Manifolds, Autoencoders, Generative Adversarial Networks, and Diffusion Models

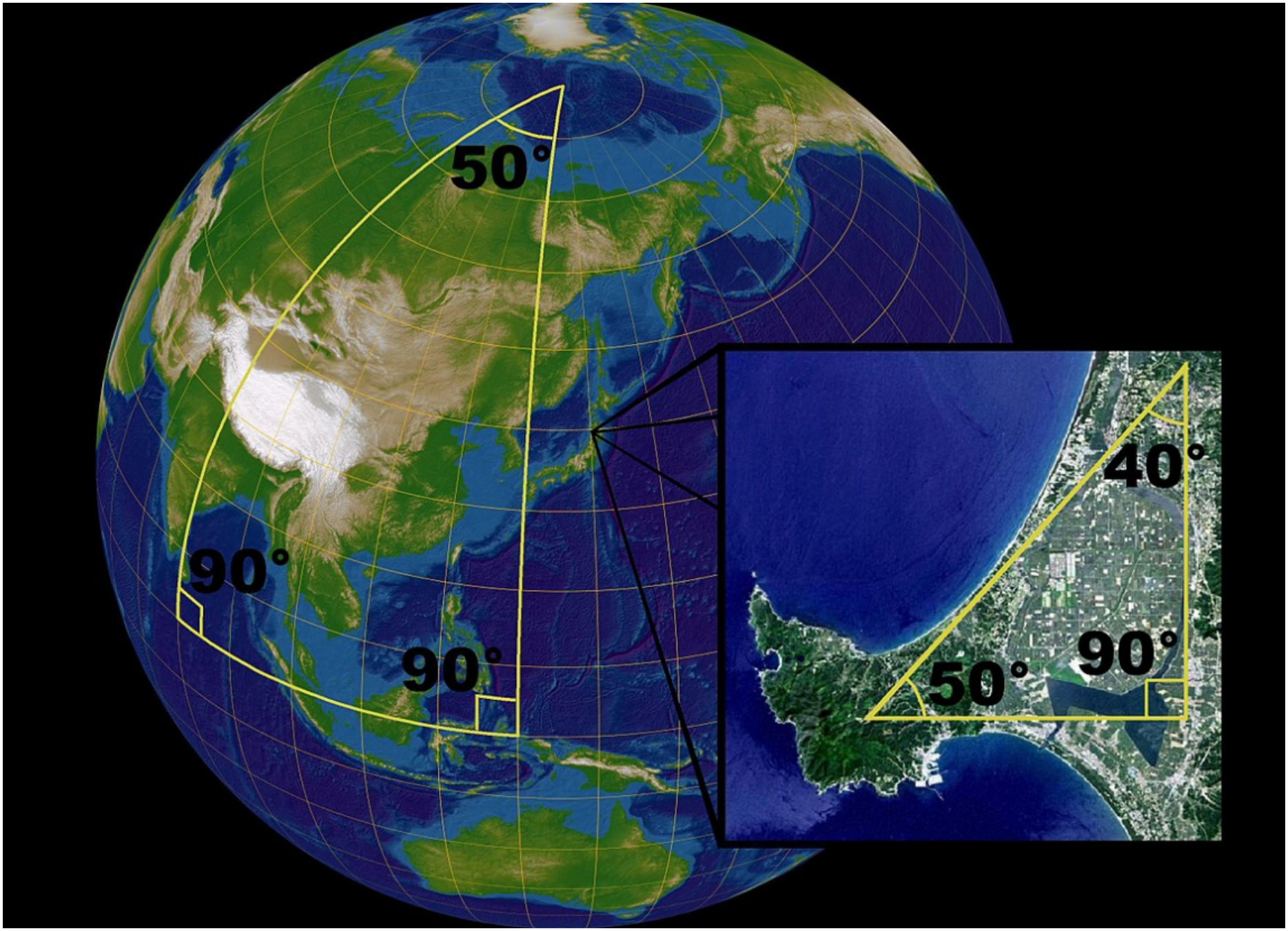
Volker Tresp
Winter 2023-2024

Generative Models

- Generating beautiful and crazy images out of noise
- So far we considered that the input space might be high-dimensional $M >> 1$; all natural inputs (images) form a manifold in the input space
- Autoencoder, GAN: We model that manifold as all points which are outputs of a generator DNN; inputs $\mathbf{h} \in \mathbb{R}^{M_{hidd}}$ to the generator are low-dimensional and random

Manifolds

- In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point
- A topological space may be defined as a set of points, along with a set of neighbourhoods for each point, satisfying a set of axioms relating points and neighbourhoods


50°
90°
90°
40°
50°
90°

Data Represented in Feature Space

- Consider Case Ib (manifold): input data only occupies a manifold
- Example: consider that the data consists of face images; all images that look like faces would be part of a manifold
- What is a good mathematical model? We assume that "nature" produces data in some low-dimensional space $\mathbf{h}^{nat} \in \mathbb{R}^{M_{hidd}}$, but nature only makes data available in some high-dimensional feature space $\mathbf{x} \in \mathbb{R}^{M}$ ($\mathbf{x}$ might describe an image, in which case M might be a million)
- Features map

$$\mathbf{x} = \text{featureMap}(\mathbf{h}^{nat})$$

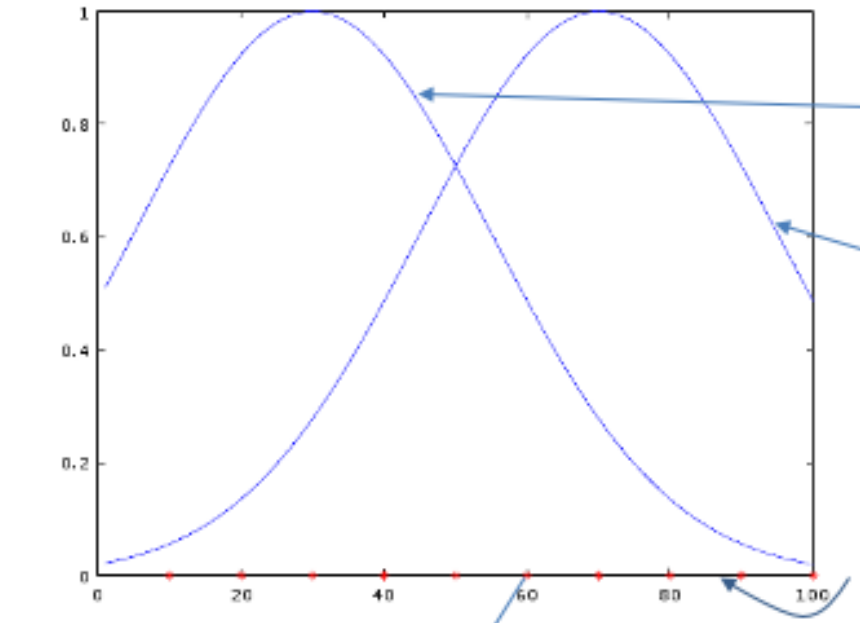
Manifold in Machine Learning

- In Machine Learning: in the observed M-dimensional input space, the data is distributed on an M_{hidd}-dimensional manifold

$$\{\mathbf{x} \in \mathbb{R}^M : \exists \mathbf{h} \in \mathbb{R}^{M_{hidd}} \ \ s.th. \ \ \mathbf{x} = g_d(\mathbf{h})\}$$

  where $g_d(\cdot)$ is smooth

- Note that for a given $\mathbf{x}$, it is not easy to see if it is on the manifold

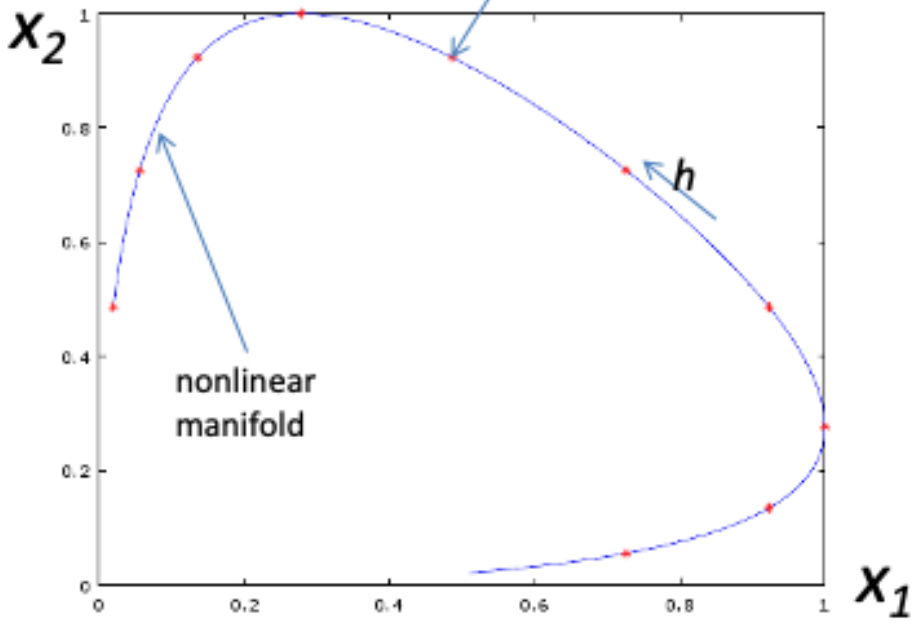


$x_1 = featureMap_1(\mathbf{h}^{nat})$

$x_2 = featureMap_2(\mathbf{h}^{nat})$

$\mathbf{h}^{nat}$

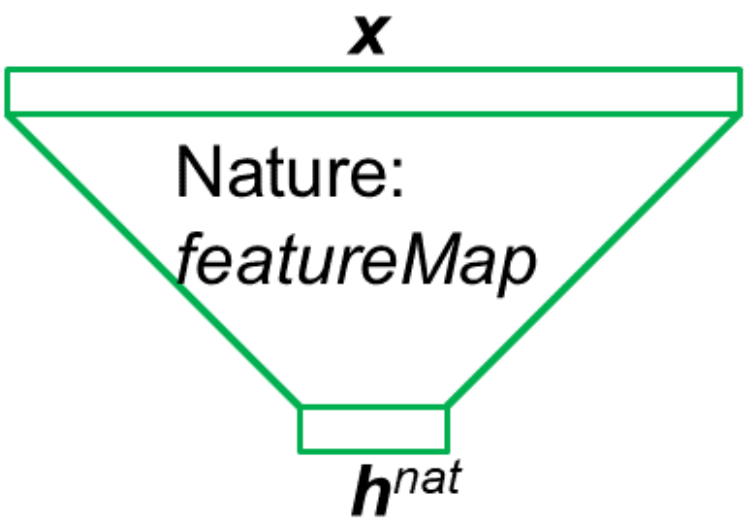
$\mathbf{h}^{nat}$ *is not measured: it is latent*
here: $M_h = 1$



The data is available in feature space $\mathbf{x}$
here: $M = 2$

Feature Engineering

- In a way, features are like basis functions, but supplied by nature or an application expert (feature engineering)
- In the spirit of the discussion in the lecture on the Perceptron: $\mathbf{h}^{nat}$ might be low-dimensional and explainable, but we can only measure $\mathbf{x}$



The feature map produces an M-dimensional observed ***x;*** but the data only occupies an M_h-dimensional manifold

Nature selects an instance of a low dimensional $\boldsymbol{h}^{nat}$ M_h-dimensional

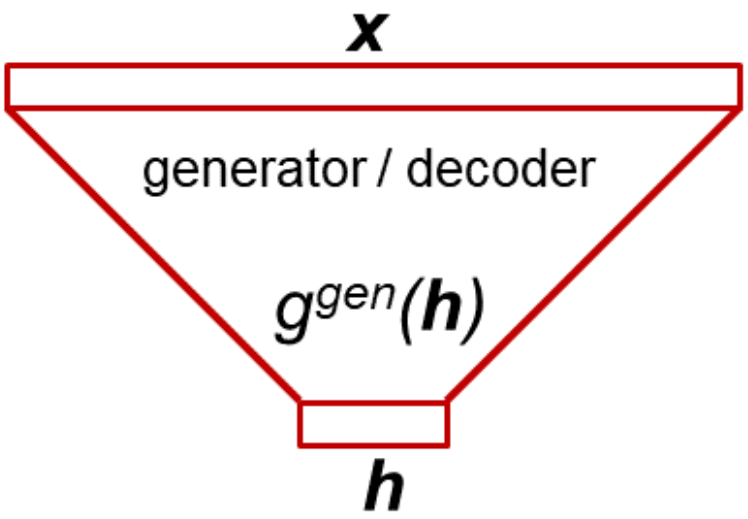
Learning a Generator / Decoder

- Example: $\mathbf{x}$ represents a face; let's assume nature selects $\mathbf{h}^{nat}$ from an M_{hidd}-dimensional Gaussian distribution with unit covariance
- Then, if the feature map is known, we can generate new natural looking faces of people who do not exist: $\mathbf{x} = \mathsf{featureMap}(\mathbf{h}^{nat})$
- We try to emulate this process by a model

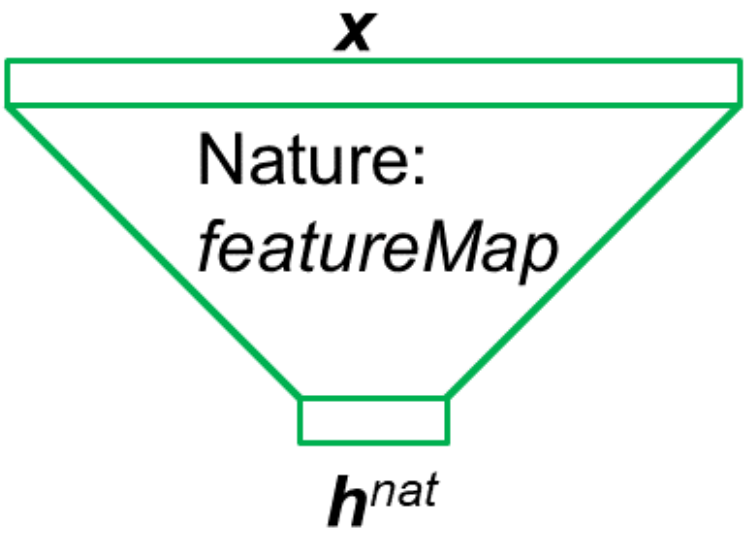
$$\mathbf{x} = g^{gen}(\mathbf{h})$$

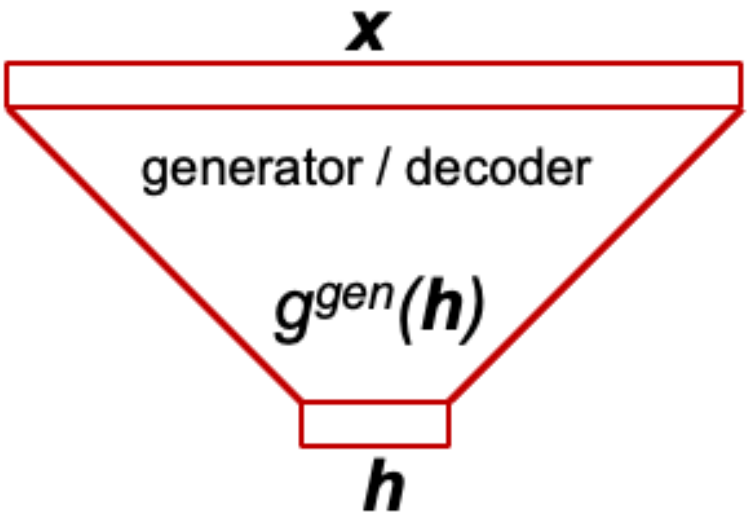
(here we drop the superscript nat, since this $\mathbf{h}$ is part of our model and not the ground truth $\mathbf{h}^{nat}$)

- In an autoencoder, the **generator** is also called a **decoder** $g_d(\cdot)$



- A generator tries to copy this generative process
- If the ***x*** represent images, one can now generate new natural looking images





- A generator maps from M_h-dimensional $\boldsymbol{h}$ to an M-dimensional $\boldsymbol{x}$
- $M_h << M$
- Any $\boldsymbol{h}$ gives a valid image $\boldsymbol{x}$

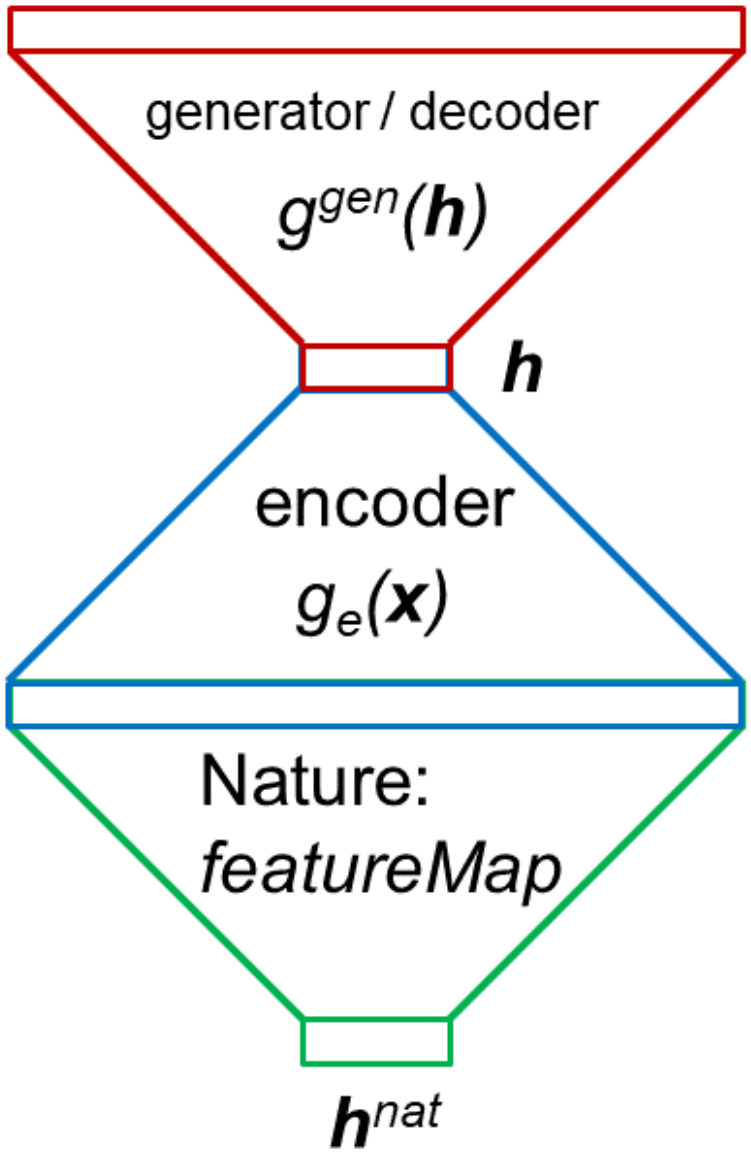
- In the simplest case, this is a map between two layers in a neural network
- In practice, the map is described by a DNN

- If we assume that the $\boldsymbol{h}$ are generated randomly, then the dimensions in $\boldsymbol{x}$ are strongly dependent (highly correlated)

Learning an Encoder

- Of course for data point i we only know $\mathbf{x}_i$ but we do not know $\mathbf{h}_i^{nat}$
- But maybe we can estimate $\mathbf{h}_i \approx \mathbf{h}_i^{nat}$ based on some **encoder** neural network

$$\mathbf{h}_i = g_e(\mathbf{x}_i)$$



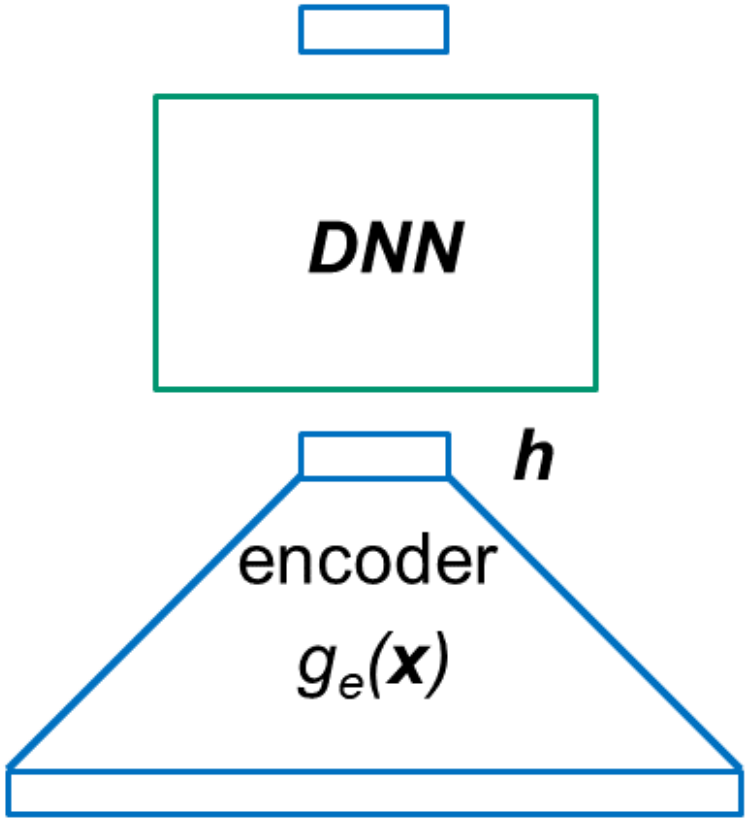
$\boldsymbol{x}$

Some models contain an explicit encoder with maps $\boldsymbol{x}$ to a latent repesentation

$\boldsymbol{x}$ (only $\boldsymbol{x}$ can be measured)

