Implementing ANNs with TensorFlow

Session 11 - Generative Models

Agenda

1. Unsupervised Learning
2. Auto-Encoders
3. GANs
4. Flow-Models
5. Interpretable Dimensions

Unsupervised Learning

Unsupervised Learning

- Unsupervised learning is the task of finding “structure” in your data.

- There are no labels.

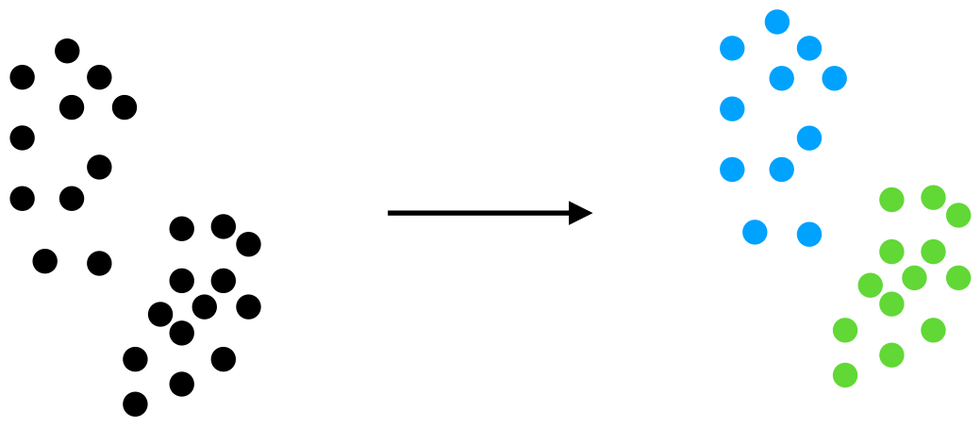
Supervised Learning: $(\vec{x}_i, \vec{t}_i)_{i=1}^{N}$

Unsupervised Learning: $(\vec{x}_i)_{i=1}^{N}$

Example: Clustering

- A simple example for an unsupervised learning algorithm is clustering.

- In clustering you are trying to find classes (the clusters) just based on the structure of the dataset.

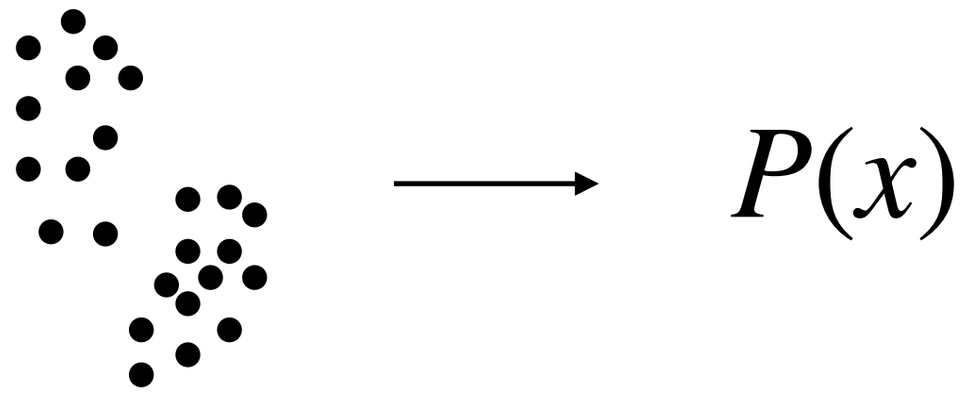


Generative Models

Generative Models

- A very strong form of unsupervised learning are generative models.

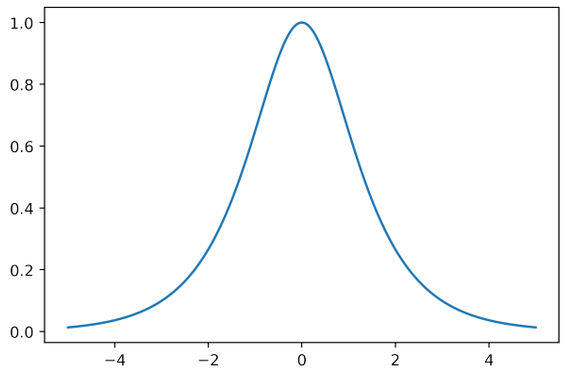
- In a generative model the goal is to model the underlying data distribution.

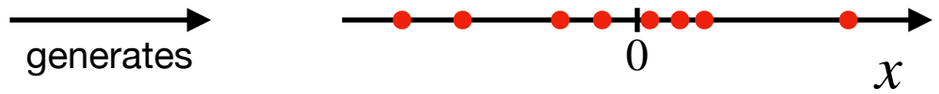


Example - 1D

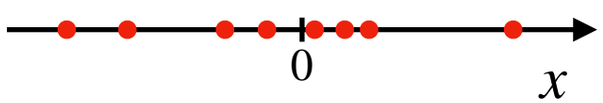
- What does it mean to model the underlying data distribution?

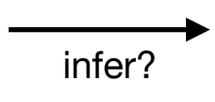
- Observing data we usually assume that a probability distribution generated the data, i.e. data was randomly drawn from this distribution.

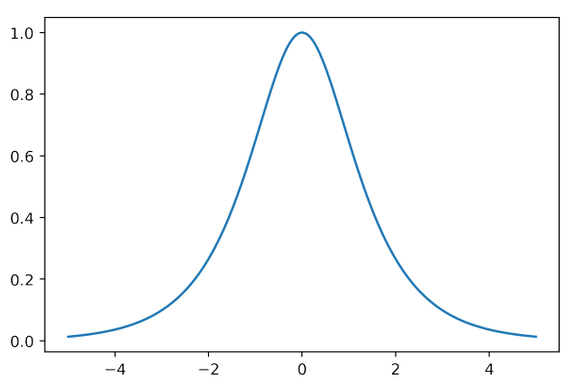




- The question is how could we infer this probability distribution, if we are only given the data?

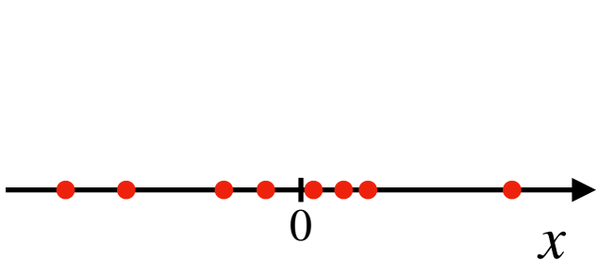




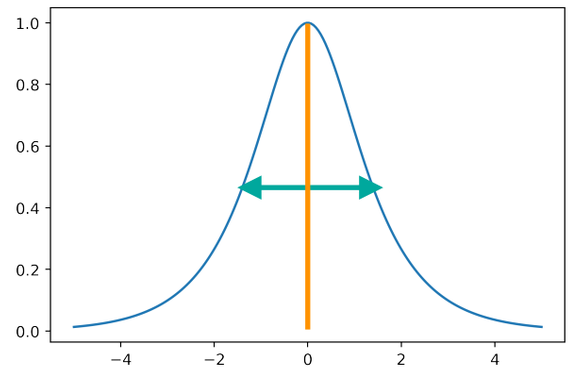


Explicit Generative Model

- Explicitly calculating the probability distribution is hard and always requires to make some prior assumption about the distribution (e.g. gaussian).



assume



What are the values of mean μ and standard deviation σ ?

- If an explicit generative model is inferred you can calculate the probability $P(x)$ for every single data point.

Implicit Generative Model

- The alternative is an implicit generative model.
- The goal is to create a model, which can “simulate” the underlying data distribution.
- The model will be able to generate random data points, as if these were drawn randomly from the underlying distribution.
- Such a model would need implicit access to the underlying data distribution, thus the name.

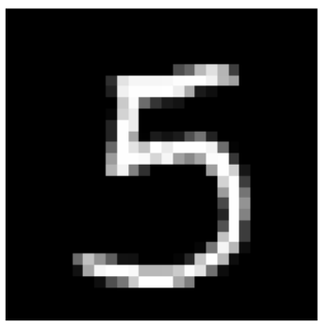
Generative Models for Images

Data Distribution for Images

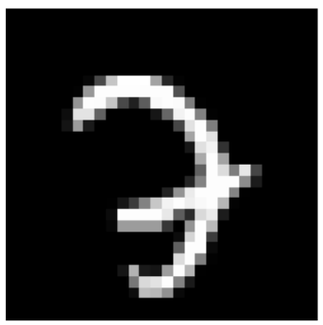
- Images are also data points, which are generated by some underlying data distribution.

- Dealing with e.g. MNIST dataset our datapoints would be $\overrightarrow{x} \in [0,254]^{28\times28}$.

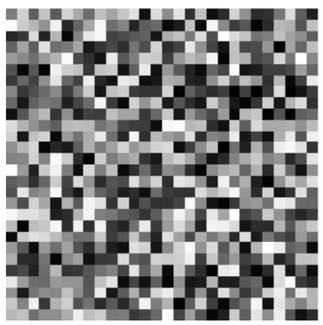
- Given an explicit generative model we would be searching for a probability distribution assigning probabilities as:



relatively high



not so much



zero

Implicit Generative Models for Images

- For low dimensional data determining the explicit probability distribution is very hard.
- For images (other complex data) it is near to impossible.
- So we will try to understand how we can build an implicit generative model for images.
- **Note: Images are only a nice to visualize example; all upcoming ideas are applicable to the data types as well.**

