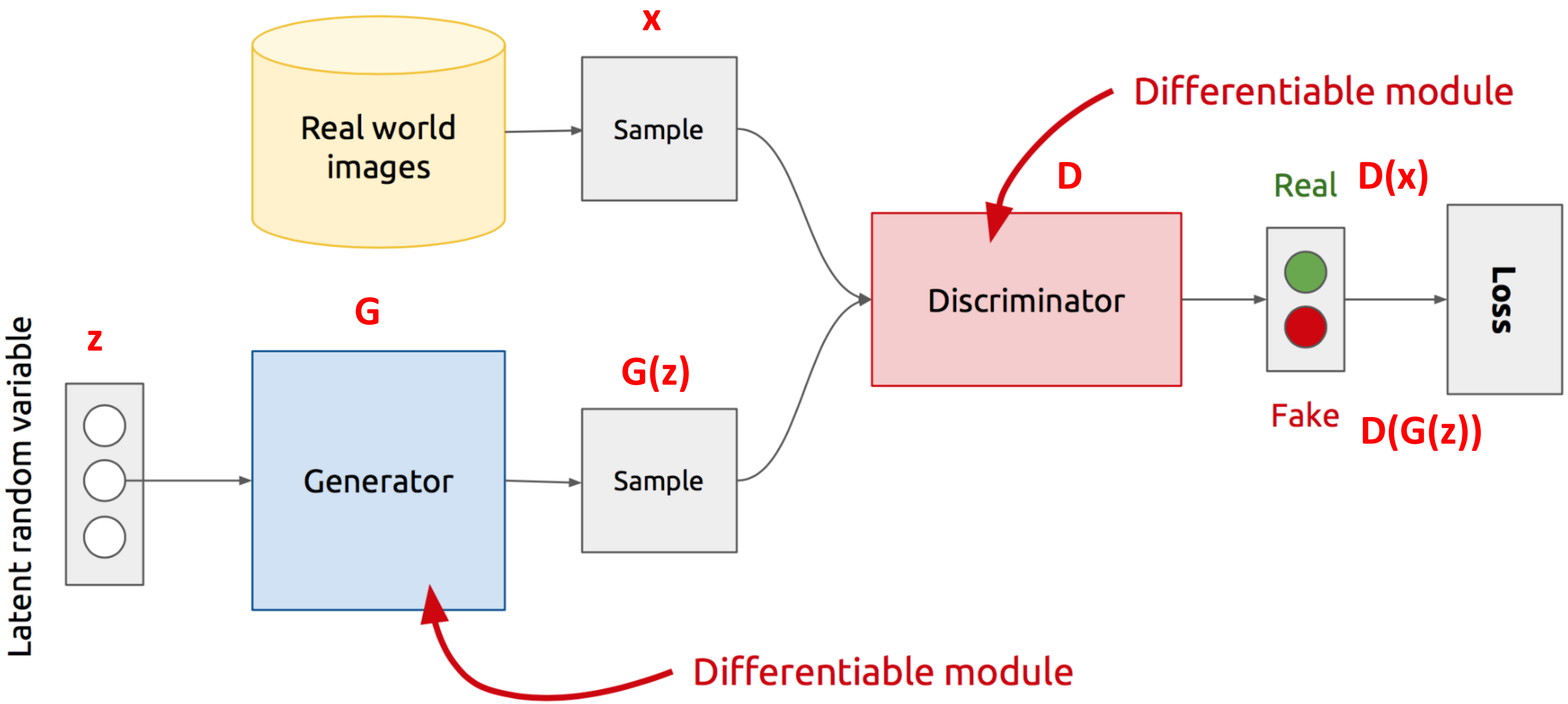
Data Analysis

Practice 9: Deep Generative models

Dr. Nataliya K. Sakhnenko

Generative Adversarial Network (GAN)