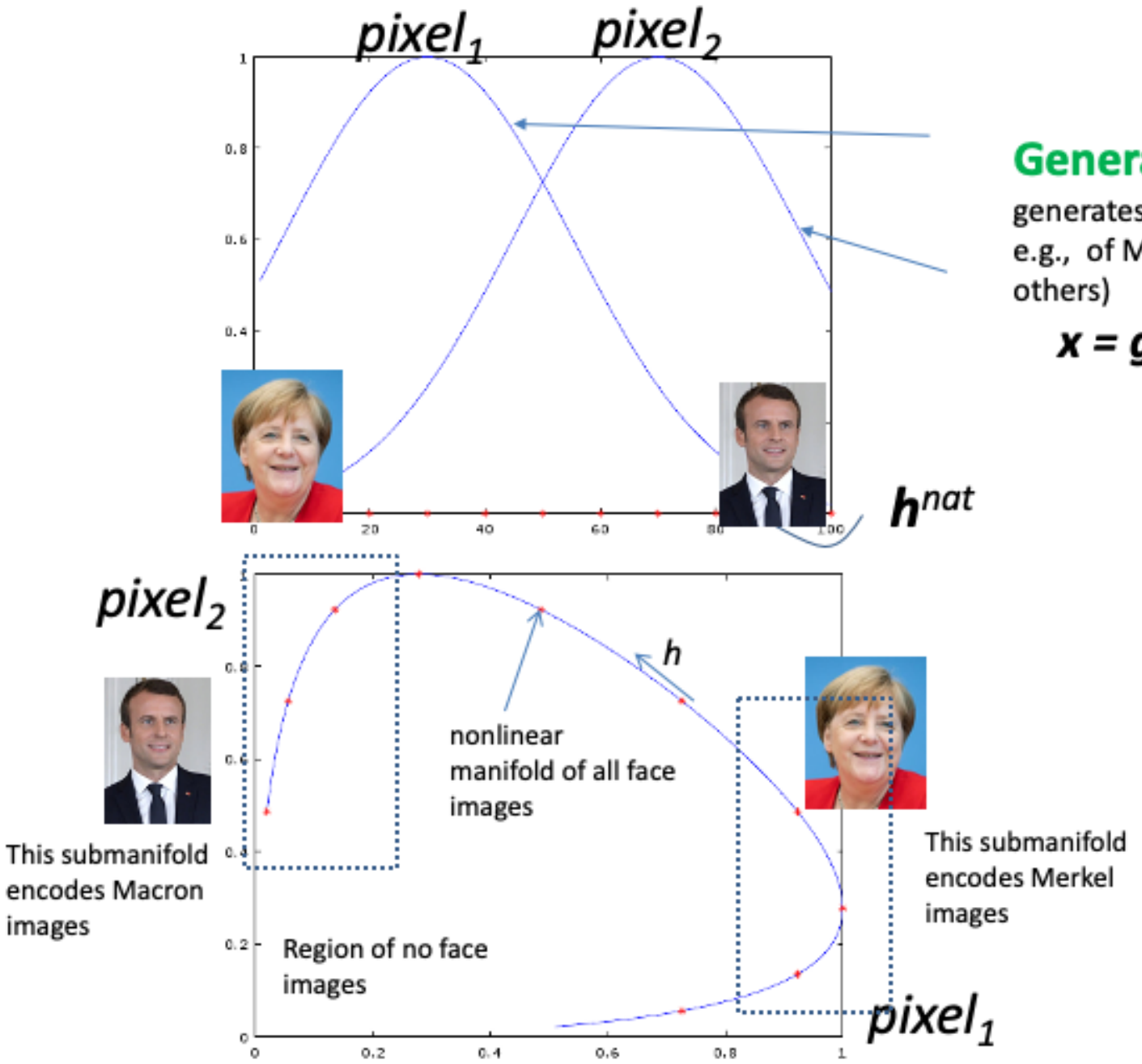
Encoder

- An encoder can be useful by itself: it can serve as a preprocessing step in a classification problem (Case Ib (manifold))



A trained encoder can be used as a preprocessing step in classification/regression

x (only ***x*** can be measured)



Generator/Decoder:

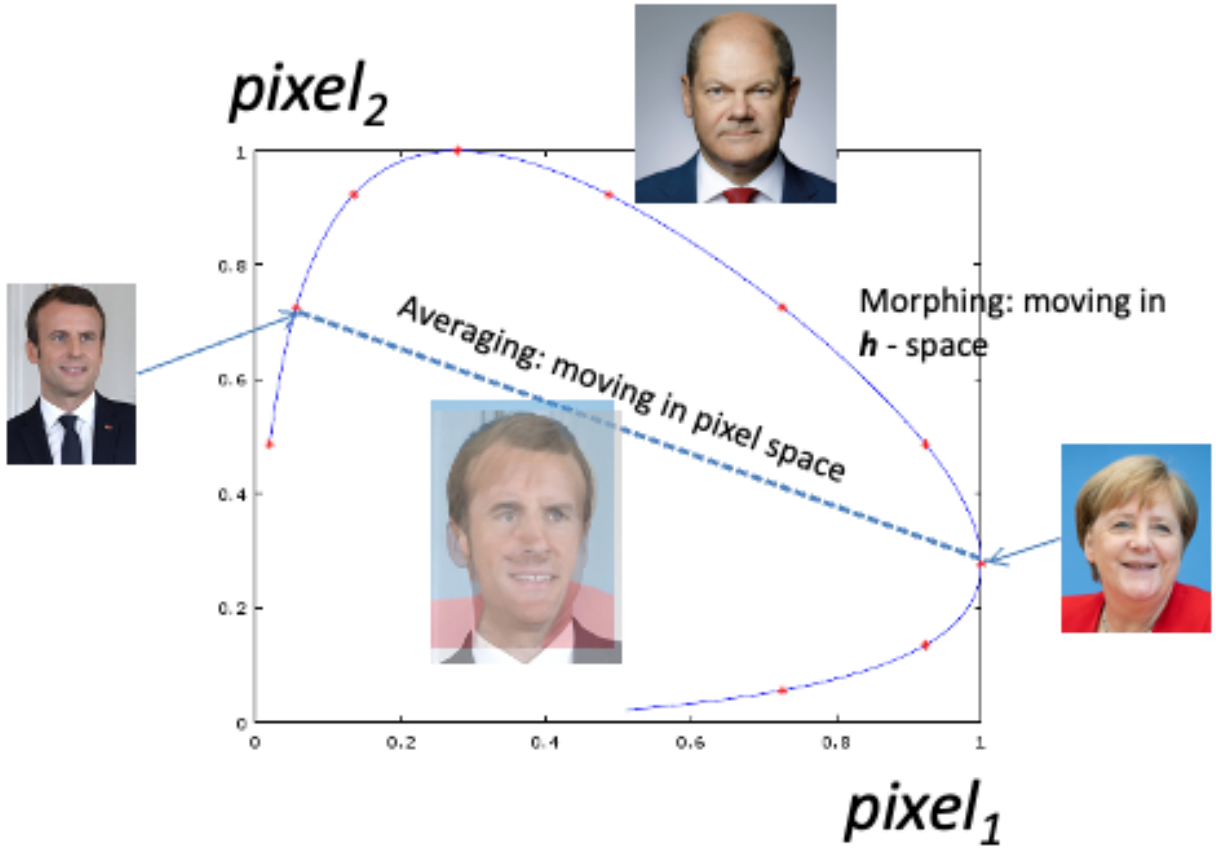
generates face images for any ***h***, e.g., of Merkel and Marcon (and others)

$$x = g_d(h)$$

Encoder:

calculates embeddings if ***x*** is on the manifold, i.e., is a valid face image

$$h = g_e(x)$$



- *If we move from Merkel to Macron in pixel space (**x**) in a straight line, we get averaged faces in between*
- *If we move from Merkel to Macron in latent space (**h**) in a straight line and then apply the generator **x** = $g_d(h)$, we get valid sharp faces in between, maybe the face of Scholz*

Learning an Autoencoder

Learning an Encoder

- If we have,

$$\mathbf{x} = \text{featureMap}(\mathbf{h}^{nat})$$

  we might want to learn an approximate inverse of the feature map

$$\mathbf{h} = g_e(\mathbf{x})$$

- $g_e(\mathbf{x})$ is called an encoder

Autoencoder

- How can we learn $\mathbf{h} = g_e(\mathbf{x})$ if we do not measure $\mathbf{h}$?
- Consider a decoder (which might be close to the feature map)

$$\mathbf{x} = g_d(\mathbf{h})$$

  But again, $\mathbf{h}$ is not measured

- We now simply concatenate the two models and get

$$\hat{\mathbf{x}} = g_d(g_e(\mathbf{x}))$$

- This is called an autoencoder

Linear Autoencoder

- If the encoder and the decoder are linear functions, we get a linear autoencoder
- A special solution is provided by the **principal component analysis** (PCA)

  Encoder:

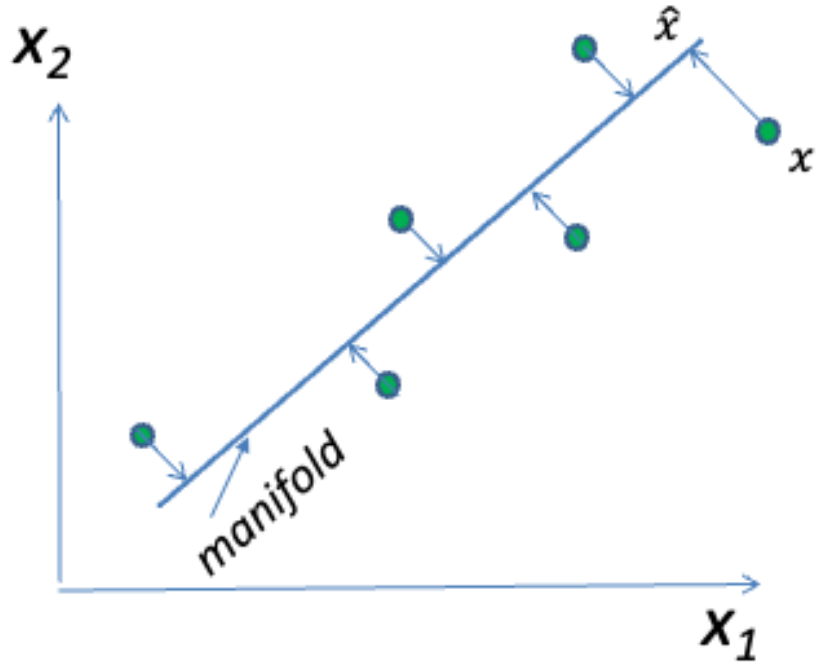
$$\mathbf{h} = g_e(\mathbf{x}) = \mathbf{V}_h^T \mathbf{x}$$

  Decoder:

$$\hat{\mathbf{x}} = g_d(\mathbf{h}) = \mathbf{V}_{M_{hidd}} \mathbf{h} = \mathbf{V}_{M_{hidd}} \mathbf{V}_{M_{hidd}}^T \mathbf{x}$$

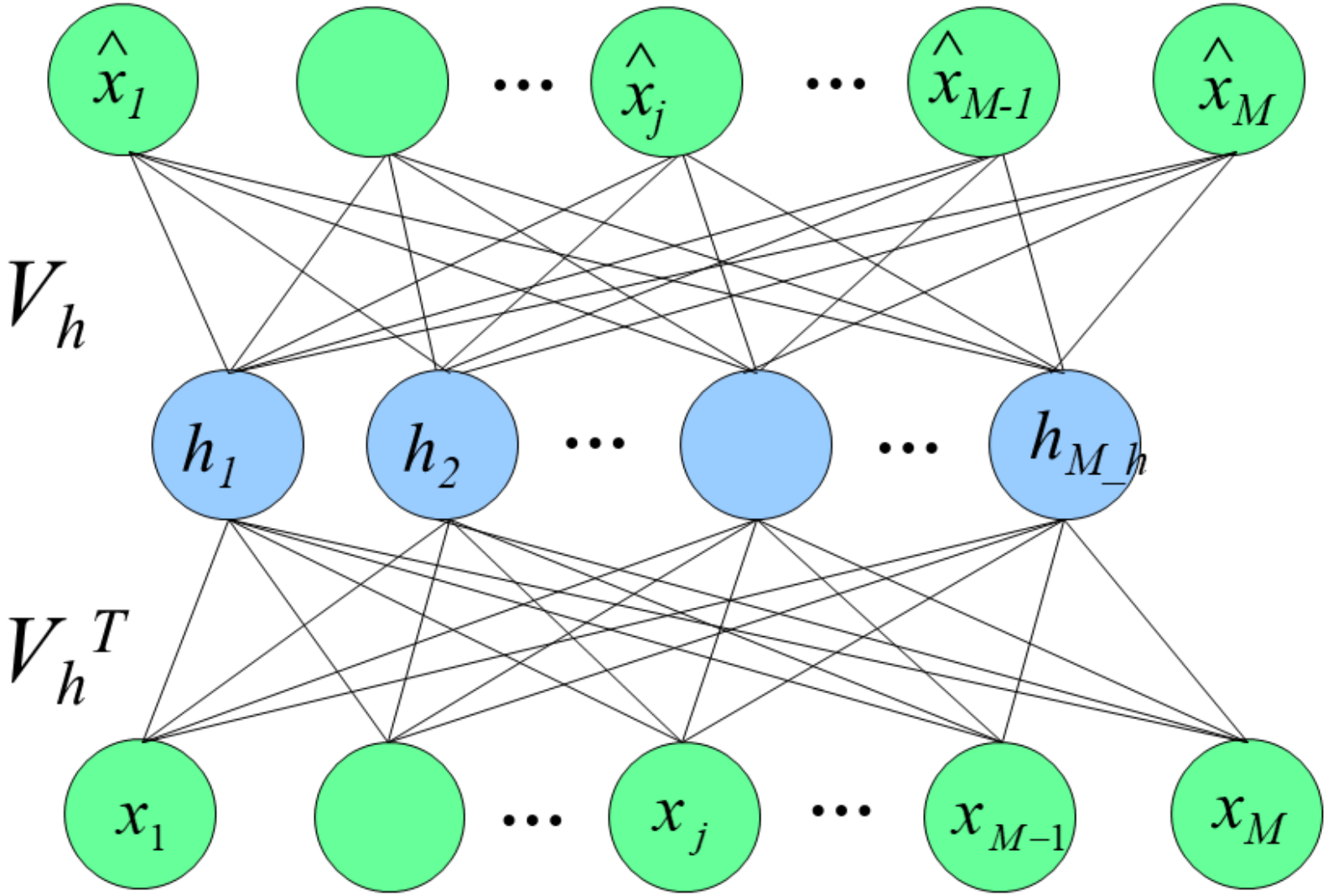
- The $\mathbf{V}_{M_{hidd}}$ are the first M_{hidd} columns of $\mathbf{V}$, where $\mathbf{V}$ is obtained from the **singular value decomposition** SVD

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$



$$\hat{x} = V_h \, V_h^T \, x$$

The manifold is a linear subspace

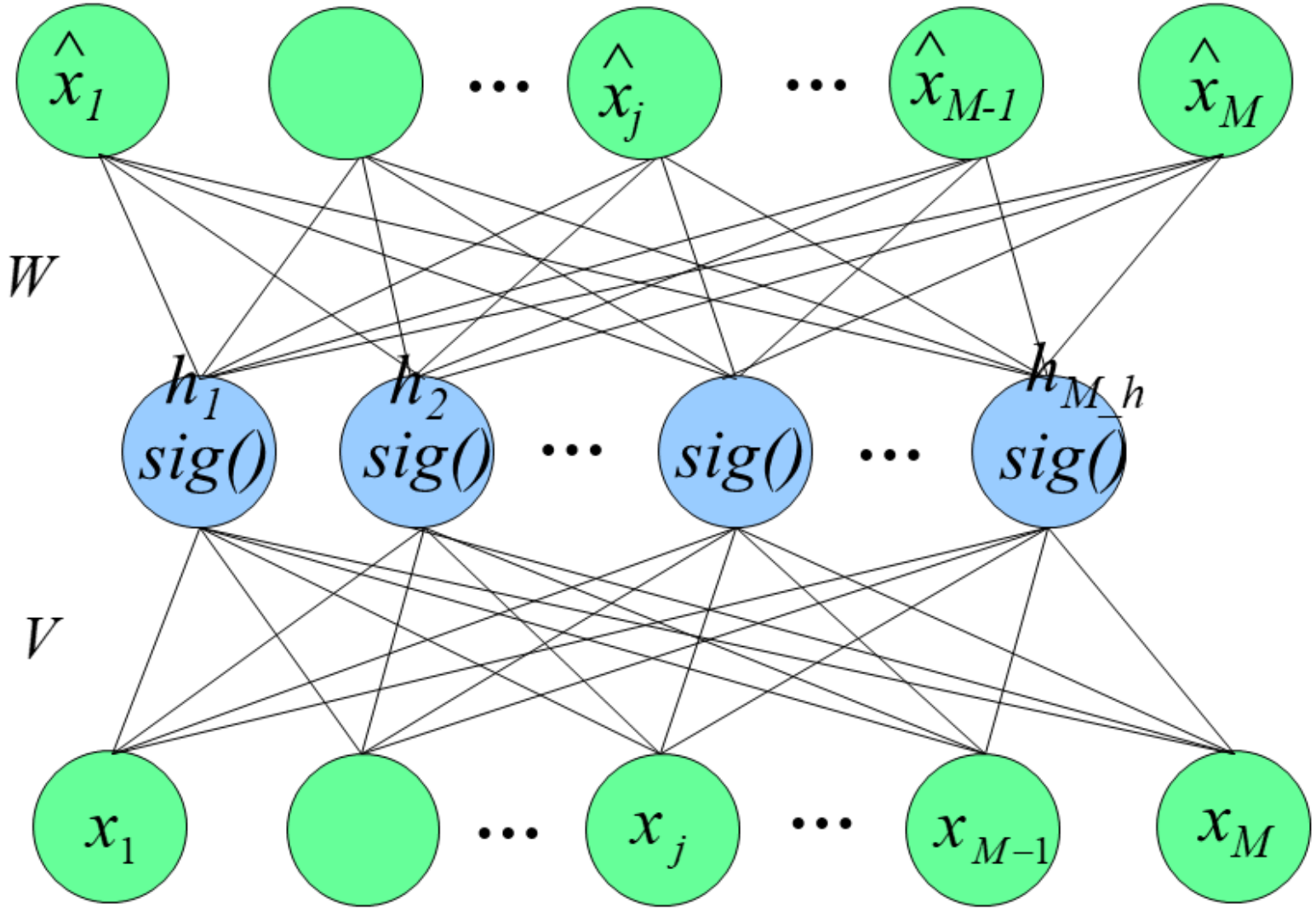

$\hat{x}_1$
$\hat{x}_j$
$\hat{x}_{M-1}$
$\hat{x}_M$
V_h
h_1
h_2
h_{M_h}
V_h^T
x_1
x_j
x_{M-1}
x_M

Neural Network Autoencoder

- In the Neural Network Autoencoder, the encoder and the decoder are modelled by neural networks
- The cost function is

$$\text{cost}(W, V) = \sum_{i=1}^{N} \sum_{j=1}^{M} (x_{i,j} - \widehat{x}_{i,j})^2$$

where $\widehat{x}_{i,1}, \ldots, \widehat{x}_{i,M}$ are the outputs of the neural network autoencoder


$\hat{x}_1$
$\hat{x}_j$
$\hat{x}_{M-1}$
$\hat{x}_M$
W
h_1
h_2
h_{M_h}
sig()
sig()
sig()
sig()
V
x_1
x_j
x_{M-1}
x_M