The Manifold Hypothesis

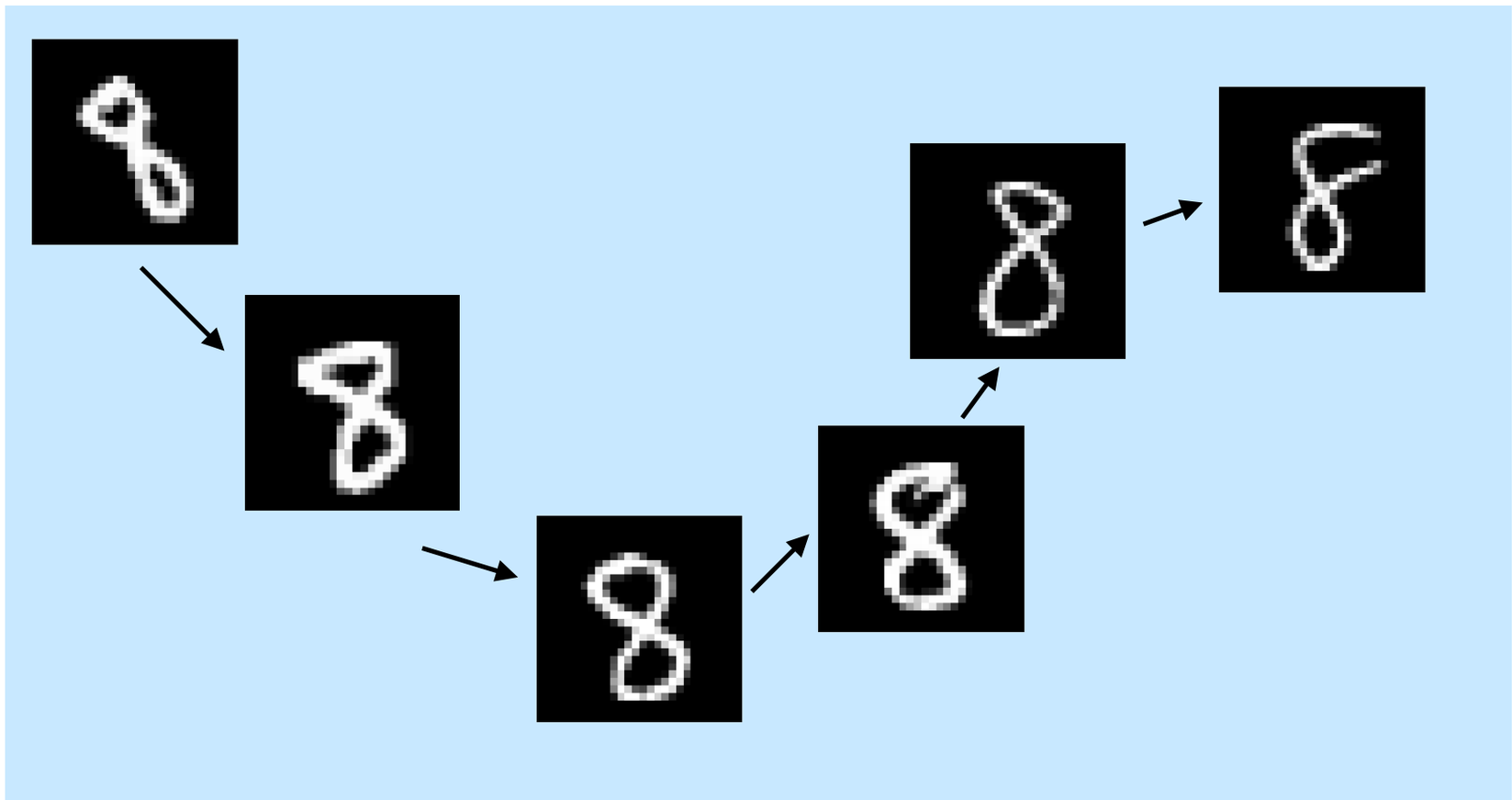
Pixel Space

- To understand implicit generative models for images we have to understand the space they are located in and how they are located in it.
- In the MNIST example $[0,254]^{28\times28}$ is the high-dimensional pixel space.

Simplified 2D Visualization of Pixel Space

Images in Pixel Space

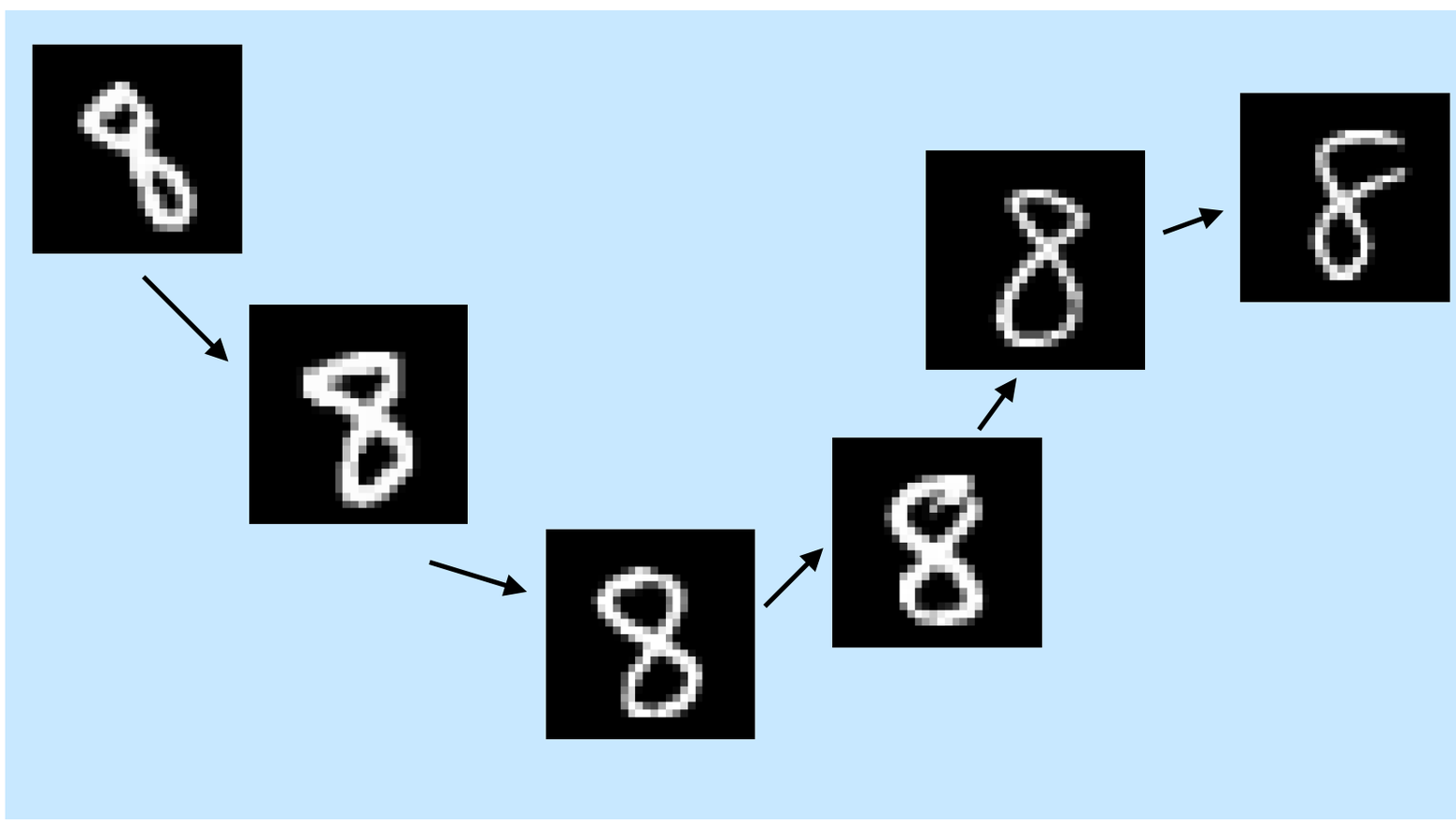
- A sensible assumption is that all images for one class are “connected” to each other.
- I can get from every image to every other image by small changes to the pixels (and all images in between are the same class).



Simplified 2D Visualization of Pixel Space

Manifold Hypothesis

- This assumption is called the manifold hypothesis.
- All images of one class exist on a low-dimensional manifold in the high-dimensional pixel space.

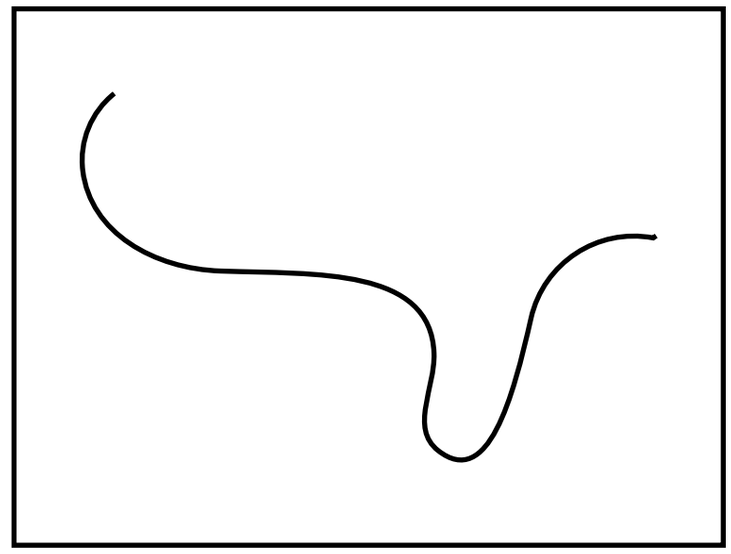


Simplified 2D Visualization of Pixel Space

Simple Manifold Example

- A manifold is simply seen as a structure, which has less dimensions than the space it is embedded in.

- Simple 1-D manifold in 2-D space:



