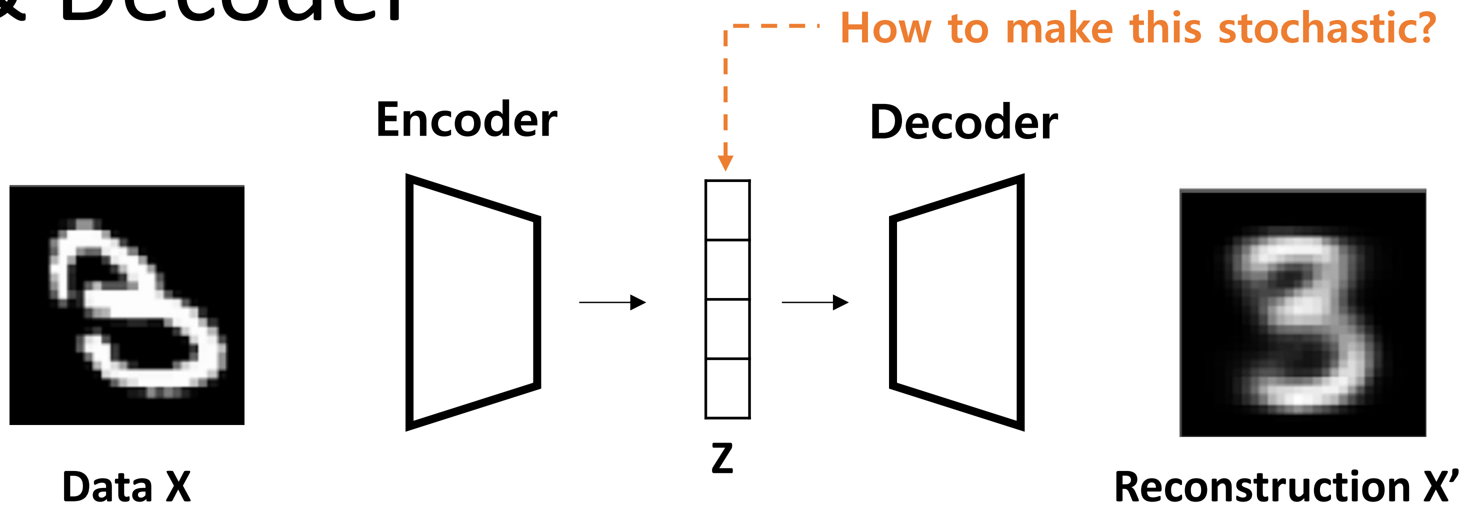
Implementation TODO

- **Encoder & Decoder Network**
- Loss function
- Training Loop
- Visualization

Encoder & Decoder

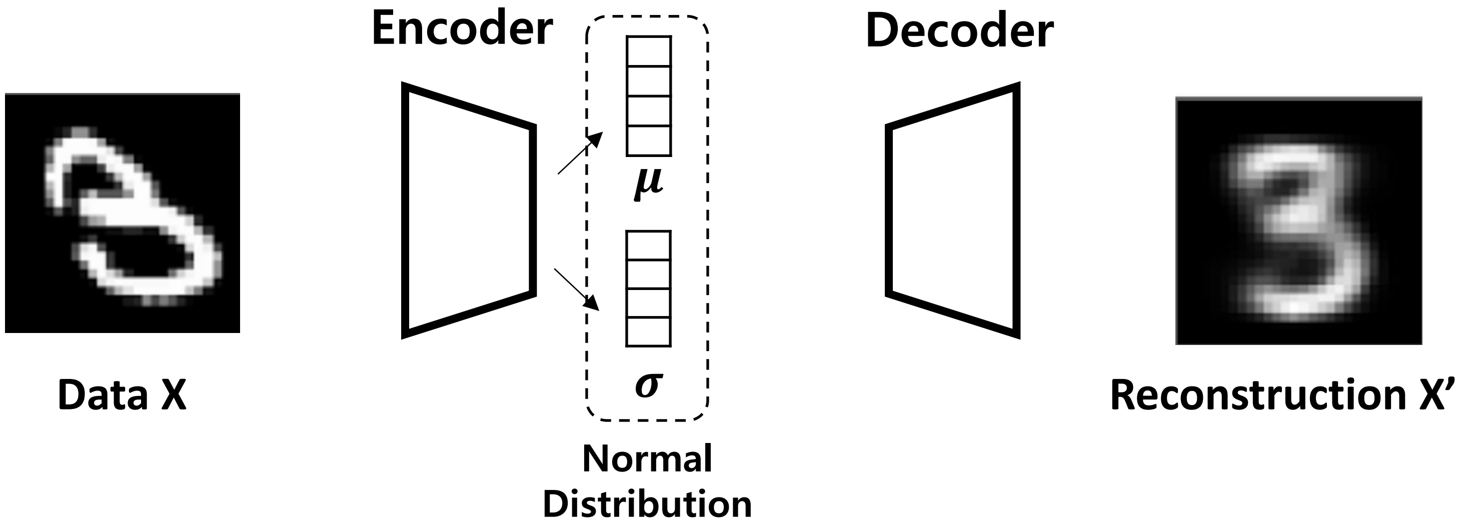


Encoder & Decoder

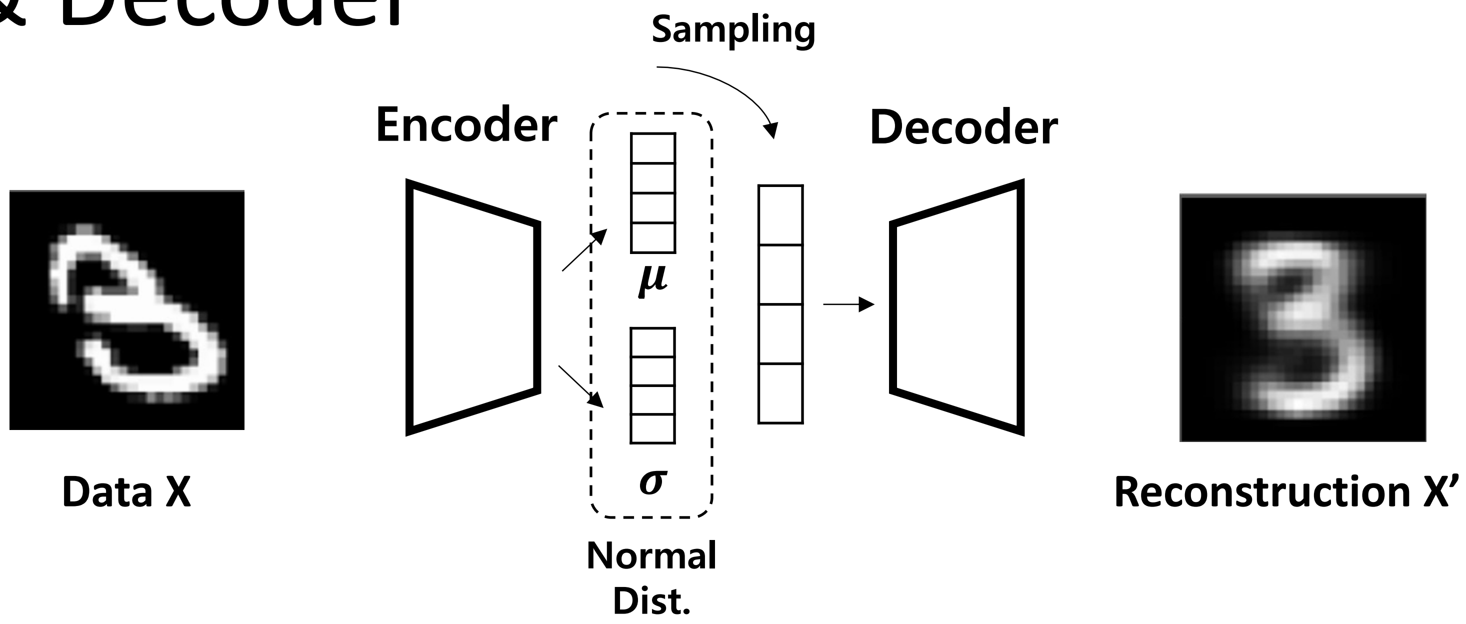


MLAI
Machine Learning & Artificial Intelligence

