THE UNIVERSITY OF MELBOURNE

Generative models

COMP90051 Statistical Machine Learning

“What I Cannot Create, I Do Not Understand” ——Richard Feynman

Qiuhong Ke

So far..

Classifier:

- SVM
- Perceptron
- Multi-layer perceptron
- CNN

Feature extraction:

- Pretrained CNN model

Using pretrained model in Keras

Feature extraction before the last classifier

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
import numpy as np

model = ResNet50(weights='imagenet',include_top=False)

img_path = '1.jpeg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Using pretrained model in Keras

Feature extraction from arbitrary layer

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
import numpy as np
from tensorflow.keras.models import Model

model = ResNet50(weights='imagenet')
layers=model.layers #get name of the layers
model = Model(inputs=model.input, outputs=model.get_layer('conv5_block3_out').output)

img_path = '1.jpeg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Using pretrained model in Keras

Perform classification

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = '1.png'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)

print('Predicted:', decode_predictions(preds, top=3)[0])
```

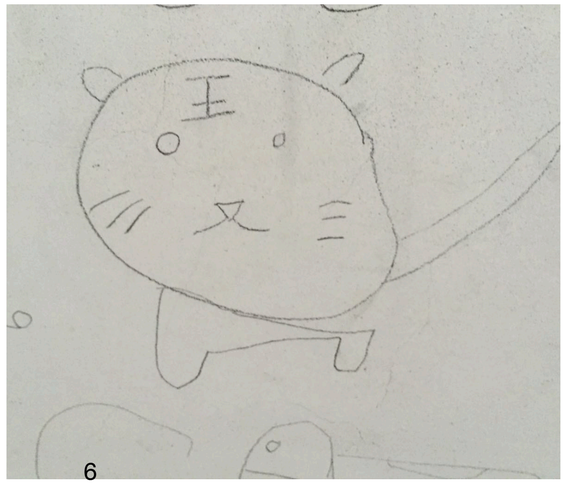