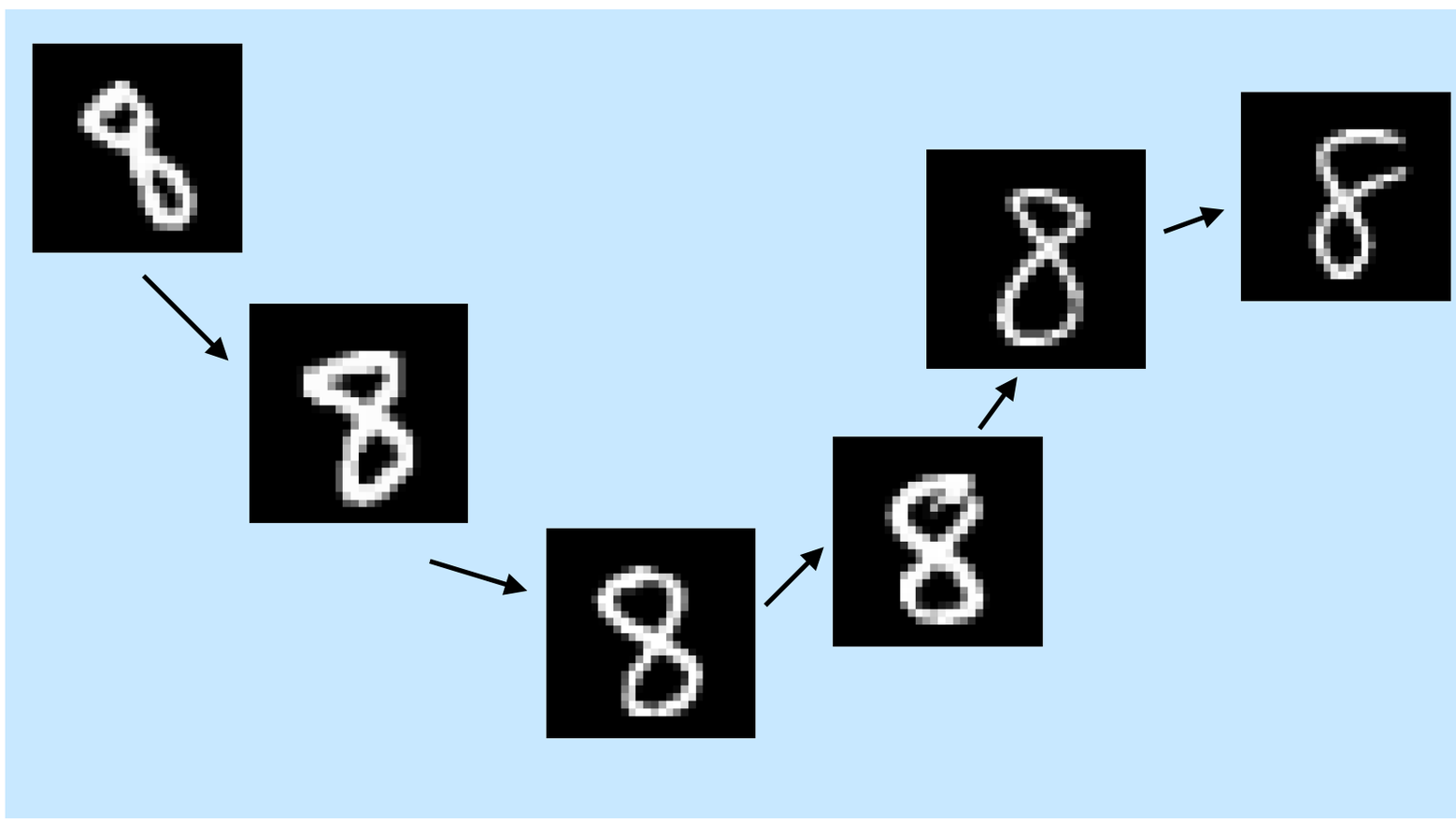
- Pseudo formalization:

$$\varphi : \mathbb{R} \rightarrow \mathbb{R}^2$$

- We map the lower dimensional space $\mathbb{R}$ into the $\mathbb{R}^2$, while keeping the essential structure of $\mathbb{R}$

Manifold Hypothesis

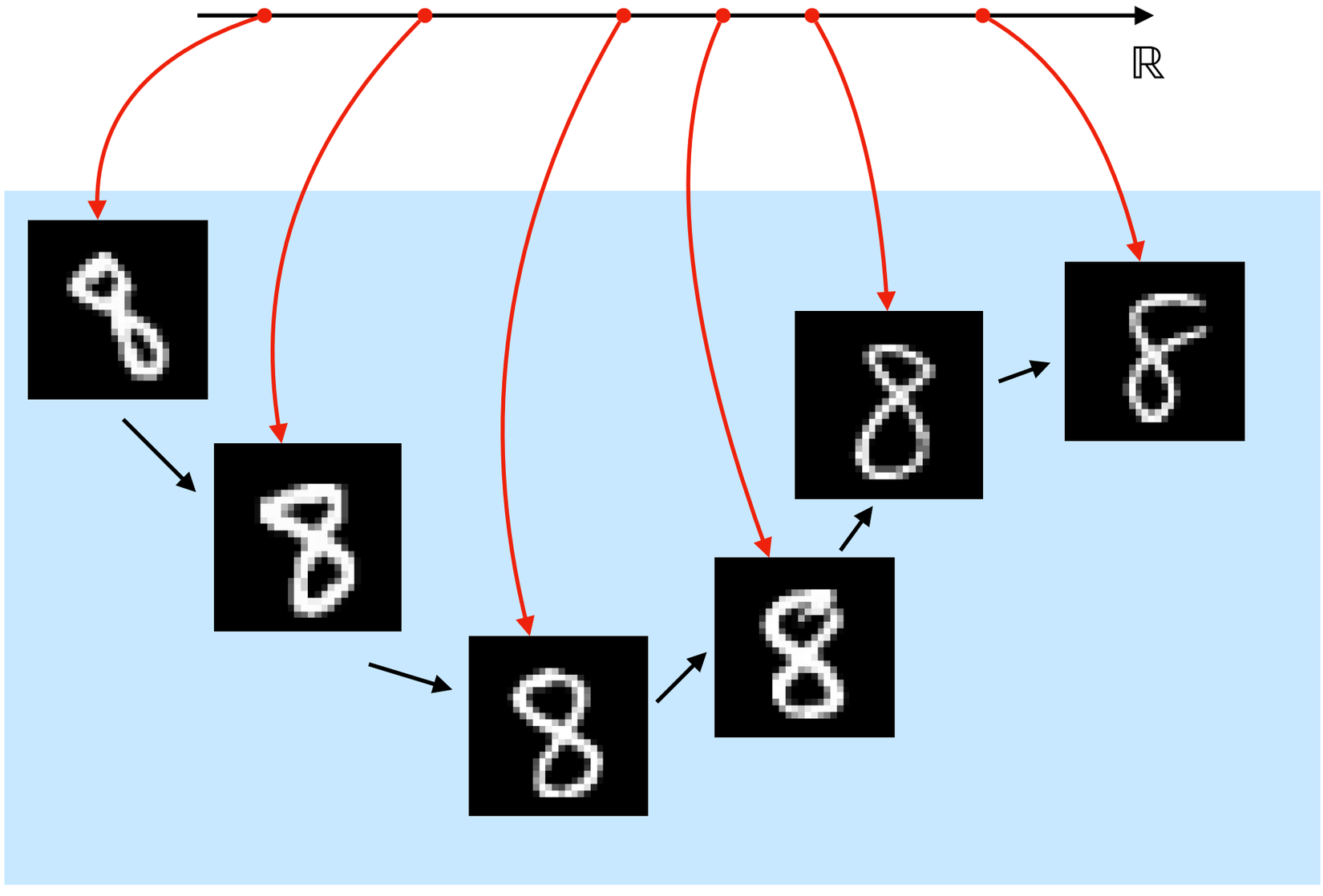
- Of course this manifold is not one dimensional.
- The actual dimensionality can’t be calculated, but is called the intrinsic dimensionality, which means by how many parameters a sample can be characterized.



Simplified 2D Visualization of Pixel Space

Finding the Mapping

The goal thus is to find the mapping from a low-dimensional space into the actual data space (here: pixel space), where each point from the low-dimensional space is mapped onto a valid point in the data space.



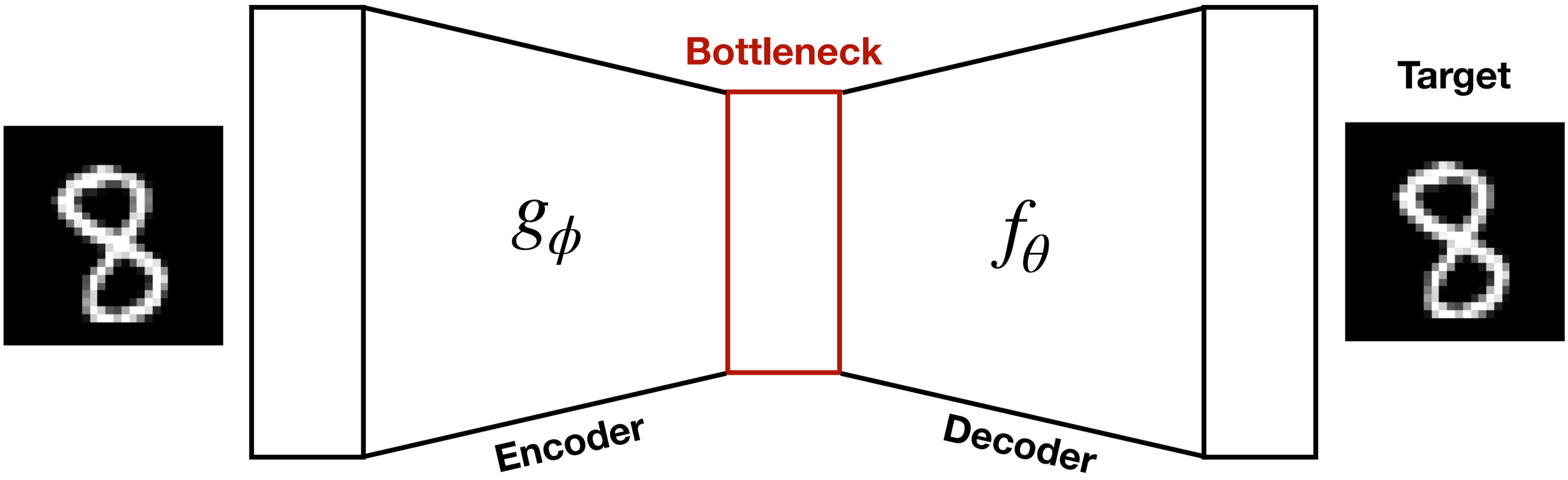
Simplified 2D Visualization of Pixel Space

Autoencoder

Nice blog post!

