GANs in action

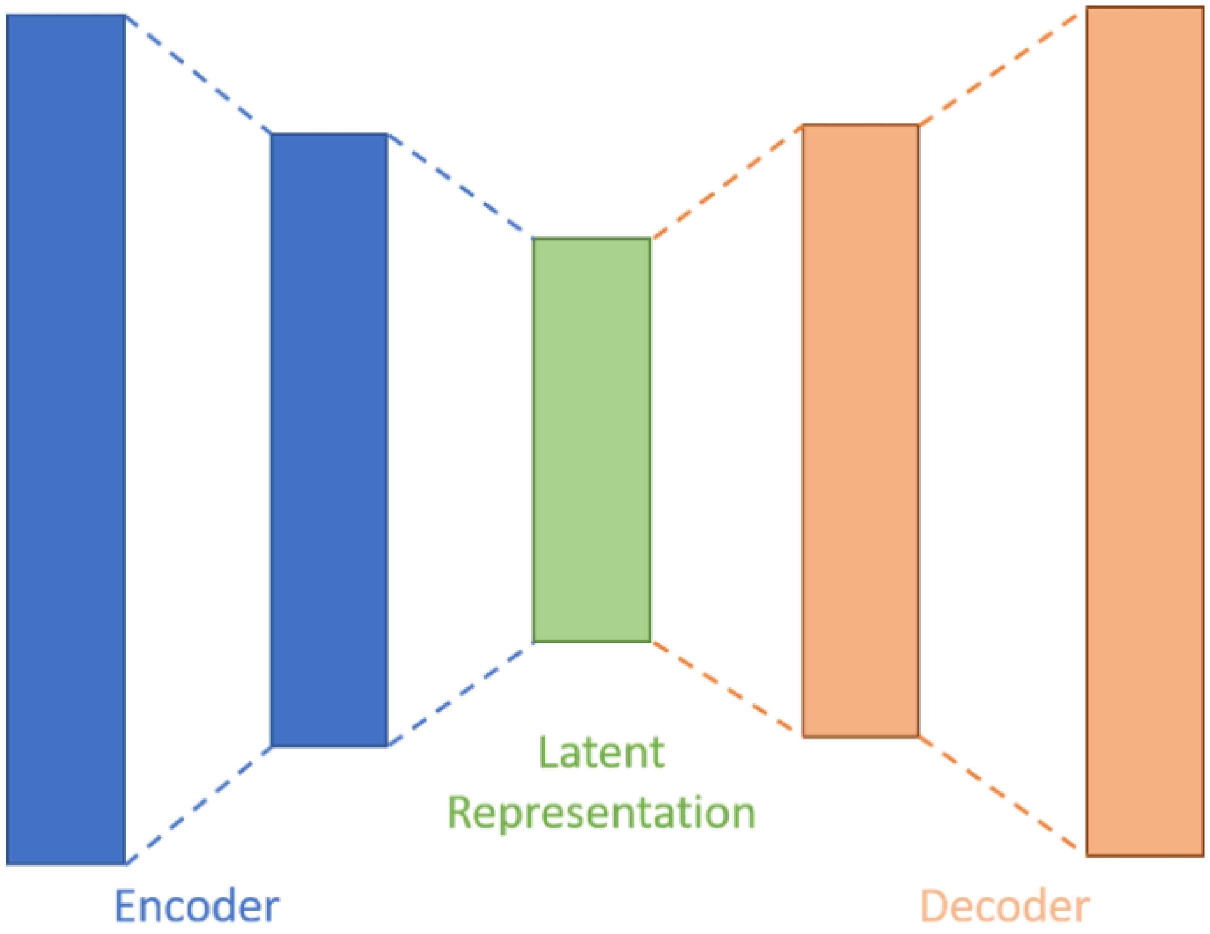
AutoEncoder, GAN, DCGAN

한병찬

Generative model

1. AutoEncoder
2. Variational AutoEncoder
3. GAN
4. DCGAN

AE



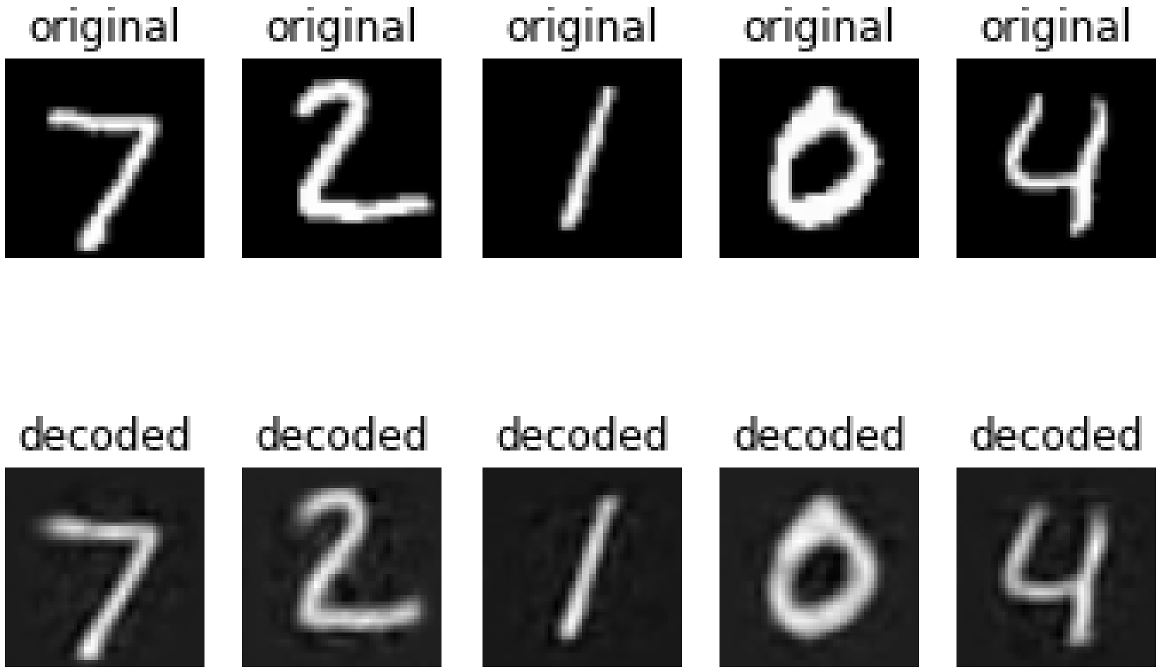
AE

```python
encoder = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(28, 28)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(32)