Predict class using a pretrained ResNet 50

Pretrained on ImageNet (1000 classes)

class	probability
espresso	0.9995716
cup	0.00022115342
coffee_mug	0.00020262193

class	probability
daisy	0.79781777
pot	0.10065505
picket_fence	0.05316206

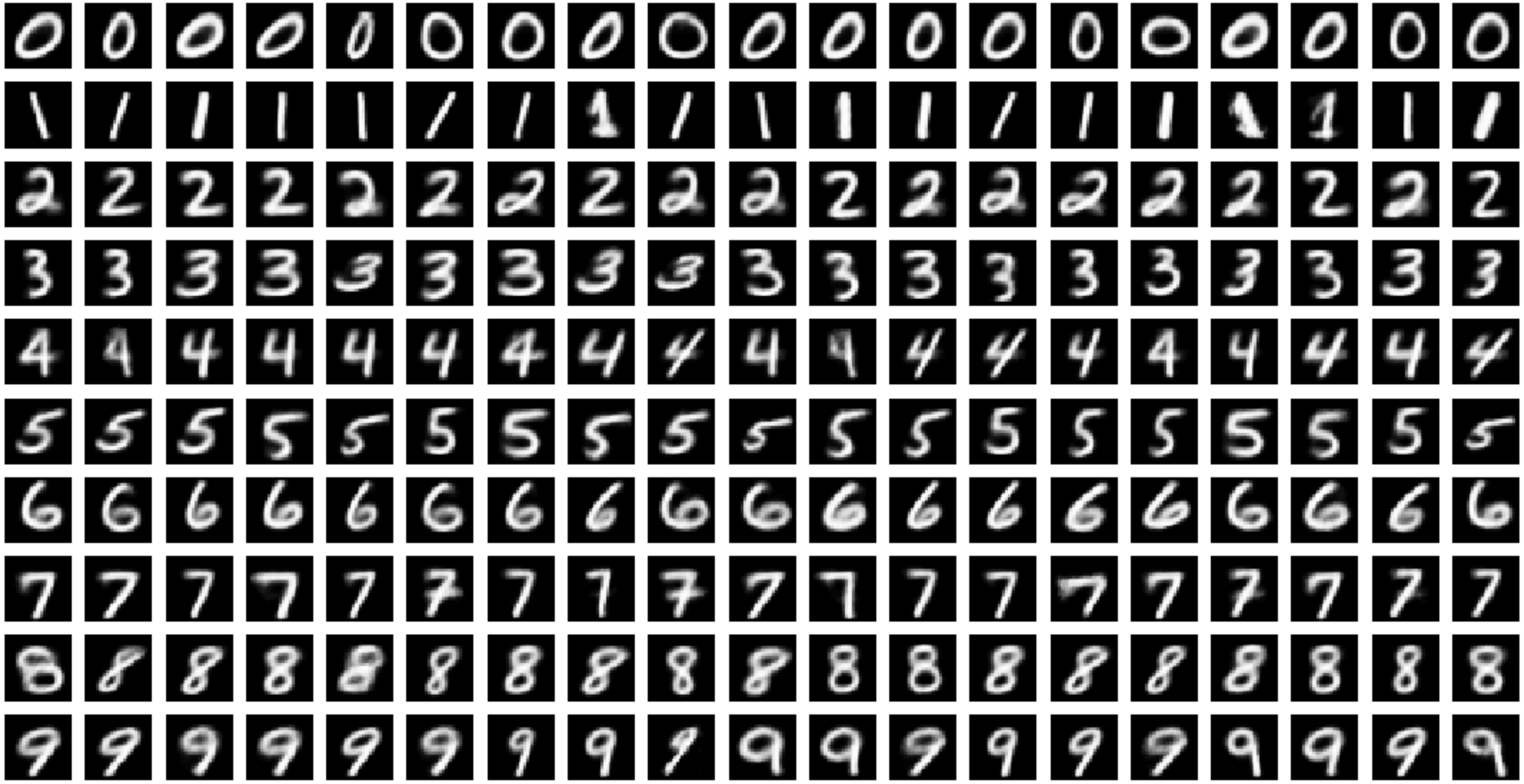
class	probability
alp (mountain)	0.7066206
geyser	0.106300876
valley	0.056517705



class	probability
rubber_eraser	0.53692865
envelope	0.1995637
paper_towel	0.040610846

Beyond classification

Image generation



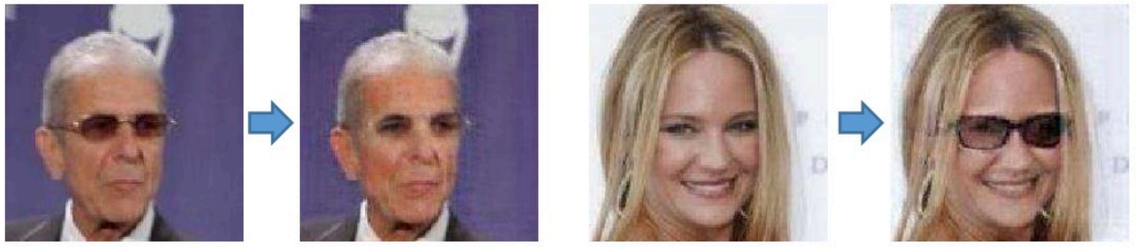
Beyond classification

Image generation

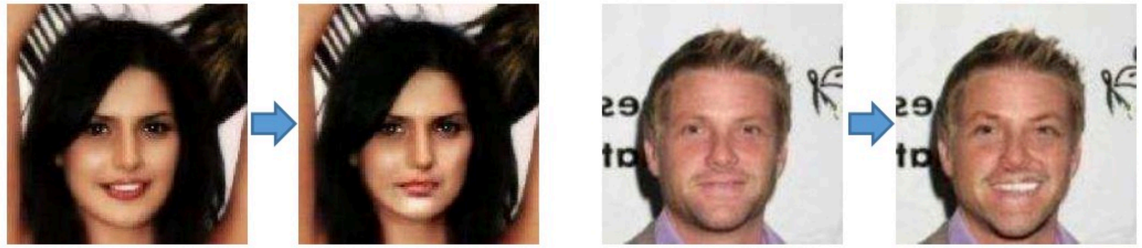
Figure 8.10 in Deep learning with python by Francois Chollet

Beyond classification

Image editing



(a) *Glasses*: remove and add the glasses



(b) *Mouth_open*: close and open the mouth

(c) *No_beard*: add and remove the beard

Shen, Wei, and Rujie Liu. "Learning residual images for face attribute manipulation." *CVPR* 2017.

Generative Models

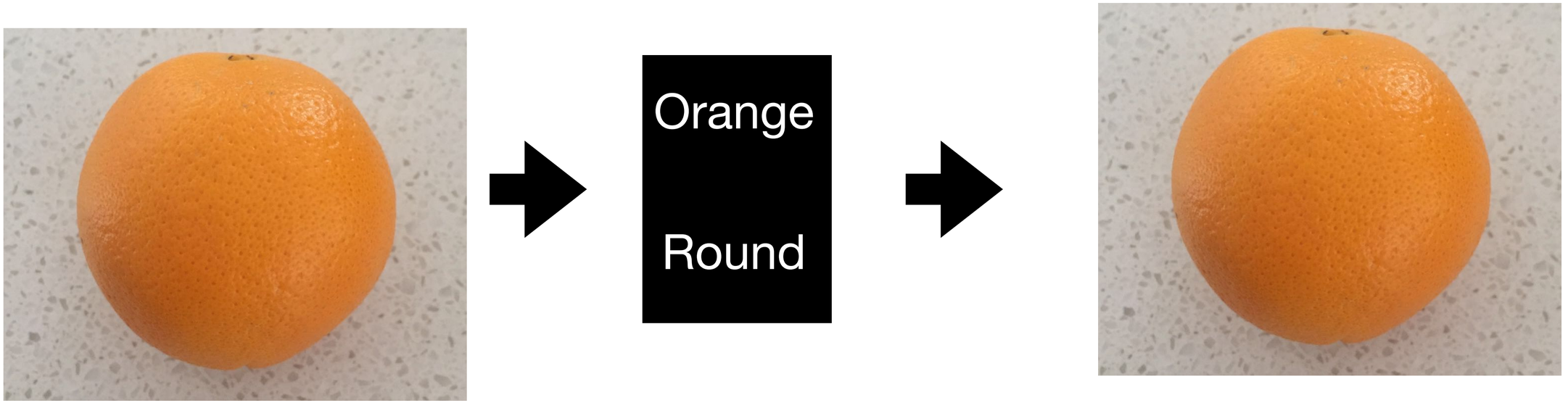
Cope with all of above tasks

- Variational Autoencoder (VAE)
- Generative Adversarial Network (GAN)

Outline

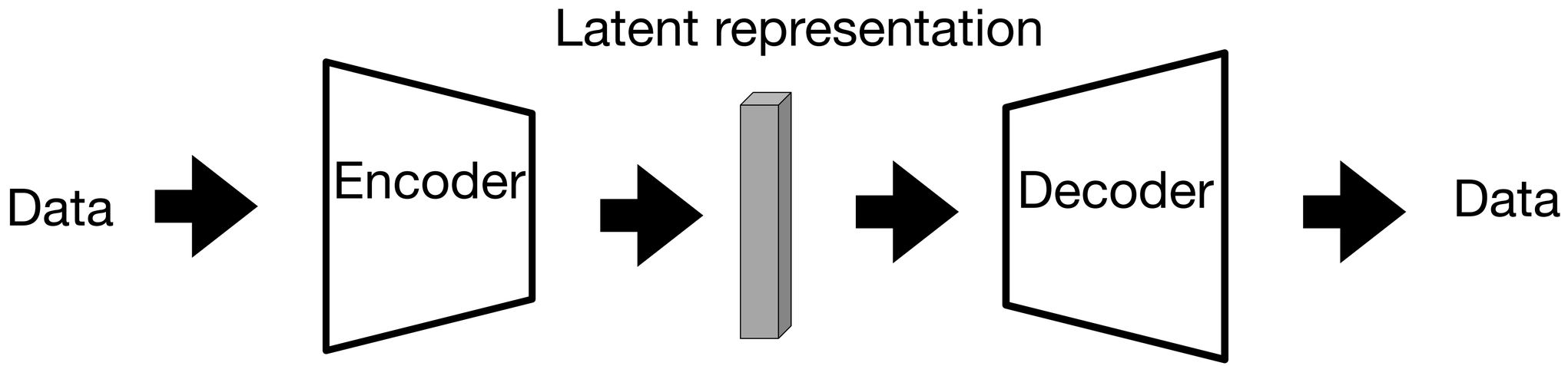
- Autoencoder (AE)
- Variational Autoencoder (VAE)
- Generative Adversarial Model (GAN)

Meaningful representations can reconstruct the input data