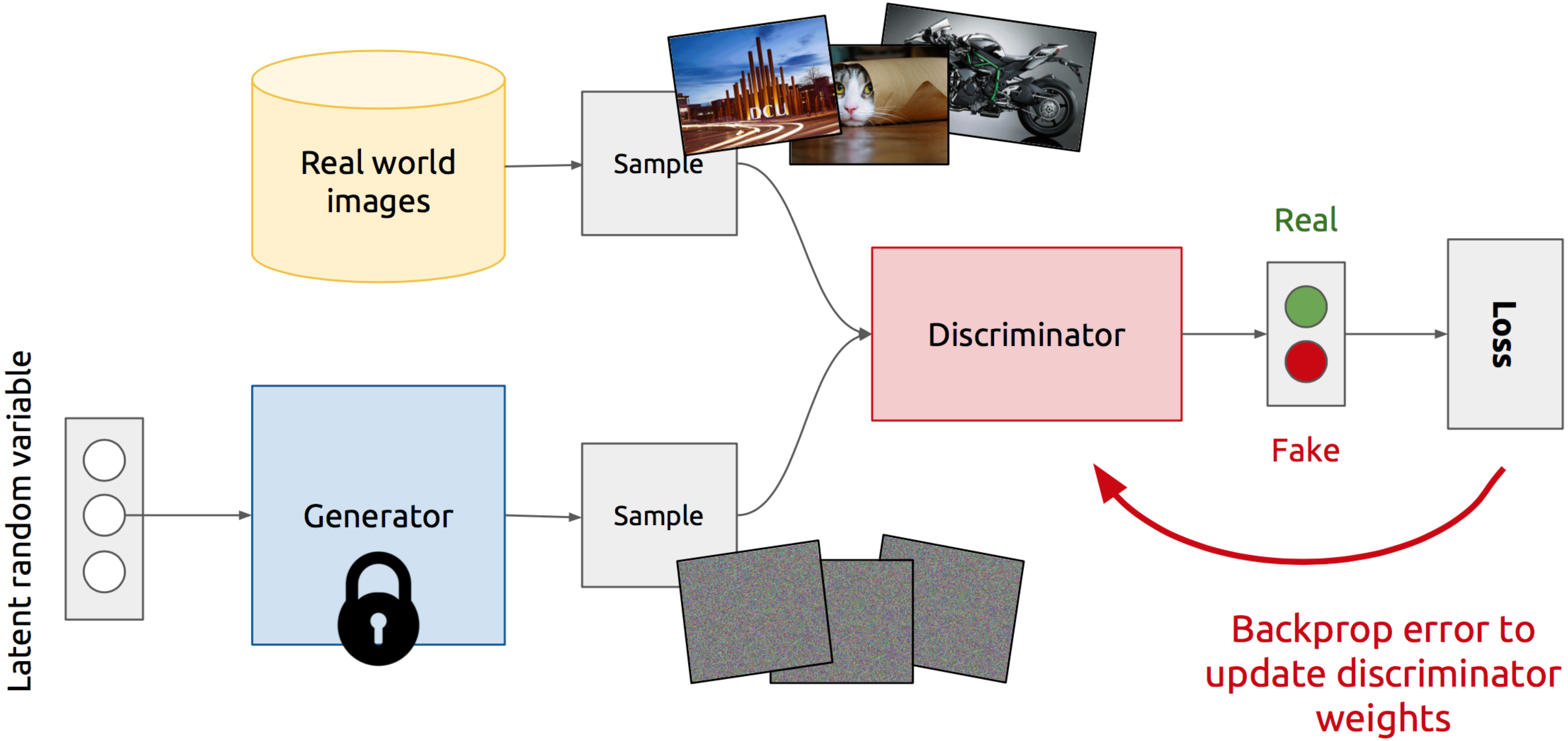
GAN's Architecture



Generator: generate fake samples, tries to fool the Discriminator
Discriminator: tries to distinguish between real and fake samples
Train them against each other
Repeat this and we get better Generator and Discriminator

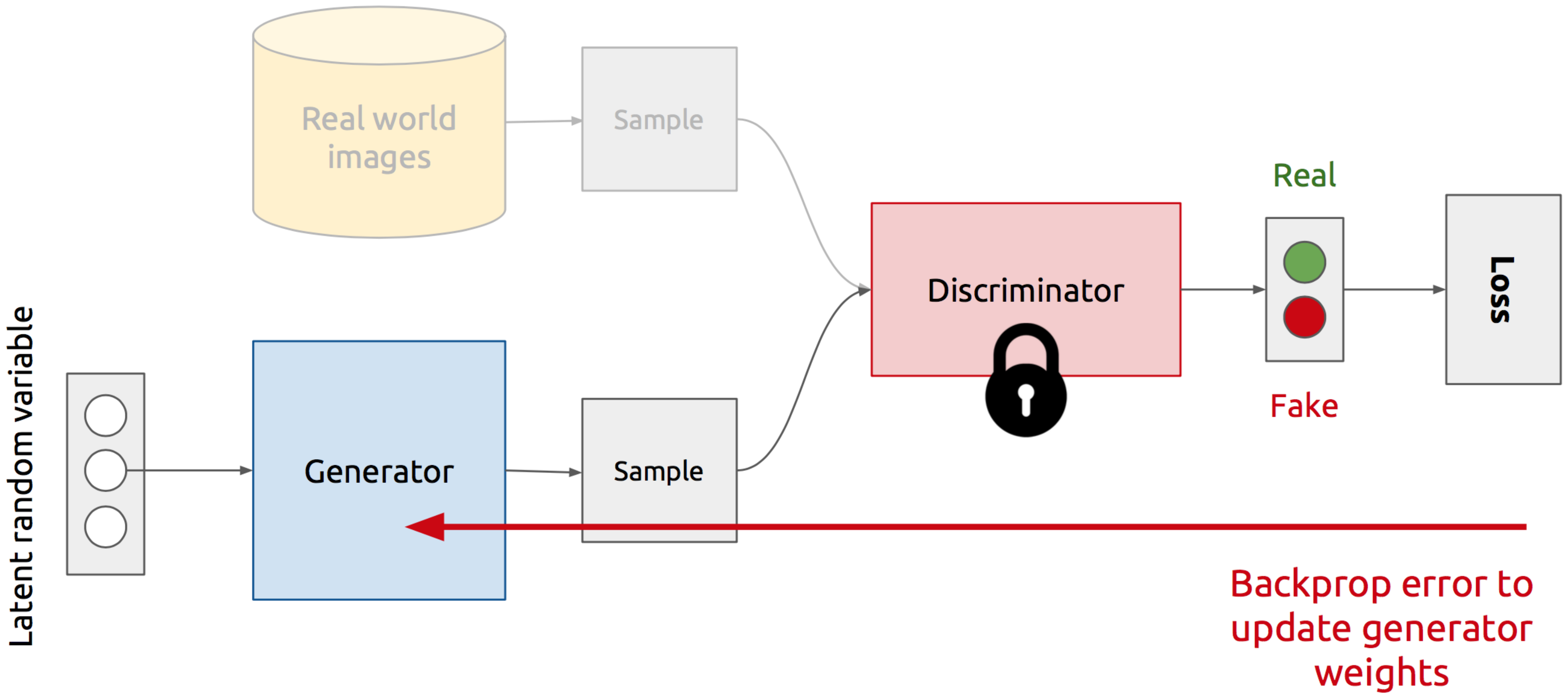
- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

$$\min_{G} \max_{D} V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

- Discriminator (θ_d) wants to **maximize objective** such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

The Nash equilibrium of this particular game is achieved at:

$$P_{data}(x) = P_{gen}(x) \ \ \forall x$$

$$\mathrm{D}(x) = \frac{1}{2} \ \ \forall x$$

MNIST GAN example

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = (X_train.astype(np.float32) - 127.5) / 127.5
X_train = X_train.reshape(60000, 784)
```

Vanilla GAN

Discriminator

```
def build_discriminator():
    discriminator = Sequential()
    discriminator.add(Dense(..., input_dim=784))
    discriminator.add(LeakyReLU(0.2))
    .....
    discriminator.add(Dense(1, activation='sigmoid'))
    discriminator.compile(loss='binary_crossentropy, optimizer=adam)
  return discriminator
```

MNIST GAN example

Latent random variable:

```
random_dim = 100
```

Generator

```
def build_generator():
    generator = Sequential()
    generator.add(Dense(..., input_dim=random_dim))
    generator.add(LeakyReLU(0.2))
    ......
    generator.add(Dense(784, activation='tanh'))
    generator.compile(loss='binary_crossentropy', optimizer=adam)
 return generator
```

MNIST GAN example

GAN