])
```

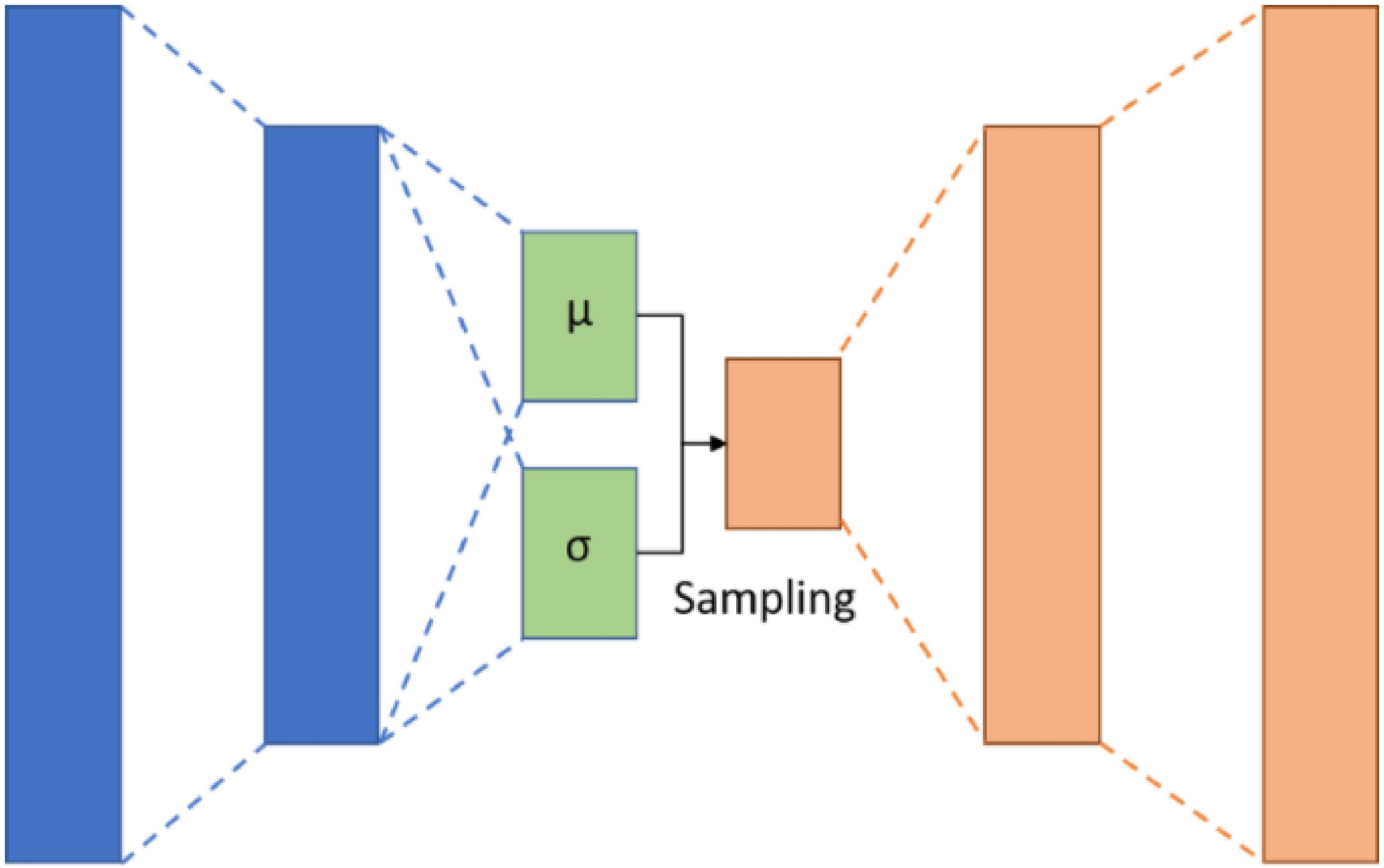
```python
AE = tf.keras.models.Sequential([
    encoder,
    decoder
])
```

```python
decoder = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(32, )),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(784),
    tf.keras.layers.Reshape((28, 28), input_shape=(784, ))
])
```