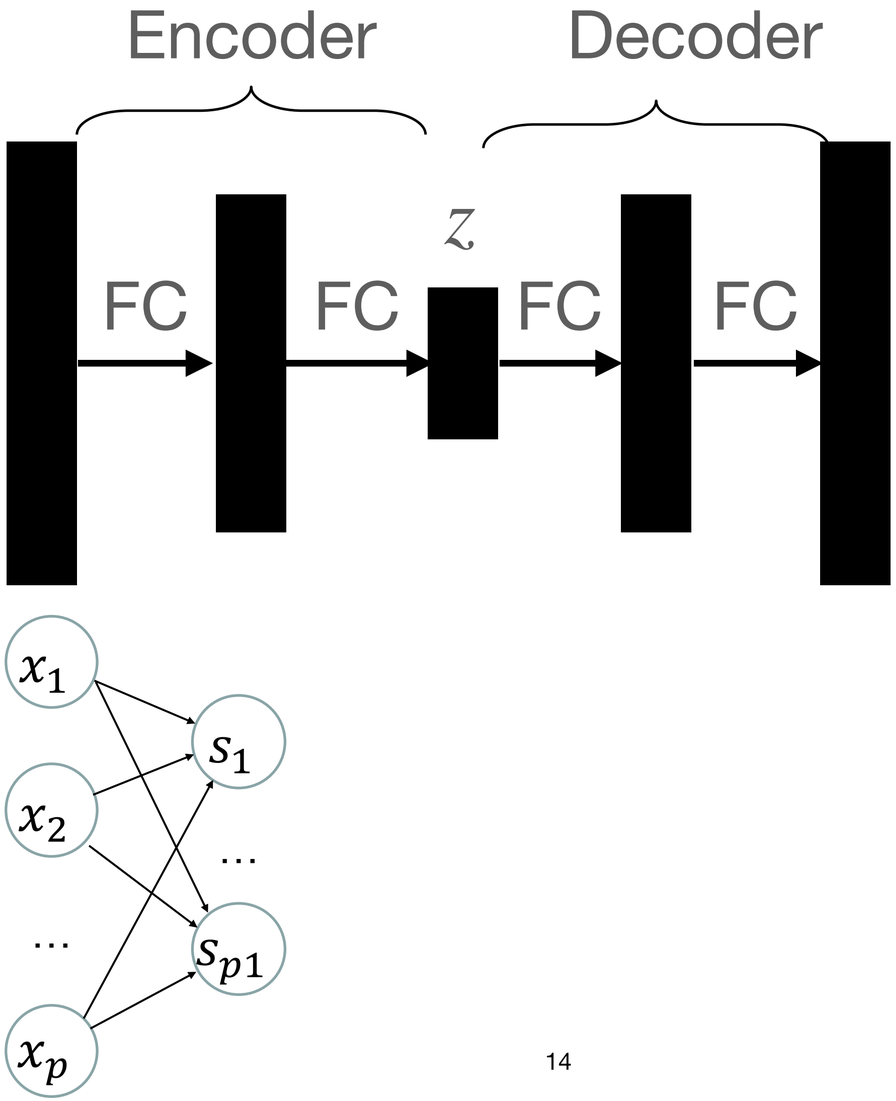


Encoder - decoder

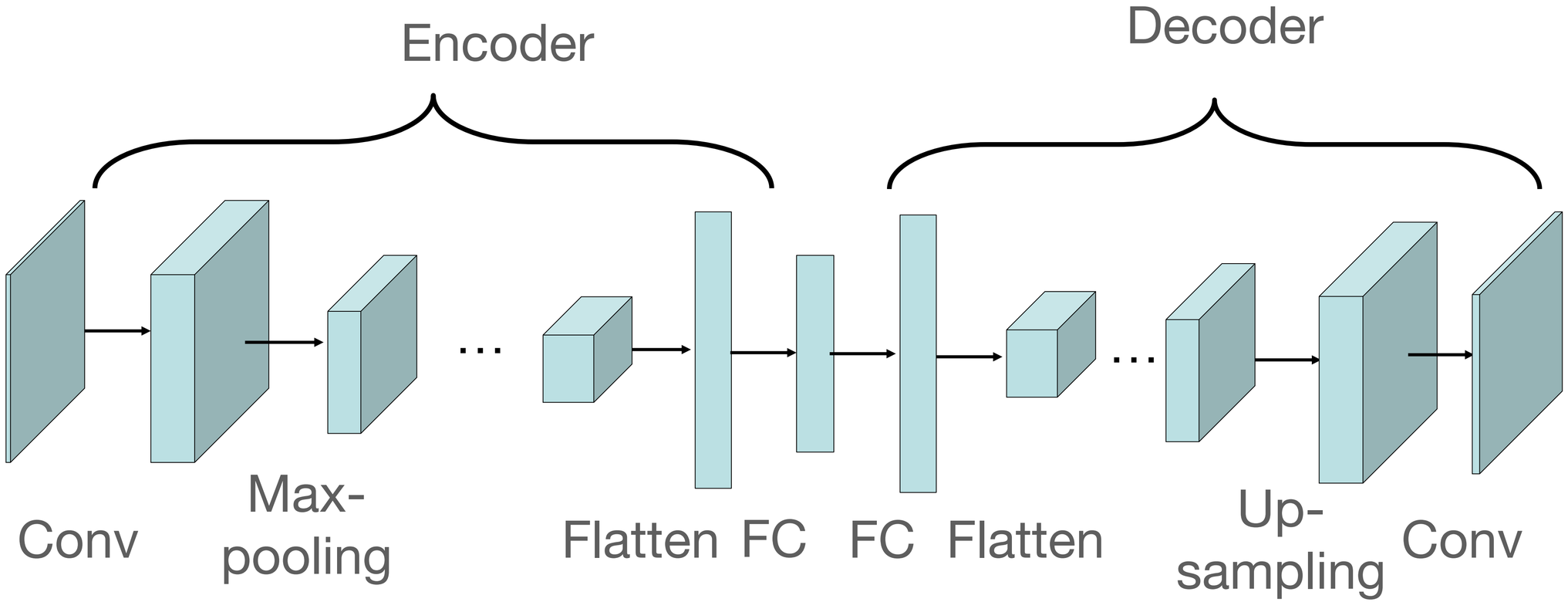
Extract meaningful representation to reconstruct the input data



Network architecture



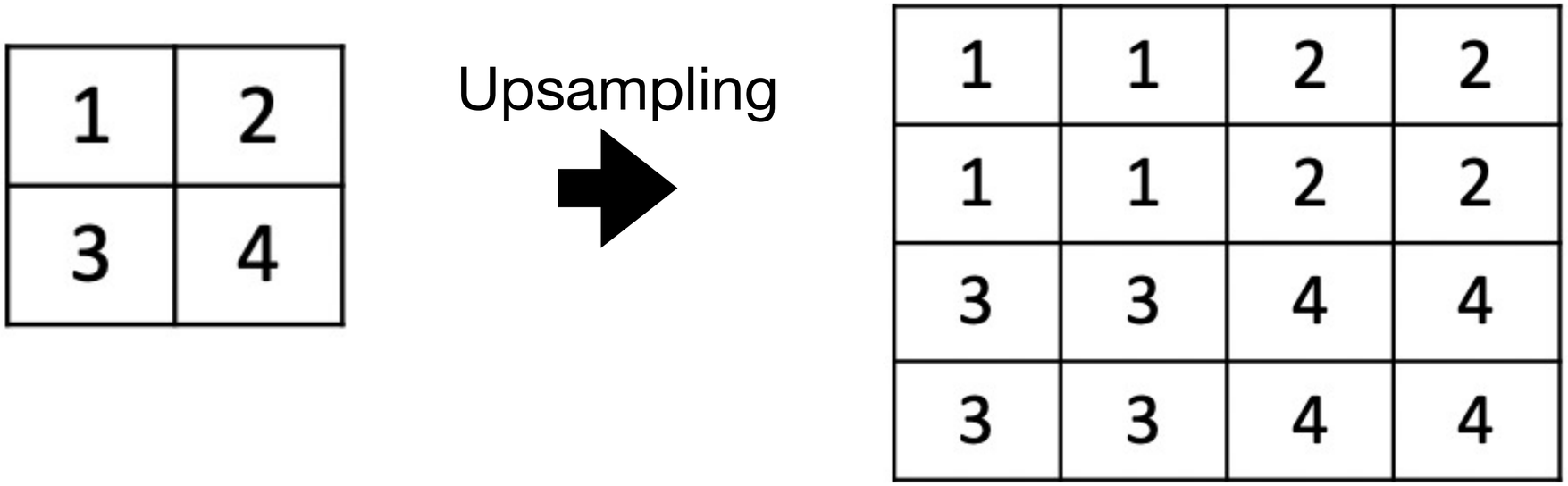
Network architecture



Upsampling

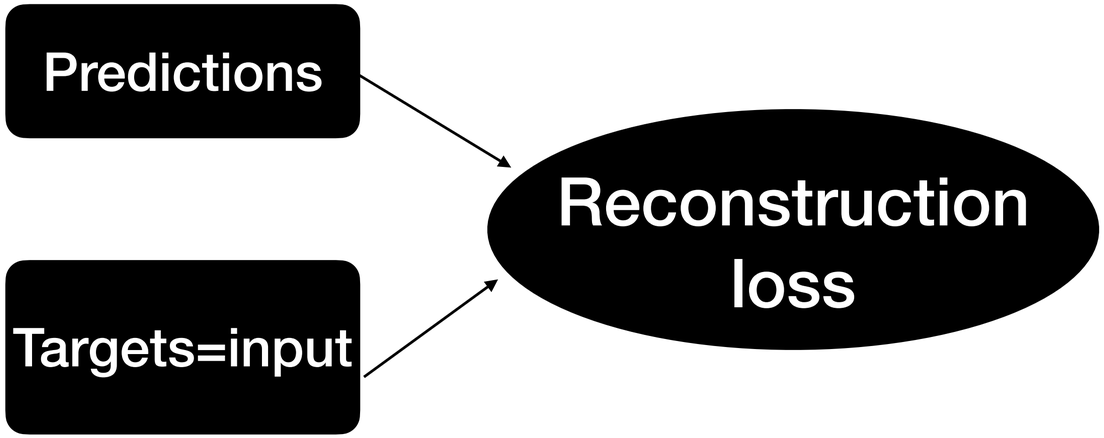
```
y = tf.keras.layers.UpSampling2D(size=(2, 2))(x)
```

Size: The upsampling factors for rows and columns

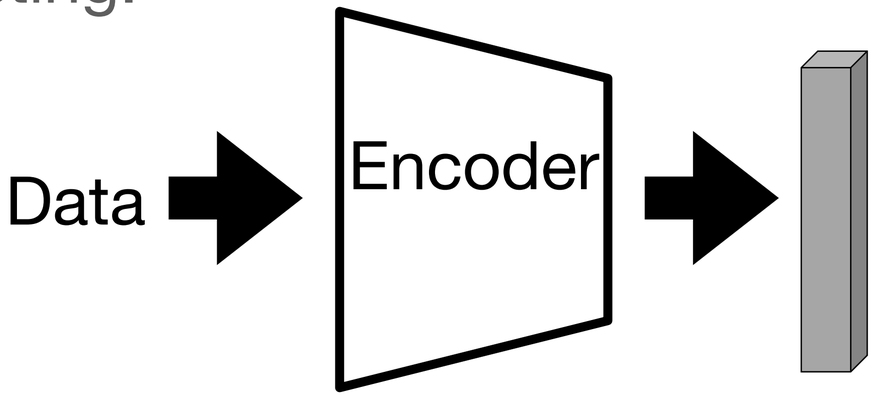


Training and testing

Training:



Testing:

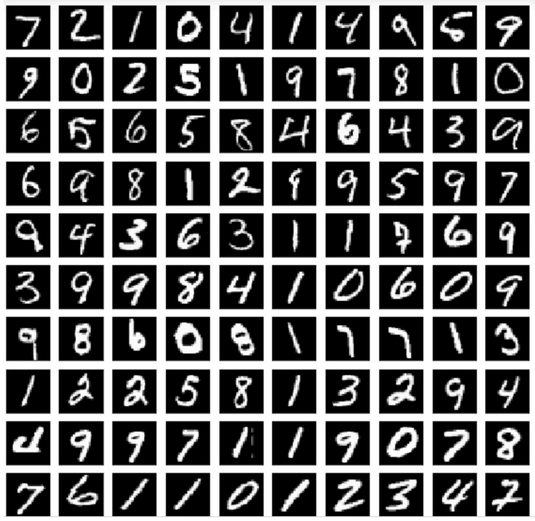


Loss:

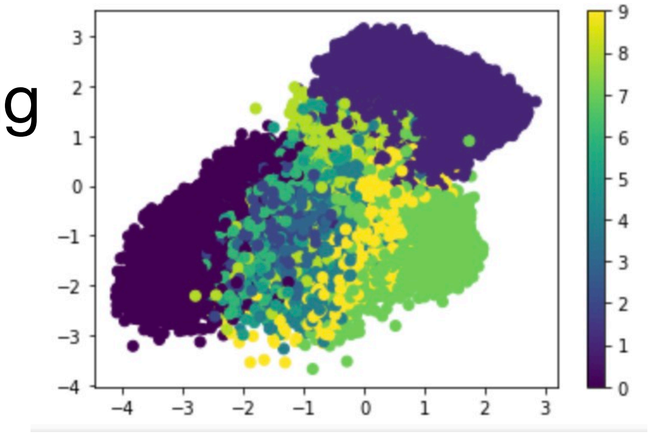
‘mse’: $L = E[(X - y_{pre})^2]$

‘binary_crossentropy’: Input is between 0~1

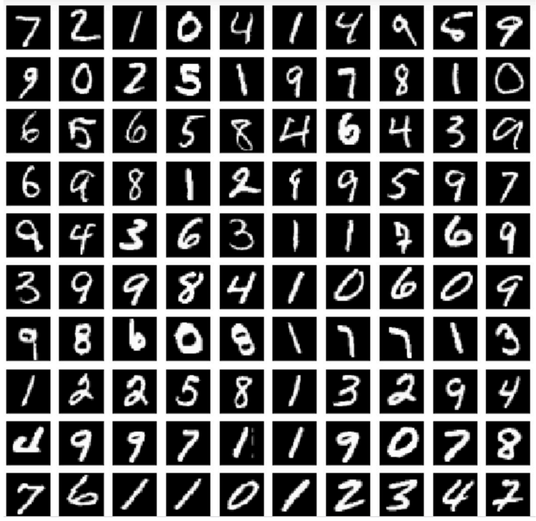
Dimension reduction



encoding

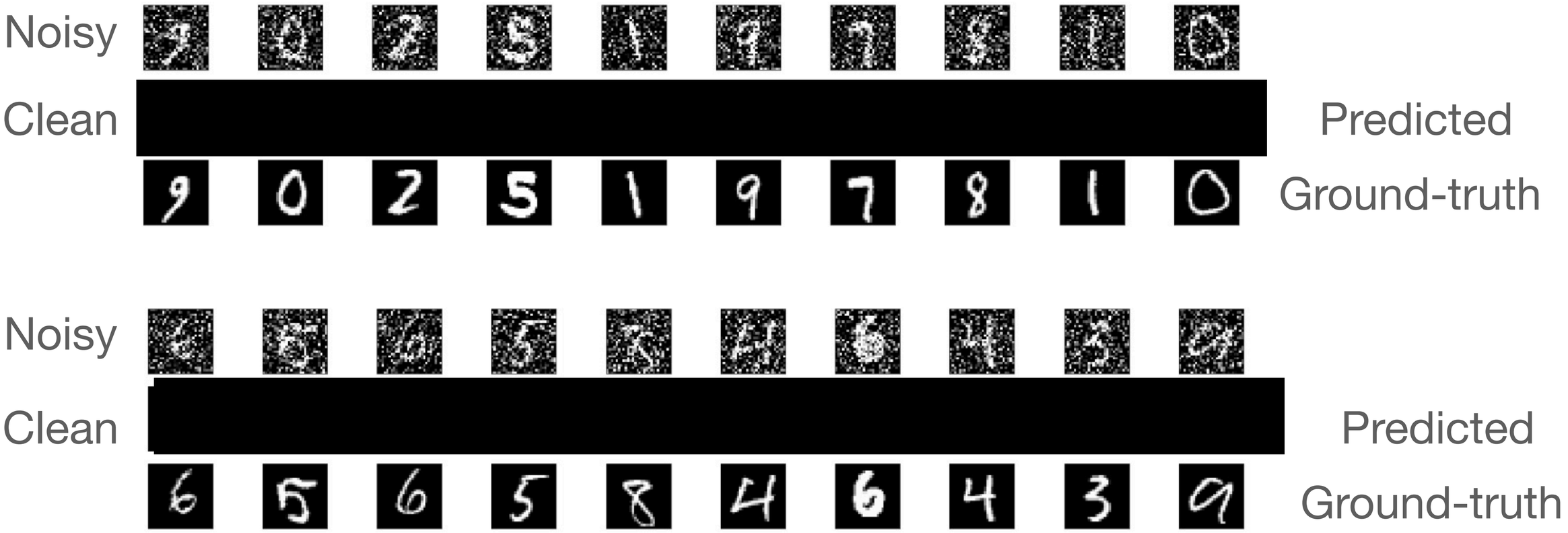


decoding



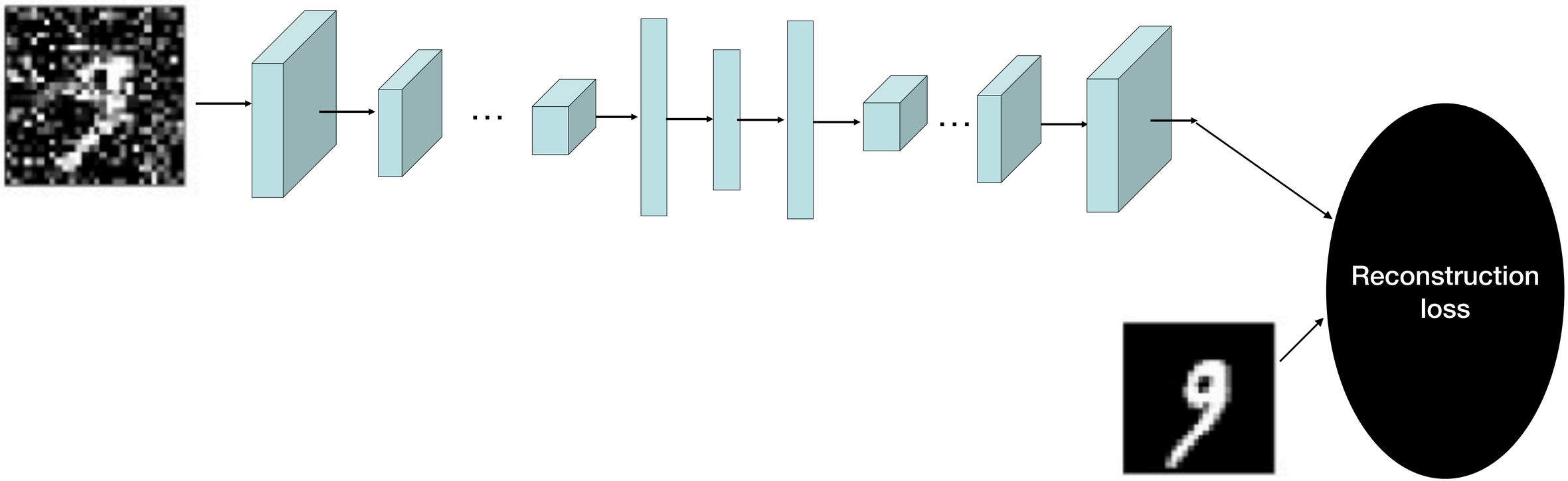
Data → Encoder → Decoder → Data

Image denoisying



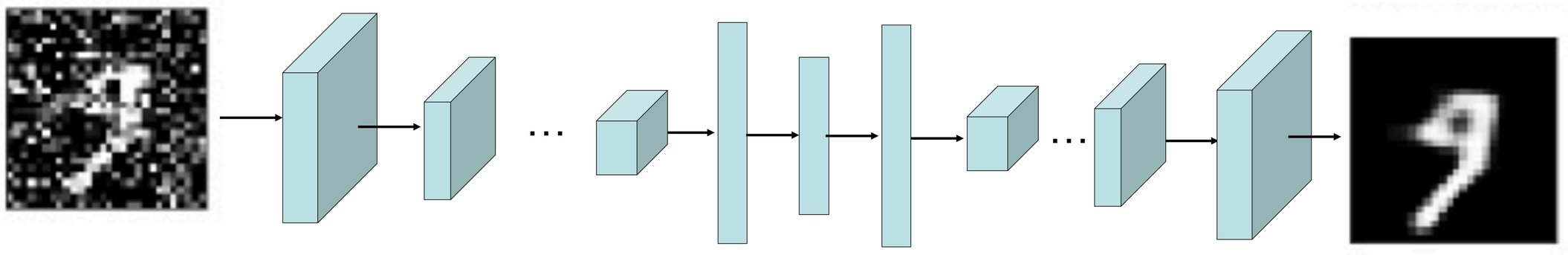
Denoisying Autoencoder

Training (have access to clean data):

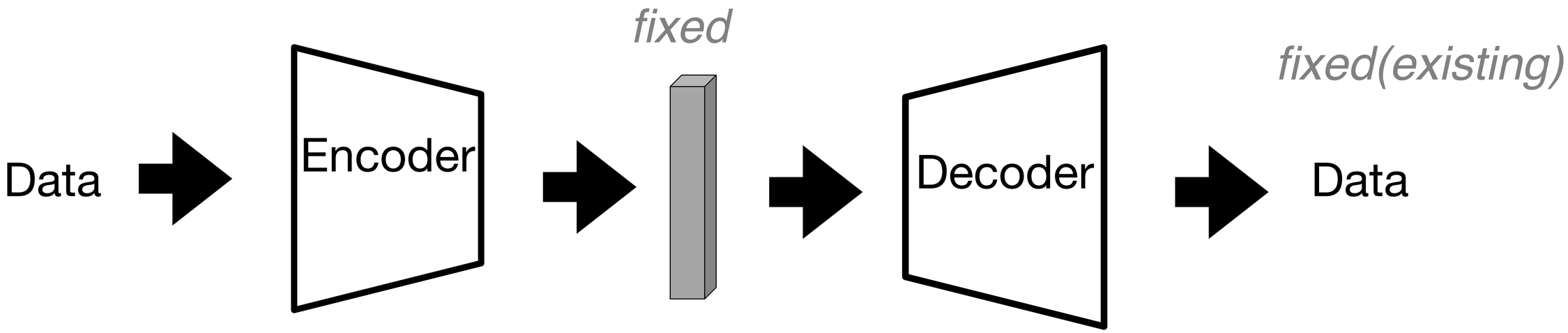


Denoisying Autoencoder

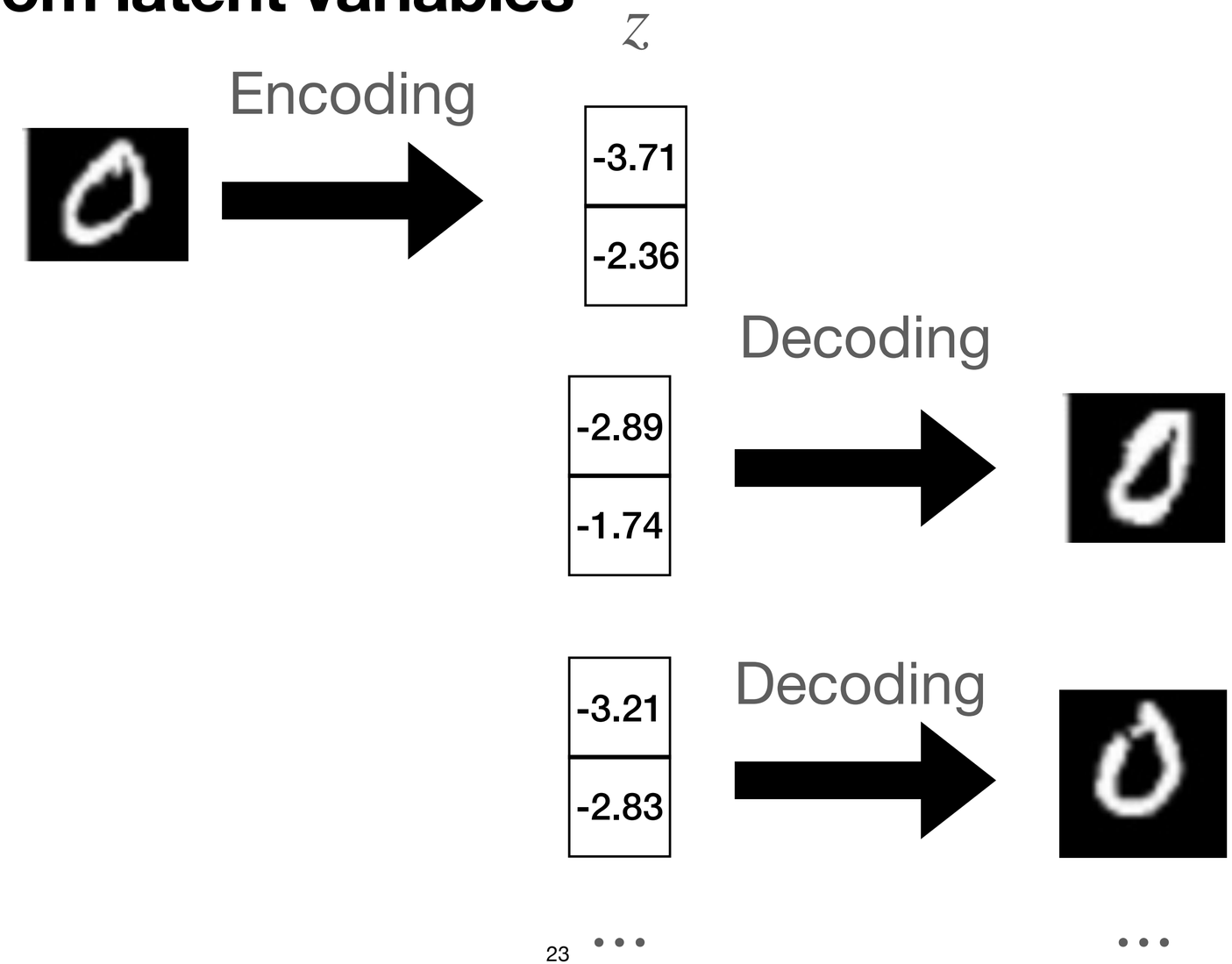
Testing:



Generate new data? NO



Data generation from latent variables



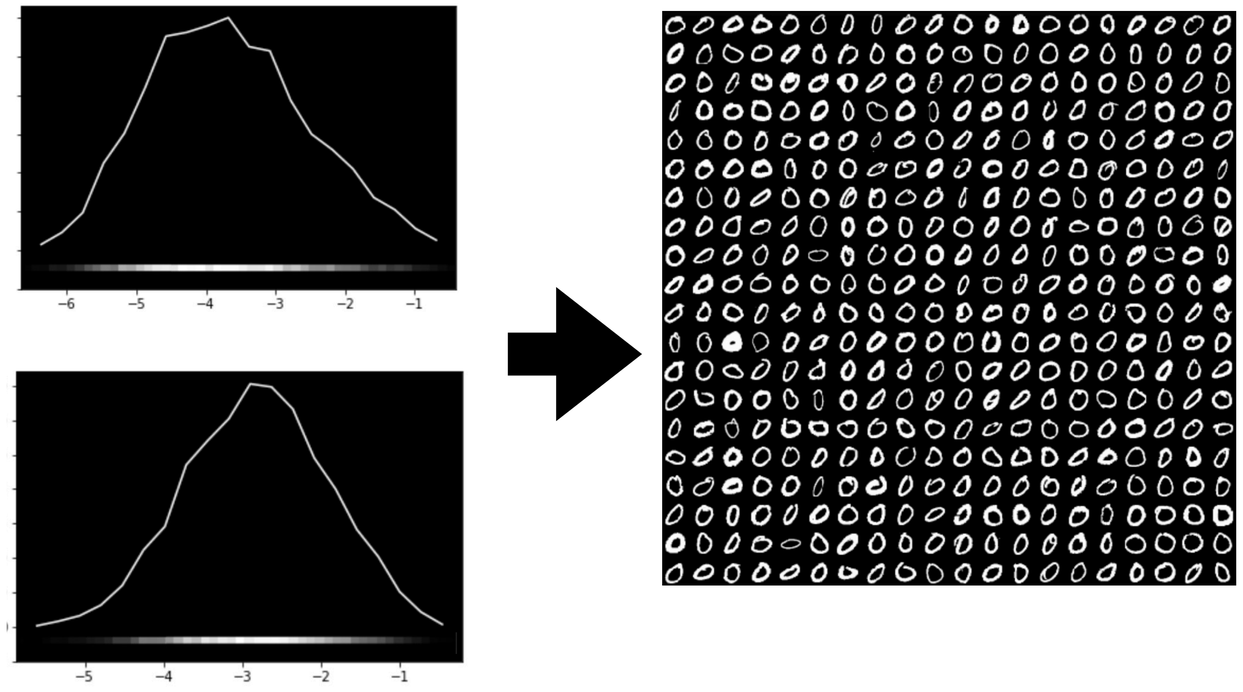
Data generation from latent variable