```
generator = build_generator()
discriminator = build_discriminator()

# Combined network
discriminator.trainable = False
gan_input = Input(shape=(random_dim,))
x = generator(gan_input)
gan_output = discriminator(x)
gan = Model(inputs=gan_input, outputs=gan_output)
gan.compile(loss='binary_crossentropy', optimizer=adam)
```

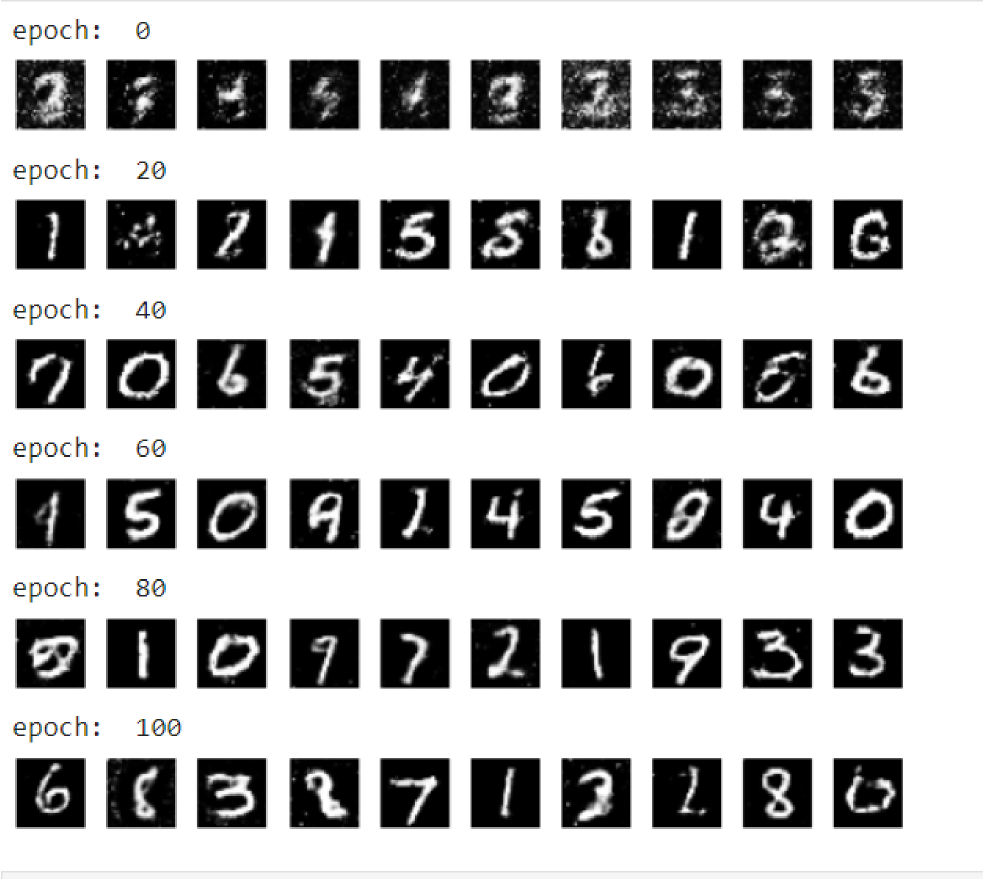
MNIST GAN example

Train function (for 1 iteration)

```
 noise = np.random.normal(0, 1, size=[batch_size, random_dim])
 image_batch = X_train[np.random.randint(0, X_train.shape[0], size=batch_size)]

generated_images = generator.predict(noise)
X = np.concatenate([image_batch, generated_images])

# Labels for generated and real data
y_dis = np.zeros(2*batch_size)
y_dis[:batch_size] = 1.0
# Train discriminator
discriminator.trainable = True
discriminator.train_on_batch(X, y_dis)
# Train generator
noise = np.random.normal(0, 1, size=[batch_size, random_dim])
y_gen = np.ones(batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, y_gen)
```

MNIST GAN example