AE


original
original
original
original
original
decoded
decoded
decoded
decoded
decoded

VAE



VAE

```python
x = tf.keras.layers.Input(shape=(784, ))
h = tf.keras.layers.Dense(256, activation='relu')(x)
z_mean = tf.keras.layers.Dense(2)(h)
z_log_var = tf.keras.layers.Dense(2)(h)
z = tf.keras.layers.Lambda(sampling)([z_mean, z_log_var])

encoder = tf.keras.Model(x, [z_mean, z_log_var, z])
```

```python
input_decoder = tf.keras.layers.Input(shape=(2, ))
decoder_h = tf.keras.layers.Dense(256, activation='relu')(input_decoder)
x_decoded = tf.keras.layers.Dense(784, activation='sigmoid')(decoder_h)

decoder = tf.keras.Model(input_decoder, x_decoded)
```

```python
output_combined = decoder(encoder(x)[2])

VAE = tf.keras.Model(x, output_combined)
VAE.add_loss(tf.reduce_mean(kl_loss)/784.)
```

VAE

$Z \sim N(\mu, \sigma^2)$ ⟶ z′(latent vector) = $\mu + \sigma \odot \epsilon \ \ (\epsilon \sim N(0,1))$

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=(tf.shape(z_mean)[0], 2), mean=0.0, stddev=1.0)
    return z_mean + tf.exp(z_log_var/2)*epsilon
```

쿨백-라이블러 발산

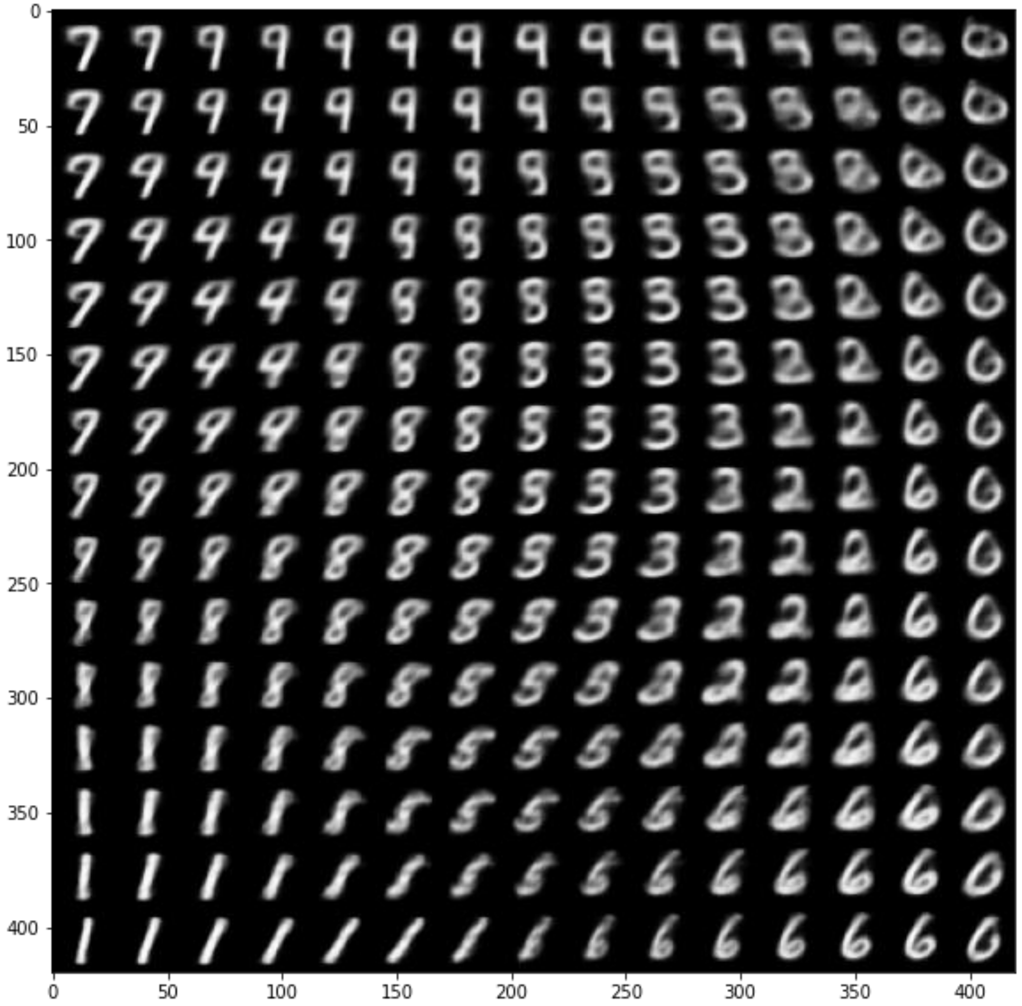
文A 17개 언어 ⌄

위키백과, 우리 모두의 백과사전.

쿨백-라이블러 발산(Kullback–Leibler divergence, **KLD**)은 두 확률분포의 차이를 계산하는 데에 사용하는 함수로, 어떤 이상적인 분포에 대해, 그 분포를 근사하는 다른 분포를 사용해 샘플링을 한다면 발생할 수 있는 정보 엔트로피 차이를 계산한다. 상대 엔트로피(relative entropy), 정보 획득량(information gain), 인포메이션 다이버전스(information divergence)라고도 한다. 정보이론에서는 상대 엔트로피, 기계학습의 결정 트리에서는 정보 획득량을 주로 사용한다.