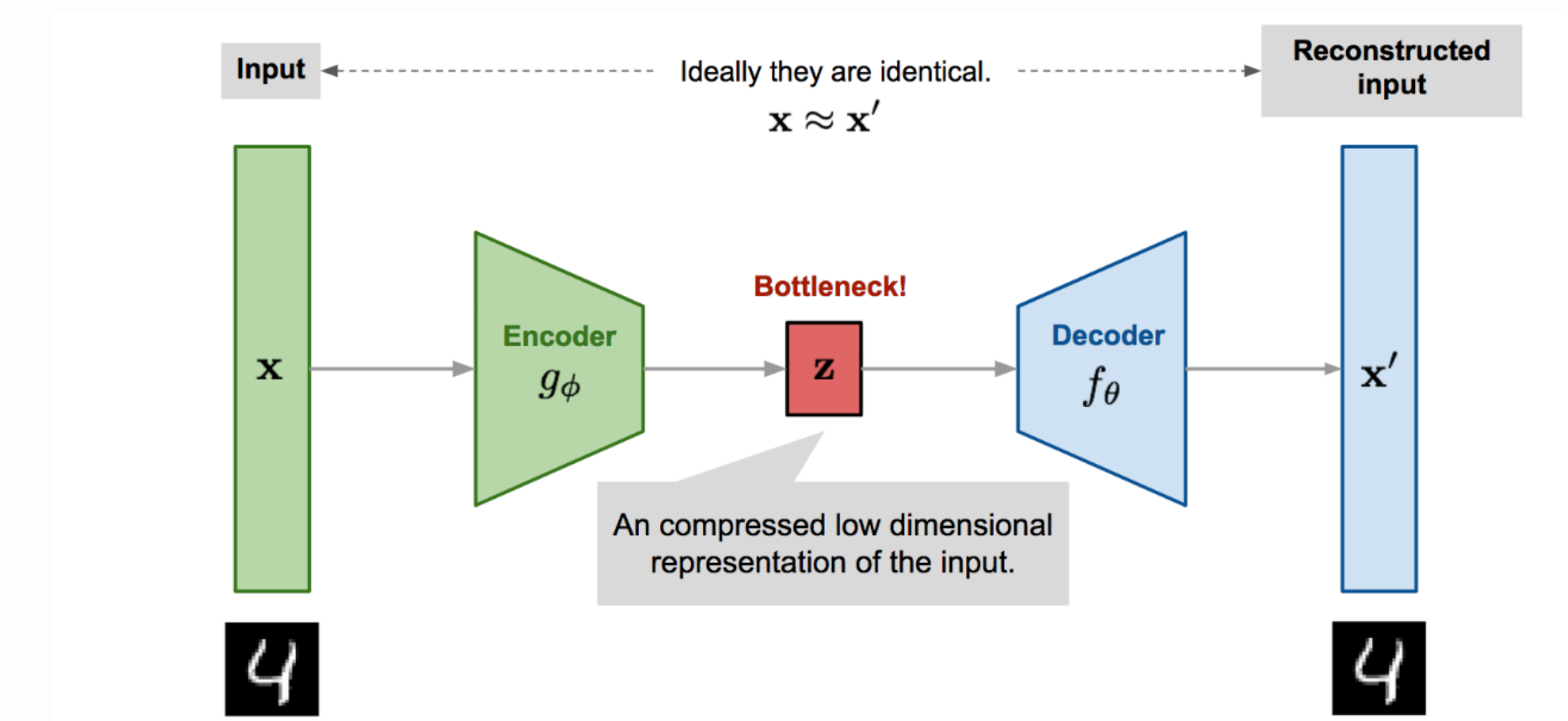
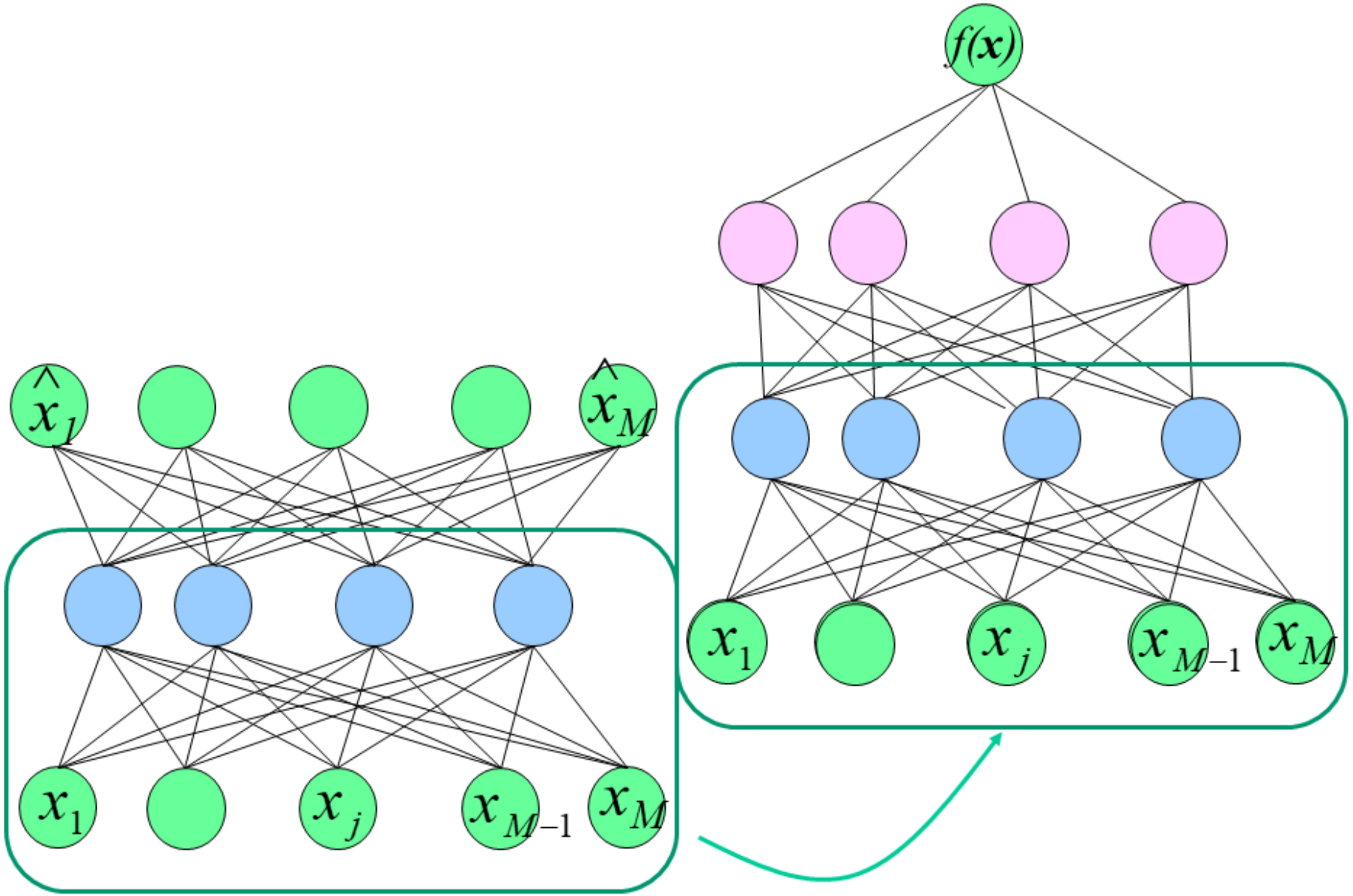


Fig. 1. Illustration of autoencoder model architecture.

Comments and Applications

- Since $\mathbf{h}$ cannot directly be measured, it is called a latent vector in a latent space. The representation of a data point $\mathbf{x}_i$ in latent space $\mathbf{h}_i$ is called its representation or embedding
- Distances in latent space are often more meaningful than in data space, so the latent representations can be used in information retrieval
- The reconstruction error $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$ is often large for patterns which are very different from the training data; the error thus measures novelty, anomality. This can be a basis for fraud detection and plant condition monitoring
- The encoder can be used to pretrain the first layer in a neural network; after initialization, the complete network is then typically trained with backpropagation, including the pretrained layer


f(x)
$\hat{x}_1$
$\hat{x}_M$
x_1
x_j
x_{M-1}
x_M
x_1
x_j
x_{M-1}
x_M

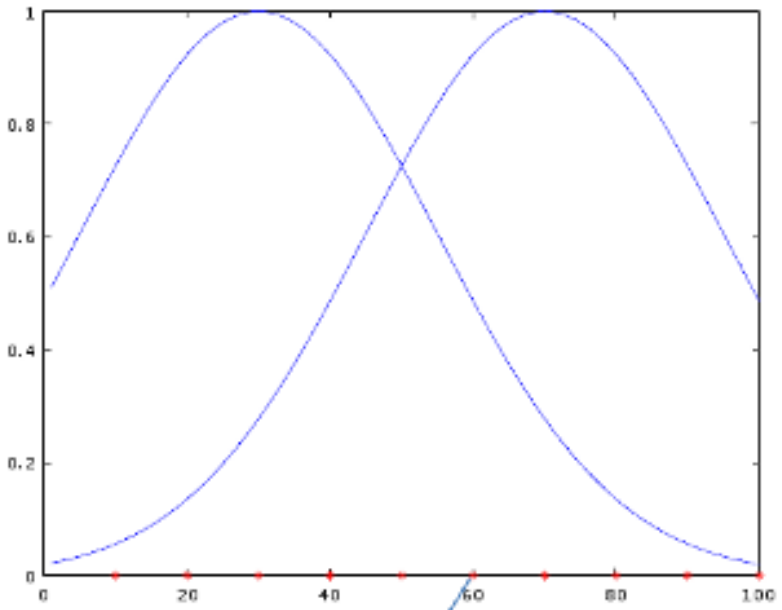
Data Represented in a Noisy Feature Space

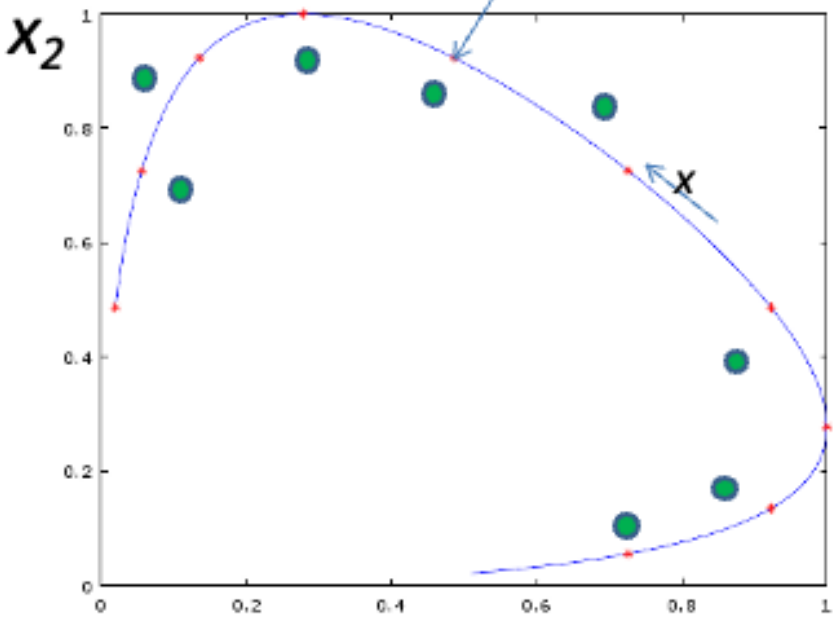
- The feature map might include some noise,

$$\mathbf{x} = \text{featureMap}(\mathbf{h}^{nat}) + \vec{\epsilon}$$

  where $\vec{\epsilon}$ is a noise vectors; then the data might only be exactly on the manifold

- One would want that $g_e(\text{featureMap}(\mathbf{h}^{nat}) + \vec{\epsilon}) \approx g_e(\text{featureMap}(\mathbf{h}^{nat}))$ such that the encoder is noise insensitive; this is enforced by the **denoising autoencoder**





Due to some noise process, the data points do not lie on the manifold

Denoising Autoencoder (DAE)

- Denoising autoencoder,

$$\mathbf{x}_i \leftarrow g_d(g_e(\mathbf{x}_i + \epsilon_i))$$

  where ϵ_i is random noise added to the input!

- This also prevents an autoencoder from learning an identity function

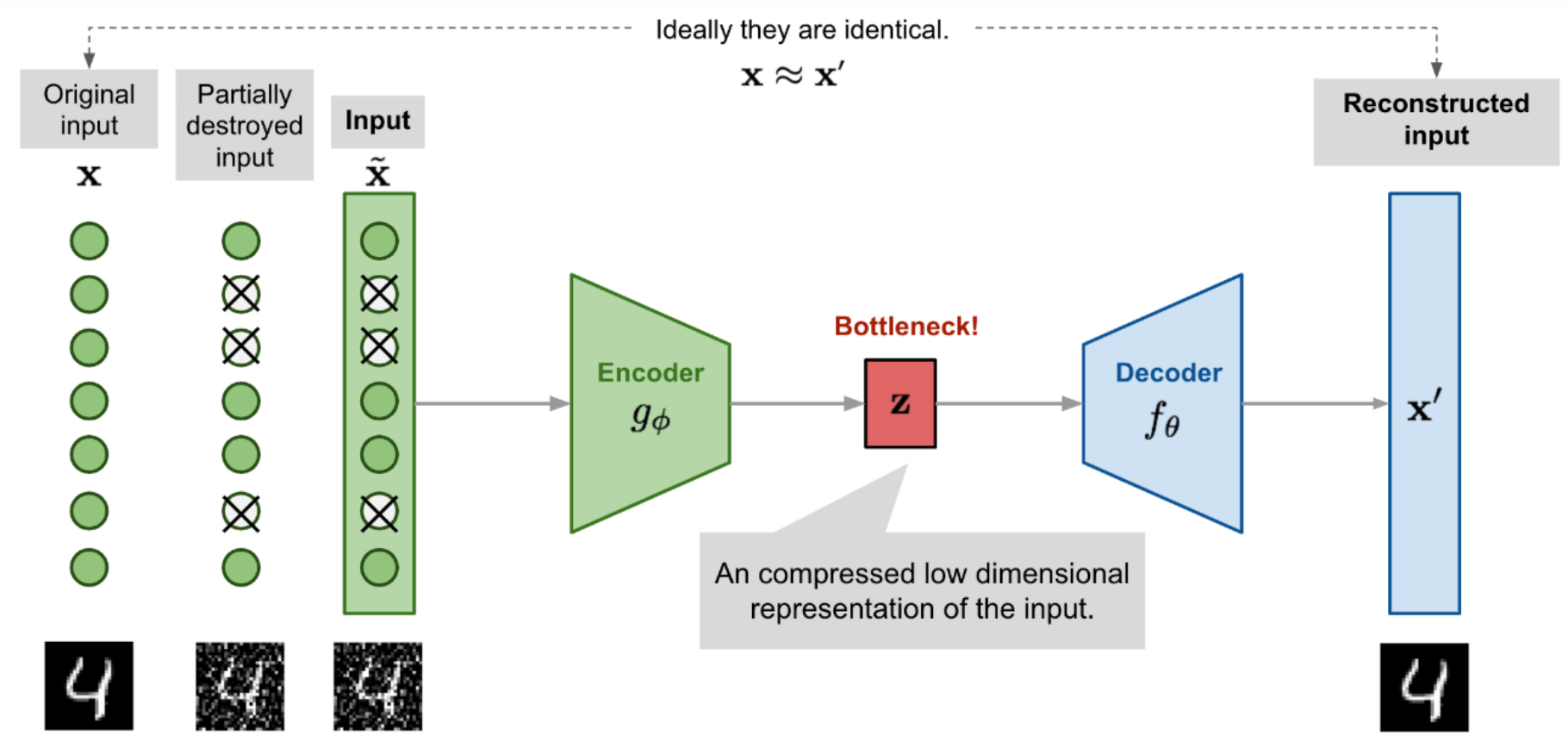


Fig. 2. Illustration of denoising autoencoder model architecture.

Learning a Generator

Learning a Decoder (Generator)

- Requirement 1: No manifold in $\mathbf{h}$-space: for any random $h_1, \ldots, h_{M_{hidd}}$,

$$\mathbf{x} = g_d(h_1, \ldots, h_{M_{hidd}})$$

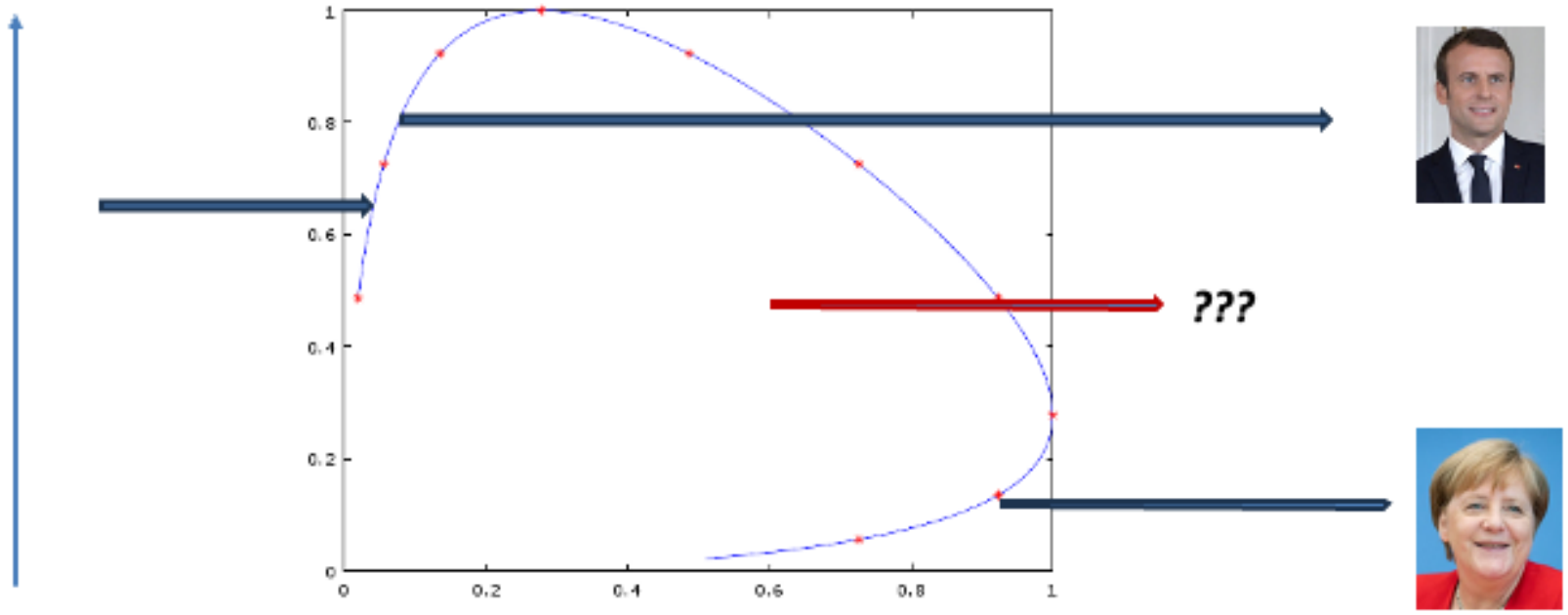
  generates a high-quality face image

- Requirement 2: Disentanglement: $h_1, \ldots, h_{M_{hidd}}$ can be interpreted

- Requirement 3: Conditional (attribute specific) models

$$\mathbf{x} = g_d(h_1, \ldots, h_{M_{hidd}}, m_1, \ldots, m_{M_m})$$

  The generated image has real features $m_1, \ldots, m_{M_m}$; do I want a smiling face, do I want a beard, glasses, sunglasses, ...?



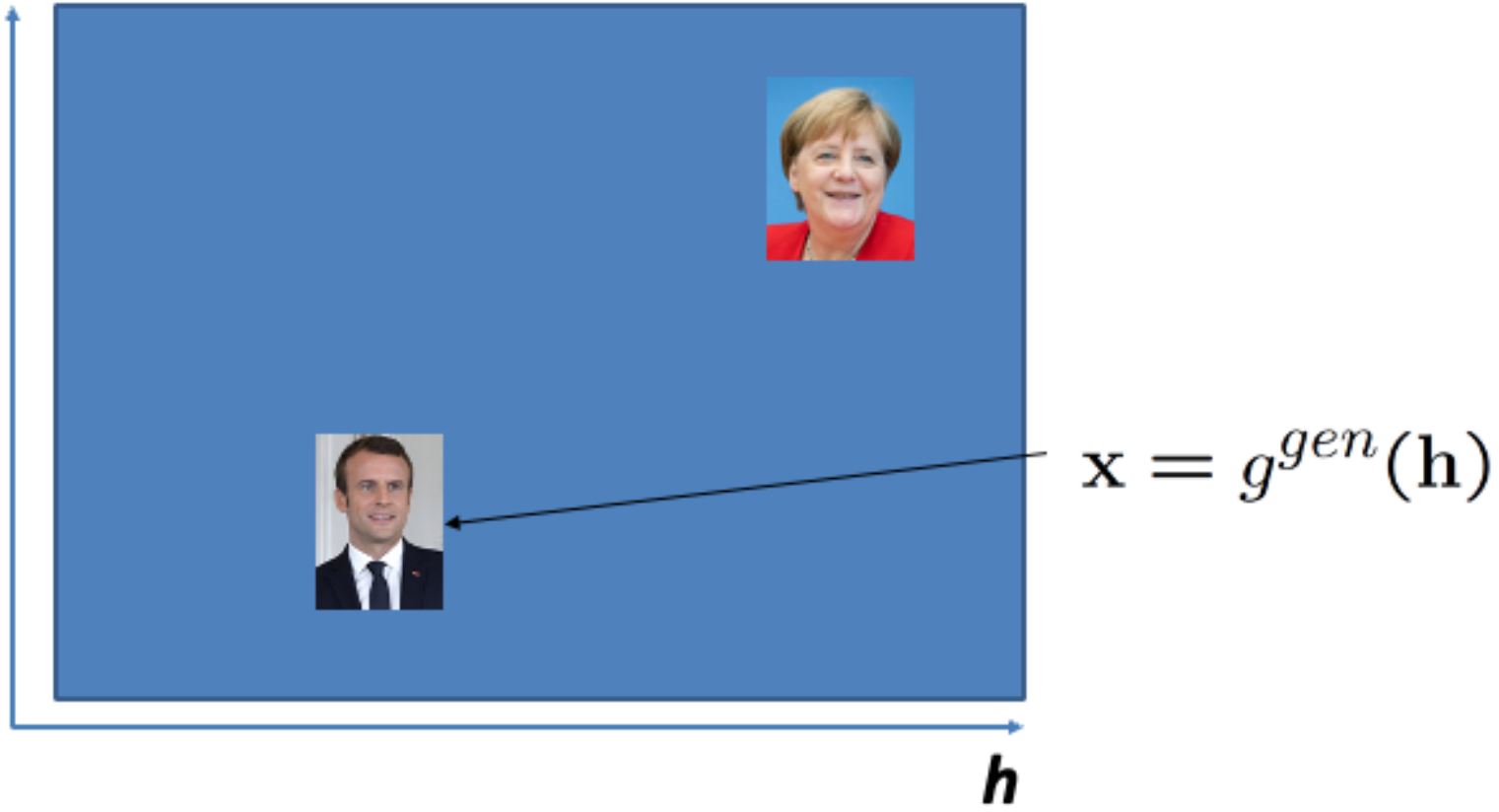
h:
Here a 1-D ***h*** is optimal:
every ***h*** gives a valid face image

h:
The model might have generated a 2-D latent space:
Only the 1-D manifold in the 2-D space ***h*** on the manifold h gives a valid face image