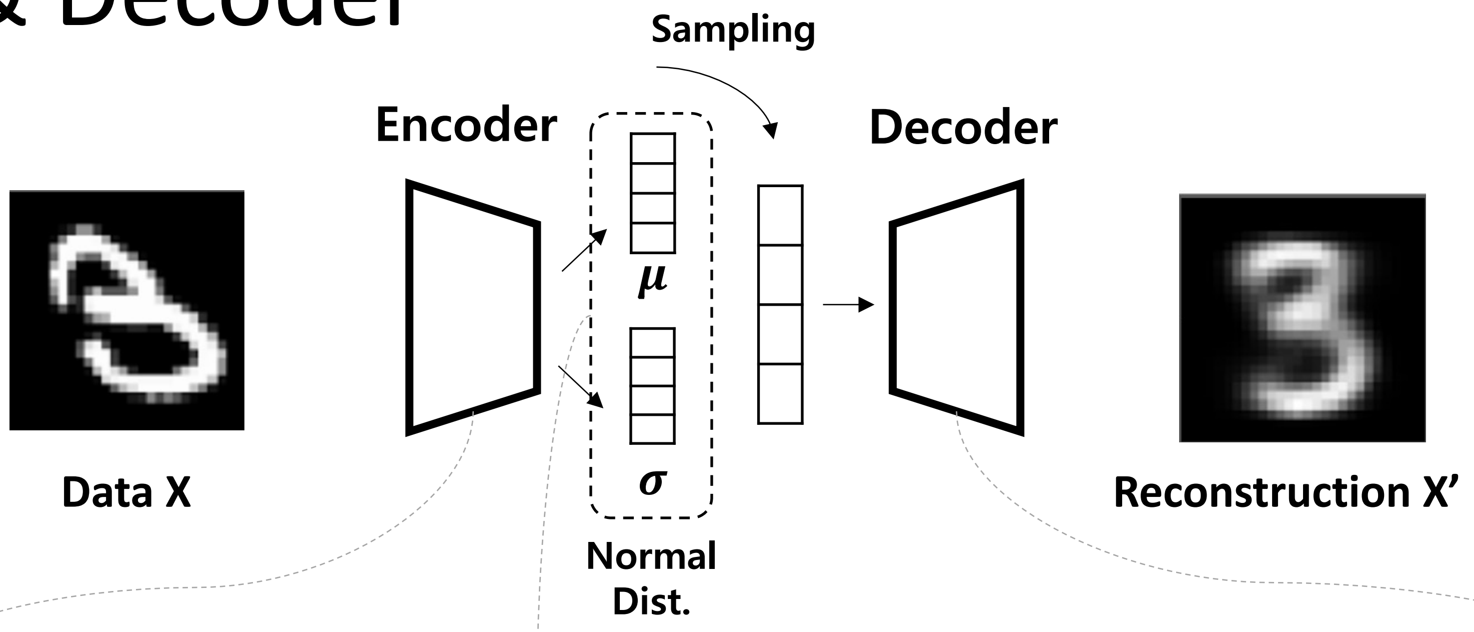
Encoder & Decoder



Encoder & Decoder



Encoder & Decoder

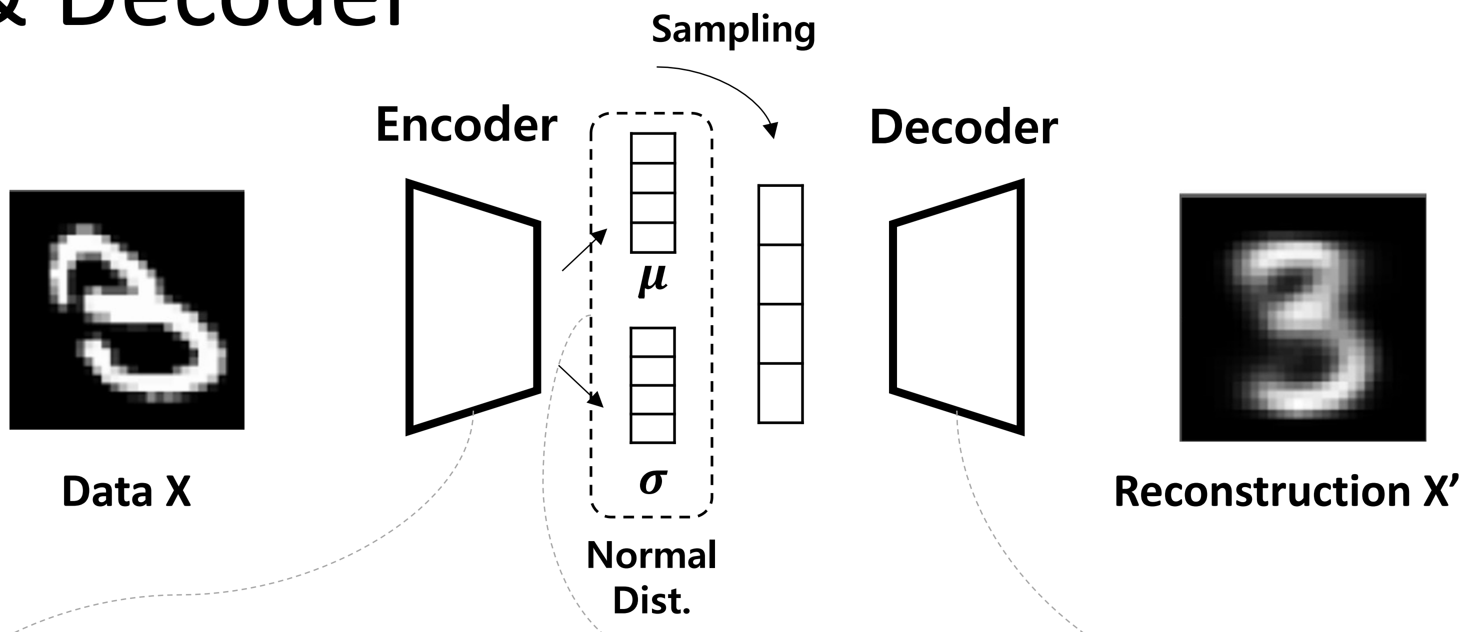



```python
self.encoder = nn.Sequential(
    nn.Linear(self.x_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.ReLU(),

)
```

```python
self.stat_net = nn.Linear(hidden_dim,
                          2 * z_dim)
```

```python
self.decoder = nn.Sequential(
    nn.Linear(z_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, hidden_dim),
    nn.ReLU(),
    nn.Linear(hidden_dim, self.x_dim),
    nn.Sigmoid(),
)
```

Encoder & Decoder




```python
def forward(self, x):
    x = x.view(-1, self.x_dim)
    h = self.encoder(x)
    stats = self.stat_net(h)
```

```python
z_mu, z_logvar = torch.chunk(stats, 2, dim=-1)
```

```python
_std = (torch.randn_like(z_mu)
        * torch.exp(0.5*z_logvar))
z = z_mu + _std
```

```python
out = self.decoder(z)
```