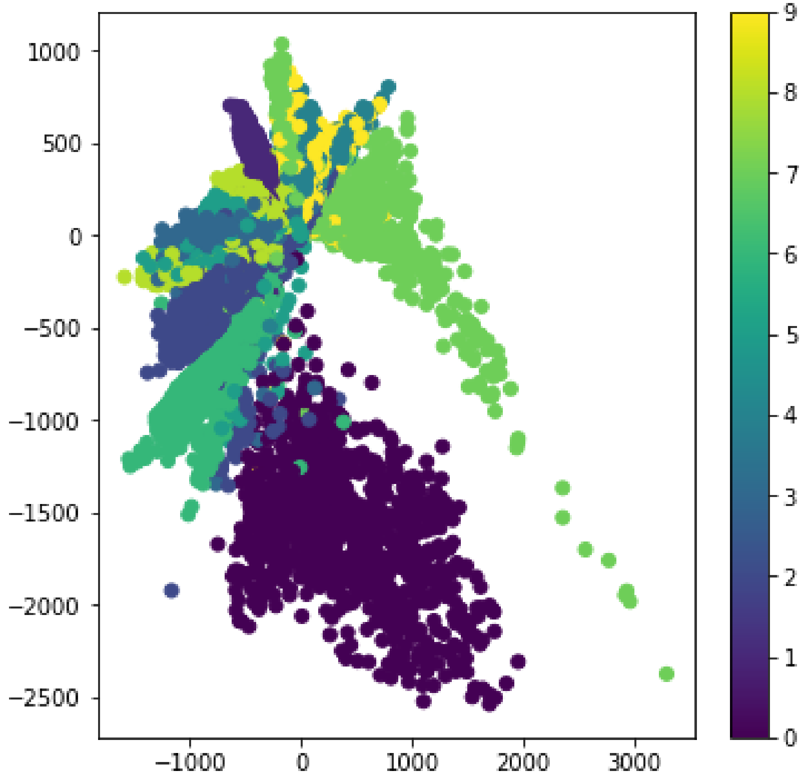
```
kl_loss = -0.5*tf.reduce_sum(1 + z_log_var - tf.exp(z_log_var) - tf.square(z_mean), axis=-1)
```

VAE

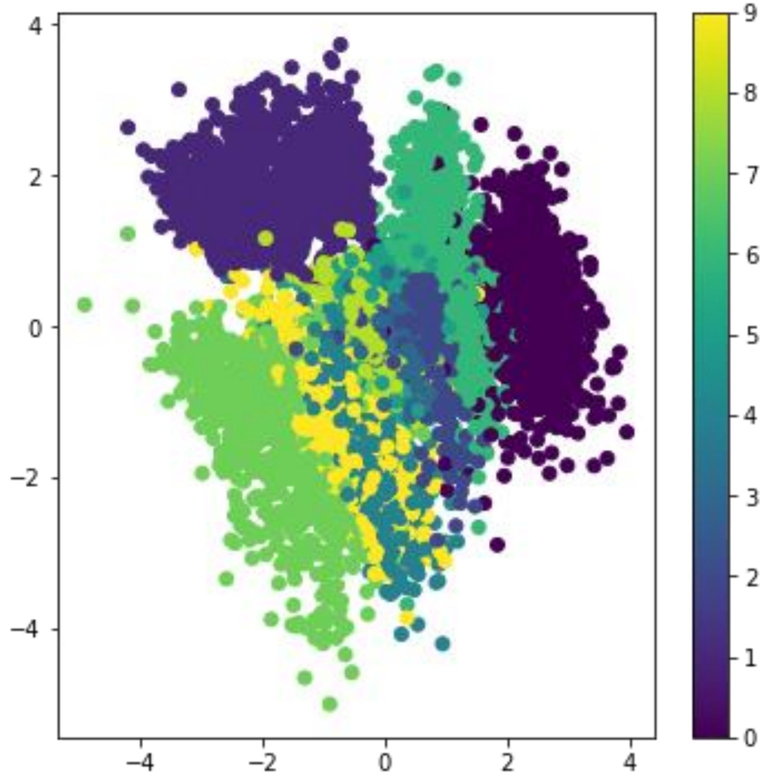


AE

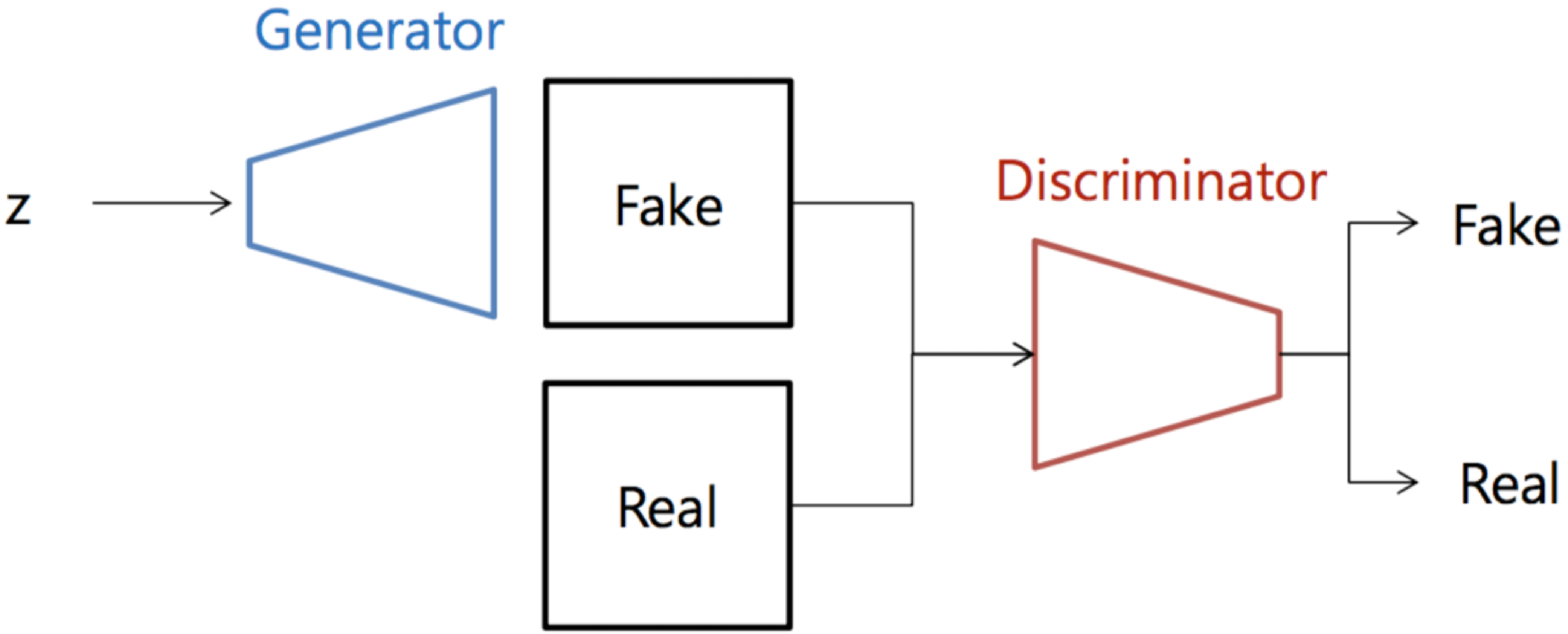
VAE



Latent Space



GAN



GAN

Binary Cross Entropy = $-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(\widehat{y}_i) + (1 - y_i) \cdot \log(1 - \widehat{y}_i)$

$Objective\ Function$ $$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

$Discriminator$: $D(x) \rightarrow 1,\quad D\big(G(z)\big) \rightarrow 0$

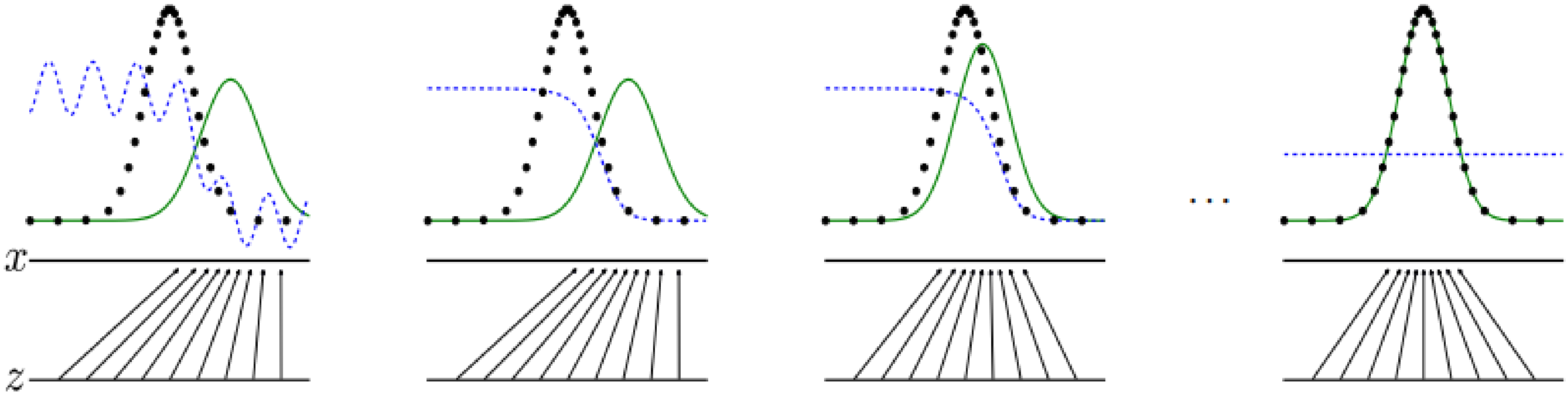
$Generator$: $D\big(G(z)\big) \rightarrow 1$

GAN