$$p(x) = \int p(x \mid z) p(z) dz$$

p(x): probability of the data x

p(z): probability of the latent variable z

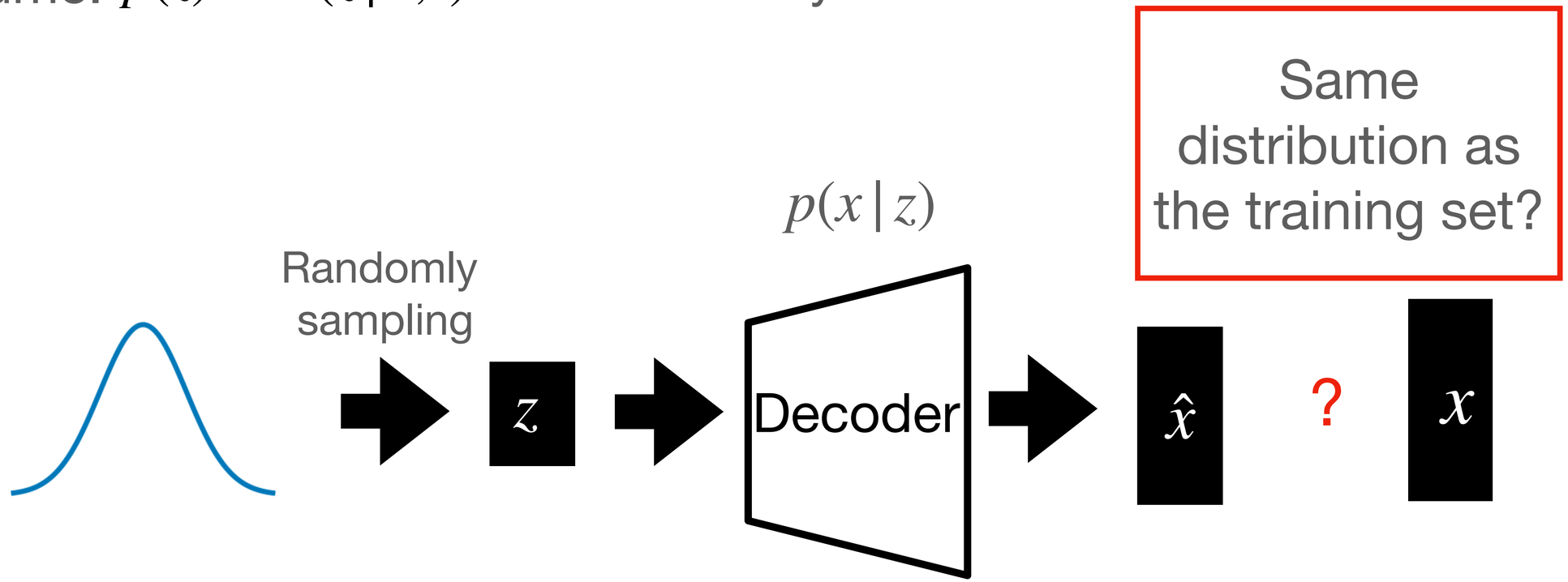
p(x|z): probability of x given z



Turn latent variable into data using a decoder (network)

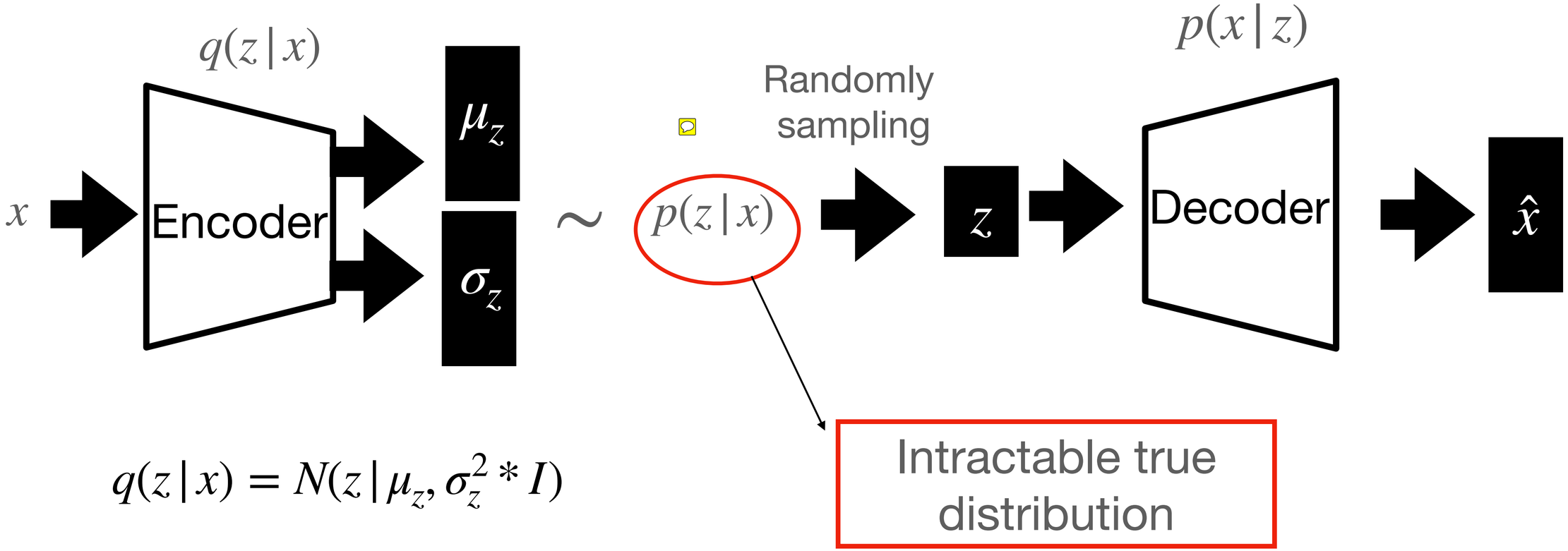
Assume: $p(z) = N(z \mid 0, I)$

I: identity matrix



Encoder: Reduce sampling space

Use posterior distribution to sample z is more likely to produce x



$$q(z|x) = N(z|\mu_z, \sigma_z^2 * I)$$

q is an approximation of p

Math is coming…

Network optimisation: Maximize log likelihood and gradient ascend

$$\log p(x^{(i)}) = E_{z\sim q(z|x)}[\log p(x^{(i)})]$$

$p(x,z)$
$= p(x|z)p(z)$
$= p(z|x)p(x)$

$$= E_{z\sim q(z|x)}\left[\log \frac{p(x^{(i)}\,|\,z)p(z)}{p(z\,|\,x^{(i)})}\right] = E_{z\sim q(z|x)}\left[\log \frac{p(x^{(i)}\,|\,z)p(z)}{p(z\,|\,x^{(i)})}\frac{q(z\,|\,x^{(i)})}{q(z\,|\,x^{(i)})}\right]$$

$$= E_{z\sim q(z|x)}\left[\log p(x^{(i)}\,|\,z)\right] - E_{z\sim q(z|x)}\left[\log \frac{q(z\,|\,x^{(i)})}{p(z)}\right] + E_{z\sim q(z|x)}\left[\log \frac{q(z\,|\,x^{(i)})}{p(z\,|\,x^{(i)})}\right]$$

KL Divergence

$$= E_{z\sim q(z|x)}\left[\log p(x^{(i)}\,|\,z)\right] - D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z)\right] + D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z\,|\,x^{(i)})\right]$$