Autoencoder

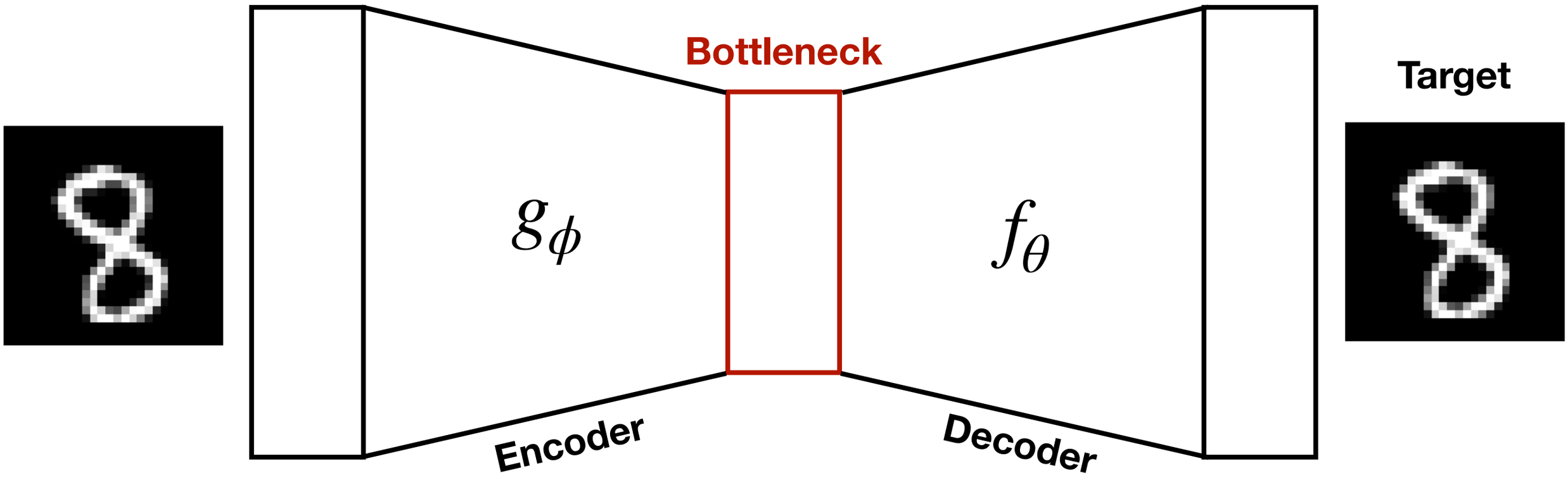
- The autoencoder consists out of an encoder and a decoder (both are multiple neural network layers).
- The encoder has to encode the input into a low-dimensional representation (bottleneck layer), while the decoder tries to reconstruct the original input.



Autoencoder

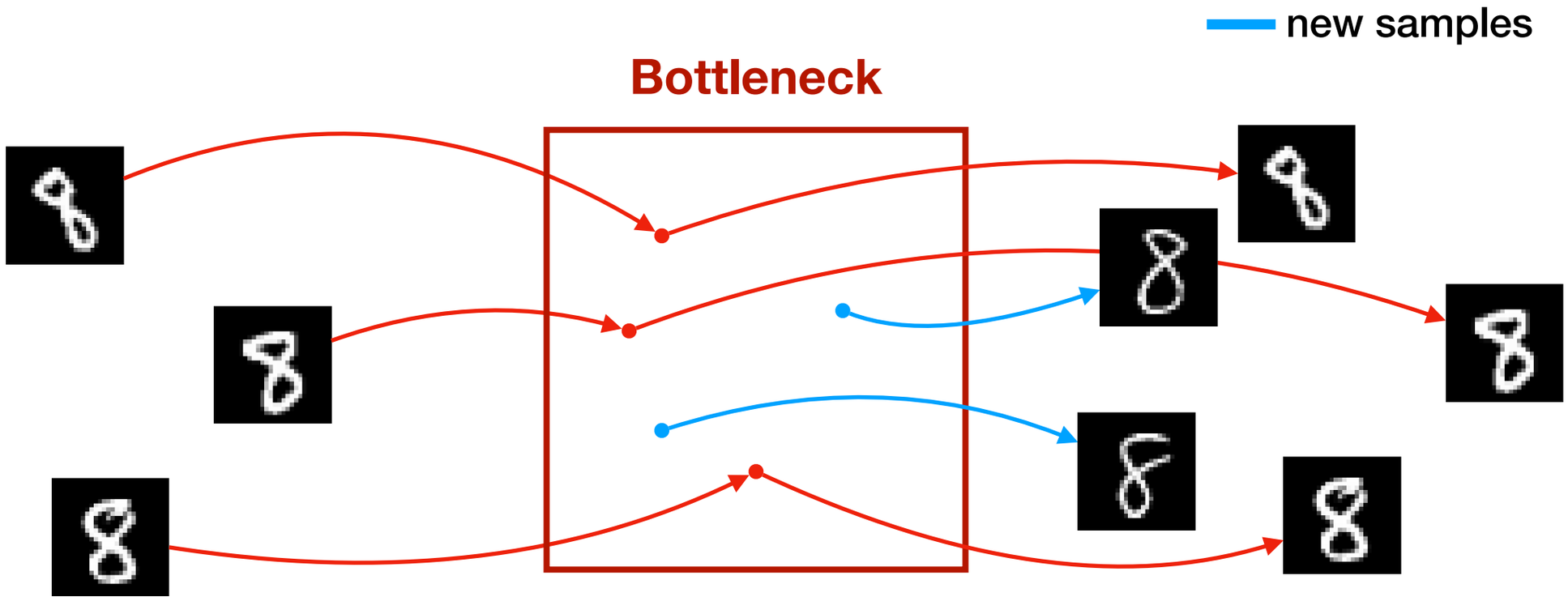
- Training an autoencoder thus is done with training pairs of $(\overrightarrow{x}, \overrightarrow{x})$ and the goal is: $f_\theta(g_\phi(\overrightarrow{x})) = \overrightarrow{x}$.

- As a loss function we can use MSE or cross-entropy (in case of sigmoidal output).



Overfitting in Autoencoder

- To solve the posed task the model needs to learn to encode an image into a low-dimensional representation and to decode the image again.
- It should learn a general representation to encode the whole data distribution (incl. samples not seen before).
- Overfitting would mean that it only learns the encoding for the training dataset.



Improving Robustness

There are several techniques which help auto encoders to be robust against overfitting:

- Denoising Autoencoder
- Sparse Autoencoder
- Contractive Autoencoder
- Variational Autoencoder (most popular)

See this blog post for an overview.

Variational Auto Encoder

- The variational autoencoder (VAE) is a rather technical machine learning principle, but can be easily incorporated into an actual autoencoder.

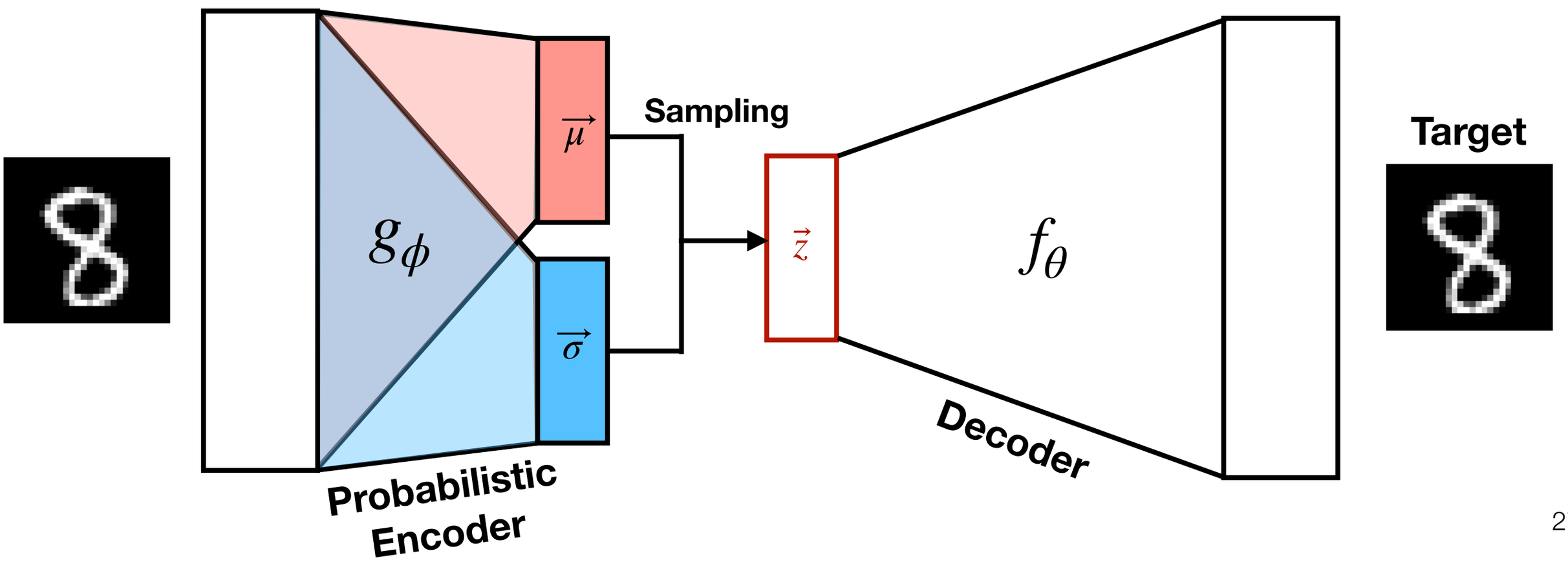
- If you want to understand the real math behind it:

    - original paper
    - blog post 1, blog post 2

- The intuitive idea is as follows: instead of mapping the images into a fixed representation map images into a distribution; because of this “smooth randomness” we can enforce a “smooth behavior” of the encoding.