Implementation TODO

- Encoder & Decoder Network
- **Loss function**
- Training Loop
- Visualization

MLAI
Machine Learning & Artificial Intelligence

ELBO

$$\max_{\theta,\phi} \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - D_{KL}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\right]$$

$$\begin{aligned}\log p_\theta(\boldsymbol{x}|\boldsymbol{z}) &= \log \prod_{d=1}^{D_x} Ber(x_d|x'_d)\\ &= \sum_{d=1}^{D} \log (x'_d)^{x_d}(1-x'_d)^{x'_d}\\ &= \sum_{d=1}^{D} x_d \log x'_d + (1-x_d)\log(1-x'_d)\end{aligned}$$

$$\begin{aligned}D_{KL}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})\right] &= \sum_{d=1}^{D_z} D_{KL}[N(z_d|\mu_d,\sigma_d^2)||N(z_d|0,1)]\\ &= \frac{1}{2}\sum_{d=1}^{D_z}(\mu_d^2+\sigma_d^2-\log\sigma_d^2-1)\end{aligned}$$

```python
def compute_elbo(x, x_reconst, mu, logvar):
    criterion = nn.BCELoss(reduction="sum")
    log_likelihood = -criterion(x_reconst, x.view(-1, 784)) / x.size(0)
    kl = -0.5 * torch.sum(1 + logvar -mu.pow(2) - logvar.exp())

    return log_likelihood - kl
```

Implementation TODO

- Encoder & Decoder Network
- Loss function
- **Training Loop**
- Visualization

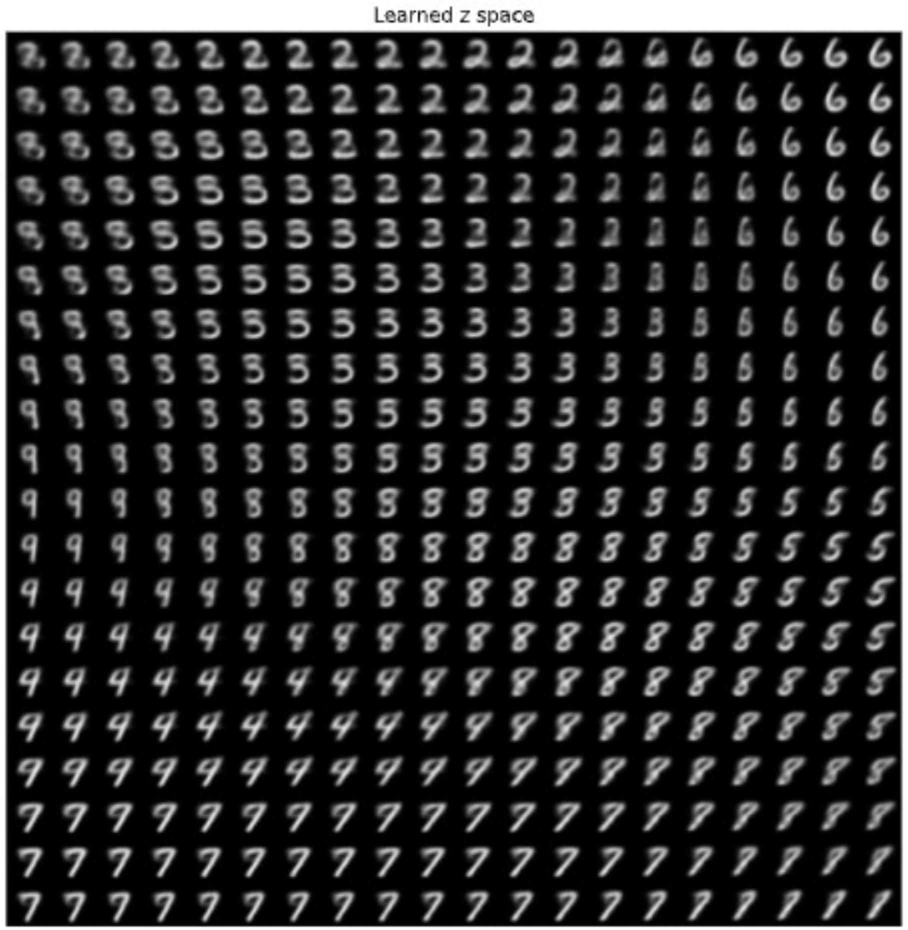
MLAI
Machine Learning & Artificial Intelligence