```
noise = np.random.normal(0, 1, size=[100, random_dim])
generated_images = generator.predict(noise)
```



Samples generated by GAN on different epochs

Conditional GAN, 2014

https://arxiv.org/abs/1411.1784

GANs can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information (e.g label)

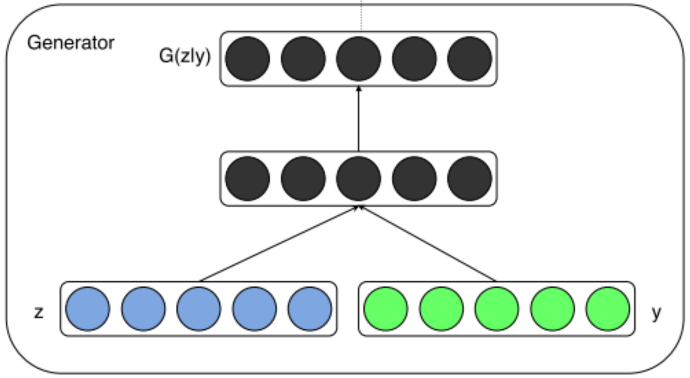
```
discriminator.train_on_batch([X, labels], y_dis)
```

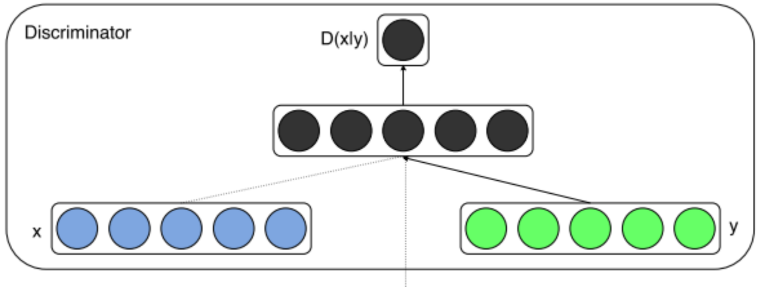
GAN objective function

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

transforms to

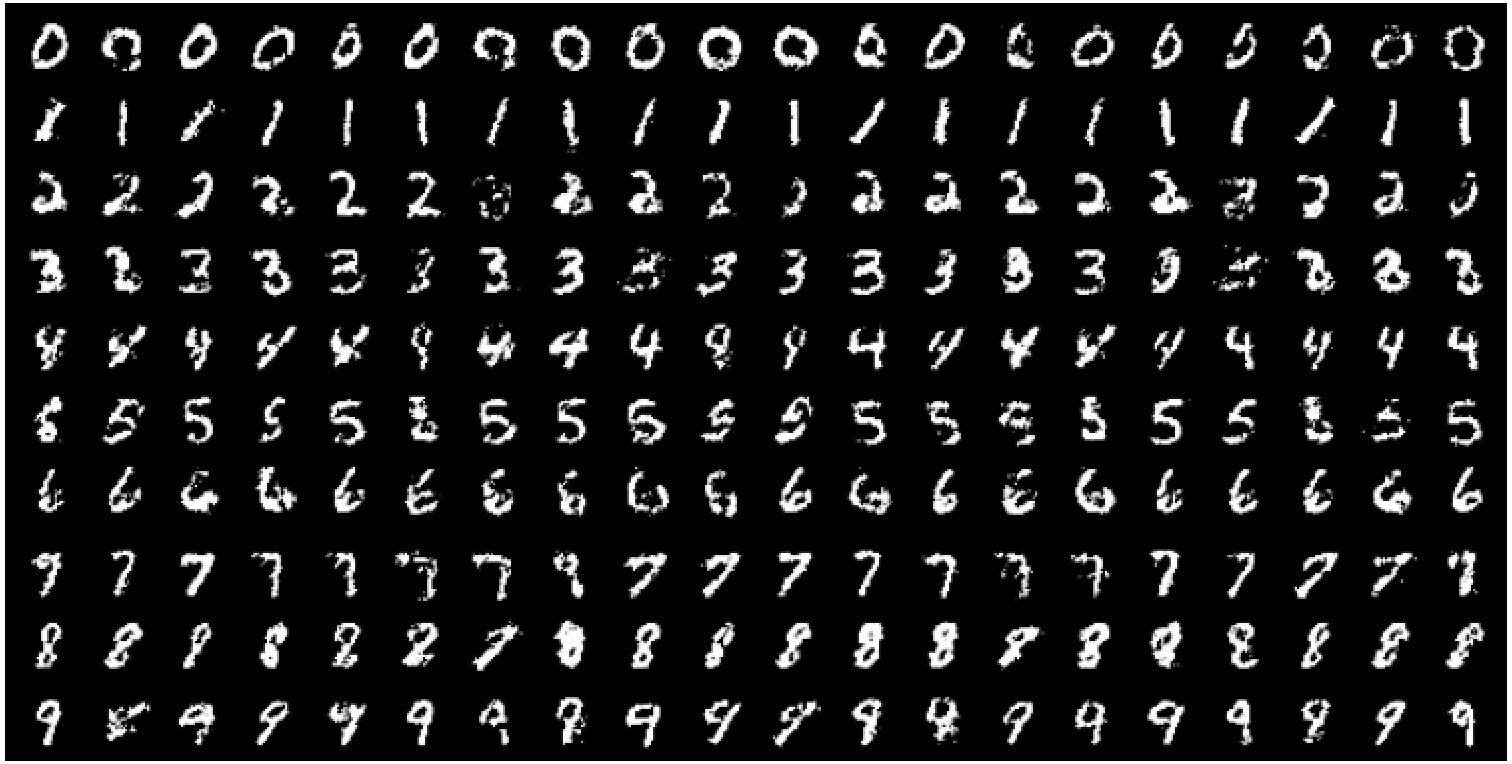