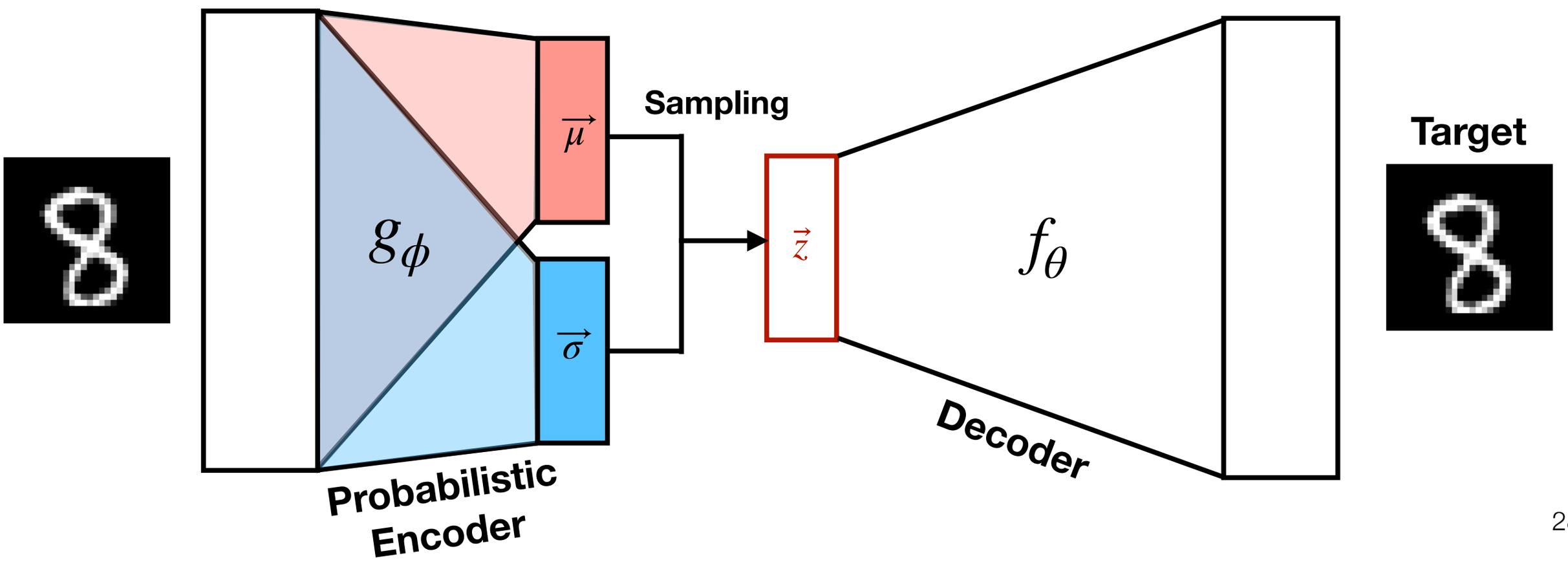
VAE

- The following is the most used form of VAE (with multivariate gaussian assumption):
- The input is mapped to a vector of means $\vec{\mu}$ and a vector of standard deviations $\vec{\sigma}$.
- The vector $\vec{z}$ (also called **latent variable**) is sampled from normal distributions following these two vectors: $z_i \sim \mathcal{N}(\mu_i, \sigma_i)$



VAE - Reparameterization

- The problem is that we can't backprop through the sampling step $z_i \sim \mathcal{N}(\mu_i, \sigma_i)$.
- But we can solve that problem through a simple reparameterization: $z_i = \mu_i + \sigma_i \cdot \epsilon_i$ where the randomness is now in $\epsilon_i \sim \mathcal{N}(0,1)$.



VAE - Loss

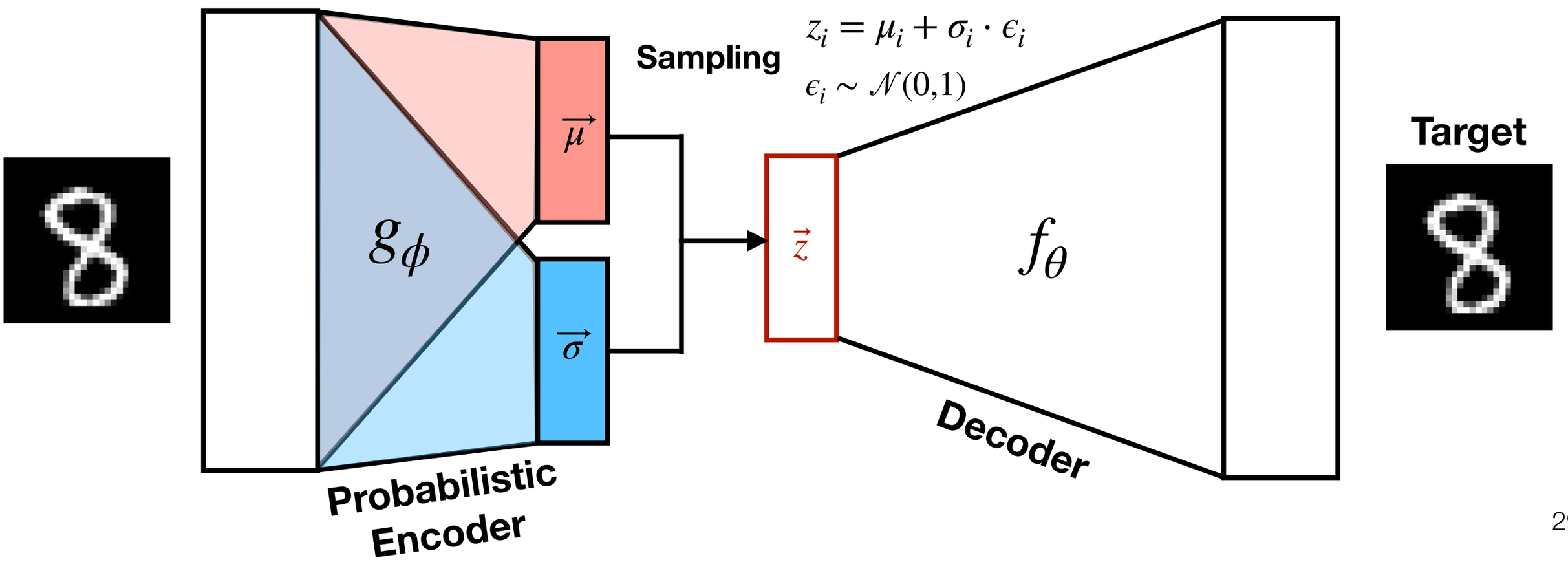
- The missing piece is the loss, which after a derivation results in:

$$Loss = -0.5 \sum_i log(\sigma_i^2) - \mu_i^2 - \sigma_i^2 \quad + \quad cross\_entropy(\vec{x}, \vec{y})$$

reconstructed sample

KL-Loss

Reconstruction error

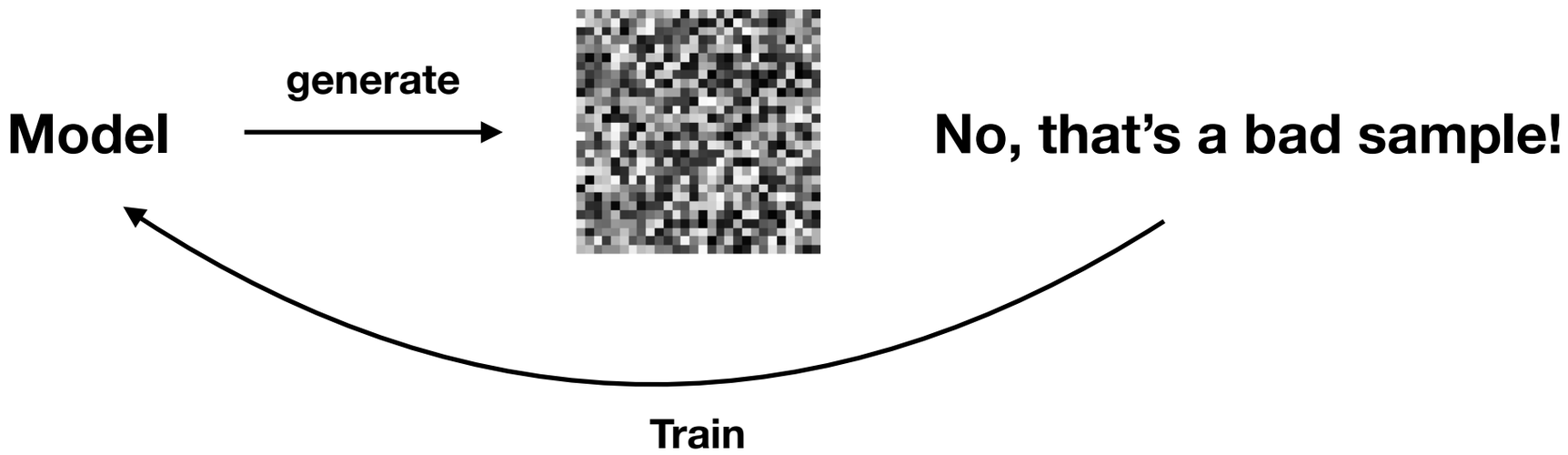


Generative Adversarial Networks

Generative Adversarial Networks

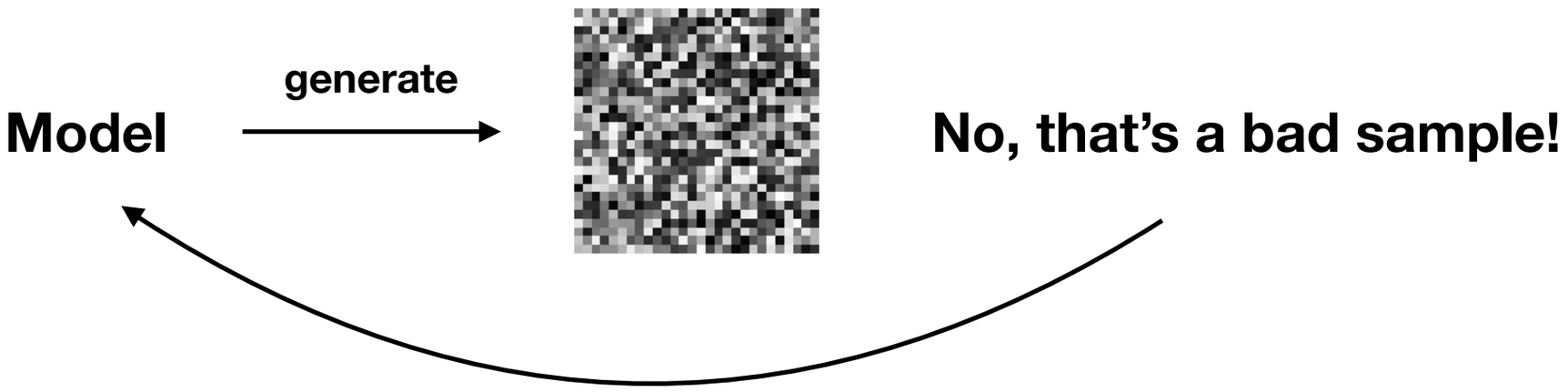
- Generative adversarial networks (GANs) are the other big type of generative models in the neural network landscape.

- The intuition is the following: let's train a model to generate realistic samples by telling it how to improve upon the so far generated samples.