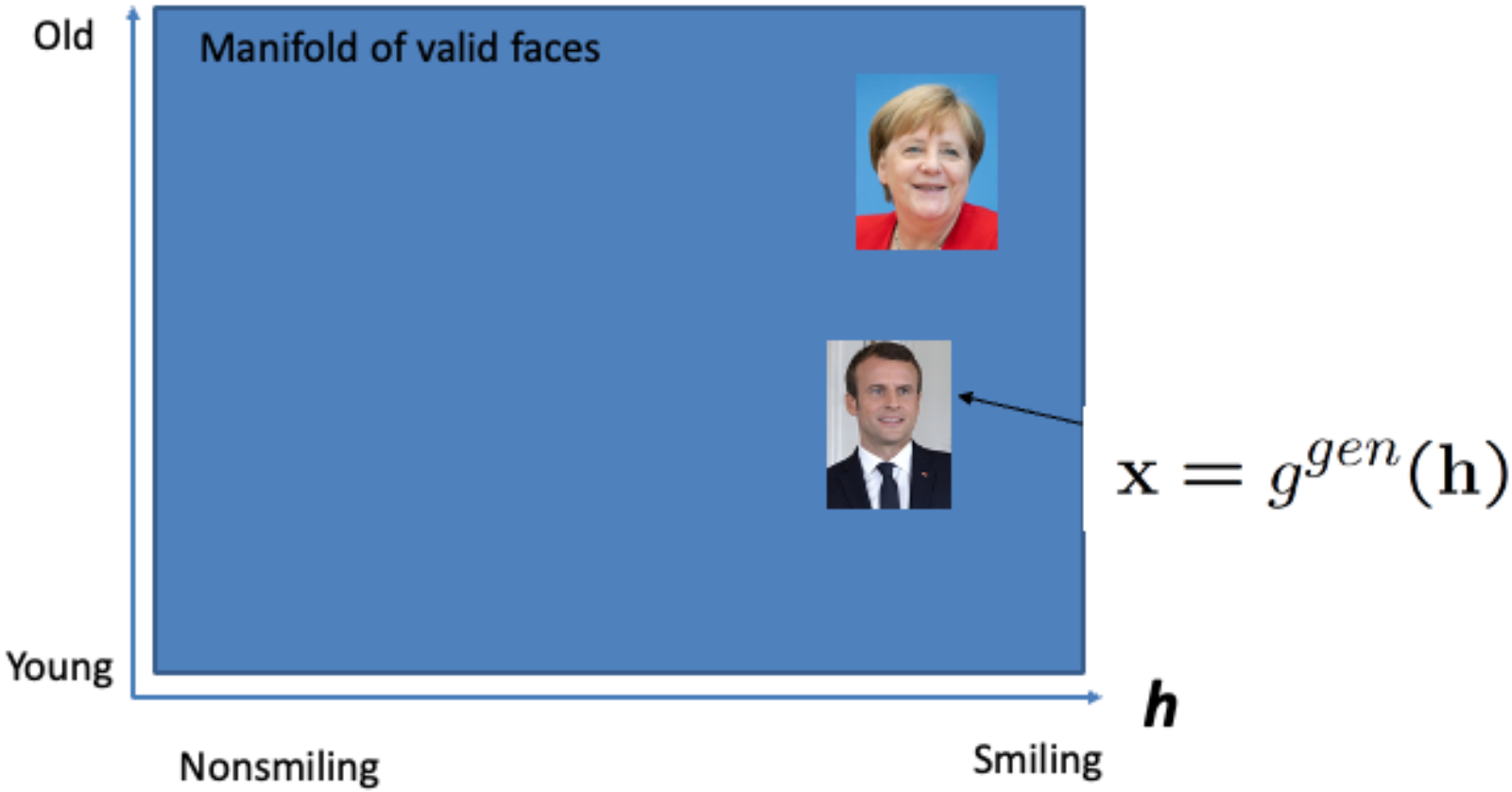
x

Requirement 1: no manifold

In the Right Space, the Whole Space Maps to Faces



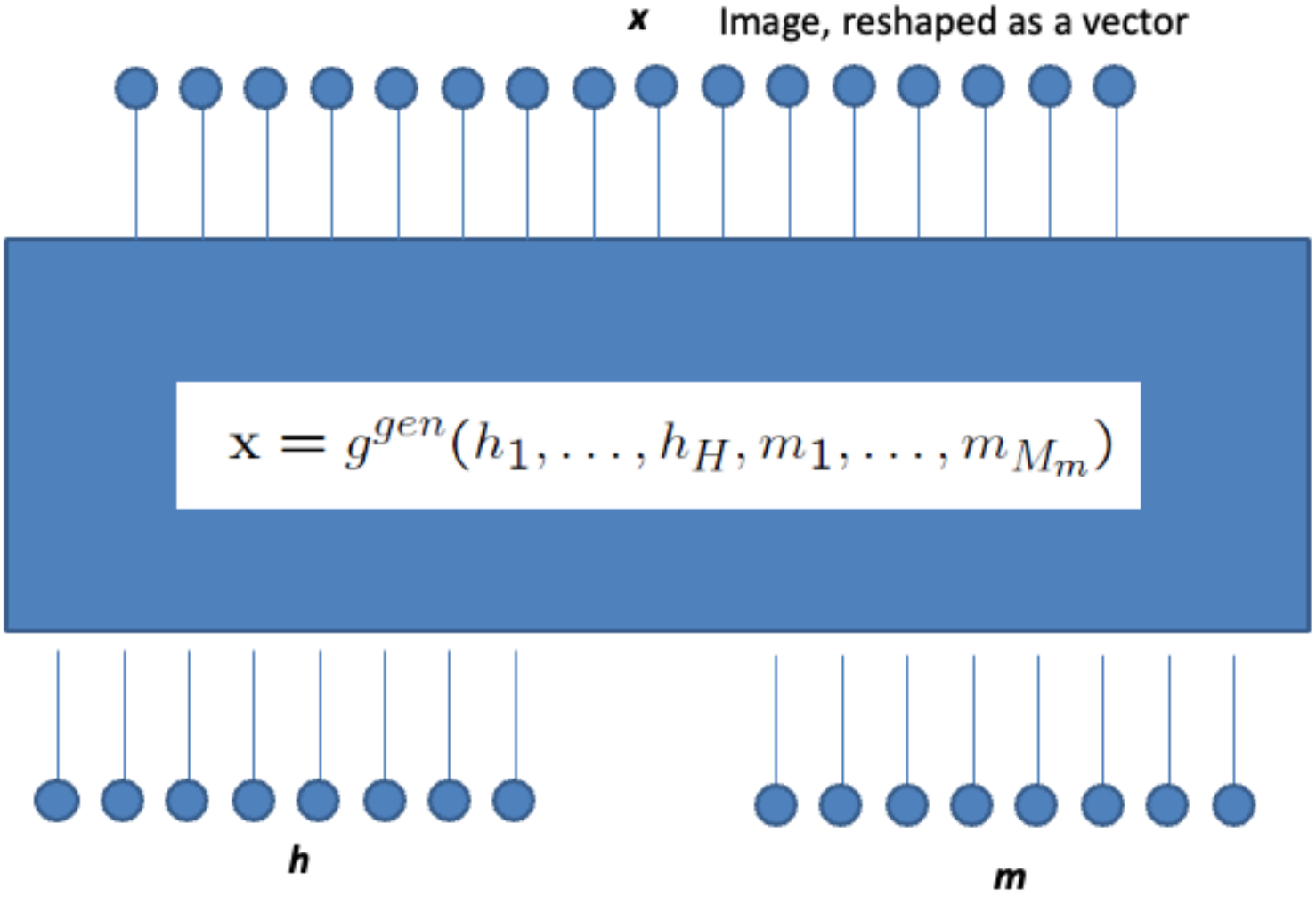
Requirement 2:
Disentangled Representations



$$\mathbf{x} = g^{gen}(\mathbf{h})$$

Even better: the dimensions have a meaning

Requirement 3: Conditional



- i.i.d. random inputs
- Disentangled: Each input might have an interpretation

Factor Analysis

- The generator generates valid images out of random noise, i.e., a random $\mathbf{h}$!
- We can consider the attributes $m_1, \ldots, m_{M_m}$ to correspond to a subset of dimensions of $\mathbf{h}$; thus the manifold stays the same, but we only generate images with the desired attributes; thus only $M_r = M_{hidd} - M_m$ dimensions are random
- The generator is

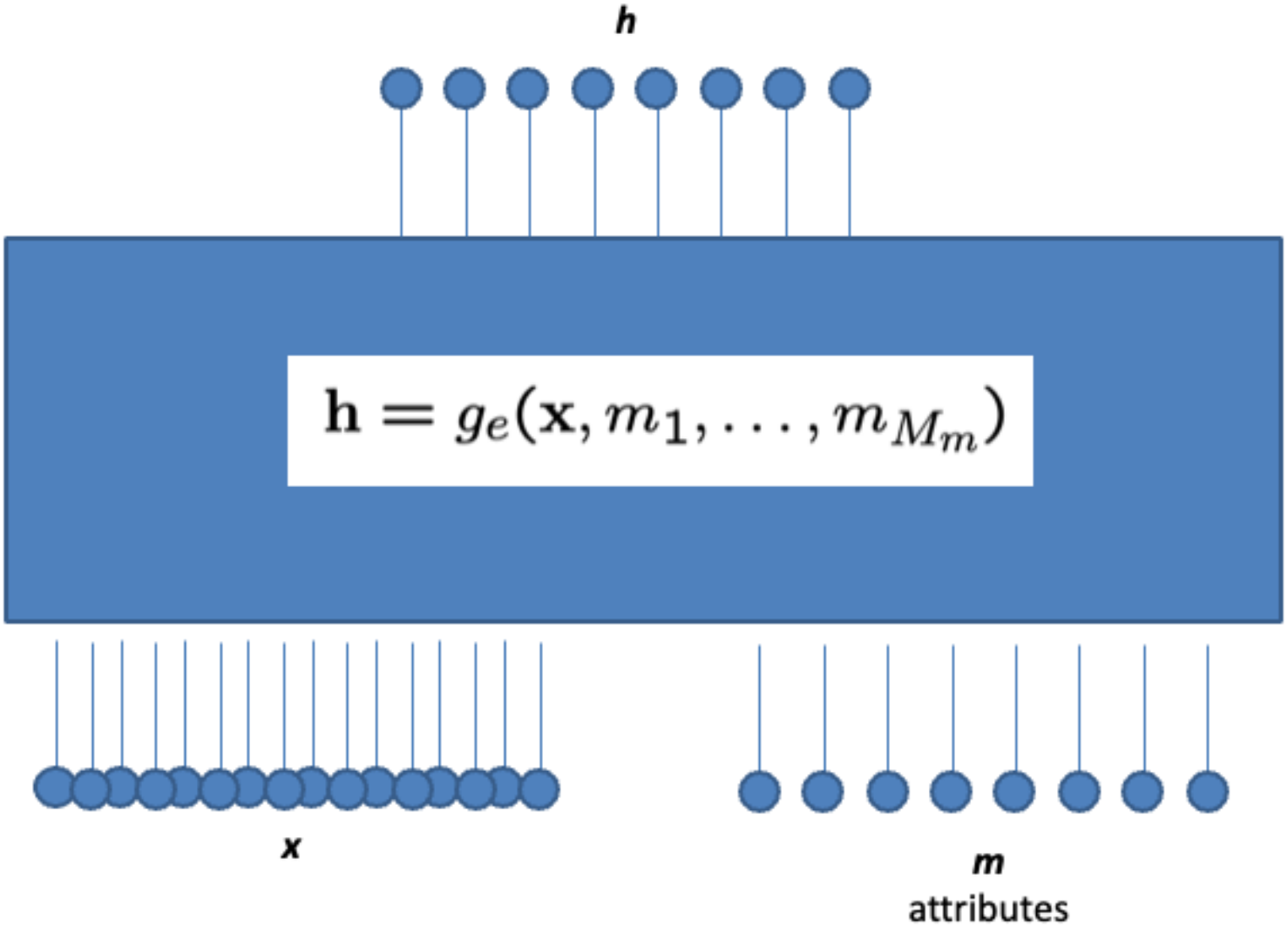
$$\mathbf{x} = g_d(h_1, \ldots, h_{M_r}, m_1, \ldots, m_{M_m})$$

  $h_1, \ldots, h_{M_r}$ are independent, e.g., independent zero-mean unit-variance Gaussians

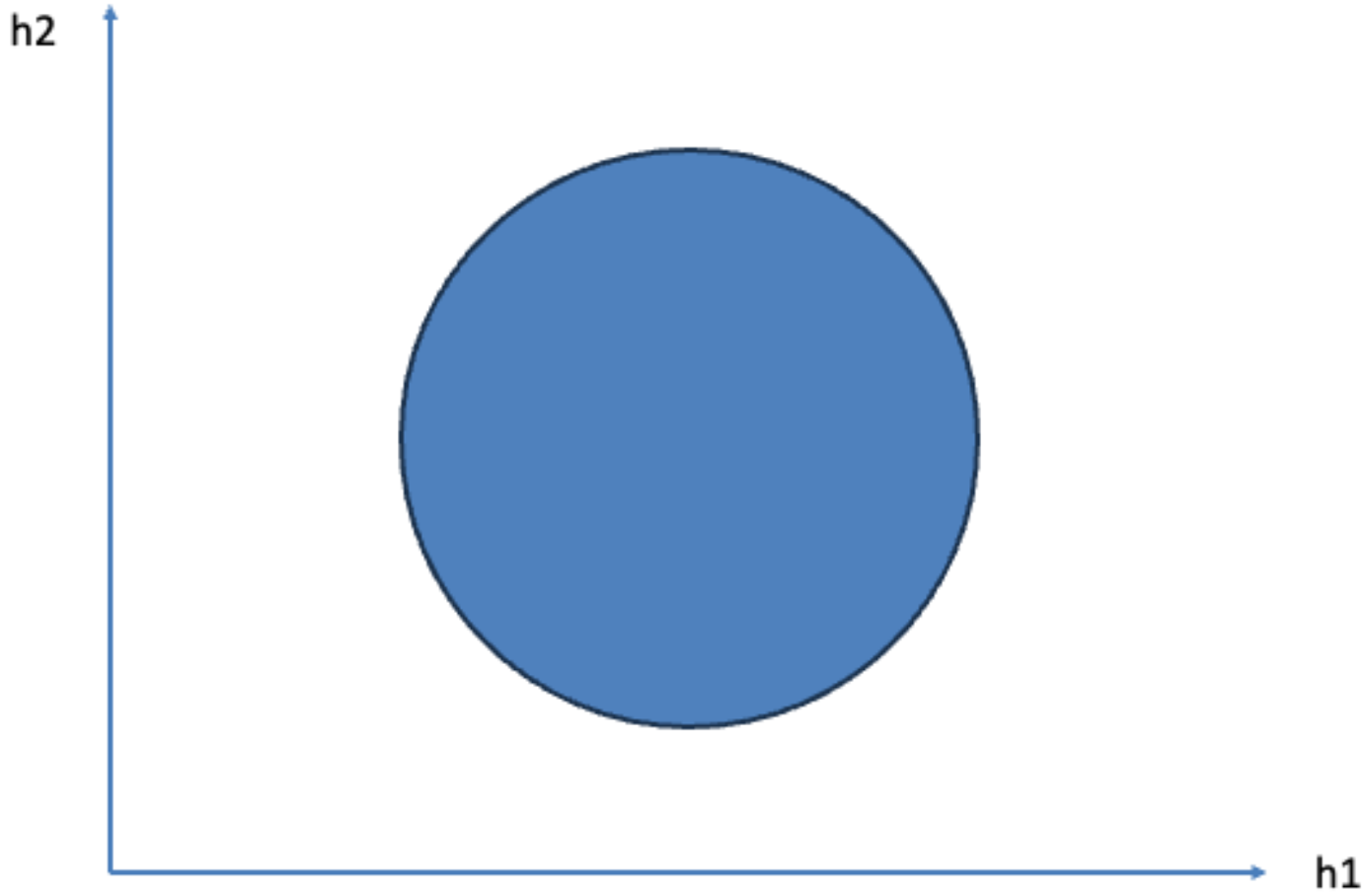
- The encoder is the also conditioned on $m_1, \ldots, m_{M_m}$

$$\mathbf{h} = g_e(\mathbf{x}, m_1, \ldots, m_{M_m})$$

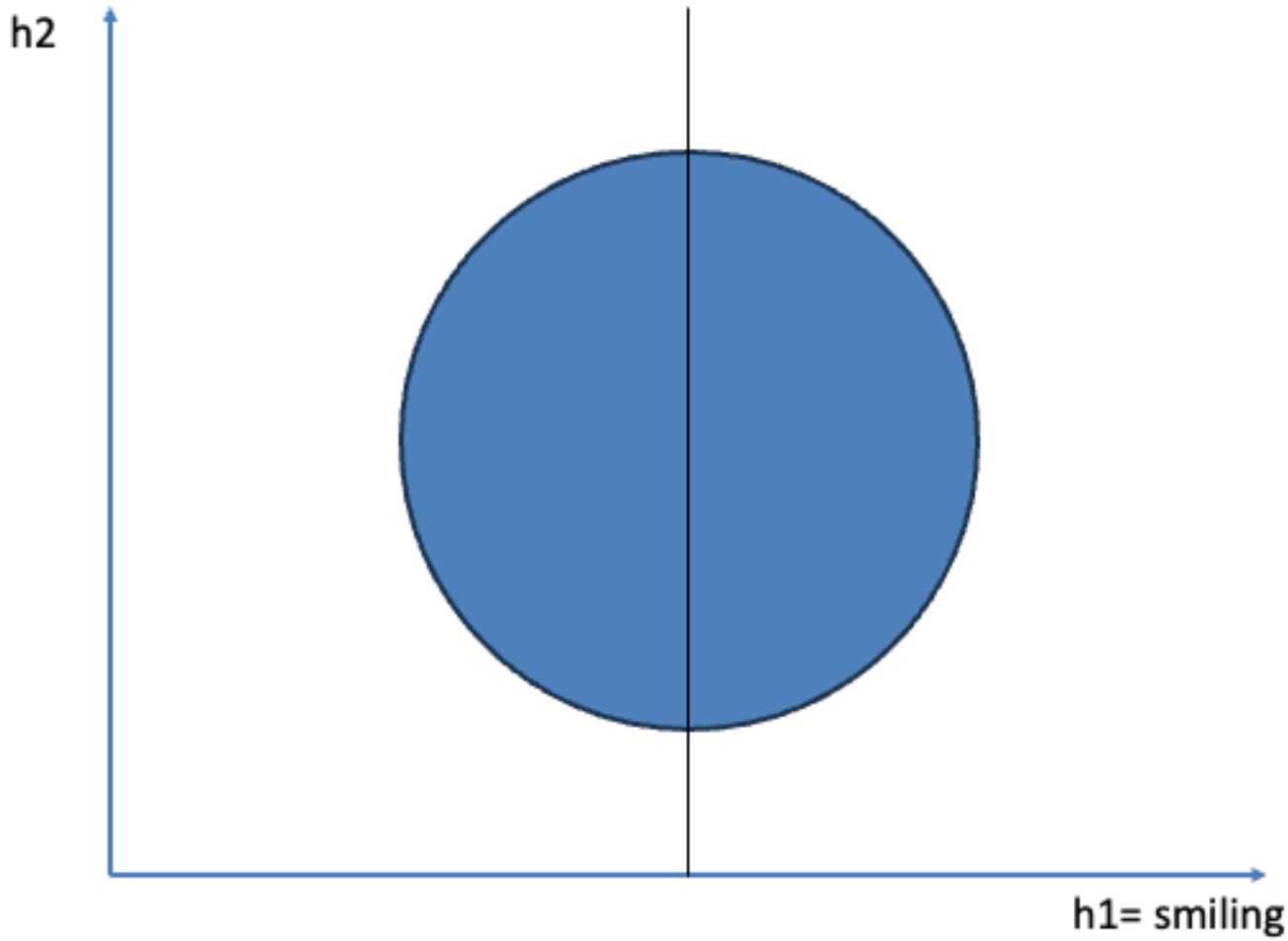
Conditional Encoder



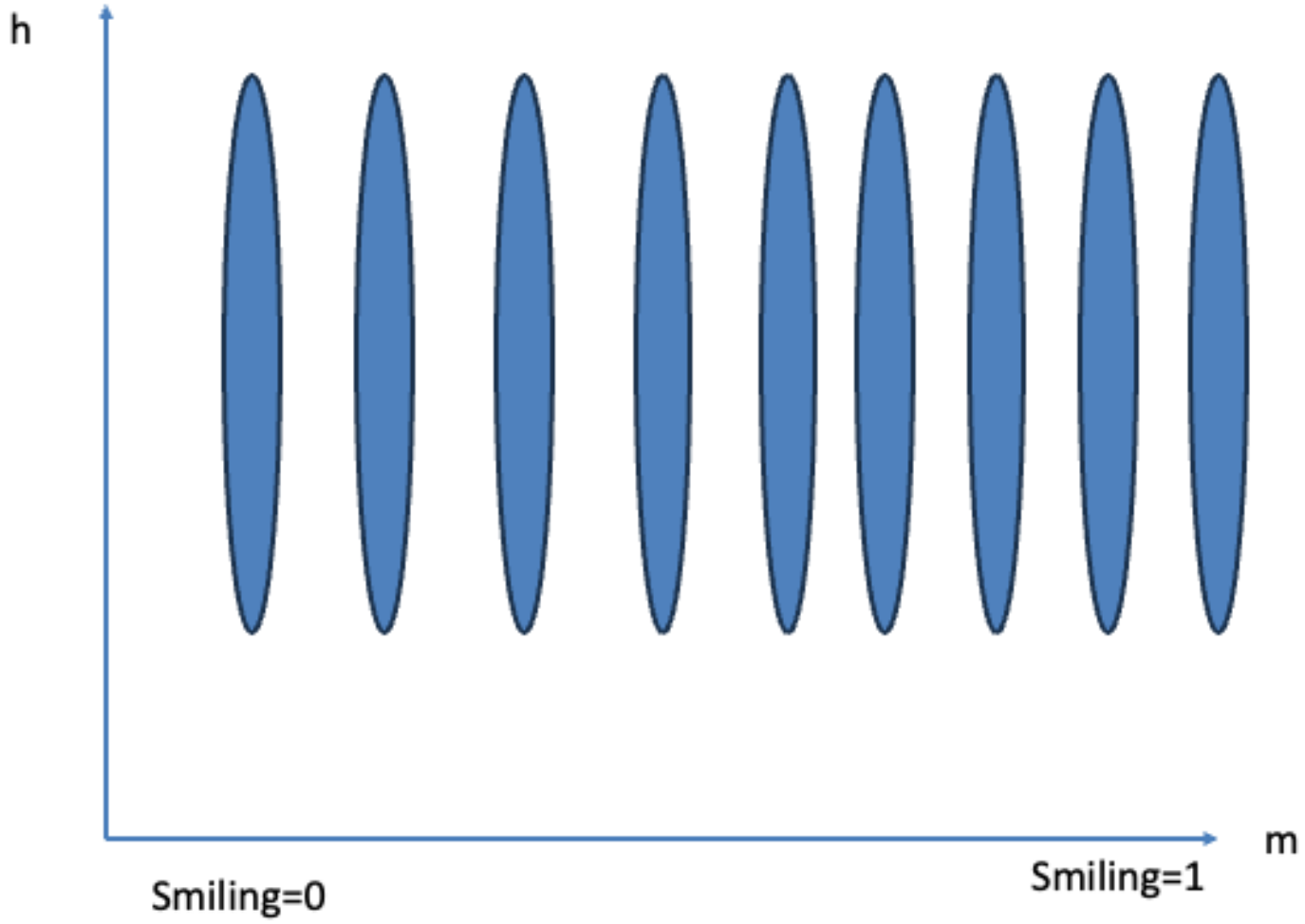
Mutli-variate Gaussian in latent space



Mutli-variate Gaussian in latent space with disentangled dimensions



Conditionned on attribute “smiling”, we get independent Gaussians



Variational Autoencoder

Learning a Decoder (Generator)