$$\min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$





https://keras.io/examples/generative/conditional_gan/

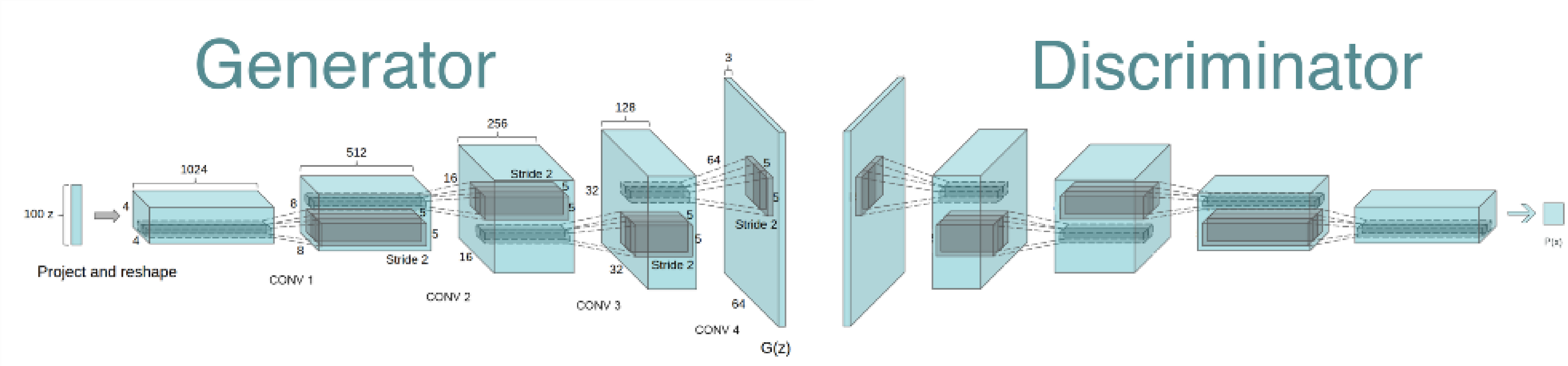
Conditional GAN, 2014



Generated MNIST digits, each row conditioned on one label

https://arxiv.org/abs/1411.1784

Deep Convolutional GAN (DCGAN), 2015



A. Radford et al. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.

(DCGAN)

Generator:

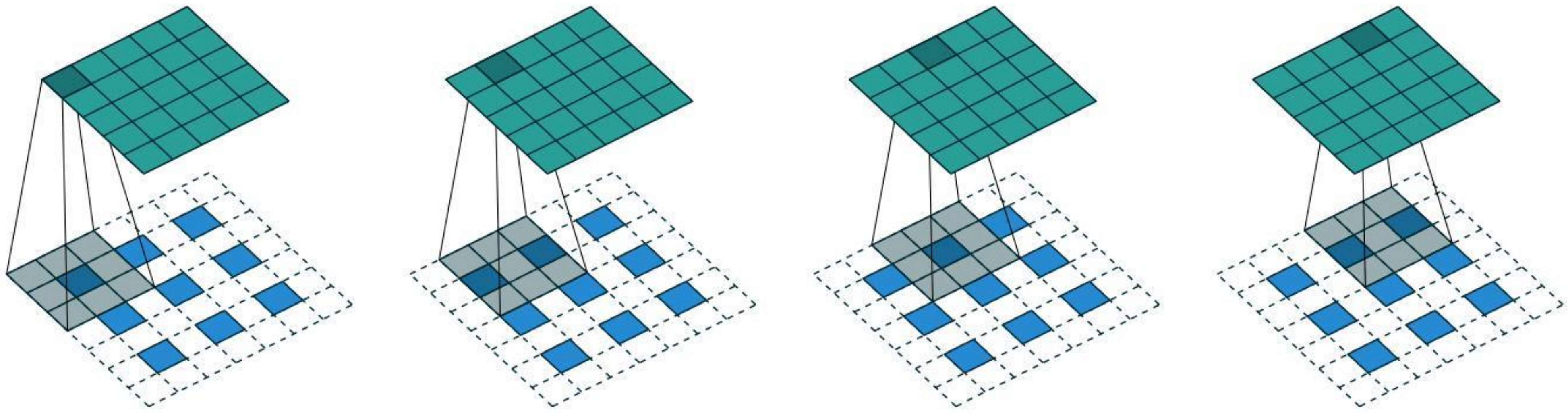
✓ upsample layer (UpSampling2D) that simply doubles the dimensions of the input

✓ or the transpose convolutional layer (Conv2DTranspose)