Maximize lower bound

Unknown true posterior

$$\log p(x^{(i)}) = E_{z\sim q(z|x)}\left[\log p(x^{(i)}\,|\,z)\right] - D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z)\right] + D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z\,|\,x^{(i)})\right]$$

$$\geq E_{z\sim q(z|x)}\left[\log p(x^{(i)}\,|\,z)\right] - D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z)\right]$$

lower bound

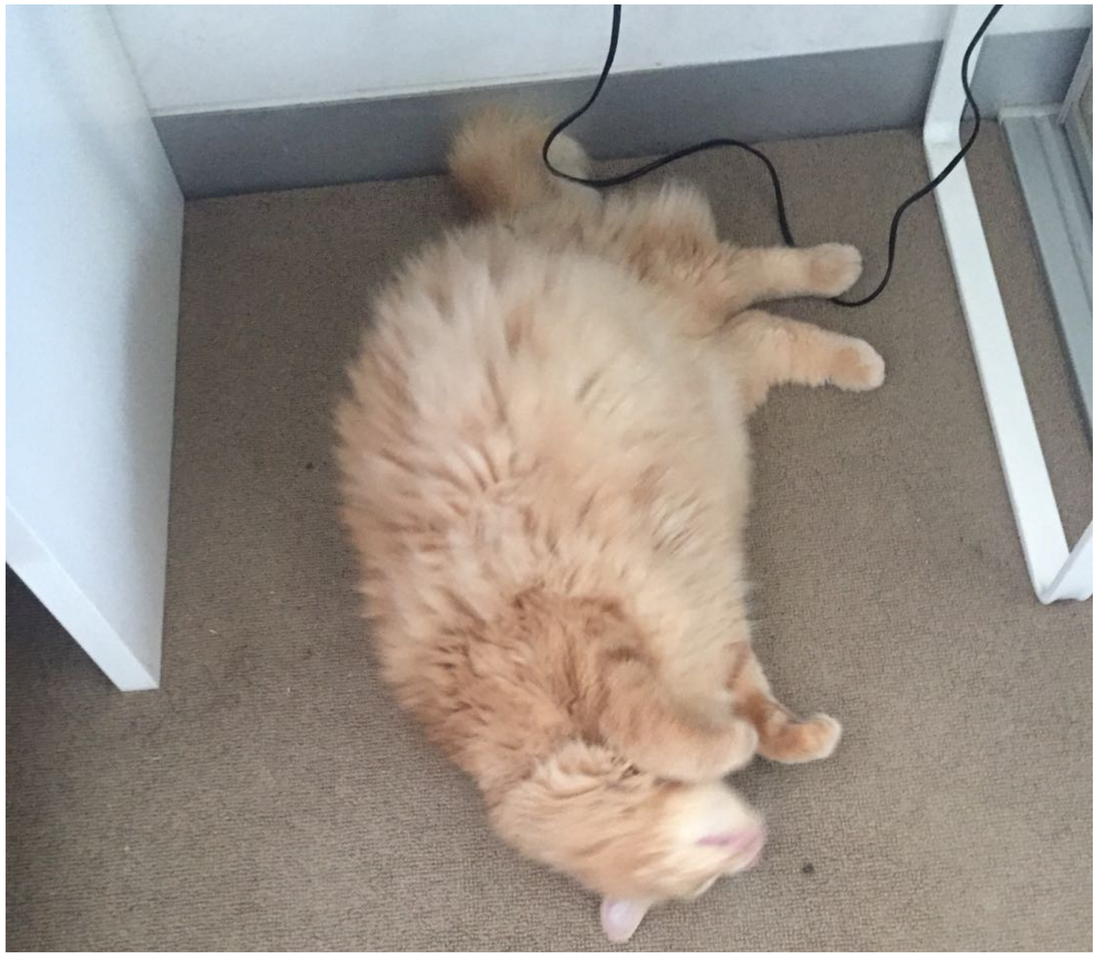
$\max E_{z\sim q(z|x)}\left[\log p(x^{(i)}\,|\,z)\right]$: Minimize reconstruction loss between output and input

$q(z\,|\,x)$: Output of encoder $q(z) = N(z\,|\,\mu, \sigma^2 I)$

$p(z)$: Prior distribution, assumed to be standard Gaussian distribution $p(z) = N(z\,|\,0, I)$

$D_{KL}\left[q(z\,|\,x^{(i)})\,||\,p(z)\right]$: KL divergence of two gaussian distributions

Math is coming AGAIN…



Warning of math: KL divergence

$$KL(p_1||p_2) = \int p_1(x) \log \frac{p_1(x)}{p_2(x)} dx \qquad p_1(x) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2} \qquad p_2(x) = \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2}$$

$$= \int [-log\sigma_1 - \frac{1}{2}(log(2\pi)) - \frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2 - (-log\sigma_2 - \frac{1}{2}(log(2\pi)) - \frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2] p_1(x) dx$$