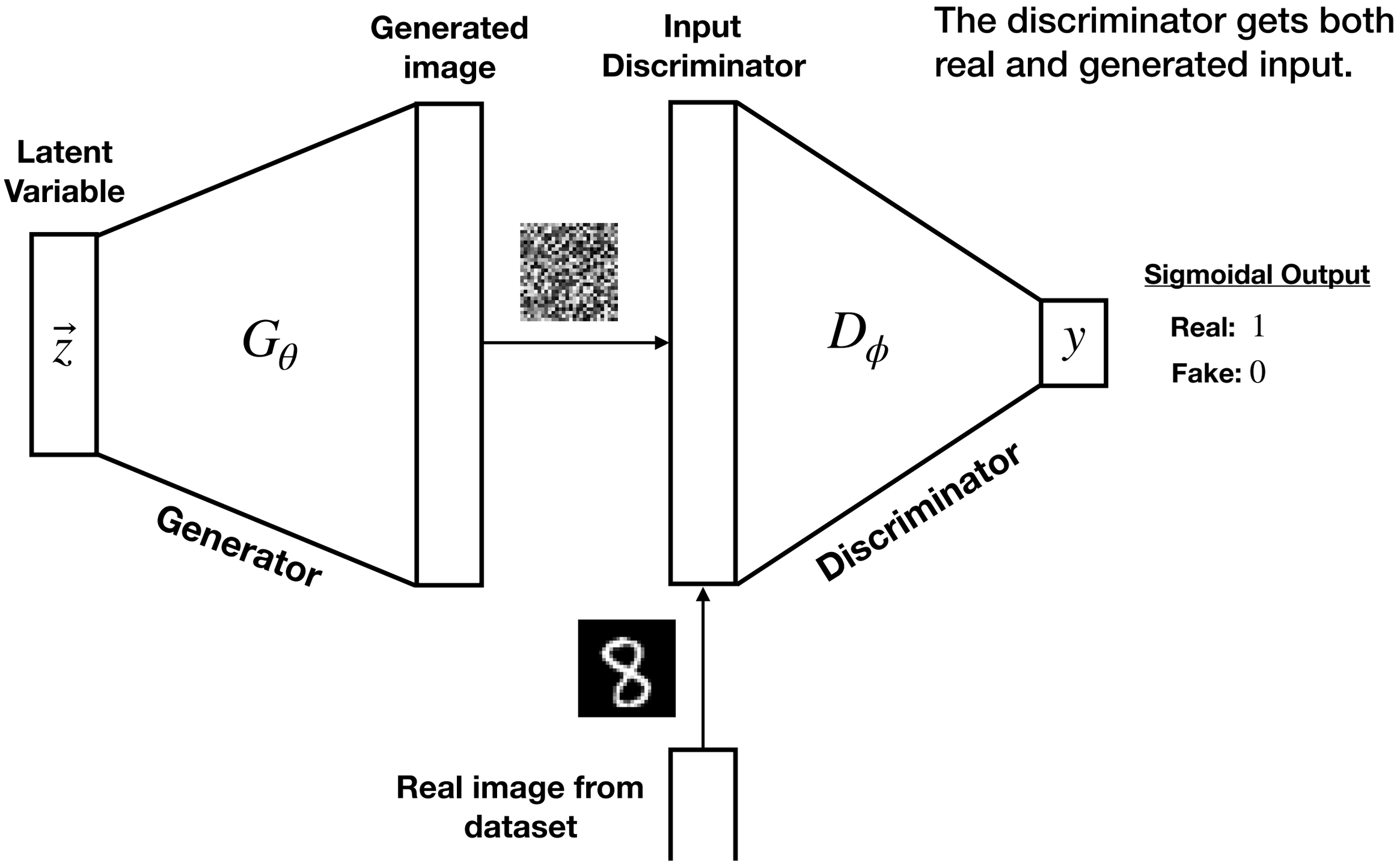
Generative Adversarial Network

- But how can we measure whether the sample is a good sample?



- Idea: Let's use a second neural network which tries to detect whether the sample was generated or whether it is an original sample - when the network is fooled the sample is good.

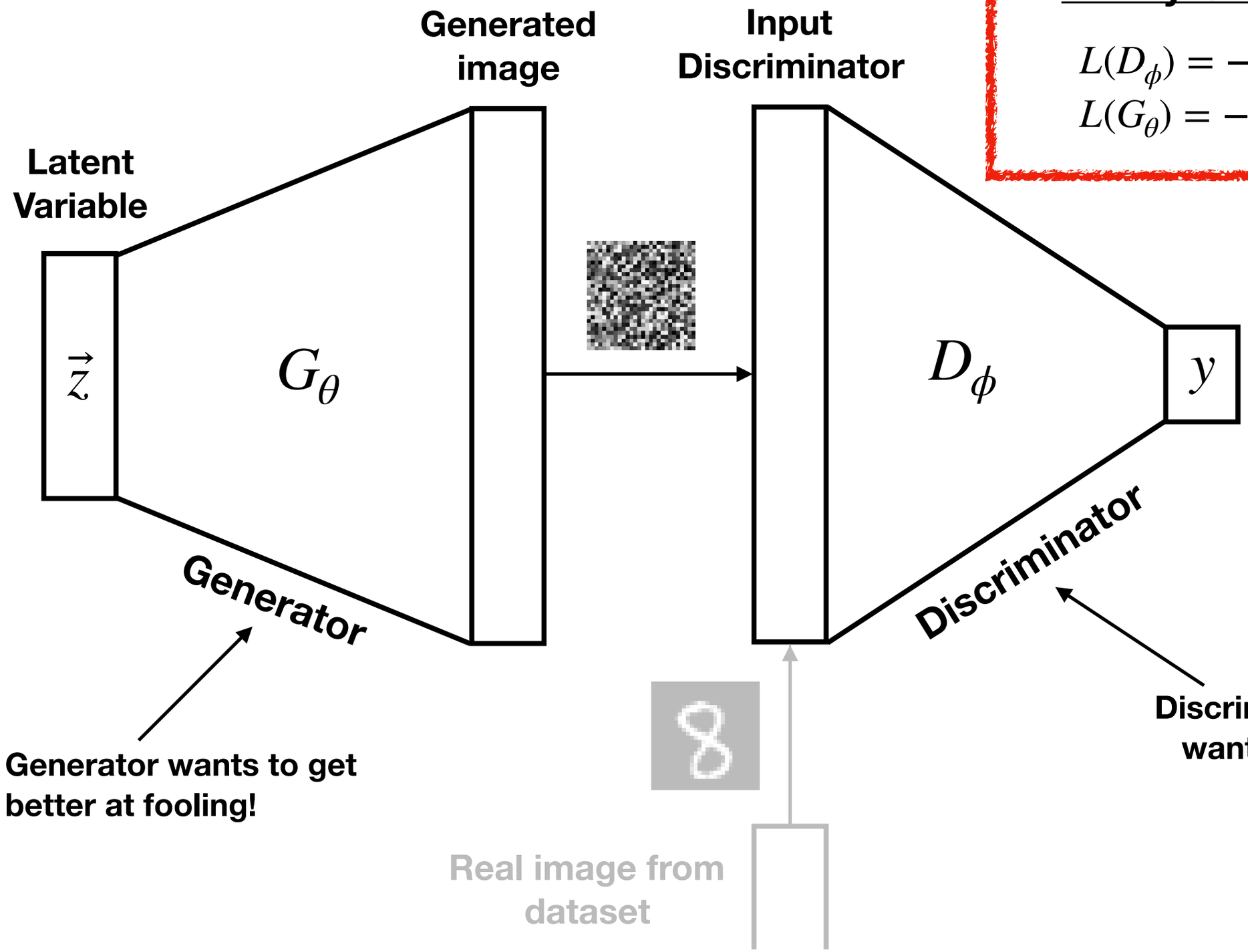
GANs


Generated image
Input Discriminator
The discriminator gets both real and generated input.
Latent Variable
$\vec{z}$
G_θ
D_ϕ
y
Sigmoidal Output
Real: 1
Fake: 0
Generator
Discriminator
Real image from dataset

GANs - Loss for Generated Input

Binary Cross Entropy

$$L(D_\phi) = -(1 - log(y))$$
$$L(G_\theta) = -log(y)$$



Sigmoidal Output

Real: 1

Fake: 0

GANs - Loss for Real Input

Binary Cross Entropy

$$L(D_\phi) = -\log(y)$$

Generated image

Input Discriminator

Latent Variable

$\vec{z}$

G_θ

D_ϕ

y

Sigmoidal Output

Real: 1

Fake: 0

Generator

Discriminator

Generator has no influence on the real images!

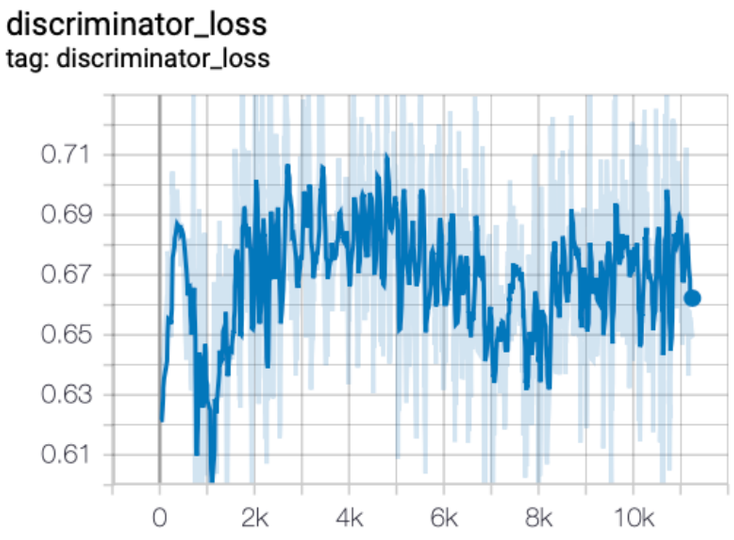
Discriminator tries to detect the real images!

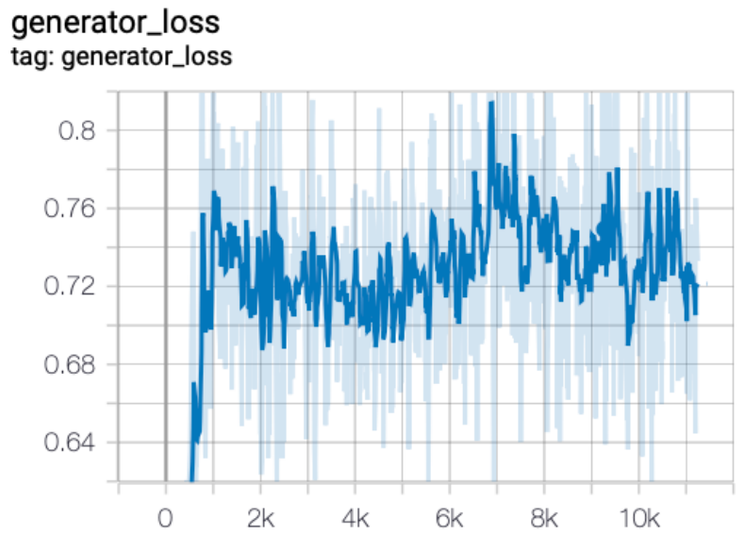
Real image from dataset

Training GANs

- Training a GAN is hard, because you are training the two networks on two different objectives.
- It can easily result in two problems:
    - too good discriminator → the generator doesn't know how to improve (vanishing gradients)
    - too bad discriminator → the generator doesn't have to improve (can lead to "mode collapse"= generator only produces a few samples)

- You won't see a loss minimization!