```
model.add(UpSampling2D(…))
```

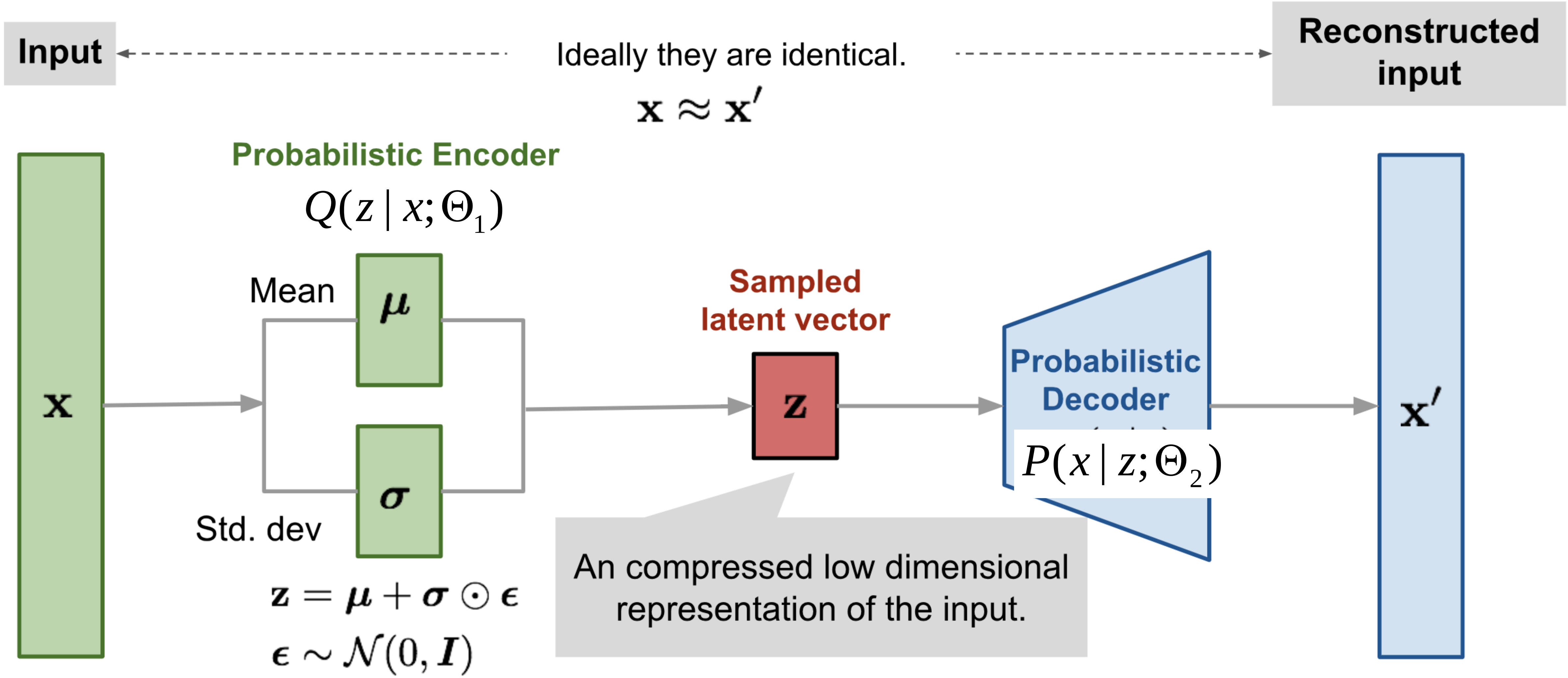
```
model.add(Conv2DTranspose(…))
```

Transpose Convolution, Fractionally Strided Convolution or Deconvolution



Variational Autoencoder (VAE)

Variational autoencoder (VAE)



VAE intuition

VAE objective function:

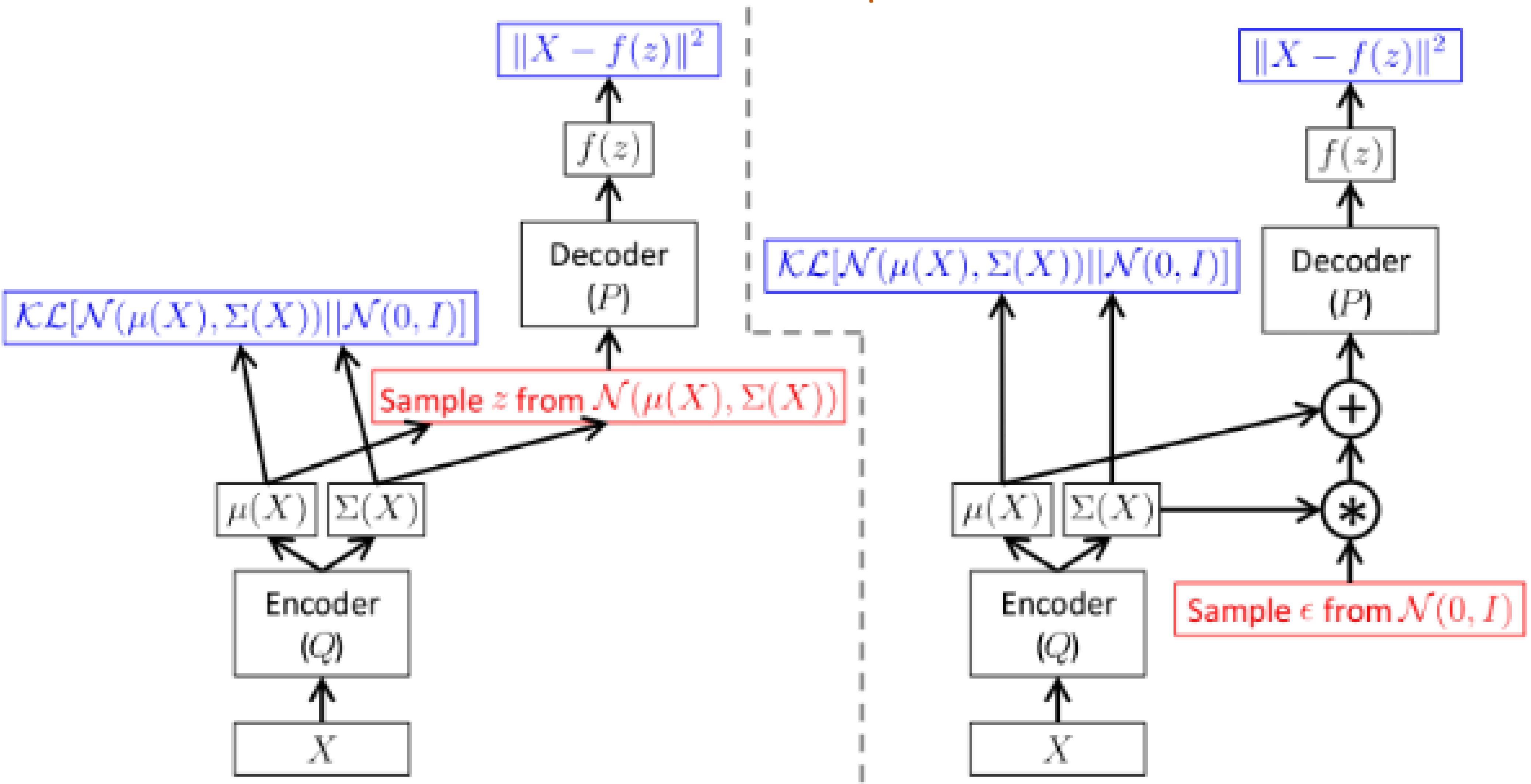
$$\boxed{\log P(X) - D_{KL}[Q(z|X)||P(z|X)] = E[\log P(X|z)] - D_{KL}[Q(z|X)||P(z)]}$$