$$= -log\sigma_1 + log\sigma_2 \quad -\frac{1}{2\sigma_1^2} E_1[(x-\mu_1)^2] \quad +\frac{1}{2\sigma_2^2} E_1[(x-\mu_2)^2]$$

$$= -log\sigma_1 + log\sigma_2 \quad -\frac{1}{2} \quad +\frac{1}{2\sigma_2^2} E_1[(x-\mu_2)^2]$$

Warning of math: KL divergence

$$KL(p_1||p_2) = \int p_1(x) \log \frac{p_1(x)}{p_2(x)} dx = -log\sigma_1 + log\sigma_2 - \frac{1}{2} + \frac{1}{2\sigma_2^2} E_1[(x-\mu_2)^2]$$

$$p_1(x) \sim N(\mu, \sigma^2) \qquad p_2(x) \sim N(0,1)$$

$$= -log\sigma + 0 - \frac{1}{2} + \frac{1}{2} E_1(x^2)$$

$$KL(p_1, p_2) = -log\sigma - \frac{1}{2} + \frac{1}{2}(\sigma^2 + \mu^2)$$

$$= \frac{1}{2}(-log\sigma^2 + \sigma^2 + \mu^2 - 1)$$

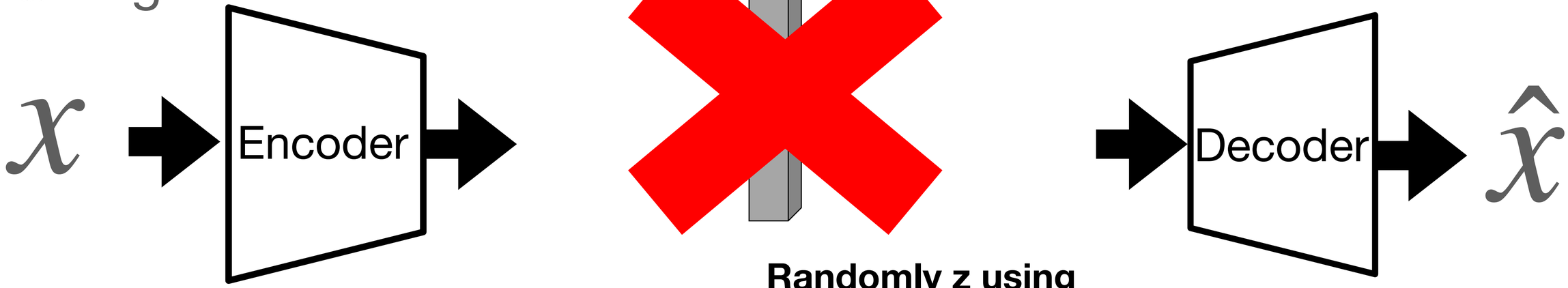
$x^2 = (x-\mu)^2 + 2\mu x - \mu^2$

$E(x^2) = E(\quad)$

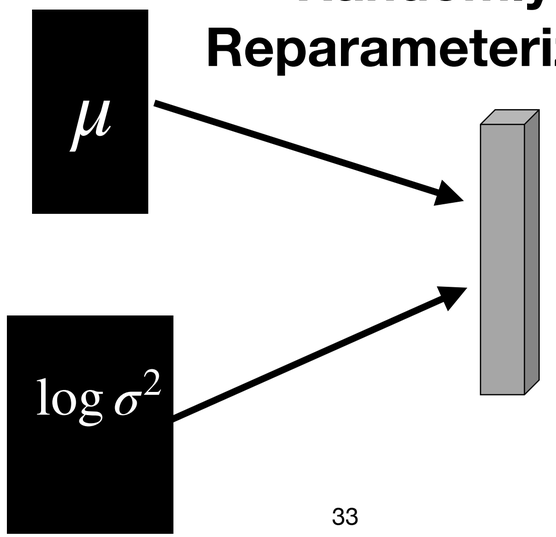
$E(x^2) = \sigma^2 + \mu^2$

(Summary) Different from AE: estimate the distribution of z

Training

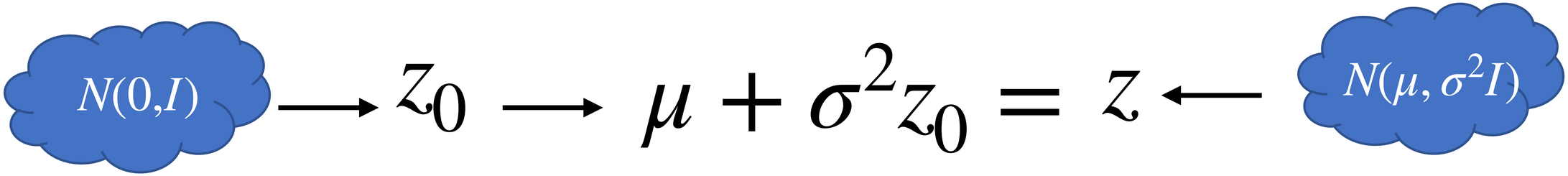


Randomly z using Reparameterization trick



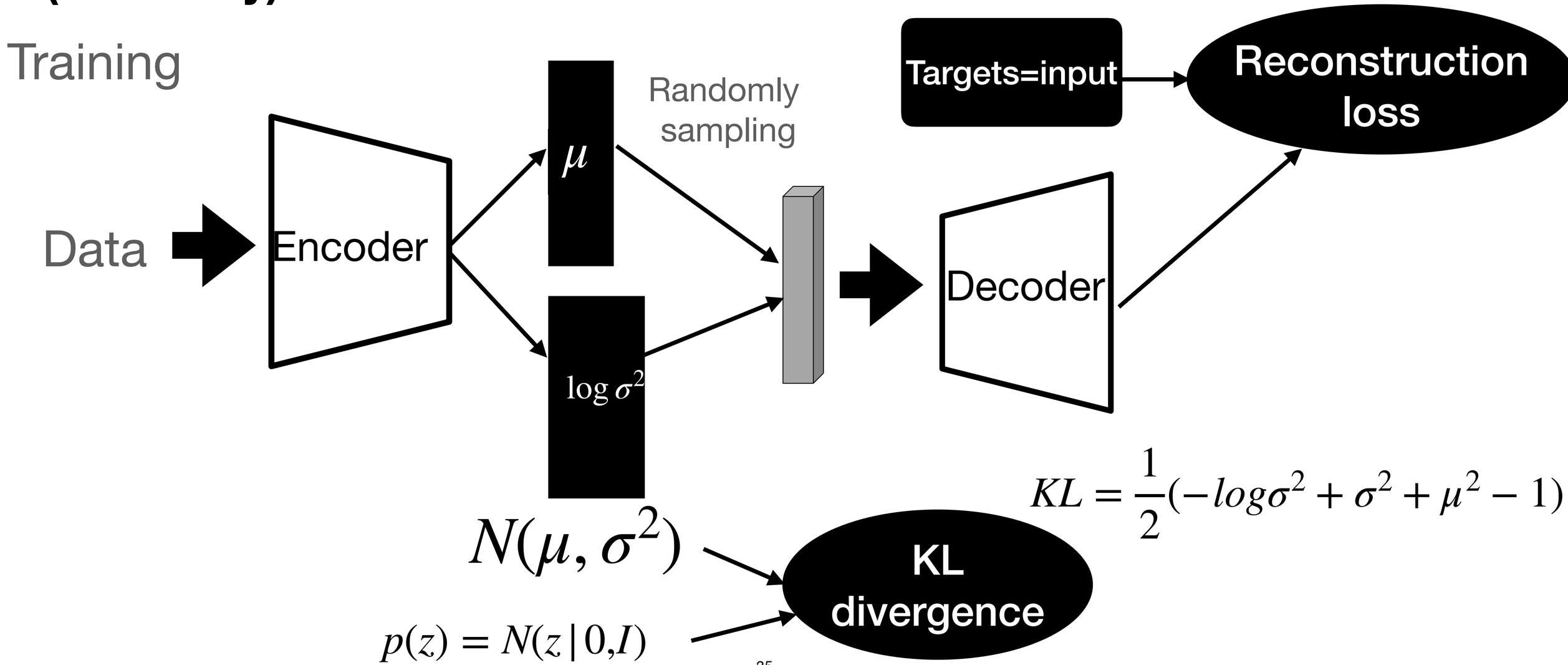
For easy and stable training: output $log\sigma^2$

Sample z: Reparameterization trick to make network differentiable

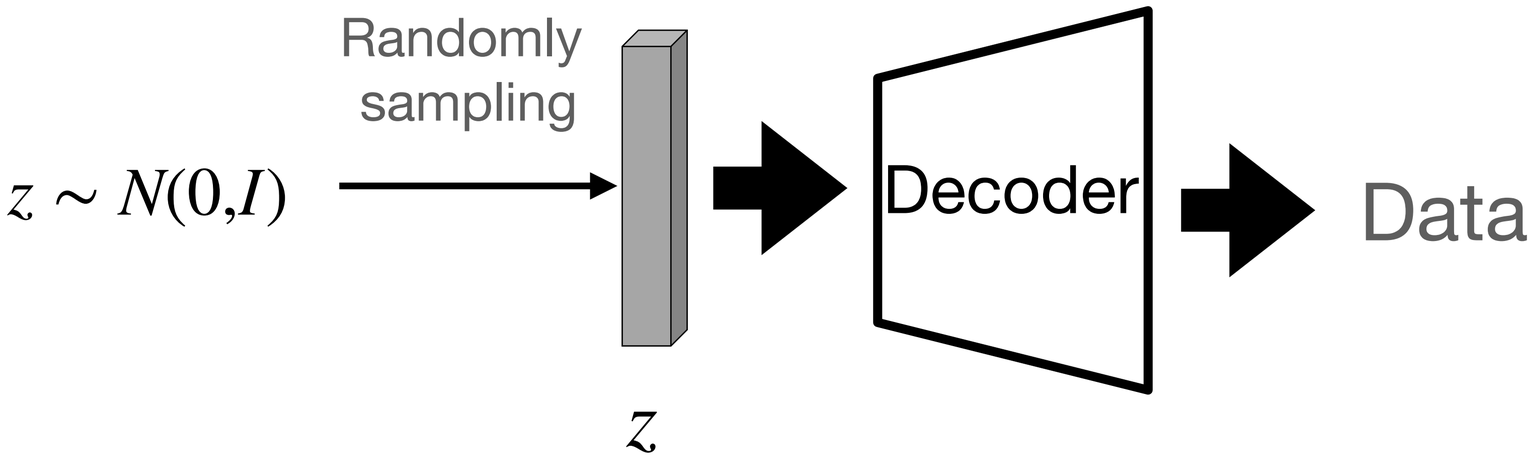