```python
img_rows = 28
img_cols = 28
img_channels = 1
img_shape = (img_rows, img_cols, img_channels)

latent_dim = 100

generator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(latent_dim, )),
    tf.keras.layers.Dense(128, activation='leaky_relu'),
    tf.keras.layers.Dense(int(np.prod(img_shape)), activation='tanh'),
    tf.keras.layers.Reshape(img_shape)
])

discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=img_shape),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='leaky_relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
discriminator.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


discriminator.trainable = False
GAN = tf.keras.models.Sequential([
    generator,
    discriminator
])
GAN.compile(loss='binary_crossentropy', optimizer='adam')
```

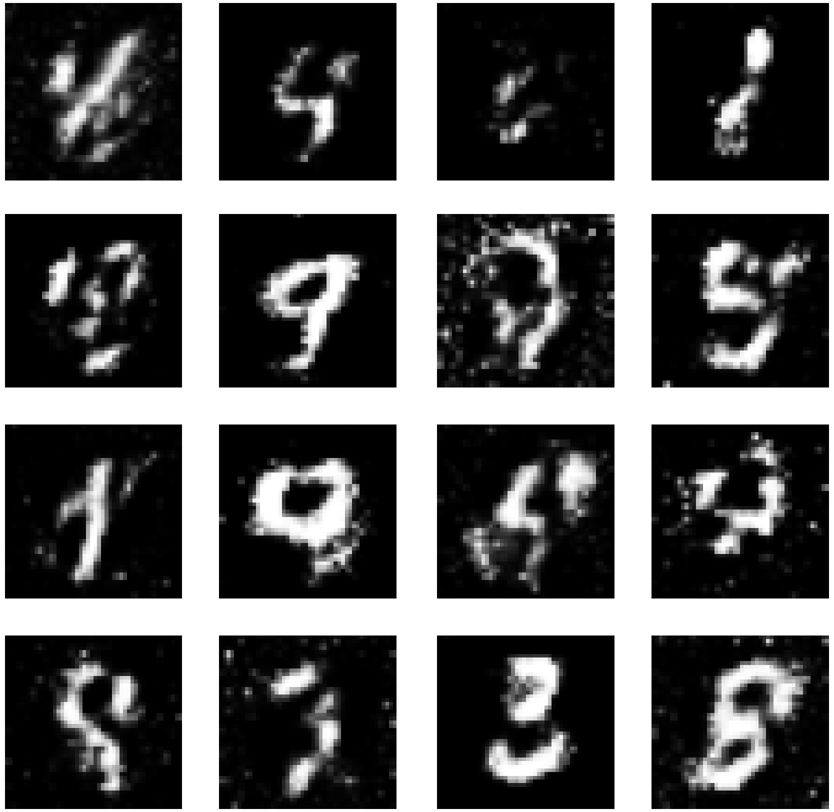
```python
real = np.ones((batch_size, 1))
fake = np.zeros((batch_size, 1))