In practice, however, it's better to model $\Sigma(X)$ as $\log \Sigma(X)$, as it is more numerically stable to take exponent compared to computing log. Hence, our final KL divergence term is:

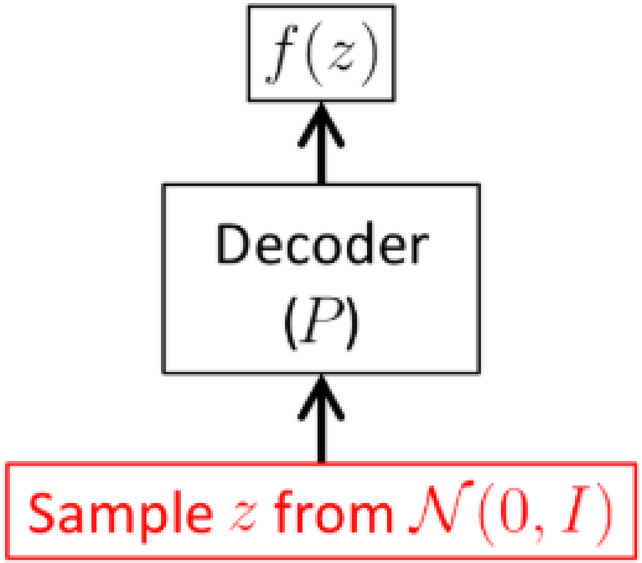
$$D_{KL}[N(\mu(X), \Sigma(X))||N(\mathbf{0}, \mathbf{1})] = \frac{1}{2} \sum_k \left(\exp(\Sigma(X)) + \mu^2(X) - \mathbf{1} - \Sigma(X)\right)$$

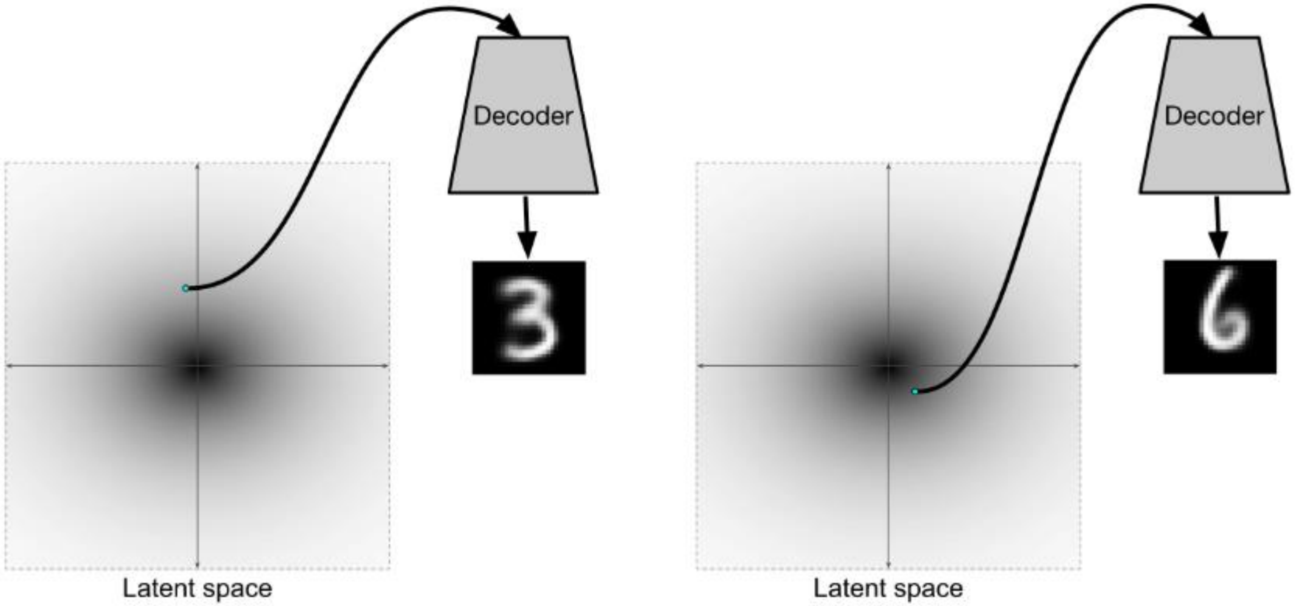
VAE

Reparametrization trick



VAE generation



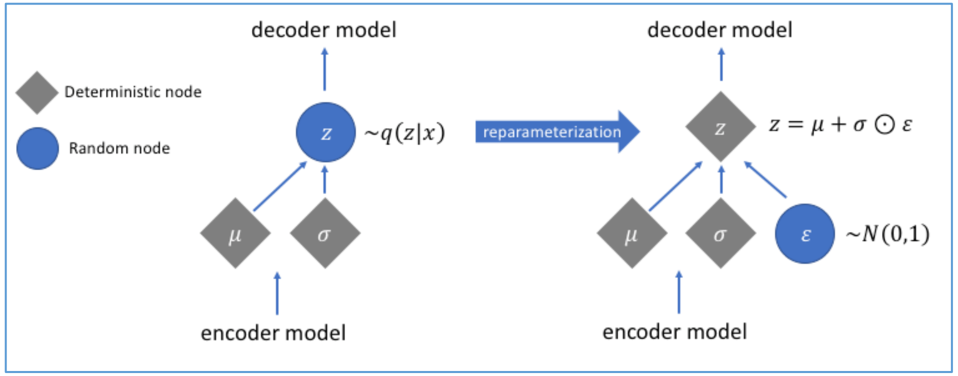


VAE MNIST Keras example