```
z0 = K.random_normal(shape=(batch_size, z_dim), mean=0., stddev=1.0)
z=mu + K.exp(log_var/2) * z0
```

(Summary) Different from AE: Two losses



Testing

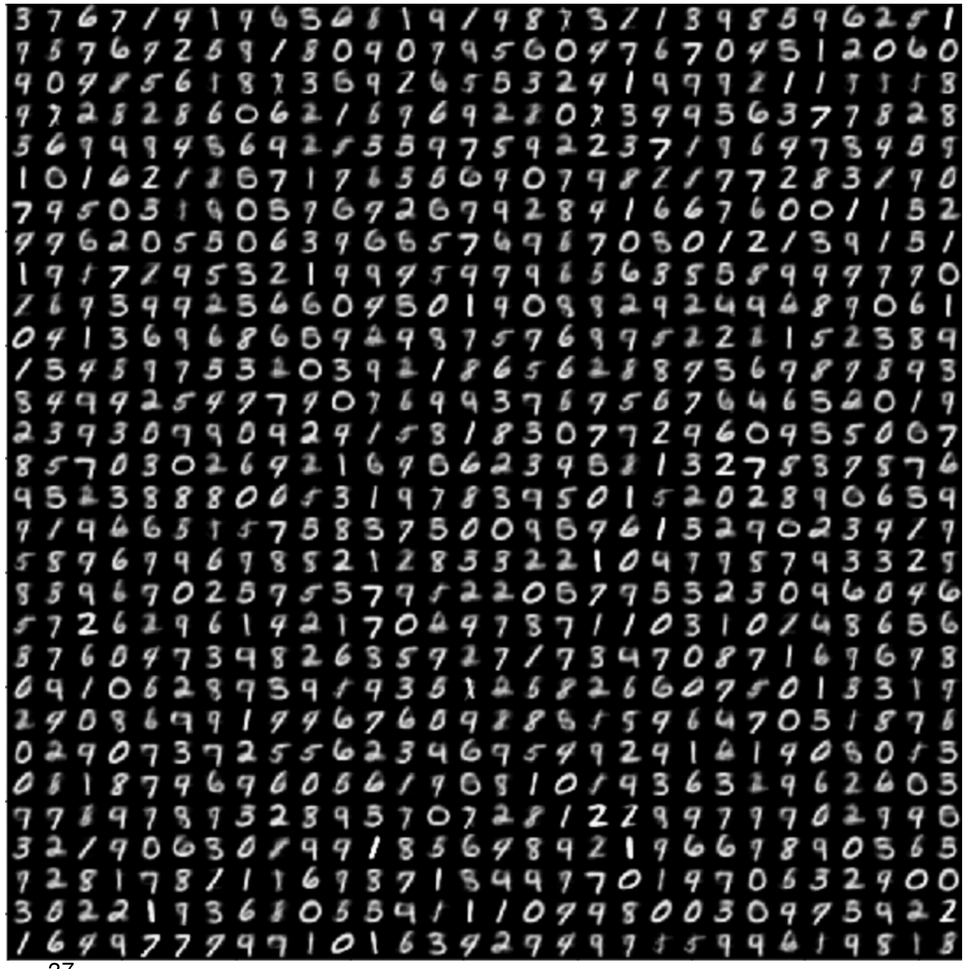


Testing

Random z

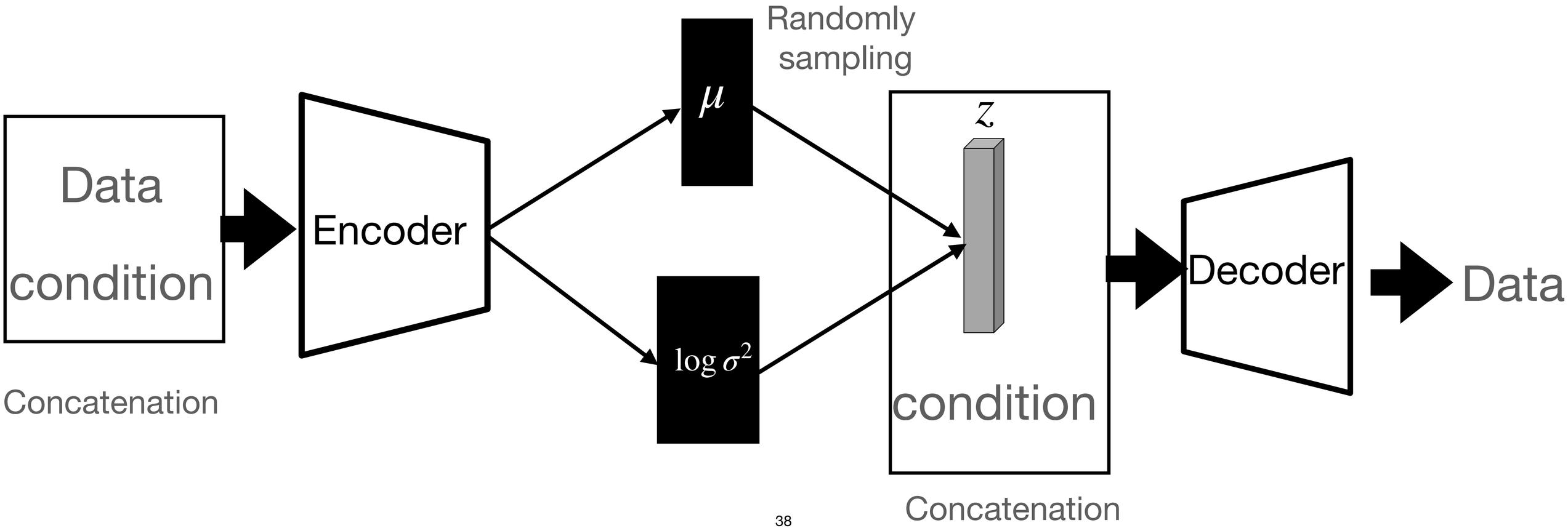
$$z \sim N(0,I)$$

Random digit



Conditional VAE

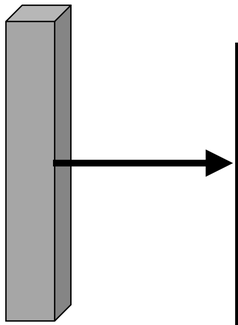
condition: $c = [1,0,0,\cdots,0]^T$



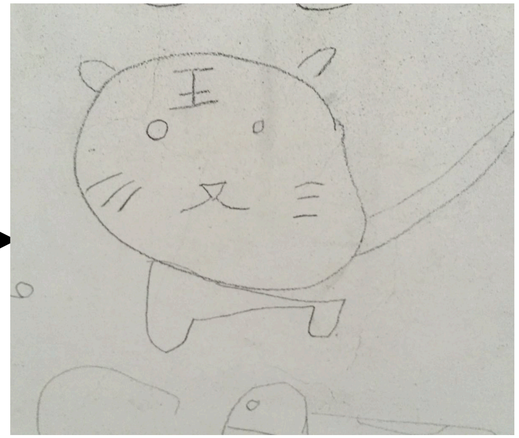
Two-player game

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

$z \sim N(0,I)$



Generator



Aims to generate more realistic images
to cheat the discriminator

Aims to tell whether the image
is generated or not

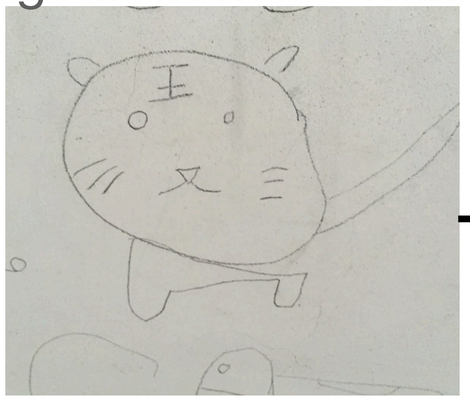
Discriminator

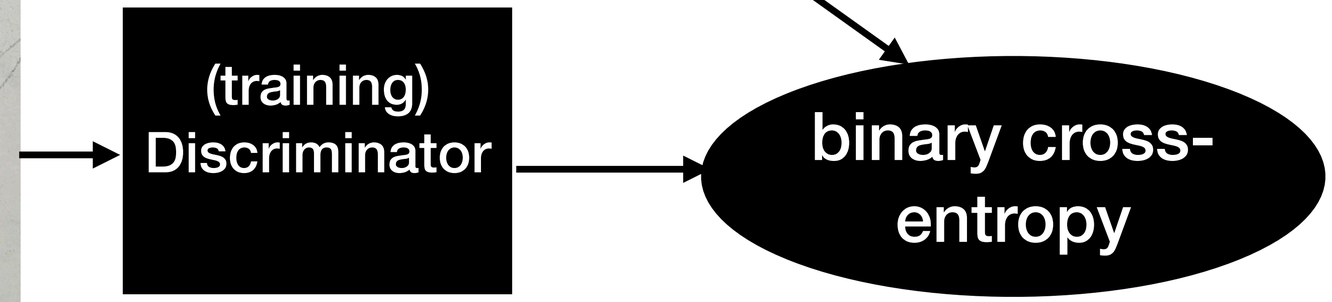
Real ?
Fake?

Source: https://pixabay.com/photos/tiger-cub-tiger-cub-sumatran-cute-164992/

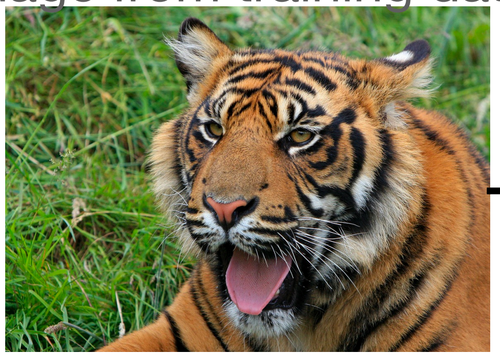
Discriminator: binary cross-entropy

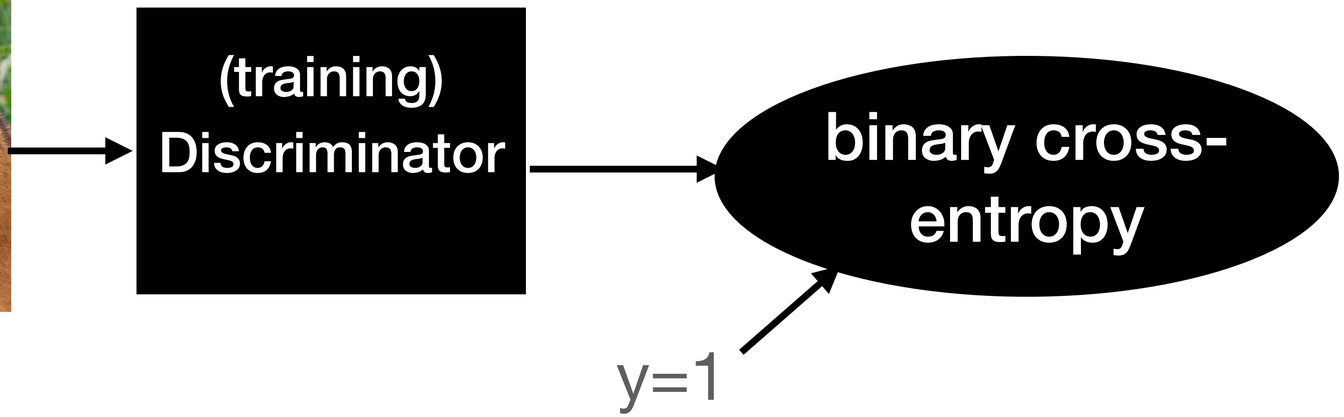
Generated image from noise with generator





Real image from training dataset





Source: https://pixabay.com/photos/tiger-cub-tiger-cub-sumatran-cute-164992/

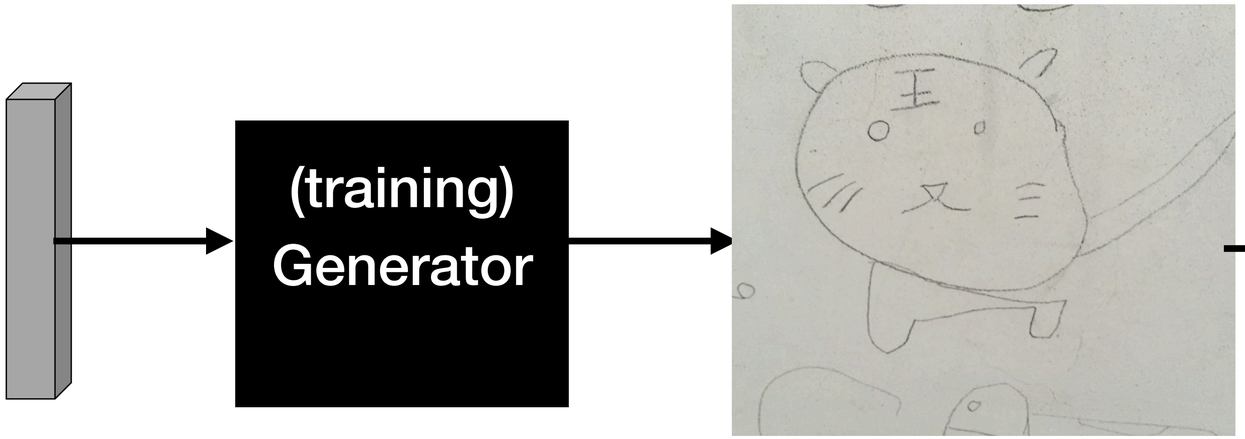
Keras code for discriminator