Training Loop

```
losses = []
for epoch_i in trange(n_epoch):
    for x, _ in loader:
        x = x.to(device)

        x_reconst, mu, logvar = vae(x)

        elbo = compute_elbo(x, x_reconst, mu, logvar)
        loss = -elbo

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
```

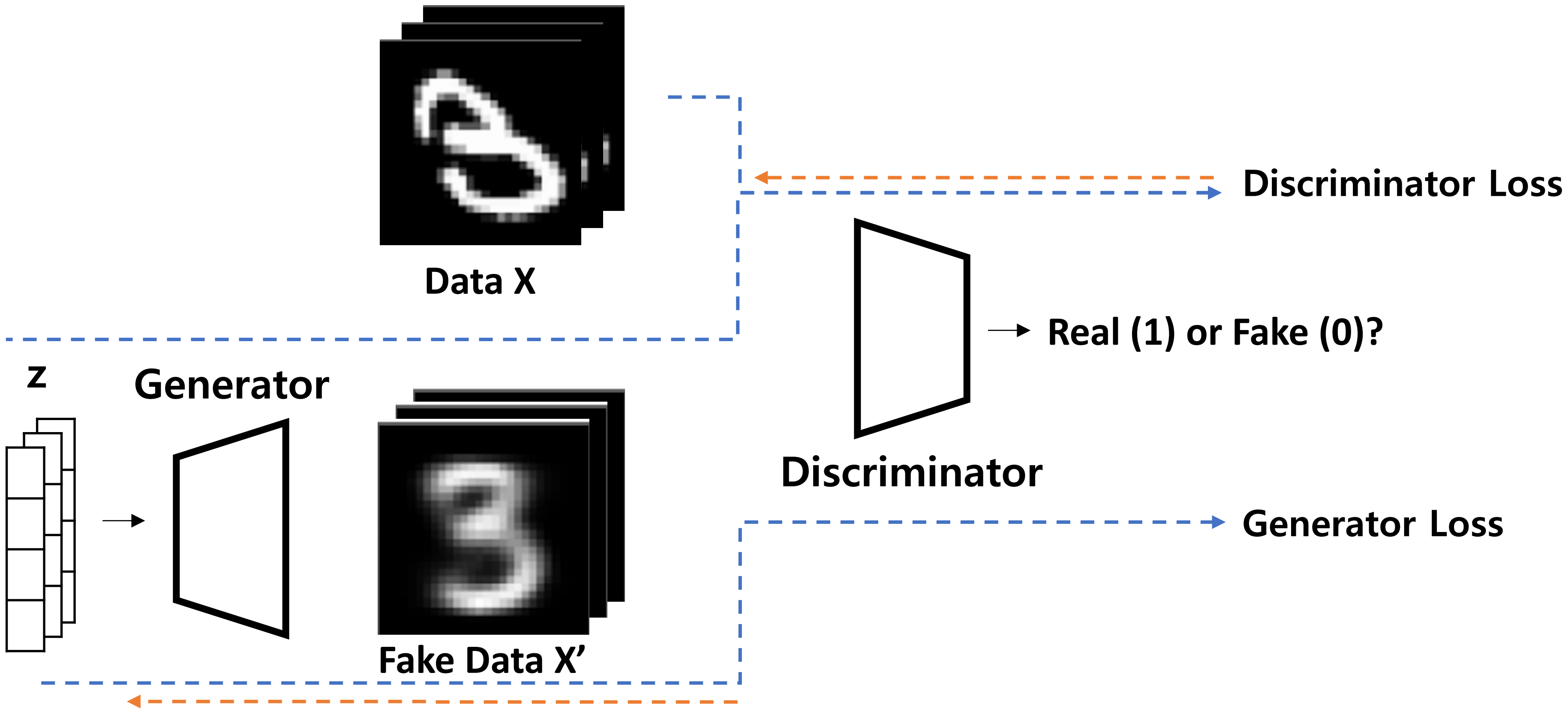
Implementation TODO

- Encoder & Decoder Network
- Loss function
- Training Loop
- **Visualization**



Generative Adversarial Networks (GAN)