```
original_dim = 28 * 28
intermediate_dim = 64
latent_dim = 2

inputs = keras.Input(shape=(original_dim,))
h = layers.Dense(intermediate_dim, activation='relu')(inputs)
z_mean = layers.Dense(latent_dim)(h)
z_log_sigma = layers.Dense(latent_dim)(h)
```

We can use these parameters to sample new similar points from the latent space:

```
from keras import backend as K

def sampling(args):
    z_mean, z_log_sigma = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim),
                              mean=0., stddev=0.1)
    return z_mean + K.exp(z_log_sigma) * epsilon

z = layers.Lambda(sampling)([z_mean, z_log_sigma])
```



Reparameterization trick

https://blog.keras.io/building-autoencoders-in-keras.html

VAE MNIST Keras example

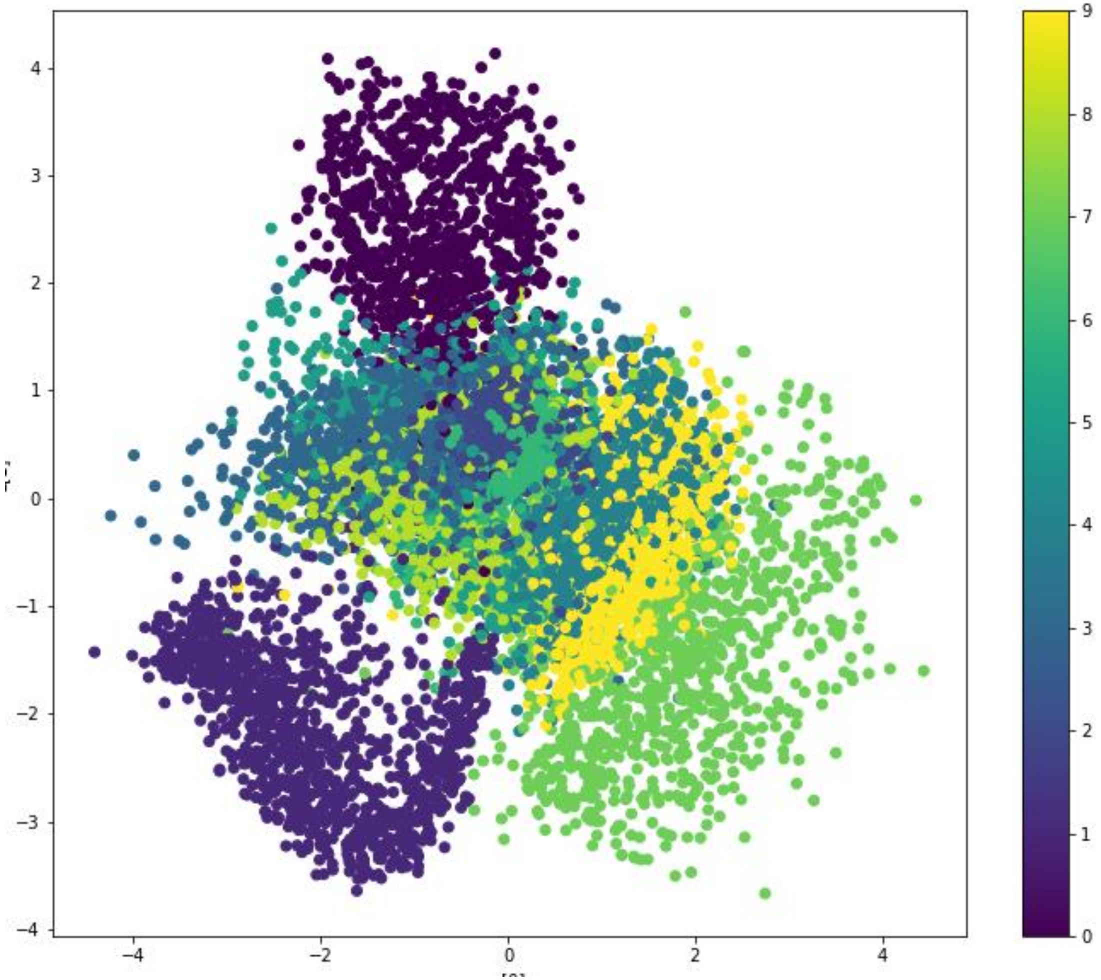
Build Encoder, Decoder, VAE

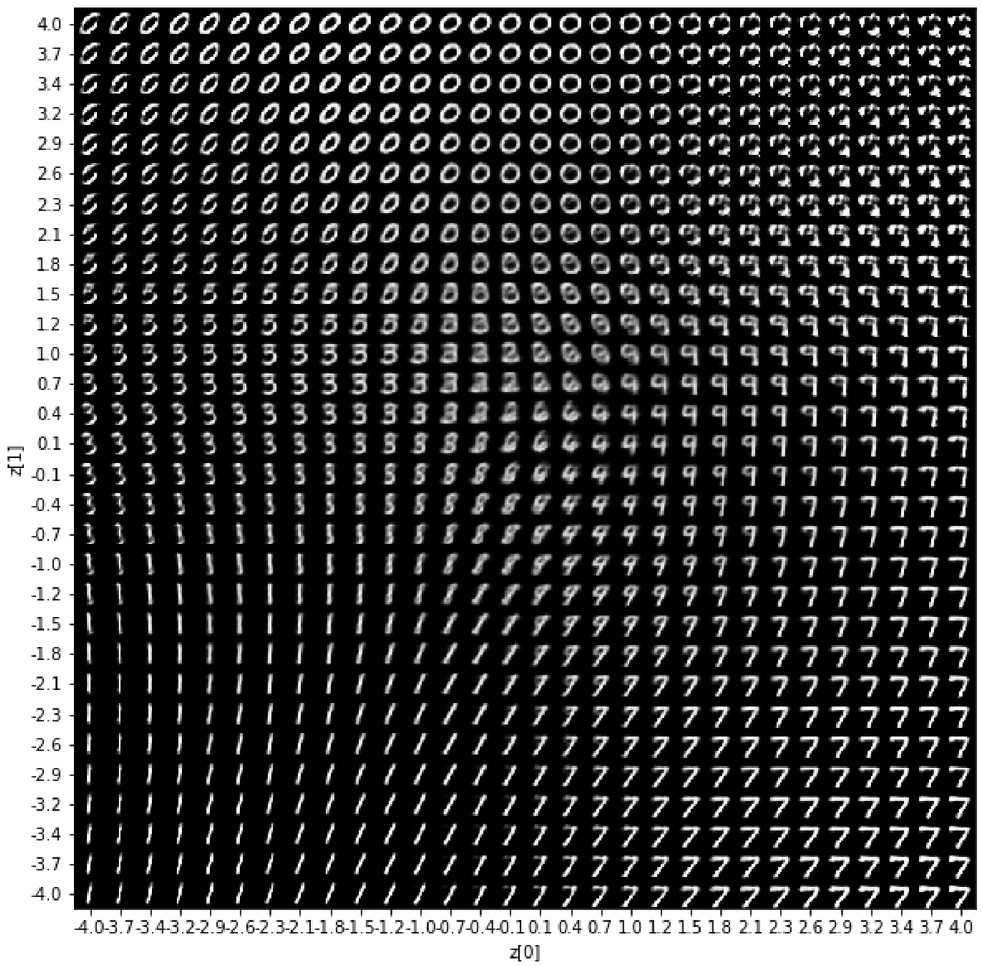
```
# Create encoder
encoder = keras.Model(inputs, [z_mean, z_log_sigma, z], name='encoder')

# Create decoder
latent_inputs = keras.Input(shape=(latent_dim,), name='z_sampling')
x = layers.Dense(intermediate_dim, activation='relu')(latent_inputs)
outputs = layers.Dense(original_dim, activation='sigmoid')(x)
decoder = keras.Model(latent_inputs, outputs, name='decoder')

# instantiate VAE model
outputs = decoder(encoder(inputs)[2])
vae = keras.Model(inputs, outputs, name='vae_mlp')
```

Custom loss function: the sum of a reconstruction term, and the KL divergence regularization term.

```
reconstruction_loss = keras.losses.binary_crossentropy(inputs, outputs)
reconstruction_loss *= original_dim
kl_loss = 1 + z_log_sigma - K.square(z_mean) - K.exp(z_log_sigma)
kl_loss = K.sum(kl_loss, axis=-1)
kl_loss *= -0.5
vae_loss = K.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
vae.compile(optimizer='adam')
```

VAE MNIST Results





Interpolating over MNIST digits by interpolating over latent variables

Conditional VAE

Conditional VAE (CVAE) is an extension of VAE to control the data generation process. Whereas VAE essentially models latent variables and data directly, CVAE models latent variables and data, both conditioned to some random variables.

CVAE objective function:

$$\log P(X|c) - D_{KL}[Q(z|X,c)\|P(z|X,c)] = E[\log P(X|z,c)] - D_{KL}[Q(z|X,c)\|P(z|c)]$$

we conditioned all of the distributions with a variable c (e.g. our labels)

Minor changes to VAE code: e.g. instead of z use [z, cond]

[1 , 0 , 0 , 0]

https://wiseodd.github.io/techblog/2016/12/17/conditional-vae/

Conditional VAE generation (style transfer)