```
optimizer = Adam(0.001)

discriminator.compile(loss='binary_crossentropy',
    optimizer=optimizer,
    metrics=['accuracy'])
```

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

# Generate a batch of new images
noise = np.random.normal(0, 1, (batch_size, self.latent_dim))
gen_imgs = generator.predict(noise)

d_loss_real = discriminator.train_on_batch(real_imgs, valid)
d_loss_fake = discriminator.train_on_batch(gen_imgs, fake)
```

Generator: Fool discriminator by adversarial loss

During the training of the generator the discriminator is fixed



Is the generated image similar to real image?

Use the discriminator to check by predicting score.

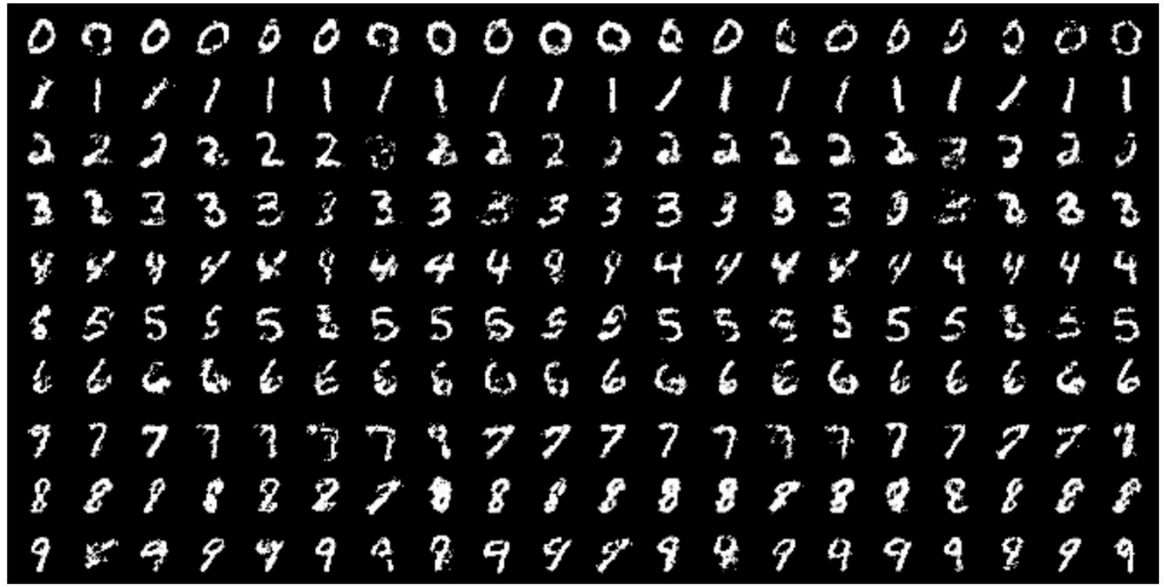
If similar to real, discriminator will output high score, and the BCE with 1 (adversarial loss) should be small

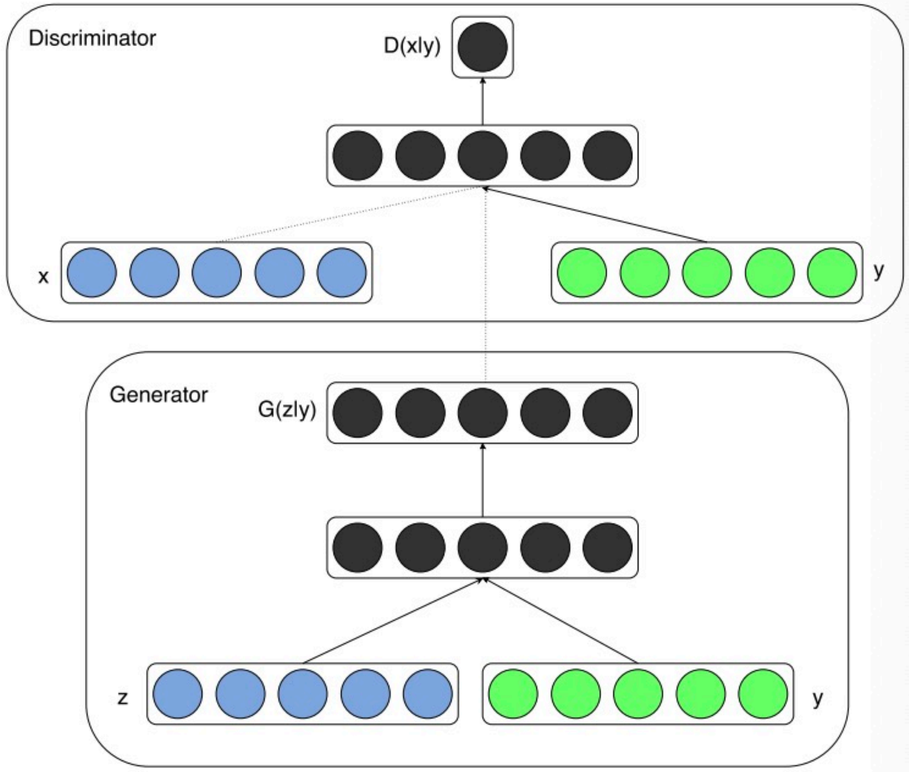
So the generator update weights to minimise the adversarial loss

Keras code for generator

```
# generate image
z = Input(shape=(self.latent_dim,))
gen_img = generator(z)

# fix weights of discriminator for prediction only
discriminator.trainable = False

# output score for generated image
validity = discriminator(gen_img)

# The combined model: input noise, output prediction of discriminator to calculate loss
combined_model = Model(z, validity)
combined_model.compile(loss='binary_crossentropy', optimizer=optimizer)


valid = np.ones((batch_size, 1))
noise = np.random.normal(0, 1, (batch_size, self.latent_dim))
# update generator to make the output of discriminator to be similar to 1
g_loss = combined_model.train_on_batch(noise, valid)
```

Conditional generative adversarial nets

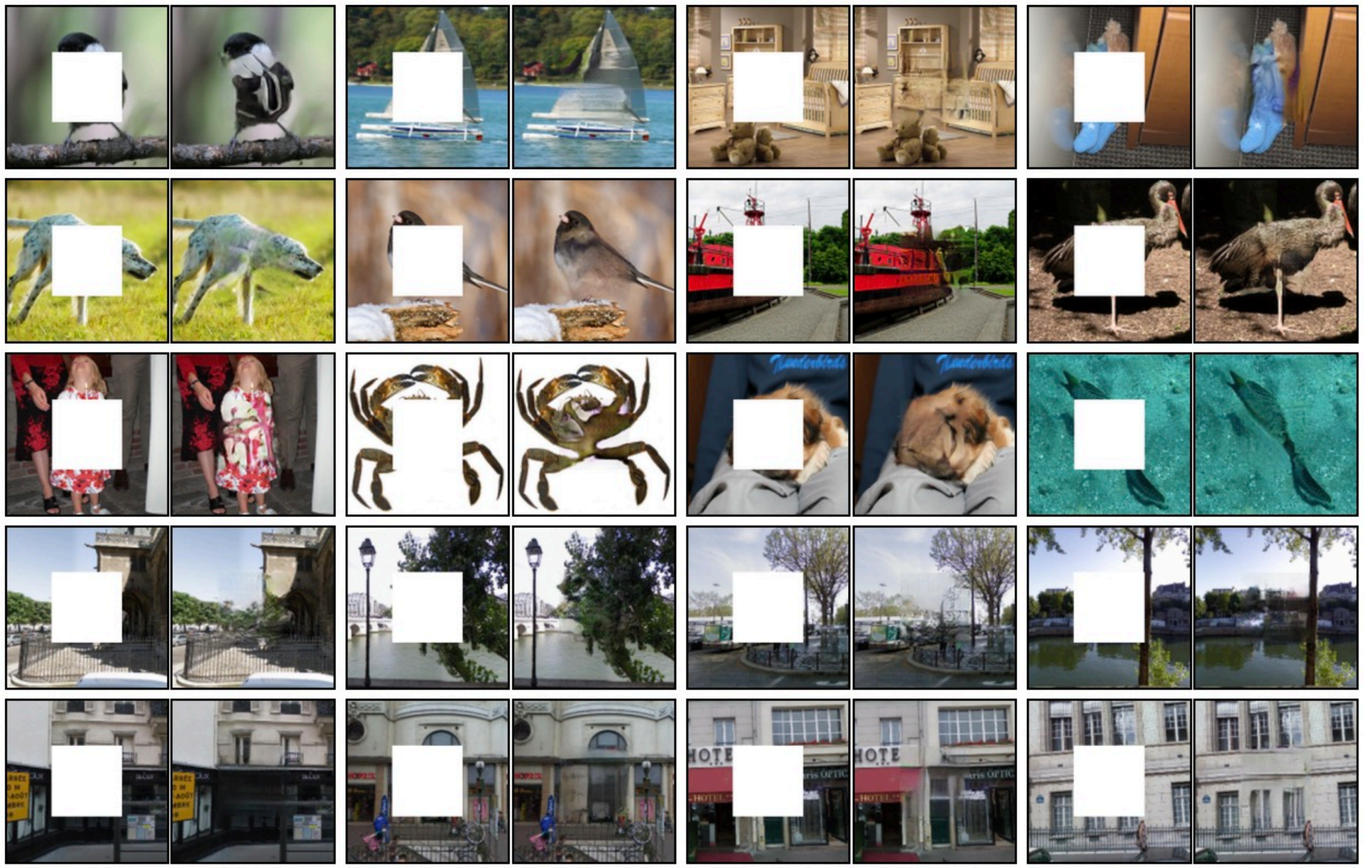


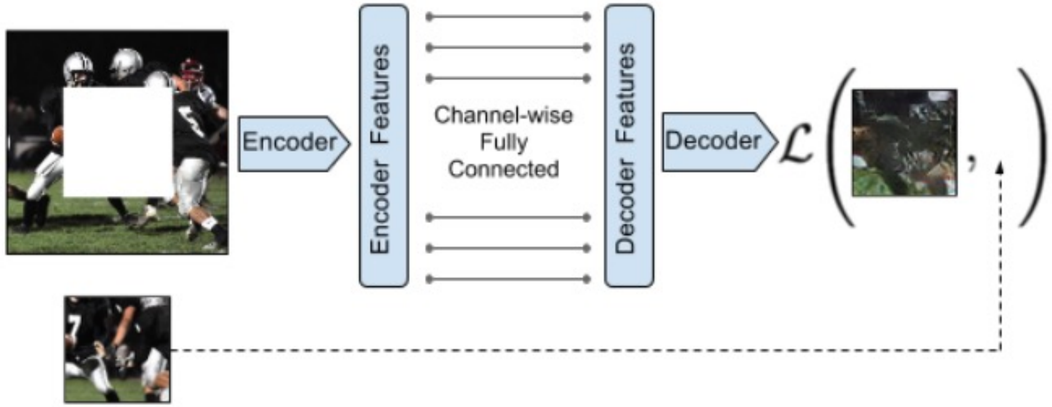


Combine the class vector with the noise

Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784* (2014).

Context encoder





Pathak, Deepak, et al. "Context encoders: Feature learning by inpainting." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Context encoder

Reconstruction loss (L2)

Generator

(training) Generator

(Frozen) Discriminator

Adversarial loss

binary cross-entropy

y=1

Discriminator is the same as the original gan

Pathak, Deepak, et al. "Context encoders: Feature learning by inpainting." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Other applications

Jin, Yanghua, et al. "Towards the automatic anime characters creation with generative adversarial networks." *arXiv preprint arXiv:1708.05509* (2017)

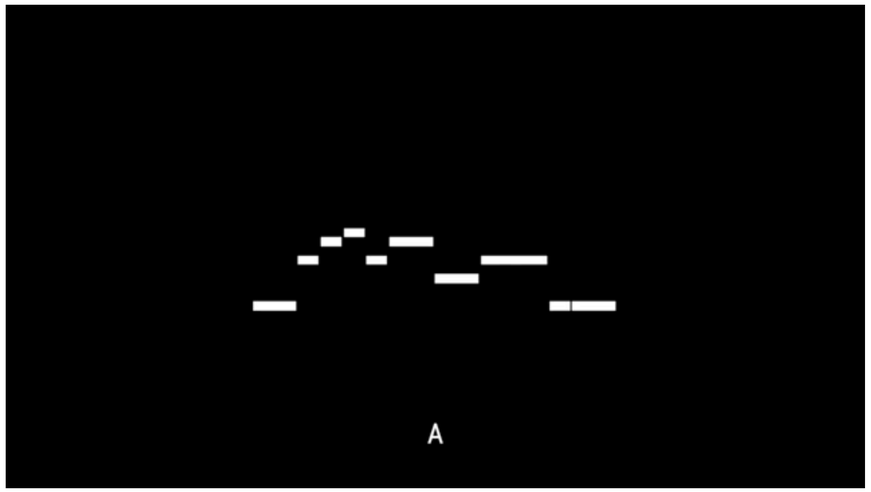
Other applications

Turning a horse video into a zebra video (by CycleGAN)

https://www.youtube.com/watch?time_continue=1&v=9reHvktowLY&feature=emb_logo

Summary

- How to train AE to learn meaningful representation and denoisy image?
- How to train VAE?
- How to train GAN?



https://www.youtube.com/watch?time_continue=87&v=G5JT16flZwM&feature=emb_logo

Next: Temporal Convolutional Network & Recurrent Neural Network