- If I knew for each data point $\mathbf{h}$ and $\mathbf{x}$, I could learn a generator $g_d(\cdot)$ simply by supervised training
- Unfortunately, $\mathbf{h}$ is unknown
- How about I assume that $\mathbf{h}$ comes from a Gaussian distribution with unit variance $P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; 0, I)$ (each dimension is independently Gaussian distributed, with unit variance)
- Goal: **any** sample $\mathbf{h}$ from this Gaussian distribution should generate a valid **x** (this was not enforced in the normal autoencoder)
- This is the idea behind the Variational Autoencoder (VAE)

Generating Data from the VAE

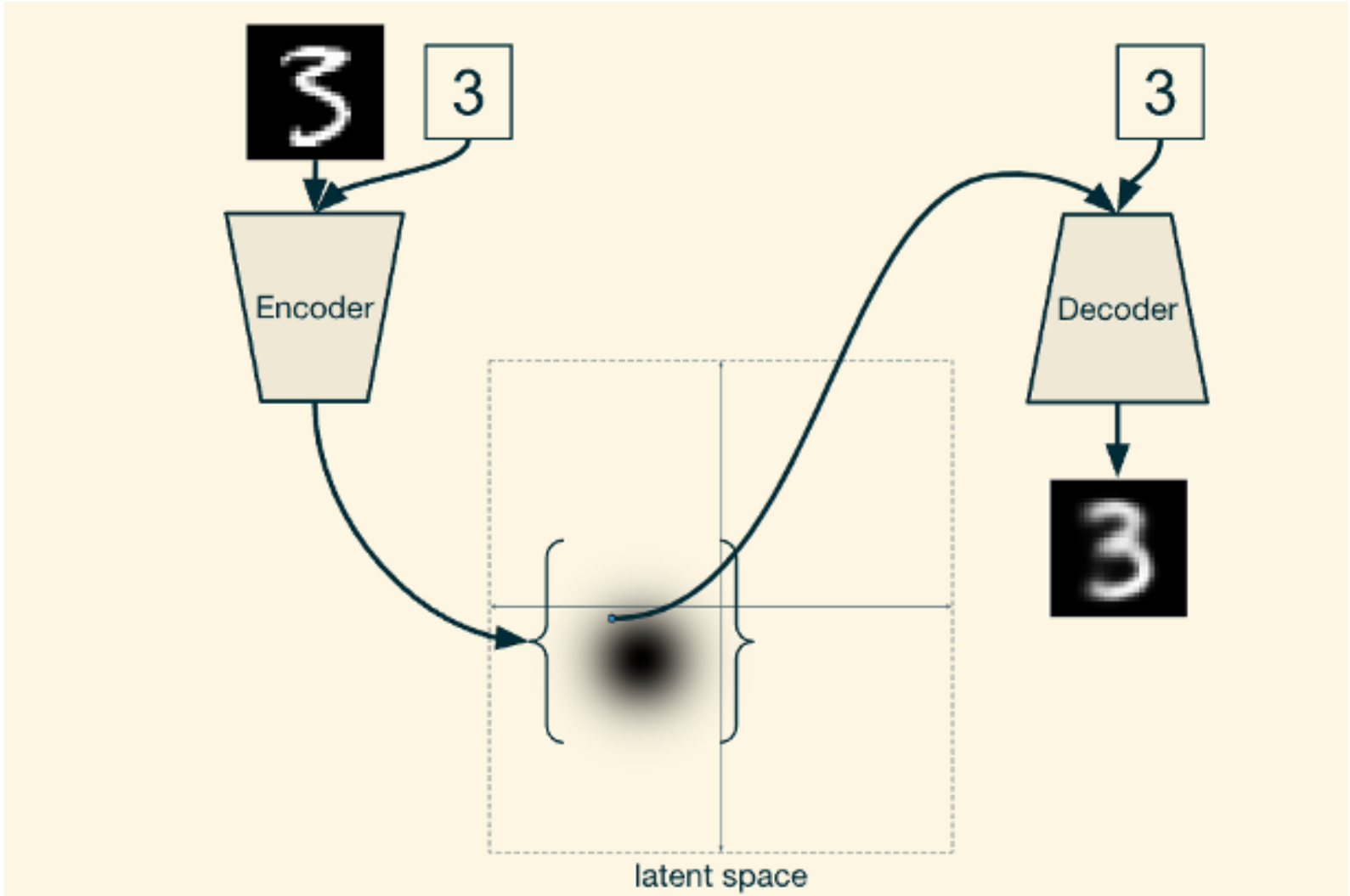
- After training, we generate a new $\mathbf{x}$ by first generating a sample $\mathbf{h}_s$ from $\mathcal{N}(0, I)$; then

$$\mathbf{x} = g_d(\mathbf{h}_s, \mathbf{v})$$

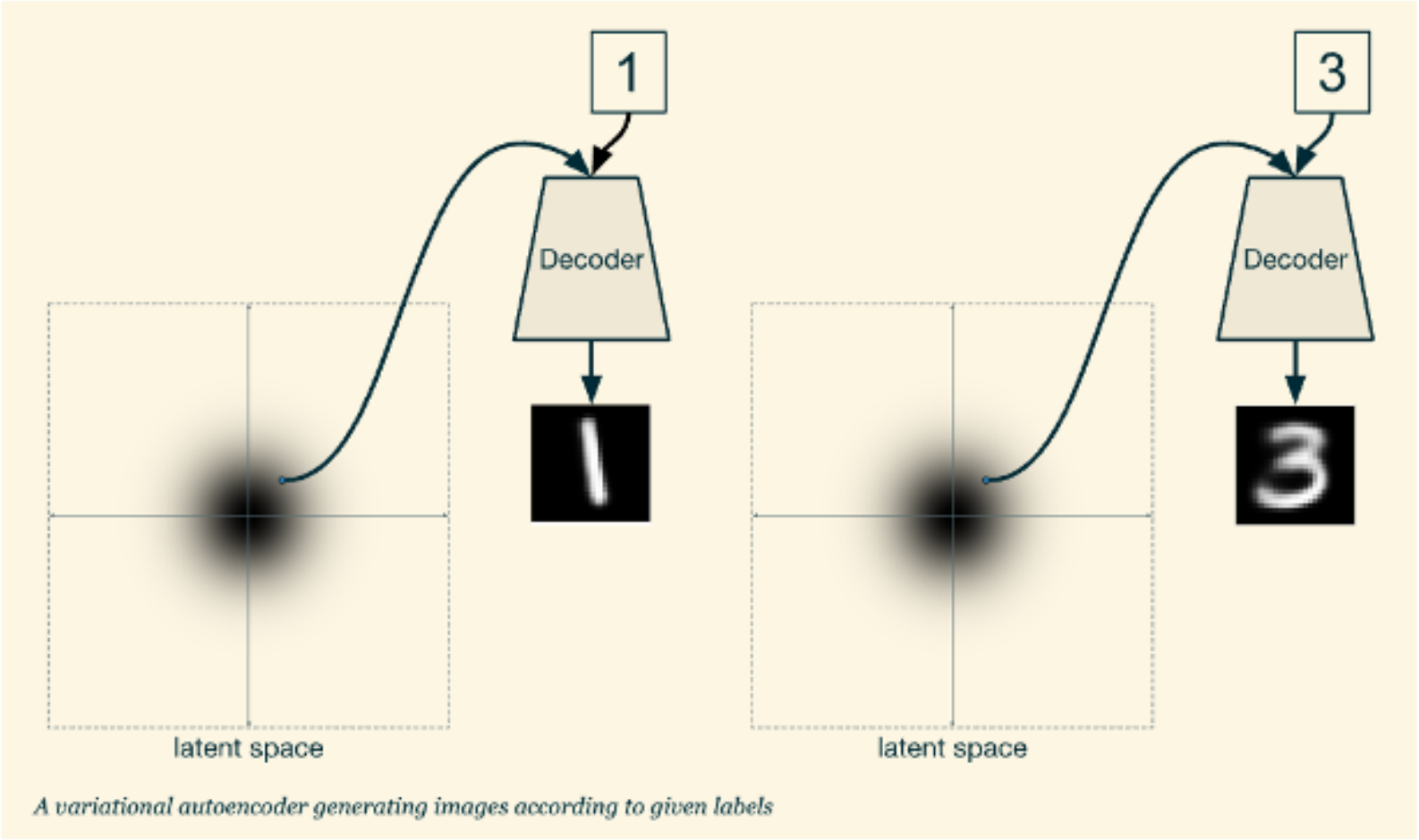
  Here, $\mathbf{v}$ are parameters of the generator $g_d(\cdot)$

- Thus generating a new $\mathbf{h}$ is trivial, but how do I learn the $g_d(\cdot)$?

- This is a bit involved and is described in the Appendix



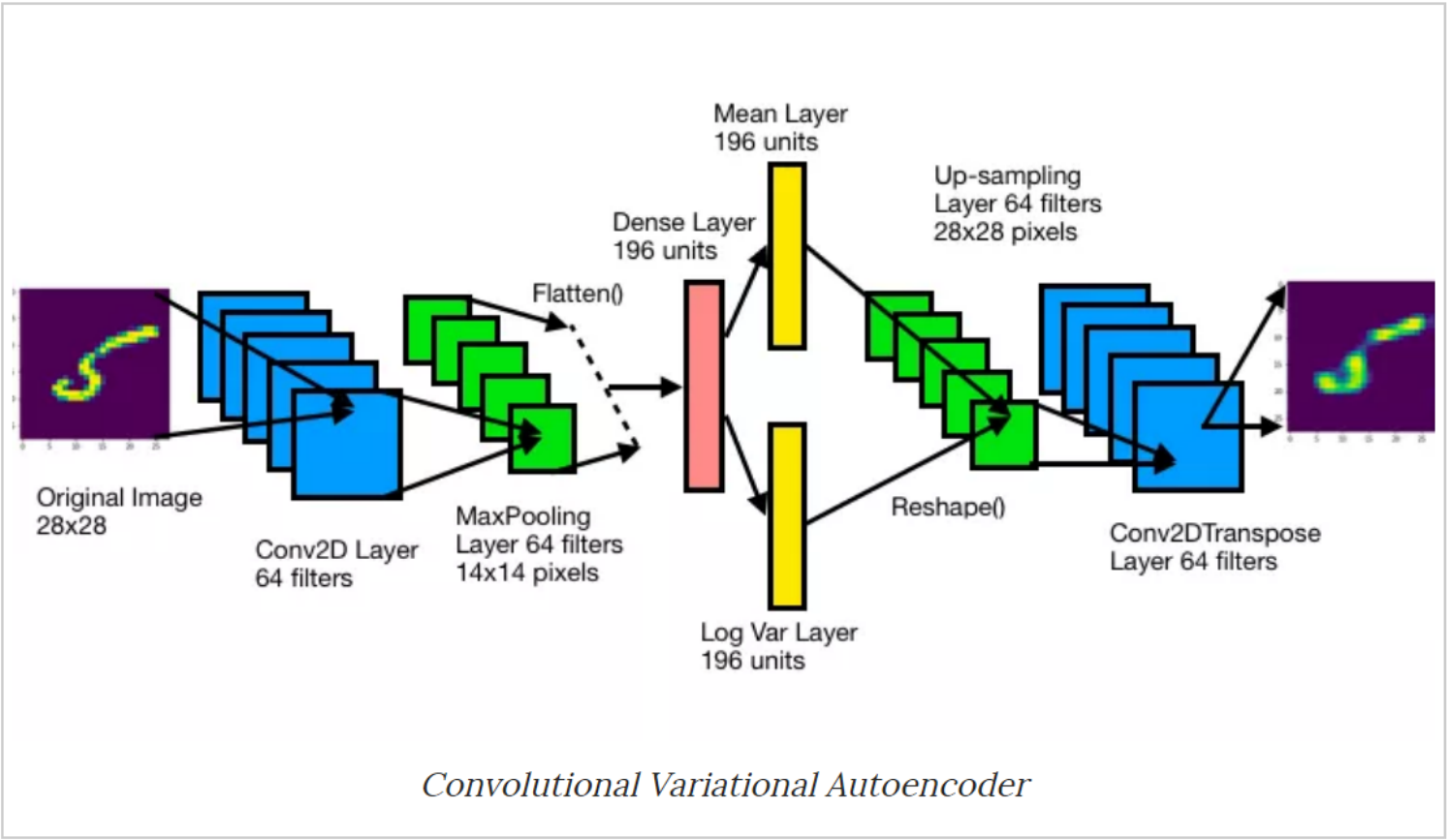
A conditional variational autoencoder



A variational autoencoder generating images according to given labels

Convolutional Variational Autoencoders

- The convolutional variational autoencoder uses convolutional layers



Convolutional Variational Autoencoder

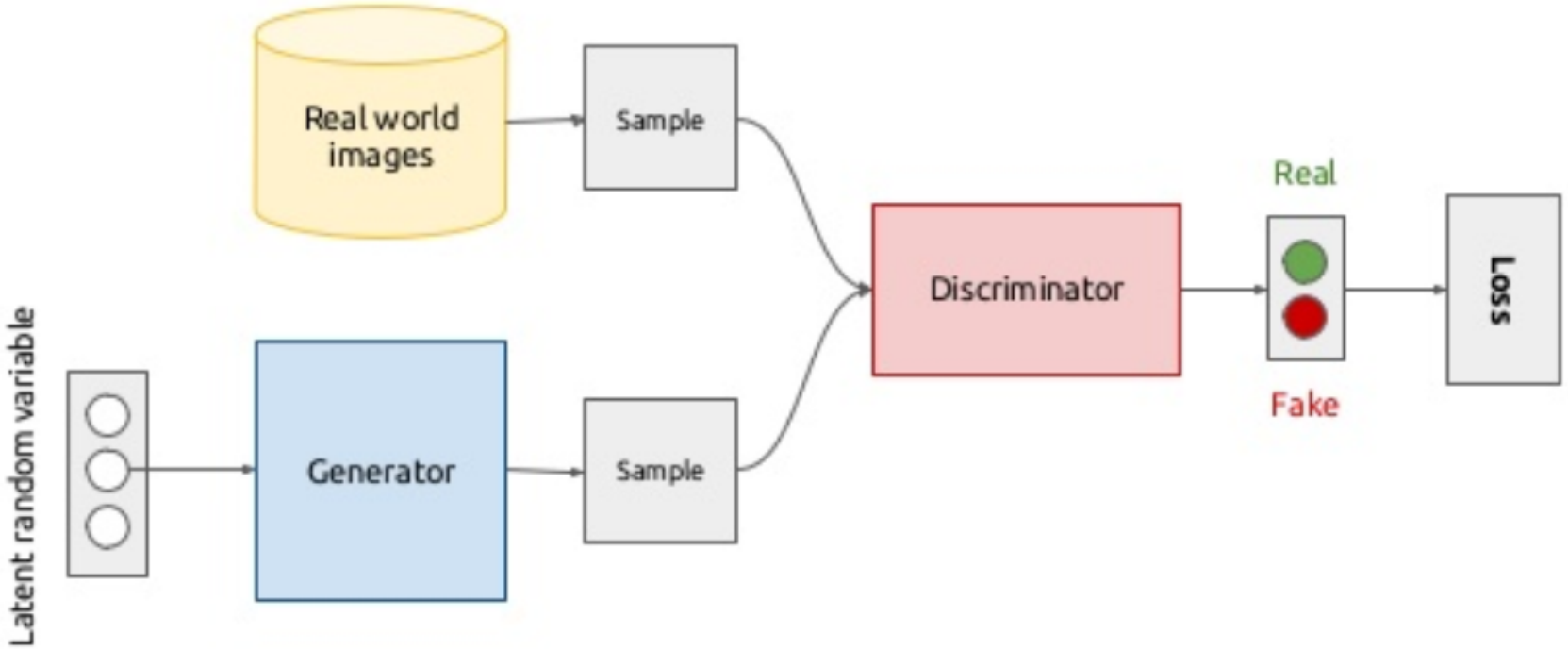
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)

- Can we train a decoder (generator) without without an encoder?
- Let's assume we have a larger number of **generators** available; let's assume that each generator generates a data set; which one is the best generator?
- The best generator might be the one where a **discriminator** (i.e., a binary neural network classifier) trained to separate training data and the data from a particular generator, cannot separate the two classes; if this is the case, one might say that

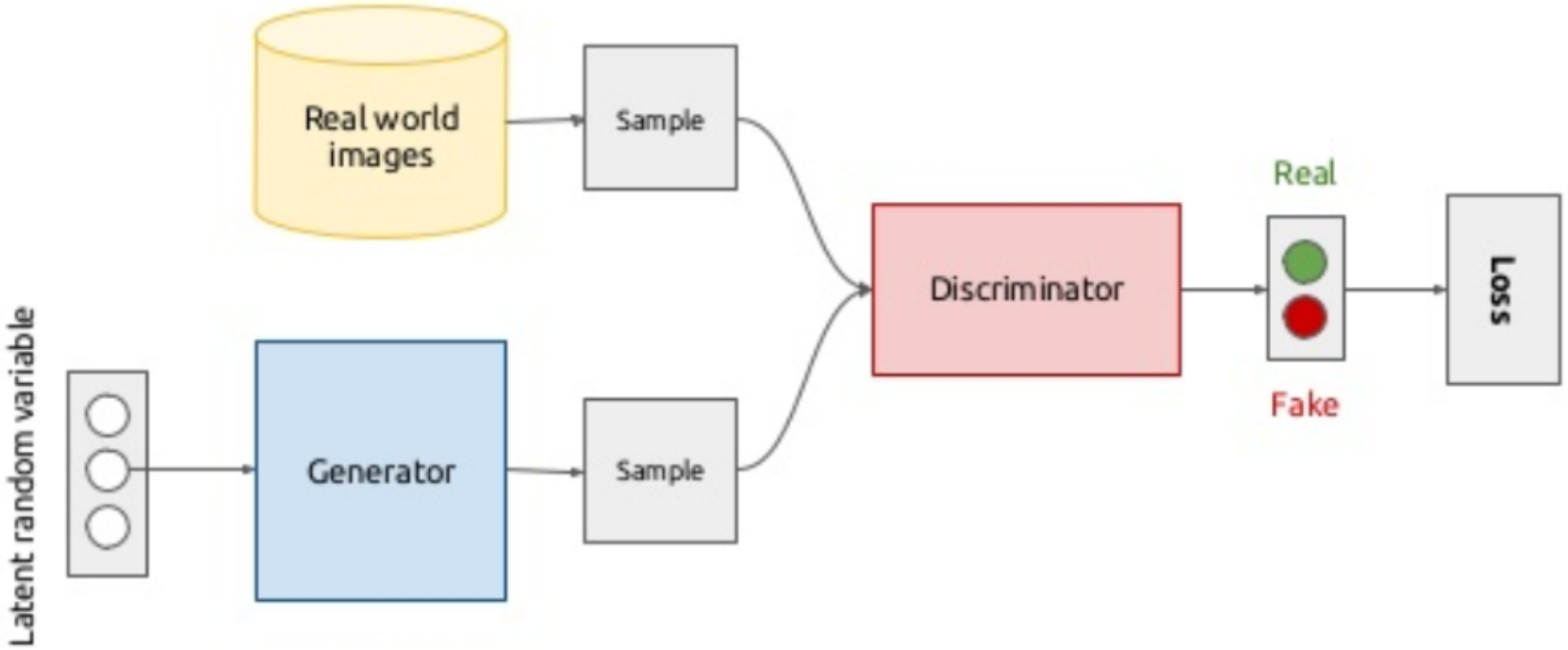
$$P_{train}(\mathbf{x}) \approx P^{gen}(\mathbf{x})$$