Advances of GANs

- Class conditioned GANs (cGANs): Allows you to include labels into the generative process. (blog post)

- Wasserstein GAN (WGAN): minimizing the Wasserstein-1 loss can resolve both problems of training GANs (blog post 1, blog post 2)

- Progressively Growing GANs: Start by generating small (8x8) images, slowly increase the size of the images; this allows the generation of very high quality images. (paper)

- StyleGAN: Gives you more control over certain aspects of the generated images. (paper)

- BigGAN: High resolution natural images. (paper)

GANs Applications

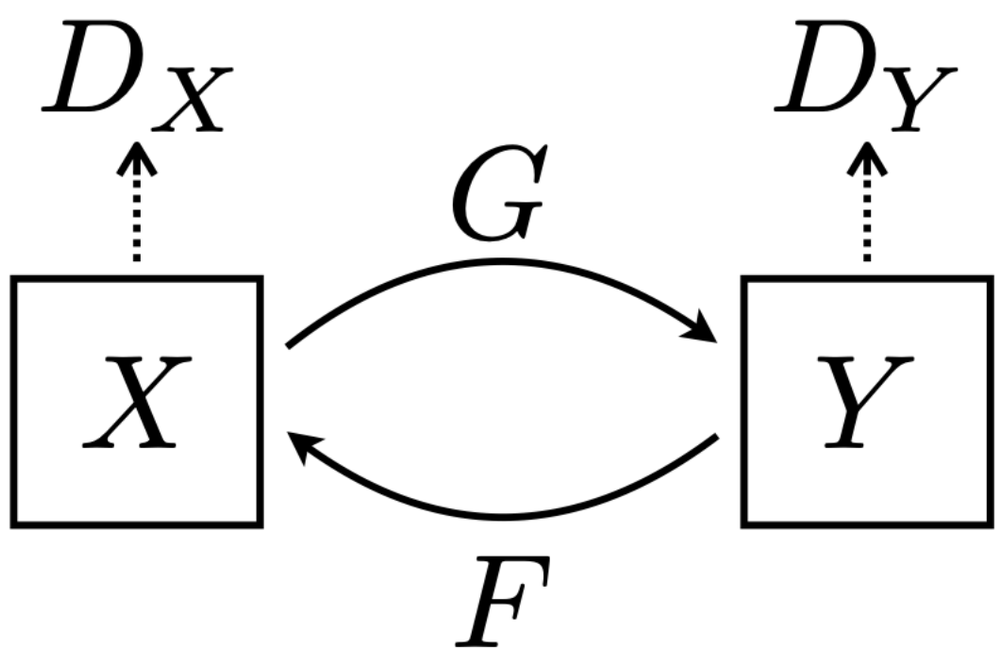
StyleGAN - Faces

Paper

Interesting game

Cycle GAN

- CycleGAN are two generative adversarial networks, which are trained to translate from one data distribution to the other.



CycleGAN - Example

Not available due to copyright issues.

Project website

Others Applications

- NVIDIA playground (incl. GauGAN)
- Deepfakes (youtube)
- Up to date mostly restricted to visual data.

Disentangled Representations

Unsupervised Learning

- Unsupervised learning is a very important field in machine learning.

- Machine learning relies on data. If the data doesn’t have to be labeled you can leverage much larger amounts of data.

- The big issue is how useful is the learned representation? Can humans relate in anyway?

Disentangled Representation

- The hypothetical solution are so called disentangled representations.
- The intuitions is that the dimensions of a representation are representing different meaningful aspects of the data distribution.
- First of all this meaningfulness is quite undefined, see this paper for discussion of a possible definition.
- But still disentangled representations can be said to be the holy grail of machine learning, because it would give machines the ability to learn symbolic meaningful aspects about the world from raw data.
- See this paper for a large-scale study of different approaches.

Example - Disentangled Representation

β-VAE

Not available due to copyright issues.

Not available due to copyright issues.

Not available due to copyright issues.

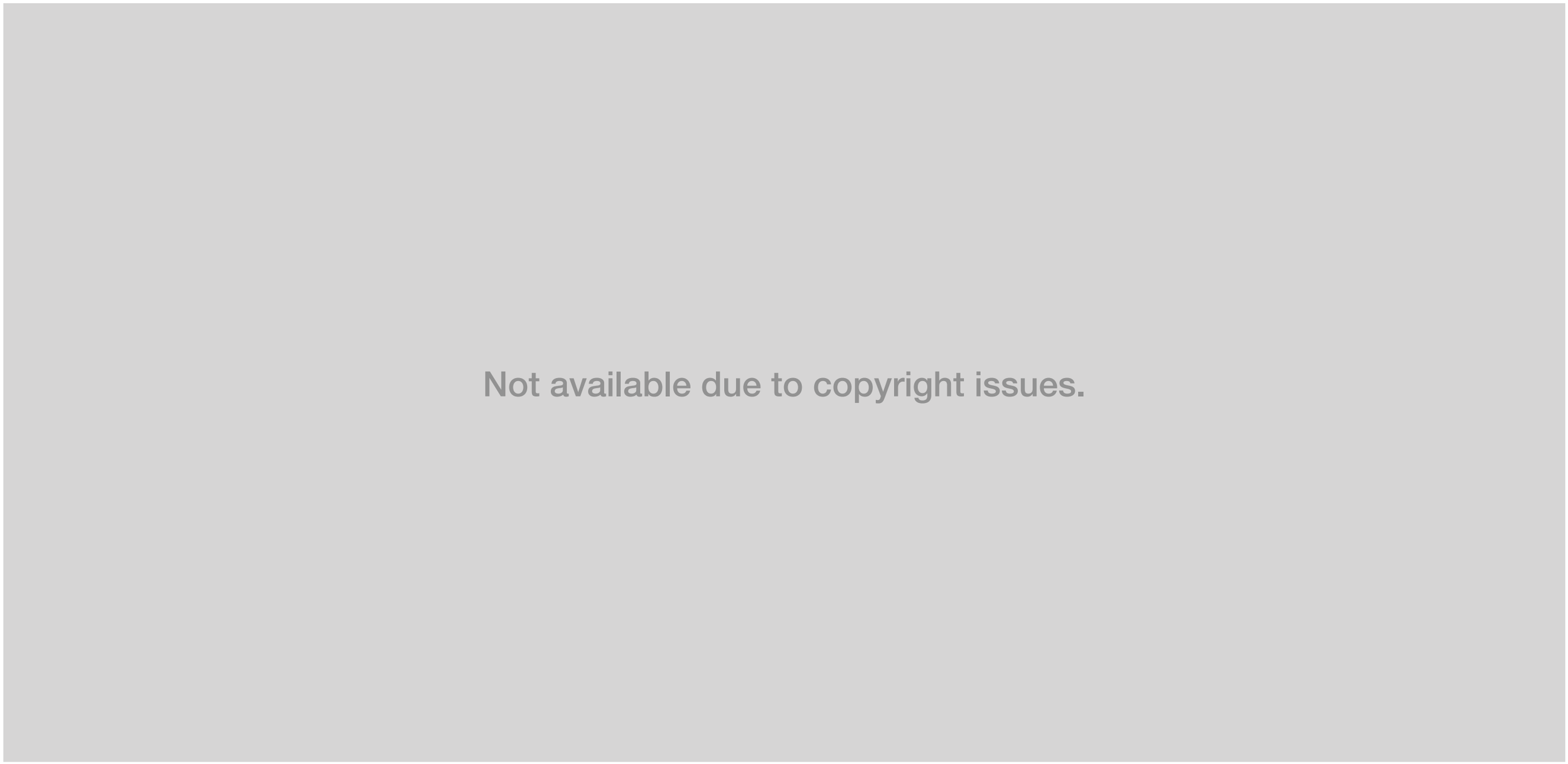
Paper

InfoGAN

Not available due to copyright issues.