for iteration in range(iterations):
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    imgs = X_train[idx]

    z = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(z)
    d_loss_real = discriminator.train_on_batch(imgs, real)
    d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
    d_loss, accuracy = 0.5*np.add(d_loss_real, d_loss_fake)

    z = np.random.normal(0, 1, (batch_size, 100))
    gen_imgs = generator.predict(z)

    g_loss = GAN.train_on_batch(z, real)
```

GAN



(a) (b) (c) (d)

GAN

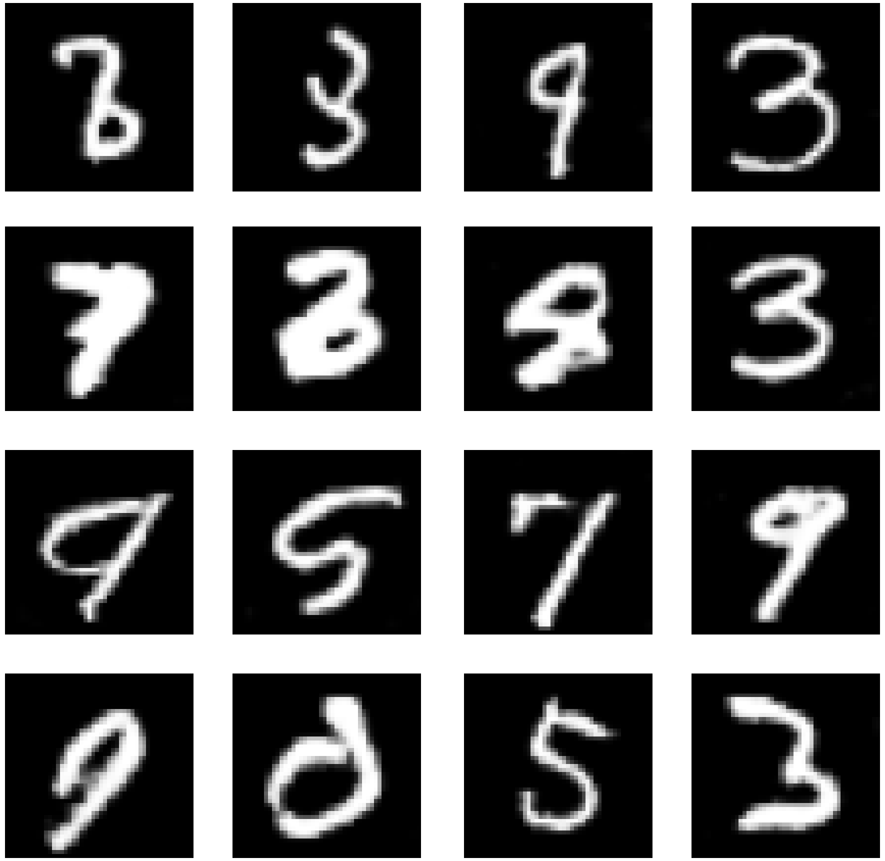


DCGAN

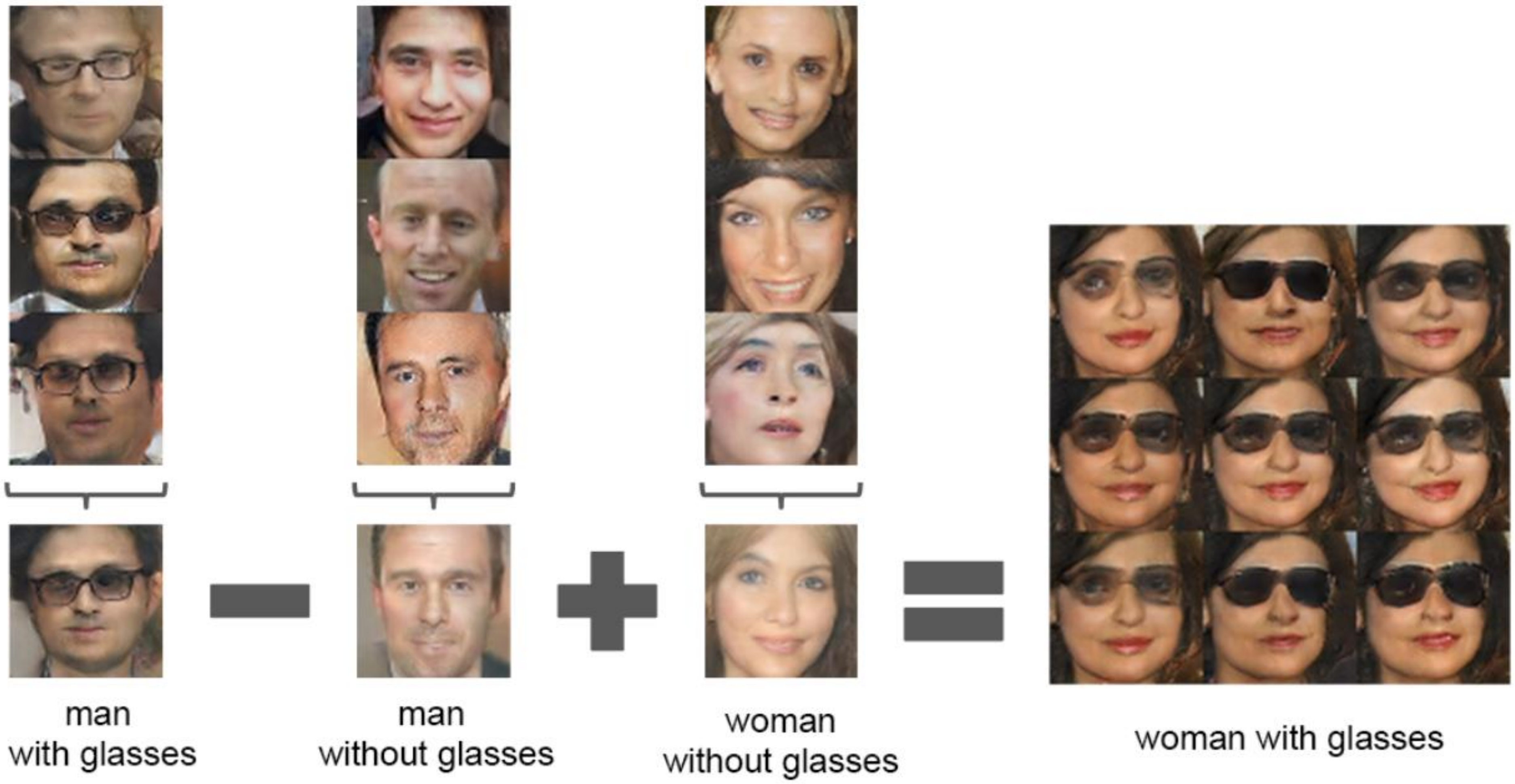
```
generator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(latent_dim, )),
    tf.keras.layers.Dense(128, activation='leaky_relu'),
    tf.keras.layers.Dense(int(np.prod(img_shape)), activation='tanh'),
    tf.keras.layers.Reshape(img_shape)
])

discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=img_shape),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='leaky_relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
generator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(latent_dim, )),
    tf.keras.layers.Dense(256*7*7),
    tf.keras.layers.Reshape((7, 7, 256)),
    tf.keras.layers.Conv2DTranspose(128, kernel_size=3, strides=2, padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(alpha=0.01),
    tf.keras.layers.Conv2DTranspose(64, kernel_size=3, strides=1, padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.LeakyReLU(alpha=0.01),
    tf.keras.layers.Conv2DTranspose(1, kernel_size=3, strides=2, padding='same'),
    tf.keras.layers.Activation('tanh')
])

discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=img_shape),
    tf.keras.layers.Conv2D(32, kernel_size=3, strides=2, padding='same'),
    tf.keras.layers.LeakyReLU(alpha=0.01),
    tf.keras.layers.Conv2D(64, kernel_size=3, strides=2, padding='same'),
    tf.keras.layers.LeakyReLU(alpha=0.01),
    tf.keras.layers.Conv2D(128, kernel_size=3, strides=2, padding='same'),
    tf.keras.layers.LeakyReLU(alpha=0.01),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

DCGAN



DCGAN


man
with glasses
man
without glasses
woman
without glasses
woman with glasses