- In GAN models, there is only one generator and one discriminator and both are trained jointly


Real world images
Sample
Real
Discriminator
Loss
Latent random variable
Generator
Sample
Fake

Cost Function

- **Discriminator:** Given a set of training data and a set of generated data: the weights $\mathbf{w}$ in the discriminator are updated to **maximize** the negative cross entropy cost function (i.e., the log-likelihood); the targets for the training data are 1 and for the generated data 0 (this is the same as minimizing the cross entropy, i.e. the discriminator is trained to be the best classifier possible)
- **Generator:** With a given discriminator and a set of latent samples: update the weights $\mathbf{v}$ in the generator, such that the generated data get closer to the classification decision boundary: the generator is trained to **minimize** the negative cross entropy cost function, where backpropagation is performed through the discriminator (this is the same as maximizing the cross entropy)


Real world images
Sample
Real
Discriminator
Loss
Latent random variable
Generator
Sample
Fake

Parameter Learning

- Optimal parameters are

$$(\mathbf{w}, \mathbf{v}) = \arg\min_{\mathbf{v}} \arg\max_{\mathbf{w}} \text{cost}(\mathbf{w}, \mathbf{v})$$

  where

$$\text{cost}(\mathbf{w}, \mathbf{v}) = \sum_{i:\mathbf{x}_i \in \text{train}} \log g^{dis}(\mathbf{x}_i, \mathbf{w}) + \sum_{i':\mathbf{x}_{i'} \in \text{gen}} \log[1 - g^{dis}(g^{gen}(\mathbf{h}_{i'}, \mathbf{v}), \mathbf{w})]$$

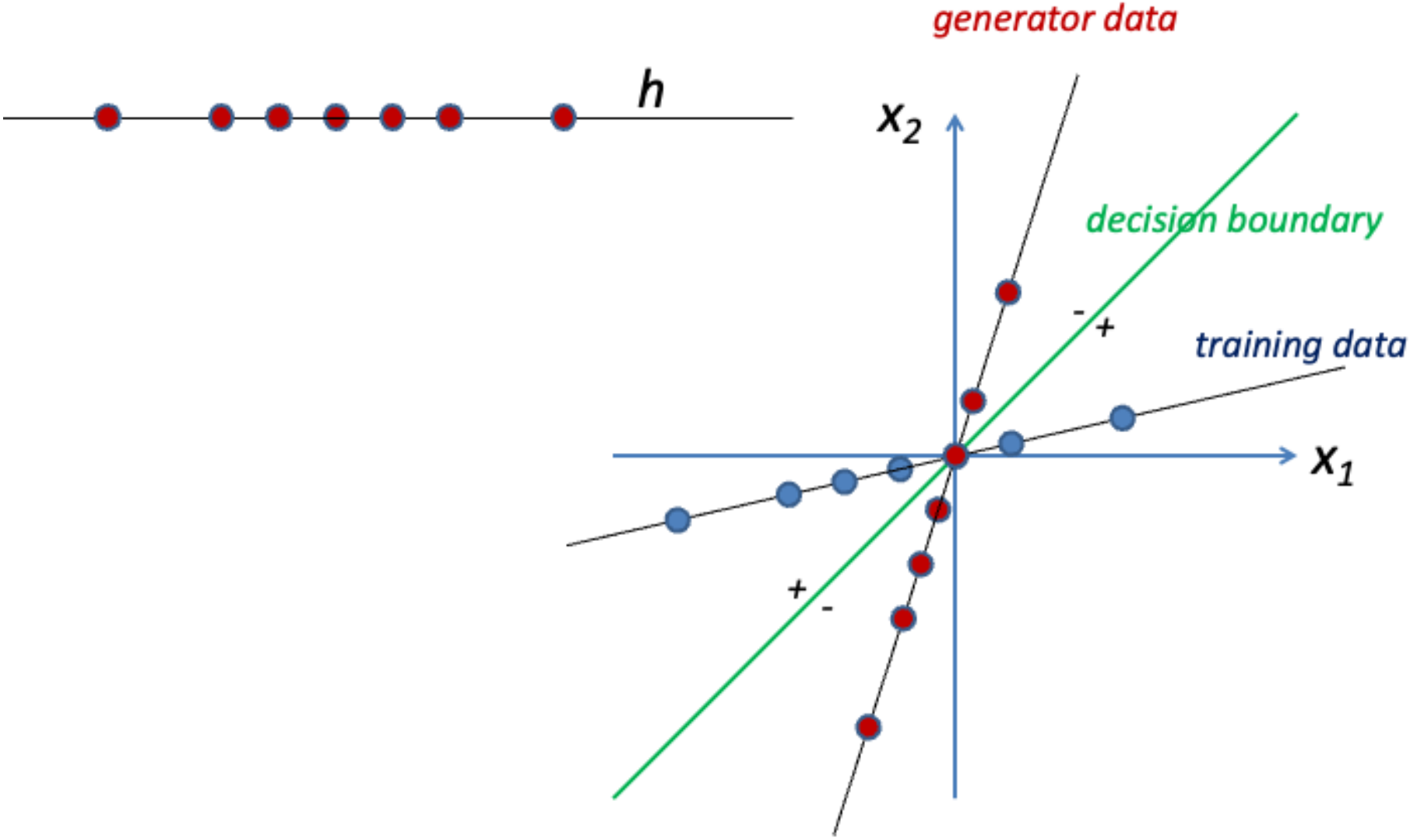
- The left sum says: all actual images should be classified by the discriminator with label 1
- The right sum says: all generated images should be classified by the discriminator with label 0
- From the perspective of the generator, the cost is a log-likelihood (to be maximized) and not a negative log-likelihood
- $g^{gen}(\mathbf{h}_{i'}, \mathbf{v})$ is a generated image with a random $\mathbf{h}_{i'}$

Minimax

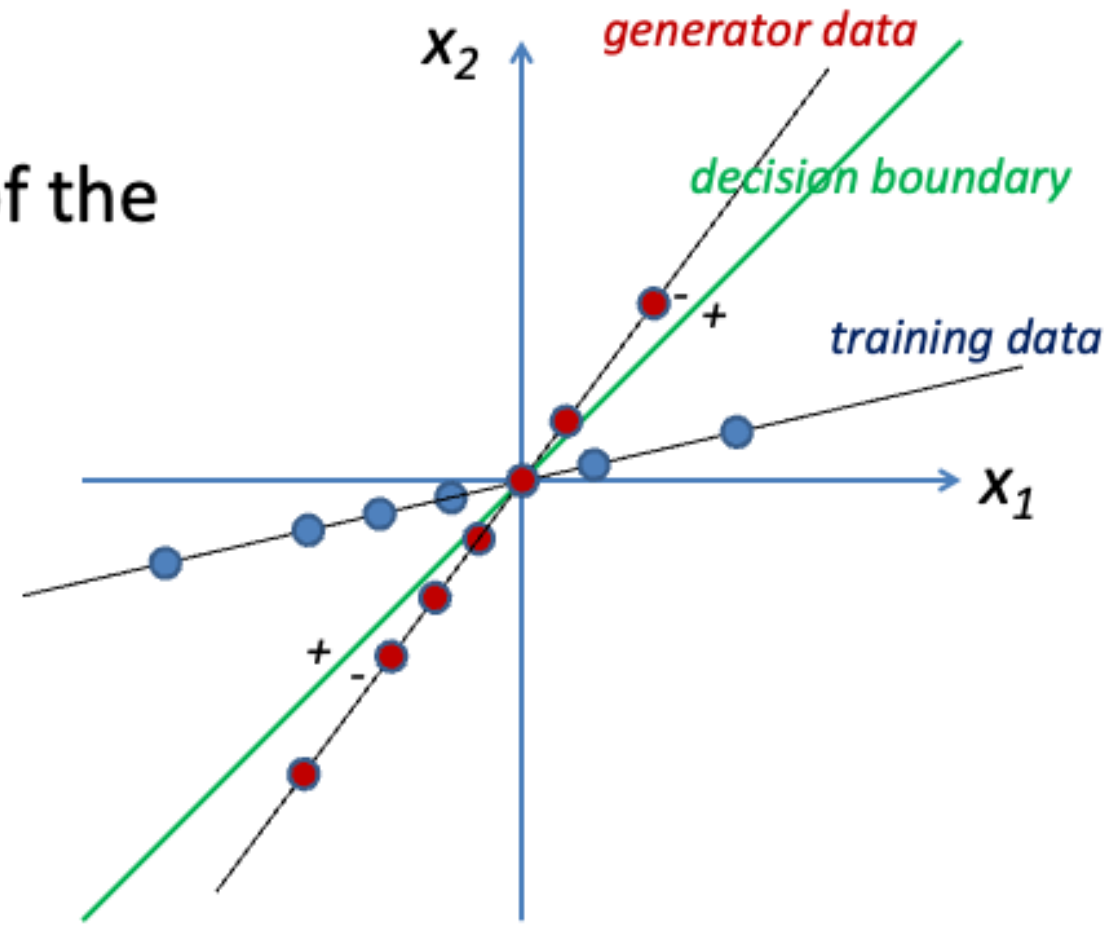
- This can be related to game theory: The solution for a zero-sum game is called a minimax solution, where the goal is to minimize the maximum cost for the player, here the generator. The **generator** wants to find the lowest cost, without knowing the actions of the **discriminator** (in two-player zero-sum games, the minimax solution is the same as the Nash equilibrium)

Illustration

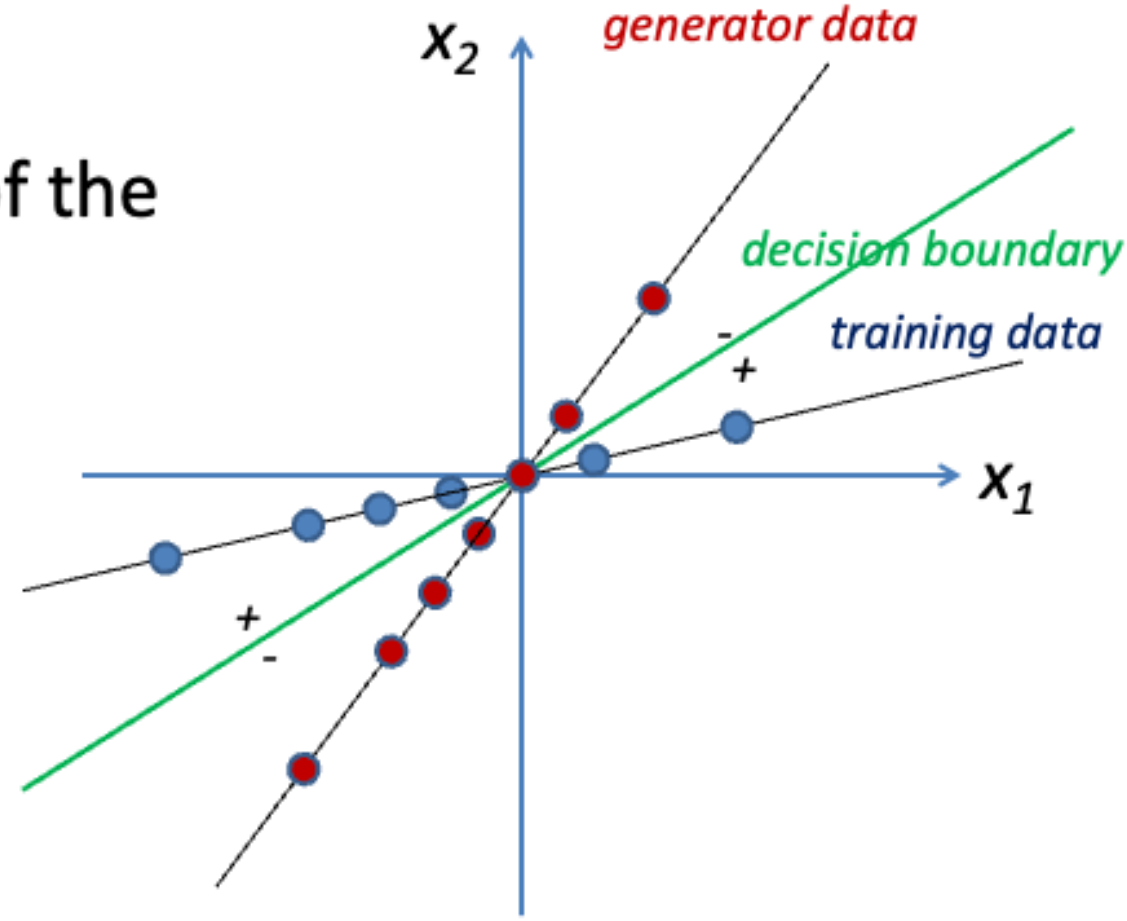
- Consider the following figure; h is one-dimensional Gaussian distributed: $P(h) = \mathcal{N}(h; 0, 1)$, $M_{hidd} = 1$
- The generator is $\mathbf{x} = h\mathbf{v}$, where $M = 2$; the data points are on a 1-D manifold in 2-D space; here: $v_1 = 0.2$, $v_2 = 0.98$
- The training data are generated similarly, but with $\mathbf{x} = h\mathbf{w}$ and $w_1 = 0.98$, $w_2 = 0.2$
- The discriminator is $y = \text{sig}(|x_1|w_1 + |x_2|w_2)$, with $w_1 = 0.71$, $w_2 = -0.71$
- After updating the generator, we might get $v_1 = 0.39$, $v_2 = 0.92$
- After updating the discriminator, we might get $w_1 = 0.67$, $w_2 = -0.74$


h
generator data
x_2
decision boundary
training data
x_1
-
+
+
-

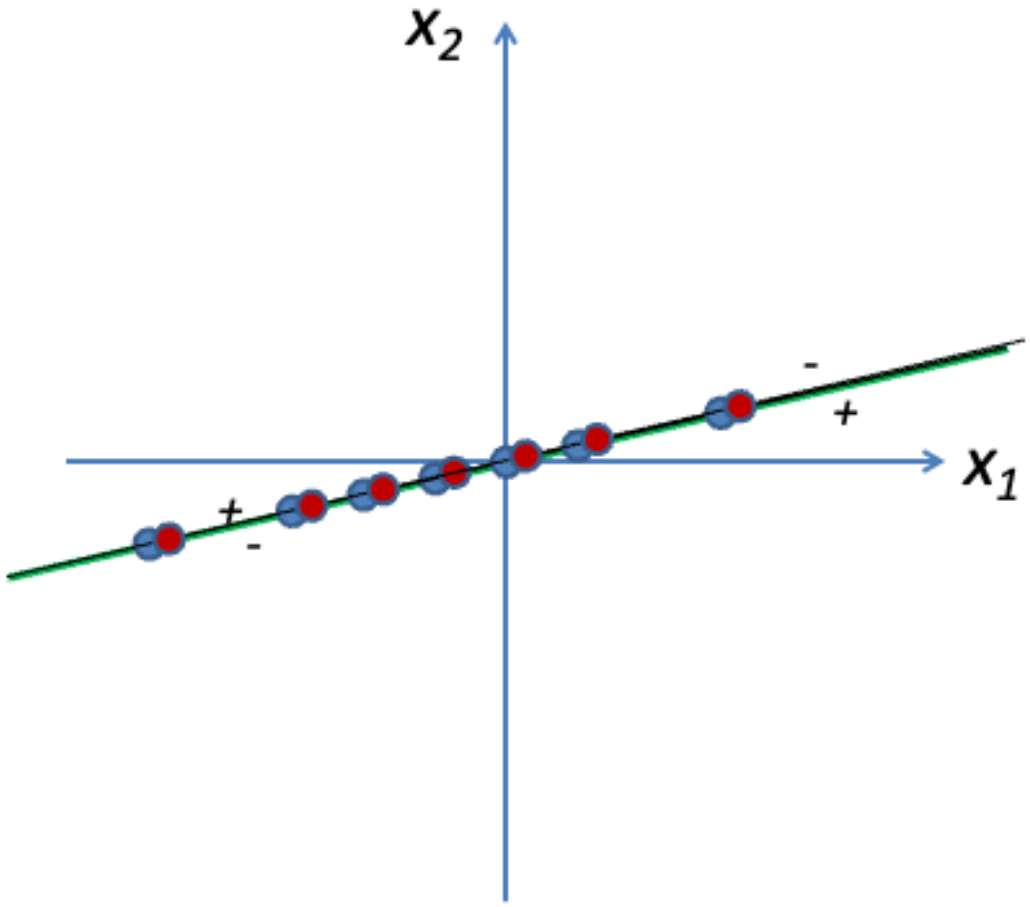
After update of the generator



After update of the discriminator



Convergence



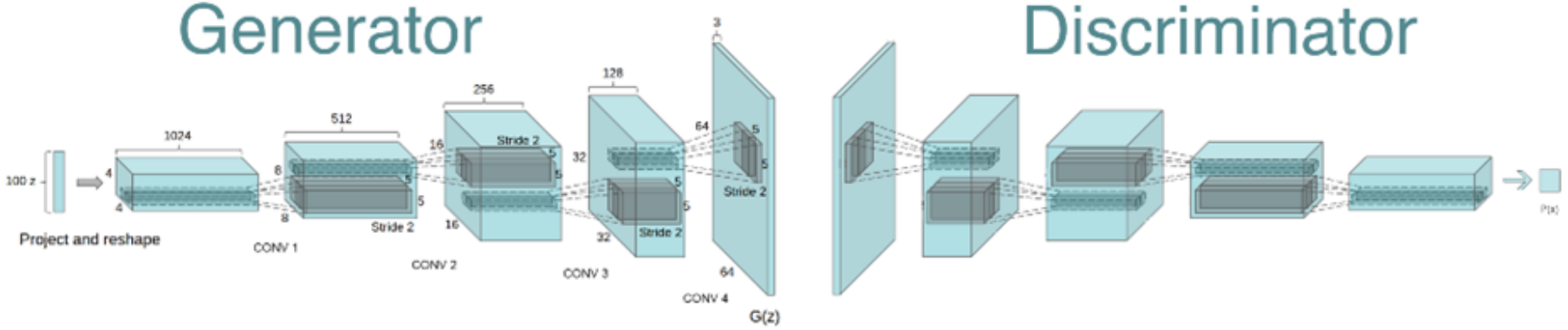
Applications

- For discriminant machine learning: Outputs of the convolutional layers of the discriminator can be used as a feature extractor, with simple linear models fitted on top of these features using a modest quantity of (image-label) pairs
- For discriminant machine learning: When labelled training data is in limited supply, adversarial training may also be used to synthesize more training samples

DCGAN

- If the data consists of images, the discriminator is a binary image classifier and the generator needs to generate images
- Deep Convolutional GAN (DCGAN): the generator and the discriminator contain convolutional layers and transposed convolution layers
- The transposed convolution layer is sometimes (incorrectly) called deconvolution layer (a deconvolution is really something different)
- The generation of sharp, photo realistic images with sharp edges and smooth regions is nontrivial!
- Radford et al. (shown below). $M_{hidd} = 100$; samples drawn from a uniform distribution (we refer to these as a code, or latent variables) and outputs an image (in this case $64 \times 64 \times 3$ images (3 RGB channels)

DCGAN


Generator
Discriminator
100 z
Project and reshape
1024
512
256
128
3
4
8
16
32
64
5
Stride 2
CONV 1
CONV 2
CONV 3
CONV 4
G(z)
P(x)