Paper

GLOW



Blog with web application

Transposed Convolutions

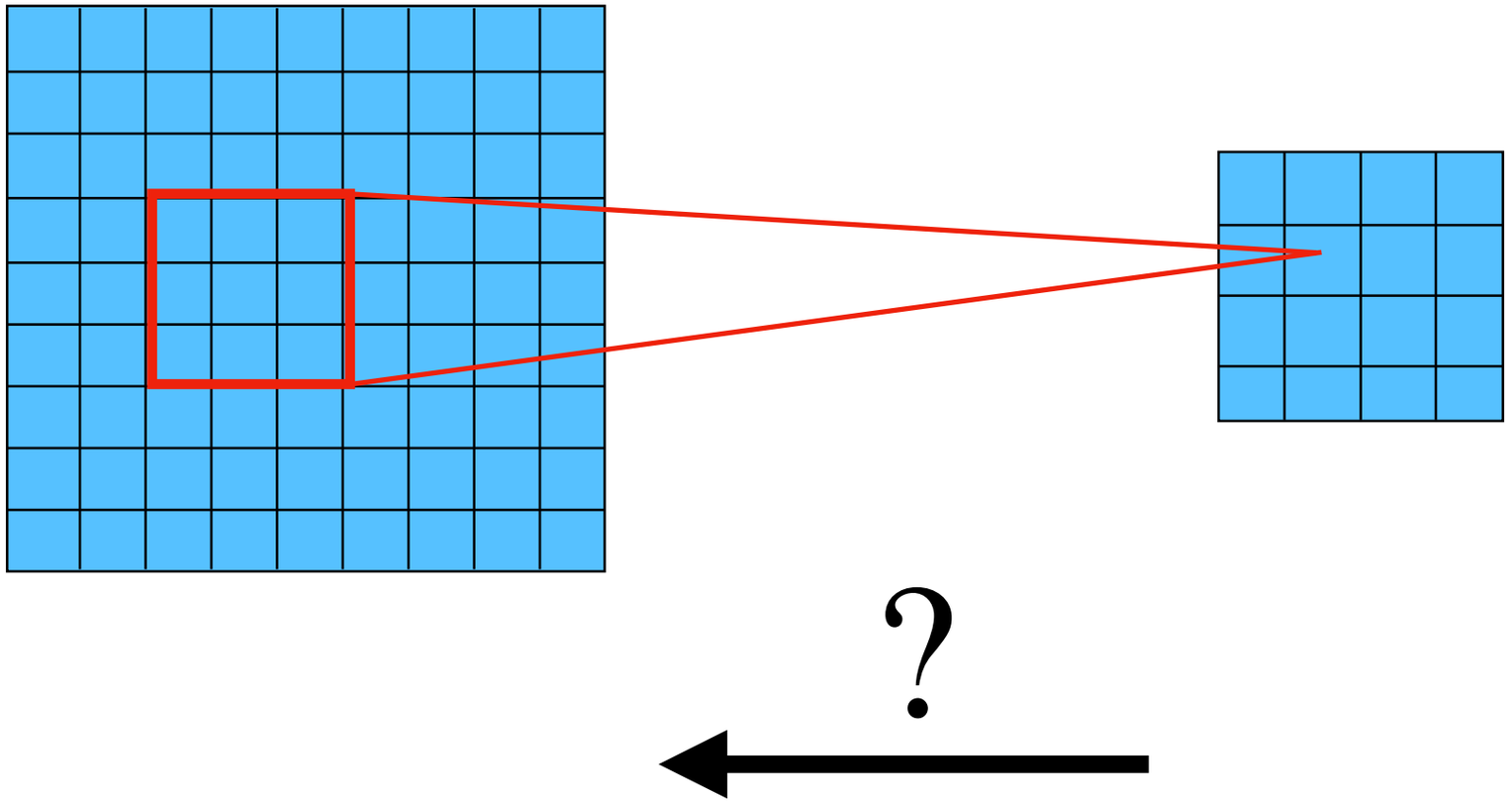
Transposed Convolutions

- Both the autoencoder and the generative adversarial network need a way of creating an image.

- You could either use ever increasing fully-connected layers and reshape the last layer to the desired image size.

- But because this is costly and we know that convolutions are a good mechanism here there is a transposed convolution which instead of decreasing the image size is increasing the images size.

Convolution - Backwards



- Convolution usually reduces the size (or same size) of an image by running a filter over the image.
- Transposed convolution reverses this process

Convolution as Matrix Multiplication

- We can rewrite a the convolution operation as a simple matrix multiplication, by flattening the input matrix and creating a specific matrix.

- By transposing the matrix we can use it to upsample instead of downsample (it is not the inverse though!).

- See this blog for detailed information on the process.

TensorFlow

- The corresponding layer in TensorFlow is very similar to the original convolutional layer.

```
self.convT_1 = tf.keras.layers.Conv2DTranspose(
                        filters=32,
                        kernel_size=5,
                        strides=1,
                        padding='SAME',
                        activation=None,
                        use_bias=False
)
```

TensorBoard

TensorBoard

- To supervise our model we need to have a look at different kind of metrics.

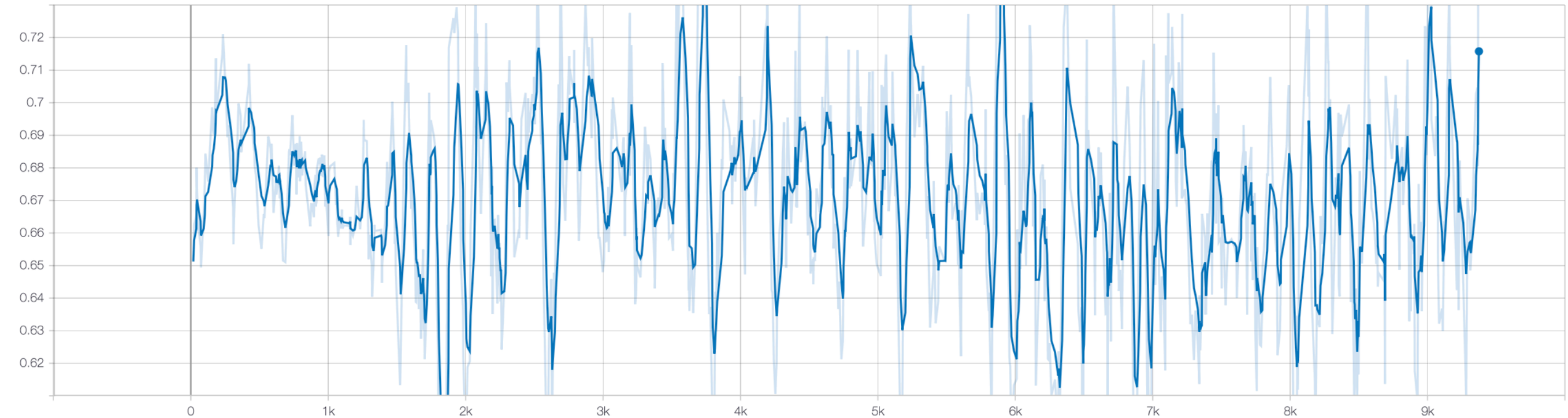
- Until now we just printed them and afterwards we usually plotted them.

- But if you actually don’t know whether a network learns as you would like it learn it would be useful to see the plots develop over time.

- TensorBoard is built-in solution coming with TF.

TensorBoard

discriminator_loss
tag: discriminator_loss



TensorBoard

```
import datetime
!rm -rf ./logs/
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
test_log_dir = 'logs/gradient_tape/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
test_summary_writer = tf.summary.create_file_writer(test_log_dir)
```

- You have to create so called summary writers (one for training and one for test).

- These will create the defined directories and be used in the next step to store your summaries in there.

TensorBoard

```
with train_summary_writer.as_default():
    tf.summary.scalar('generator_loss', gen_loss, step=step)
    tf.summary.scalar('discriminator_loss', dis_loss, step=step)
```

```
with test_summary_writer.as_default():
    tf.summary.image('fake_images', fake, step=step, max_outputs=8)
```

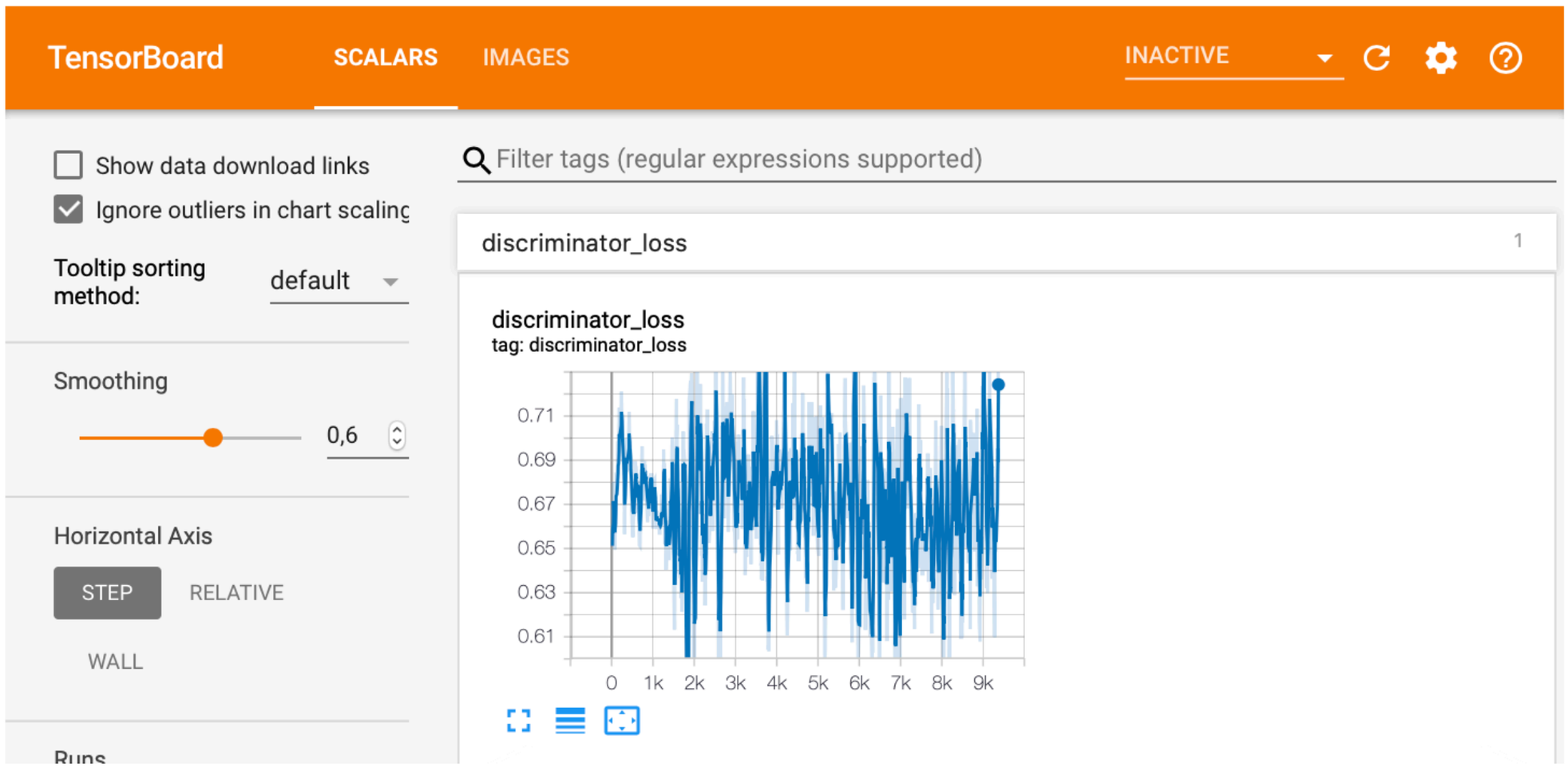
- To store a summary you simply activate the corresponding summary writer after you computed e.g. the loss.

- Then you create a summary, e.g. a *summary.scalar( )* or a *summary.image( )*.

- These summaries are then automatically logged to your directory.

TensorBoard

```
%tensorboard --logdir logs/
```



- In a last step you call the magic function above and TensorBoard is opened inside the notebook.

TensorBoard

- Alternatively you can open it via terminal:

```
$ tensorboard --logdir='./logs'

TensorBoard 2.0.0 at http://localhost:6006/ (Press CTRL+C to
```

- You can then simply open the link in your browser.

Any questions?

See you next week!