GAN Overview



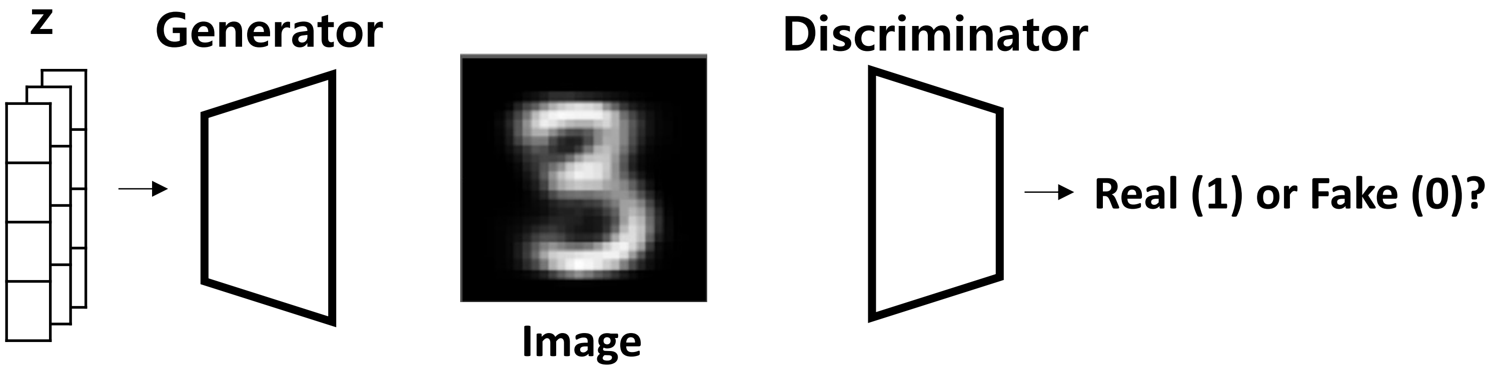
Implementation TODO

- Generator & Discriminator Network
- Training Loop
- Visualization

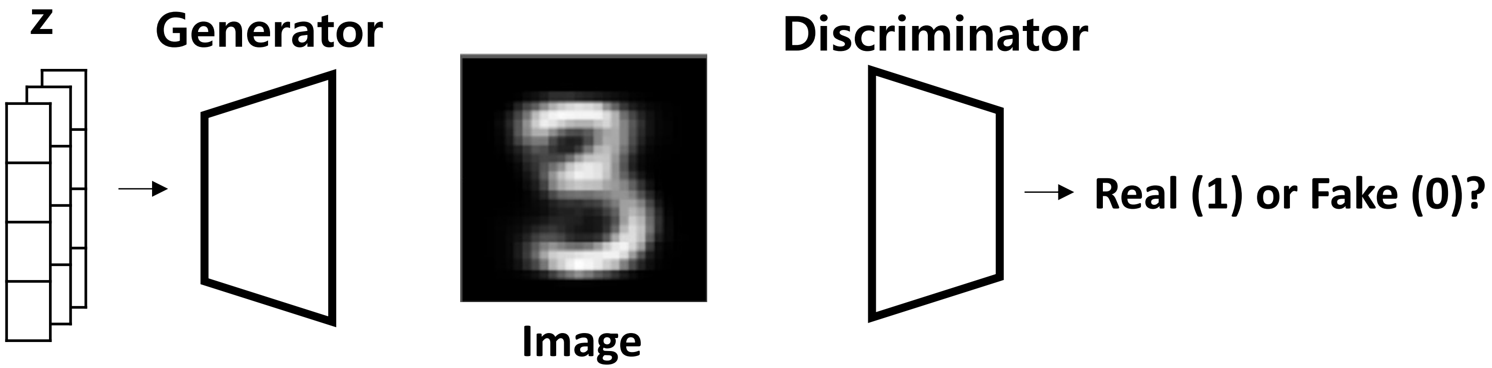
Implementation TODO

- **Generator & Discriminator Network**
- Training Loop
- Visualization

Generator & Discriminator



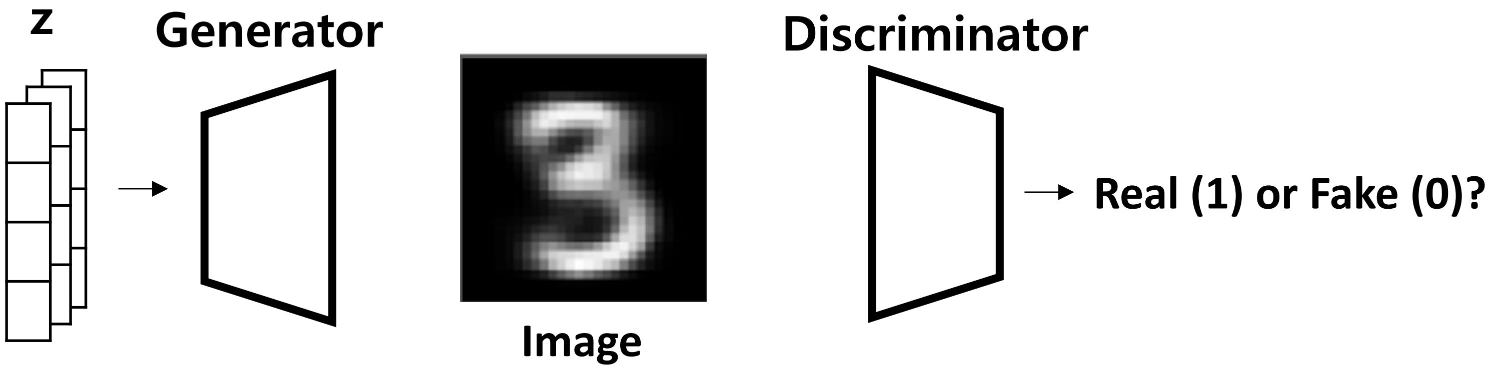
Generator & Discriminator




```python
class G(nn.Module):
    def __init__(self, z_dim, x_shape, hidden_dim=500):
        super().__init__()

        self.x_shape = x_shape
        self.x_dim = np.prod(x_shape)
        self.z_dim = z_dim

        self.net = nn.Sequential(
            nn.Linear(z_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, self.x_dim),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.net(input).view(len(input), *self.x_shape)
```

MLAI
Machine Learning & Artificial Intelligence

Generator & Discriminator




```
class D(nn.Module):
    def __init__(self, x_shape, hidden_dim=500):
        super().__init__()

        self.x_dim = np.prod(x_shape)

        self.net = nn.Sequential(
            nn.Linear(self.x_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 1),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.net(input.view(len(input), self.x_dim)).squeeze(-1)
```

MLAI
Machine Learning & Artificial Intelligence

Implementation TODO

- Generator & Discriminator Network
- **Training Loop**
- Visualization

Training Loop

```
for epoch_i in trange(n_epoch):
    for real_x, _ in loader:
```

```
        d.zero_grad()

        # discriminator for real data
        real_x = real_x.to(device)
        real_pred_y = d(real_x)
        real_y = make_real_y(len(real_x)).to(device)

        real_d_loss = criterion(real_pred_y, real_y)
        real_d_loss.backward()

        # discriminator for fake data
        z = torch.randn(len(real_x), z_dim, device=device)
        fake_x = g(z)
        fake_pred_y = d(fake_x.detach())
        fake_y = make_fake_y(len(real_x)).to(device)

        fake_d_loss = criterion(fake_pred_y, fake_y)
        fake_d_loss.backward()

        d_optimizer.step()
```

```
        # generator
        g.zero_grad()
        fake_pred_y = d(fake_x).view(-1)
        g_loss = criterion(fake_pred_y, real_y)
        g_loss.backward()
        g_optimizer.step()
```

MLAI
Machine Learning & Artificial Intelligence

Implementation TODO

- Generator & Discriminator Network
- Training Loop
- **Visualization**

Q&A