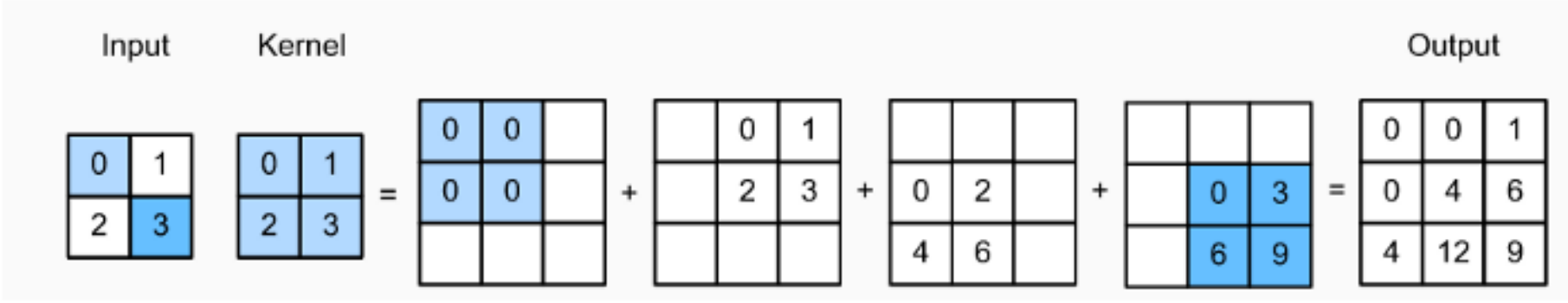
Convolution and Transposed Convolution

- With linear neurons, and the first hidden layer is

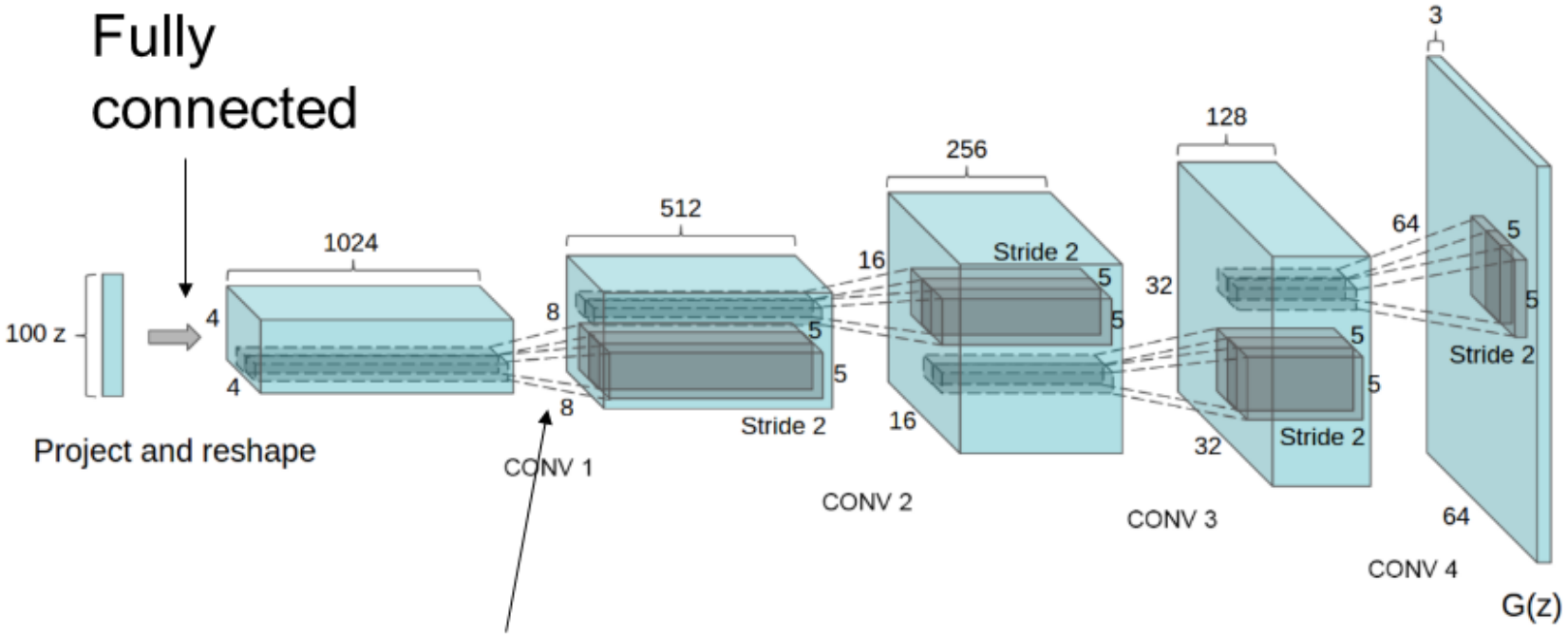
$$\mathbf{h} = \mathbf{W}^T \mathbf{x}$$

- If $\mathbf{W}$ is orthonormal, then $\mathbf{x} = (\mathbf{W})_{:,k}$ will only activate hidden unit k
- Similarly, if we only activate hidden unit k, and propagate towards the inputs, we will get again the same pattern, $\mathbf{x} = (\mathbf{W})_{:,k}$
- In a real convolutional NN, $\mathbf{W}$ is defined by the kernels and is not orthonormal; propagating back is known as a **transposed convolution** (which is also a convolution)
- Transposed convolution generates the characteristic input pattern of the hidden neurons, i.e. for hidden unit k, $\mathbf{x} = (\mathbf{W})_{:,k}$
- If $\mathbf{W}$ is not orthonormal, the transposed convolution is not a deconvolution; the latter would implement the inverse of the convolution and would require a very different connection matrix

Transposed Convolution (“Deconvolution”)



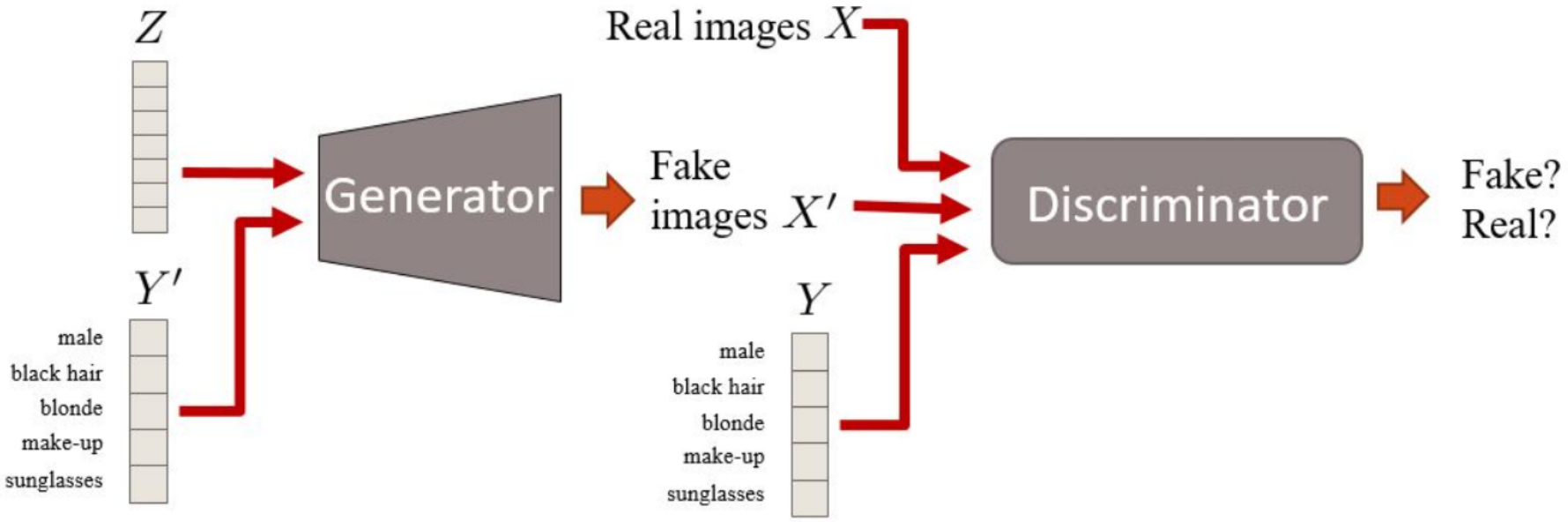
DCGAN: Generator



(the general scheme is :
transposed convolution→ batch normal→leaky ReLU)

cGAN

- Consider that class/attribute labels are available; in a normal GAN, one would ignore them; another extreme approach would be to train a different GAN model for each class; cGAN and InfoGans are compromizes (related to the idea of Conditional Variational Autoencoders)
- Conditional GAN (cGAN): An additional input to the generator and the discriminator is the class/attribute label



Overview of a conditional GAN with face attributes information.

cGAN Applications

- The attributes can be quite rich, e.g., images, sketches of images
- cGANs: GAN architecture to **synthesize images from text descriptions**, which one might describe as reverse captioning. For example, given a text caption of a bird such as "white with some black on its head and wings and a long orange beak", the trained GAN can generate several plausible images that match the description
- cGANs not only allow us to synthesize novel samples with specific attributes, they also allow us to develop **tools for intuitively editing images** - for example editing the hair style of a person in an image, making them wear glasses or making them look younger
- cGANs are well suited for **translating an input image into an output image**, which is a recurring theme in computer graphics, image processing, and computer vision

Unpaired Image-to-Image Translation

- Example task: turn horses in images into zebras
- One could train a generator *Generator A2B* with horse images as inputs and the corresponding zebra images as output; this would not work, since we do not have matching zebra images
- But consider that we train a second generator *Generator B2A* which has zebra images as inputs and generates horse images
- Now we can train two autoencoders

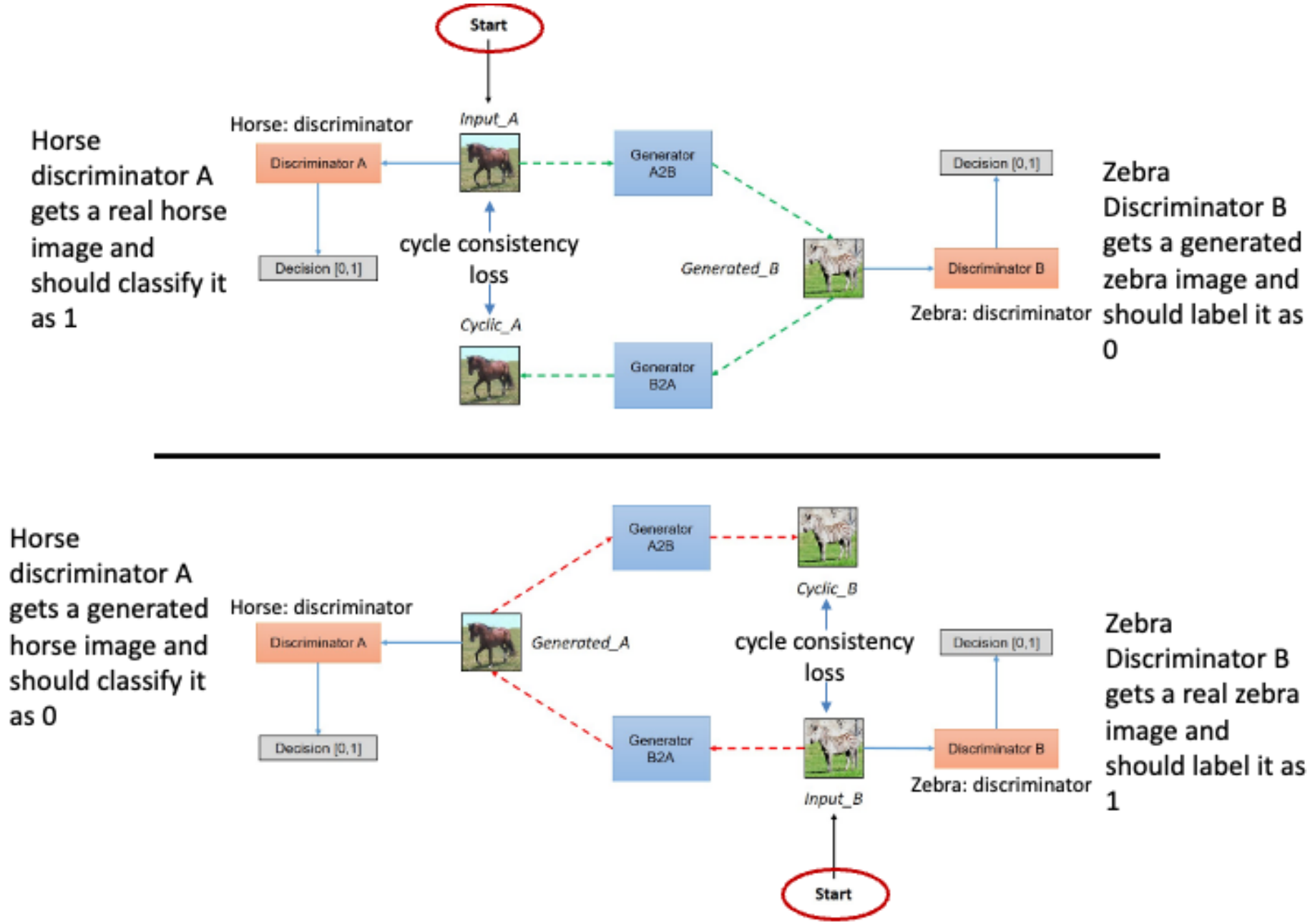
$$\hat{\mathbf{x}}_{horse} = g_{B2A}(g_{A2B}(\mathbf{x}_{horse}))$$

$$\hat{\mathbf{x}}_{zebra} = g_{A2B}(g_{B2A}(\mathbf{x}_{zebra}))$$

- These constraints are enforced using the *cycle consistency loss*, $\|\mathbf{x}_{horse} - \hat{\mathbf{x}}_{horse}\|^2$ and $\|\mathbf{x}_{zebra} - \hat{\mathbf{x}}_{zebra}\|^2$

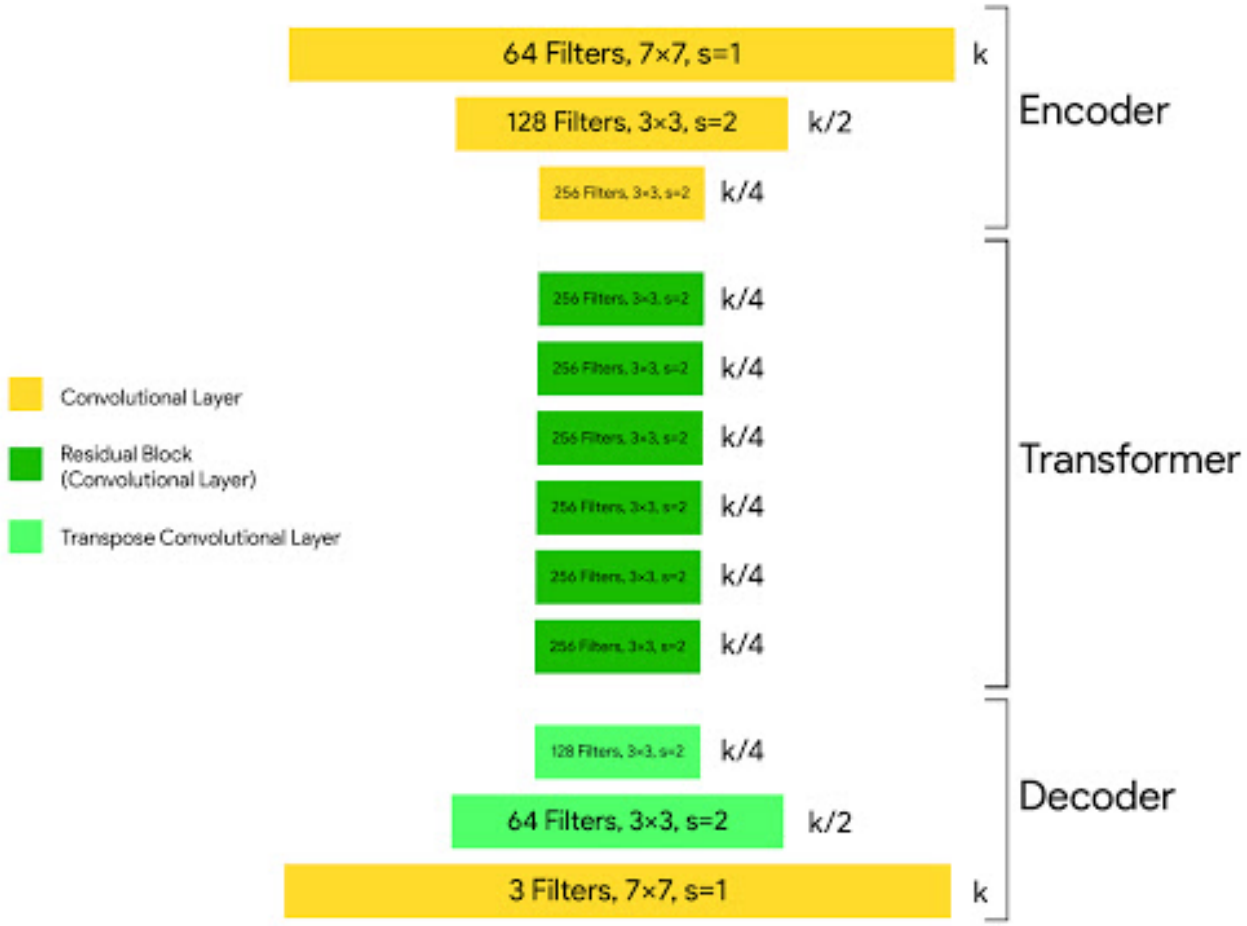
CycleGAN

- CycleGAN does exactly that
- CycleGAN adds two discriminators, trained with the *adversarial loss*
- *discriminator*$_A$ tries to discriminate real horses from generated horses
- *discriminator*$_B$ tries to discriminate real zebras from generated zebras
- If the generated horses and zebras are perfect, both fail to discriminate
- Both the *cycle consistency loss* and the *adversarial loss* are used in training
- Note that the random seeds here are the images!



Simplified view of CycleGAN architecture

Generator for CyleGAN



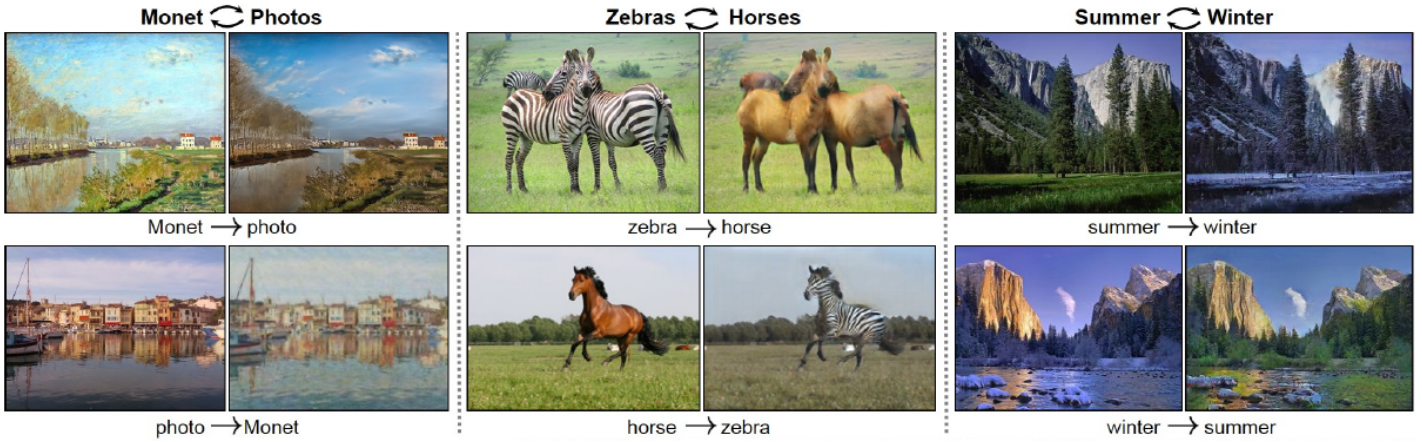
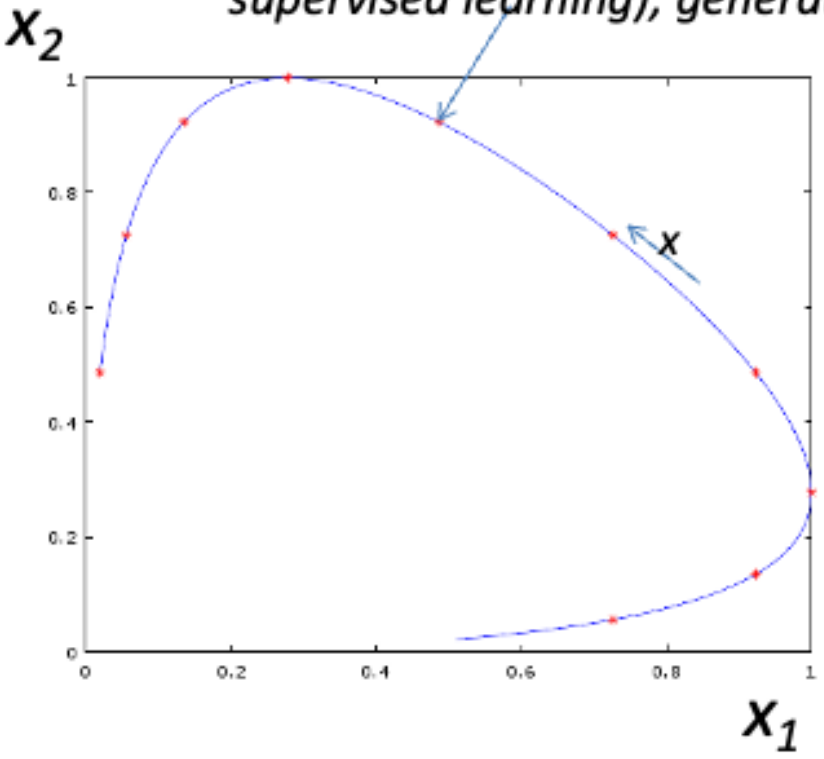


Fig. 8. CycleGAN model learns image to image translations between two unordered image collections. Shown here are the examples of bi-directional image mappings: Monet paintings to landscape photos, zebras to horses, and summer to winter photos in Yosemite park. Figure reproduced from [4].

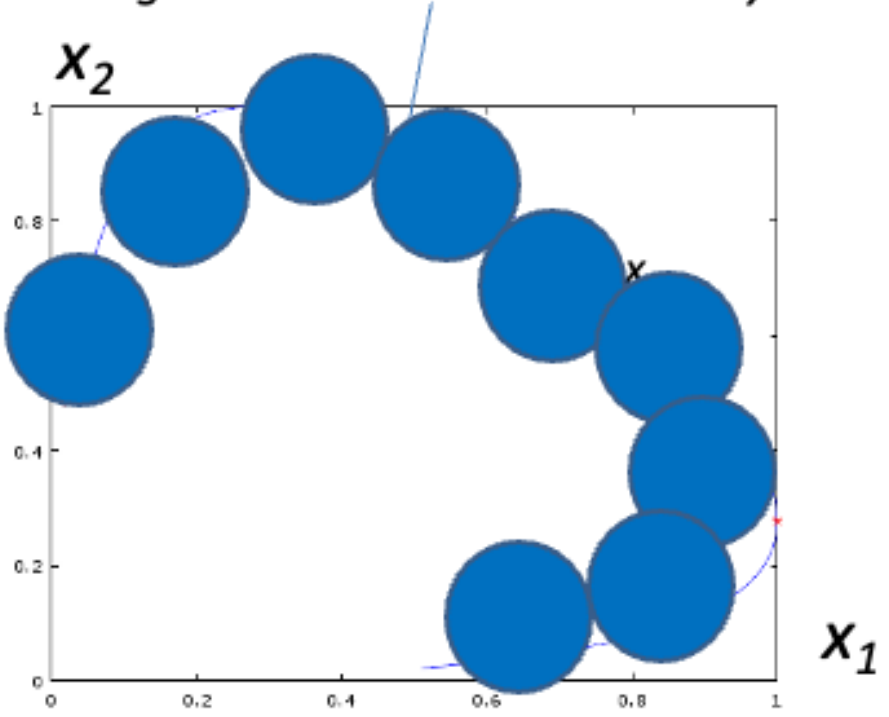
Why Not Use Classical Approaches?

- Classically, one would start with a probabilistic model $P(\mathbf{x}; \mathbf{w})$ and determine parameter values that provide a good fit (maximum likelihood)
- Examples for continuous data: Gaussian distribution, mixture of Gaussian distributions
- These models permit the specification of the probability density for a new data point and one can sample from these distributions (typically by transforming samples for a normal or uniform distribution; this would be the generator here)
- These approaches typically work well for low dimensional distributions, but not for image distributions with 256×256 pixels and where data is essentially on a manifold

AE and GANs can learn the manifold distribution (see semi-supervised learning); generated face images are sharp and realistic



Classical models are too rough and cannot model the manifold very well: generated face images are nonrealistic and blurry



Related Approaches and Applications

- The GAN generator generates data $\mathbf{x}$ but we cannot easily evaluate $P(\mathbf{x})$
- In many applications it is possible to generate data but one cannot generate a likelihood function (likelihood free methods)
- Moment matching is one approach to evaluate the quality of the simulation
- Optimization: in physics and other fields it is sometimes easy to evaluate the cost (e.g., energy) of a solution and the problem is to find good proposal solutions $\mathbf{x}$ with a low $\text{COST}(\mathbf{x})$ (combinatorical optimization)

A First Look at DALLE and Diffusion Models

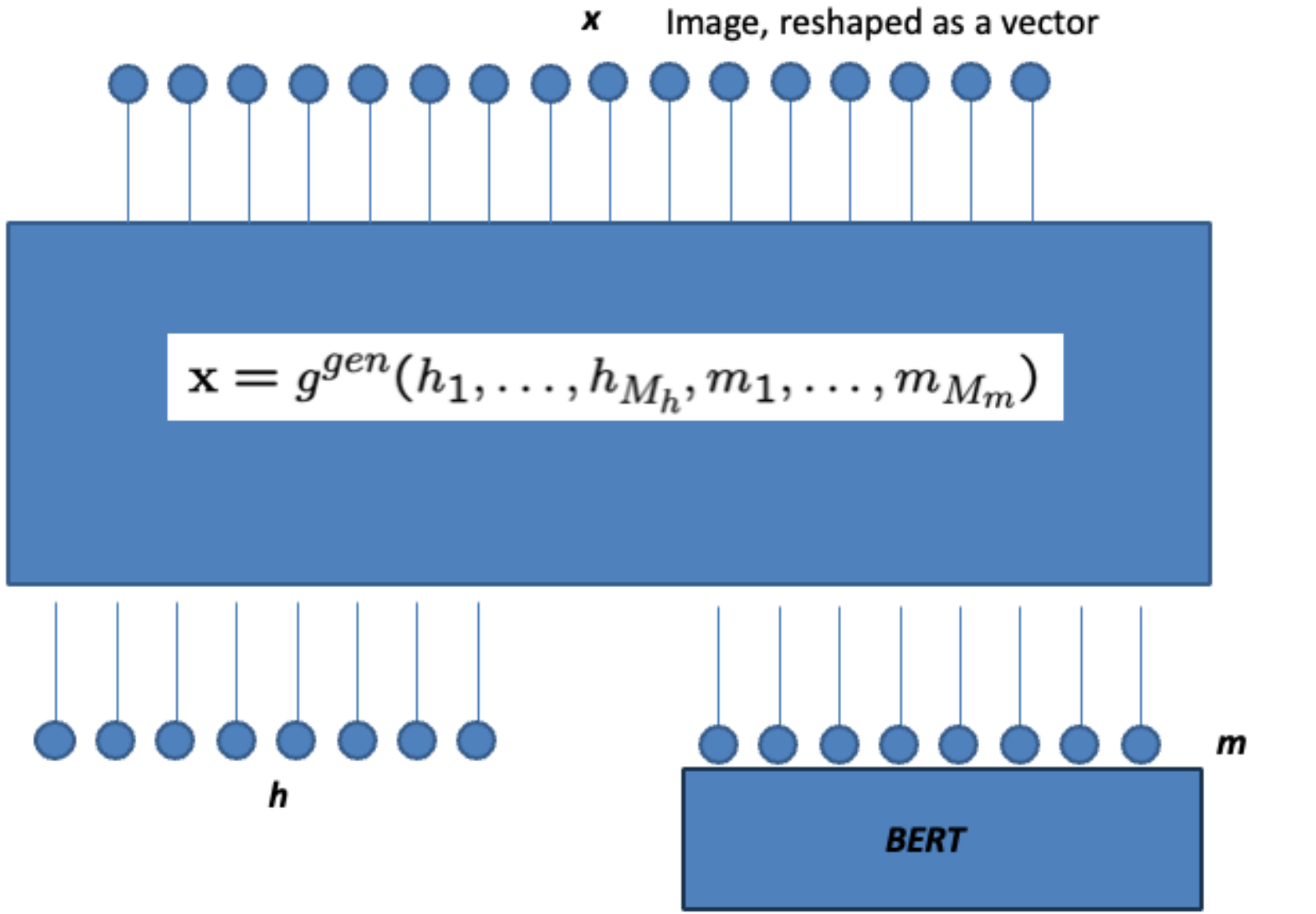
DALLE-2: Basic Idea

- Conditional (attribute specific) models

$$\mathbf{x} = g_d(h_1, \ldots, h_{M_{hidd}}, m_1, \ldots, m_{M_m})$$

- $m_1, \ldots, m_{M_m}$ is the output representation layer of a language encoder, e.g., BERT
- Challenge: $m_1, \ldots, m_{M_m}$ should be understood by the generator
- Example text input: "a corgi playing a flame throwing trumpet" (there is no training image even close)
- Hierarchical Text-Conditional Image Generation with CLIP Latents. Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, Mark Chen (DALLE-2-Paper; OpenAI)

An embedding vector as input to the generator

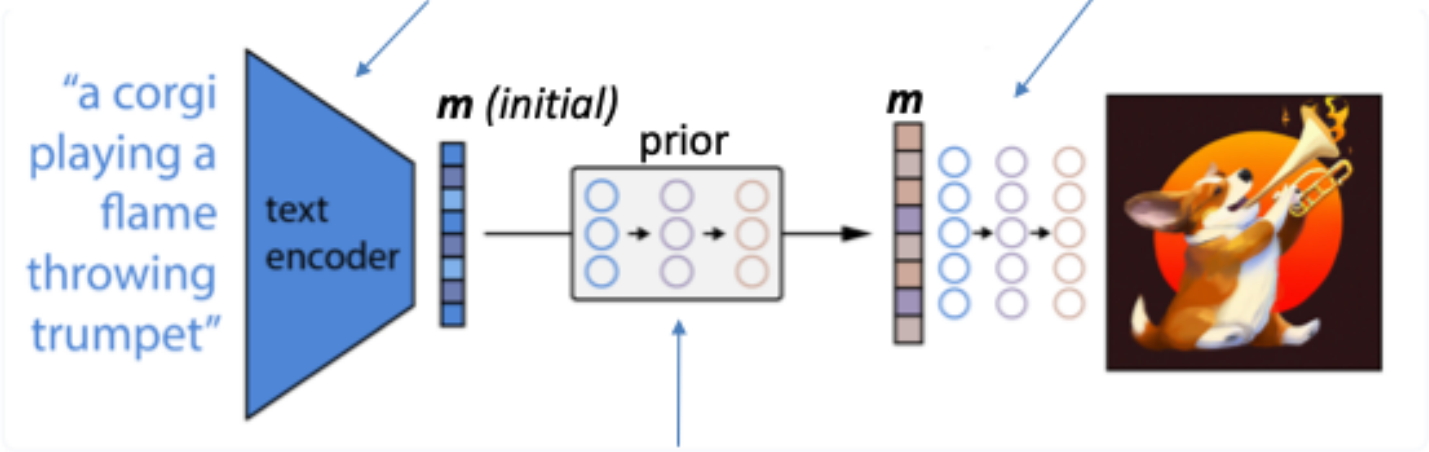


DALLE-2

A diffusion model which starts with a random image (x_T=h) which is iteratively updated where the updates depend on the embedding vector:

x_{t-1}= f(x_t, m)

A BERT model trained with contrastive learning based on image/image annotations from the web

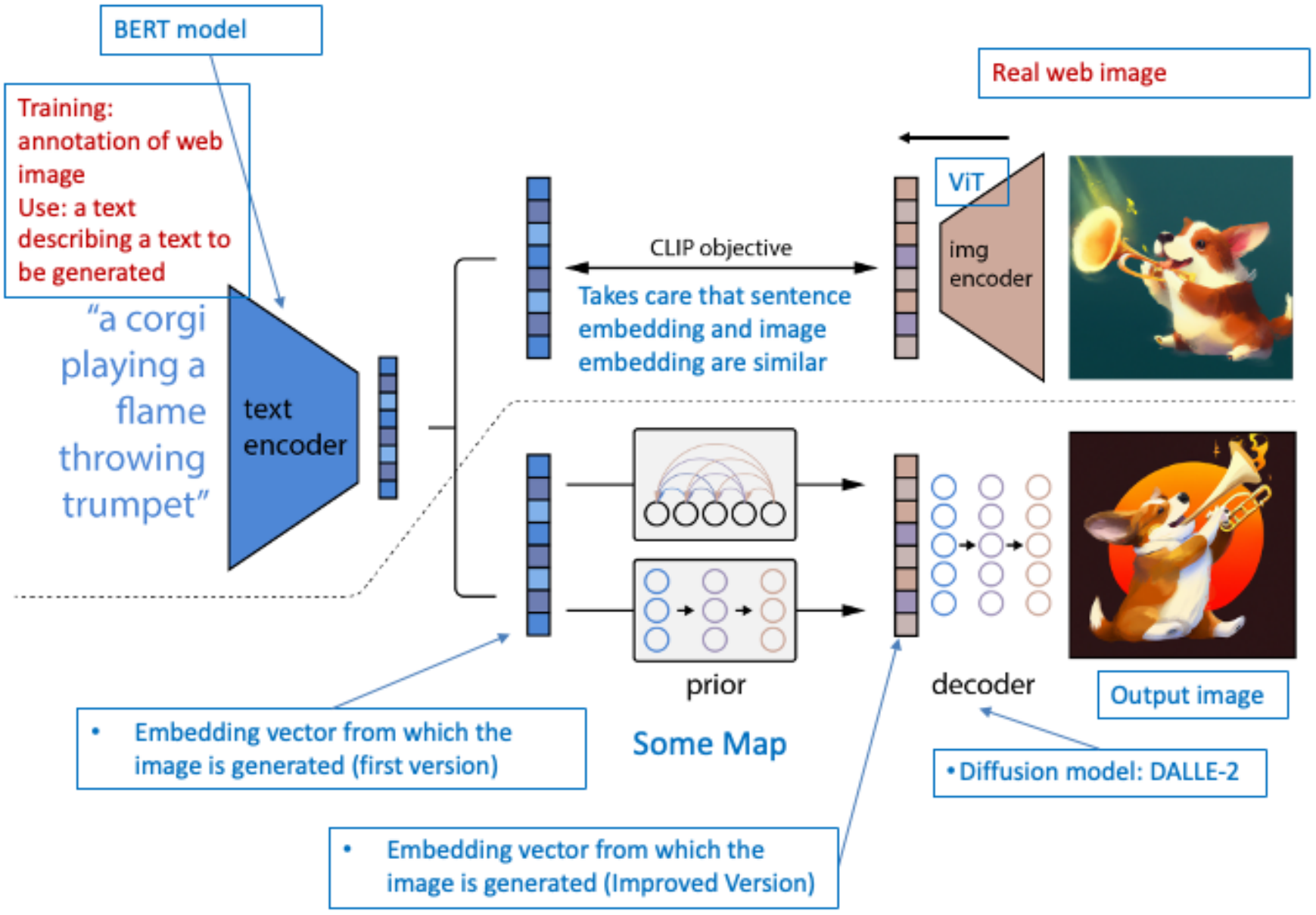


Transfers to an embedding vector used by the diffusion model

1: a text prompt is input into a text encoder that is trained to map the prompt to a representation space.
2: a model called the prior maps the text encoding to a corresponding image encoding that captures the semantic information of the prompt contained in the text encoding.
3: an image decoder stochastically generates an image which is a visual manifestation of this semantic information.

Contrastive Pretraining

- DALLE-2 is pretrained using contrastive learning in CLIP


BERT model
Training:
annotation of web image
Use: a text describing a text to be generated
"a corgi playing a flame throwing trumpet"
text encoder
CLIP objective
Takes care that sentence embedding and image embedding are similar
Real web image
ViT
img encoder
prior
decoder
Output image
Some Map
• Embedding vector from which the image is generated (first version)
• Embedding vector from which the image is generated (Improved Version)
• Diffusion model: DALLE-2

Contrastive Pretraining Using Clip

- The BERT model is pretrained on image annotations from the web; Contrastive learning enforces similar embeddings for matching text-image pairs
- Cost function, e.g.,

$$\text{softmax}((\mathbf{h}_i^{text})^T \mathbf{h}_i^{image})$$

  $\mathbf{h}_i^{text}$ is the text embedding vector generated from BERT and $\mathbf{h}_i^{image}$ is the image embedding vector generated, e.g., from a vision transformer (ViT)
- Often cosine distance is used instead of inner product
- Use embedding vector pairs (image-text) and mismatches of texts and mismatches of images (Batch size for the softmax: e.g., 2000)

Contrastive Pre-Training using CLIP

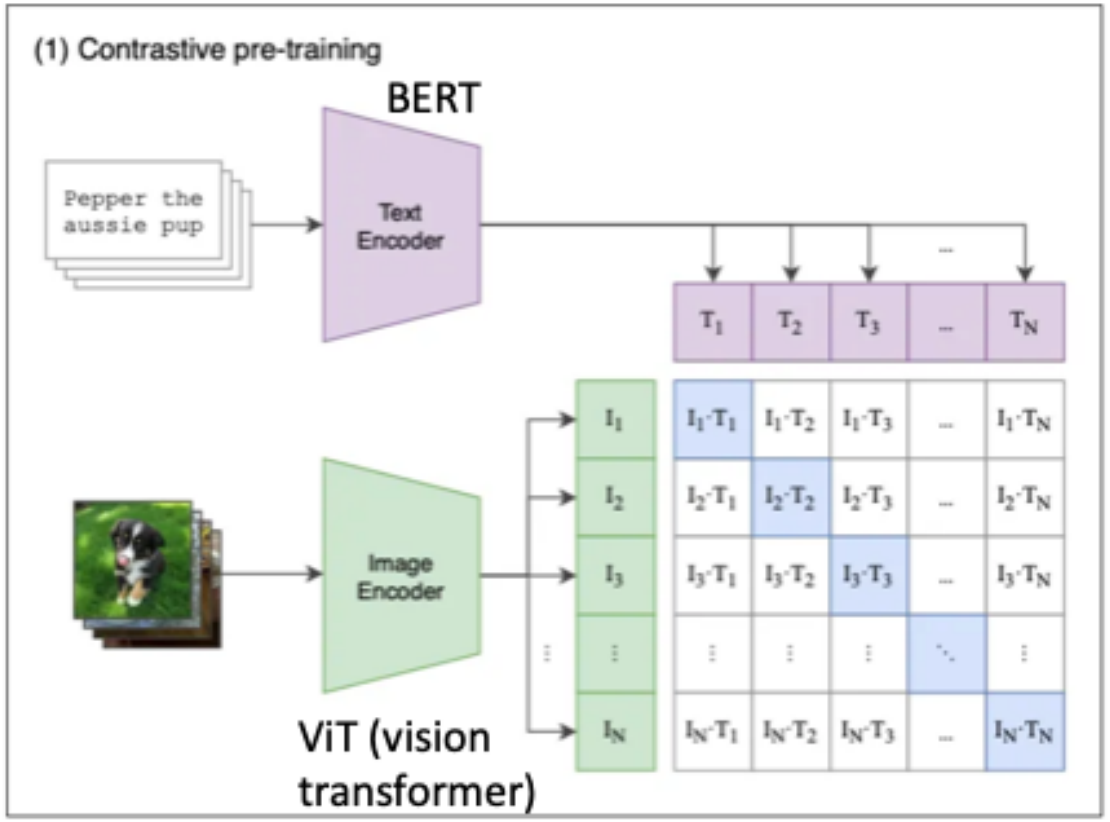


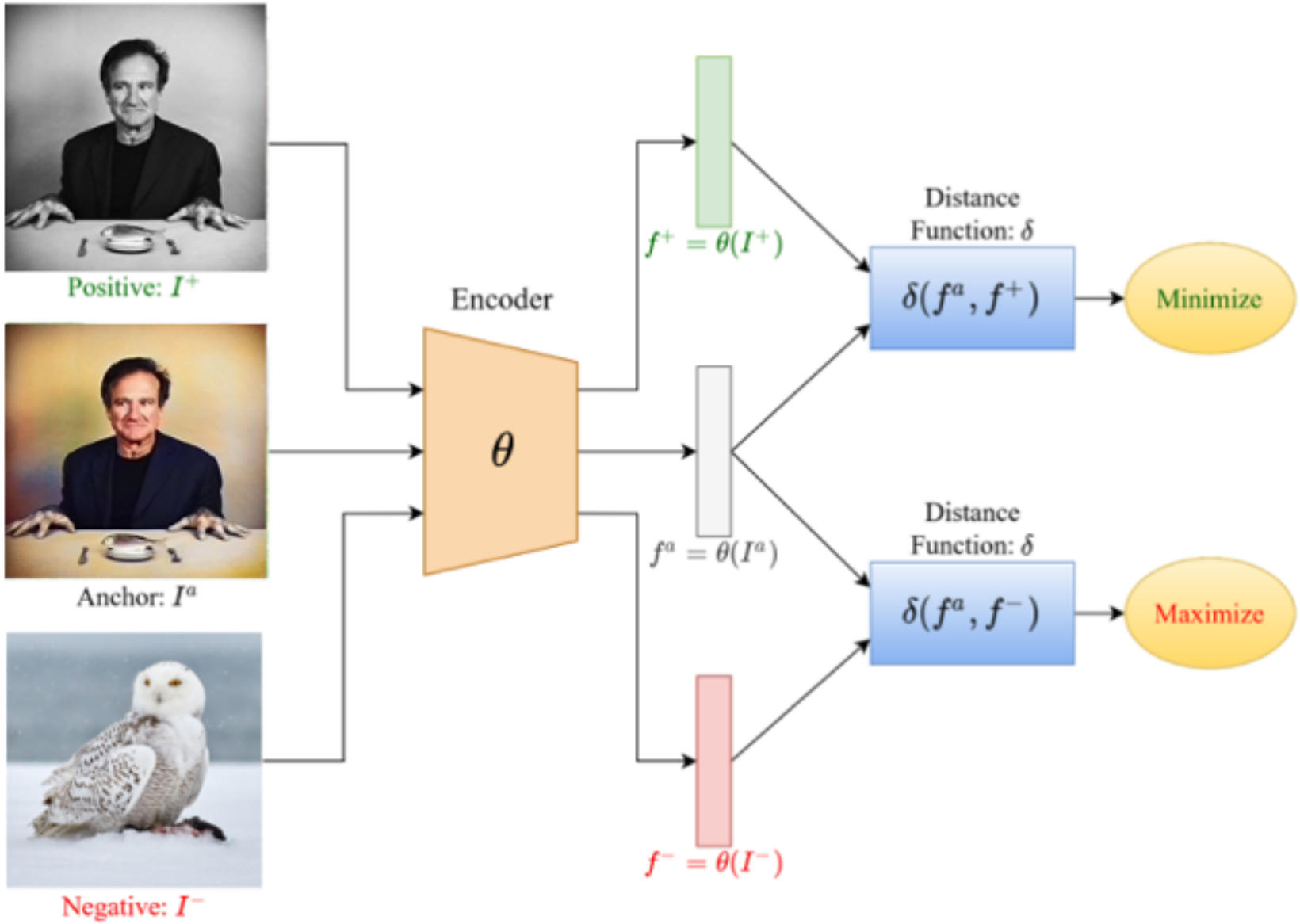
Figure 1: Contrastive Pre-training step of CLIP (Source)

Contrastive Pretraining for Learning Embeddings in Object Recognition

- Pretraining of embeddings using unlabelled data
- We select a training image i and, temporarily, $\mathbf{z} = \mathbf{h}(\mathbf{x}_i)$ is its current embedding (last hidden layer of a DNN)
- We disturb image $\mathbf{x}_i$ (rotation, scaling, adding noise,...) and generate image $\tilde{\mathbf{x}}_i$; we temporarily replace image i with this generated imagein the training data, and in particular, $\mathbf{h}_i \leftarrow \mathbf{h}(\tilde{\mathbf{x}}_i)$
- The cost function for image i then is

$$-\log \frac{\exp \mathbf{h}_i^T \mathbf{z}}{\sum_j \exp \mathbf{h}_j^T \mathbf{z}}$$

- Thus, the embedding of the original image i, i.e., $\mathbf{z}$, should be close to its disturbed version, i.e., $\mathbf{h}_i$, (they show the same items), but distant to the embeddings of all other images


Positive: I^+
Anchor: I^a
Negative: I^-
Encoder
θ
$f^+ = \theta(I^+)$
$f^a = \theta(I^a)$
$f^- = \theta(I^-)$
Distance Function: δ
$\delta(f^a, f^+)$
Minimize
Distance Function: δ
$\delta(f^a, f^-)$
Maximize