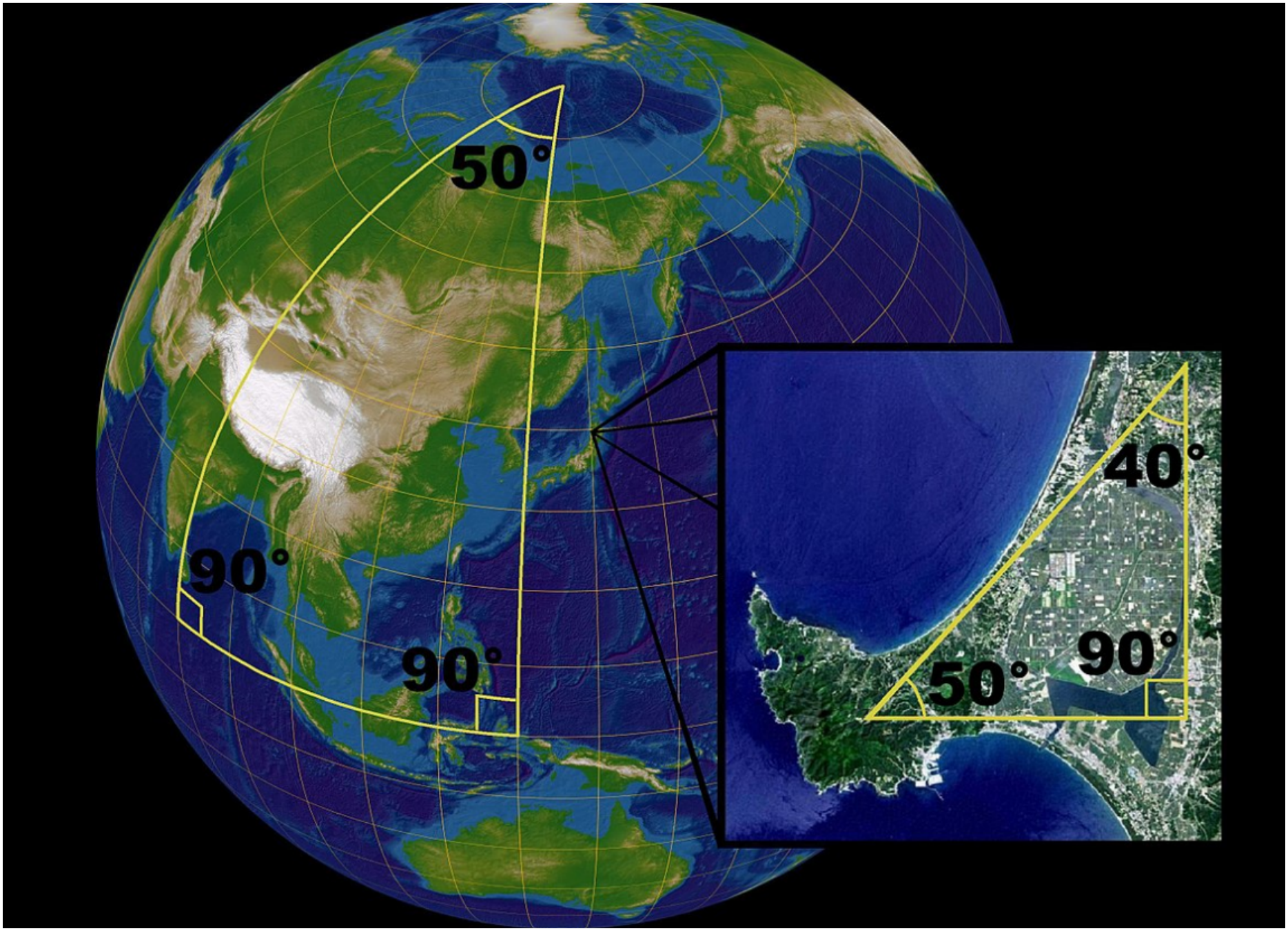
Manifolds, Autoencoders and Generative Adversarial Networks

Volker Tresp
Summer 2019

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 $\mathbf{h}^{nat} \in \mathbb{R}^{M_h}$, 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

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

- 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

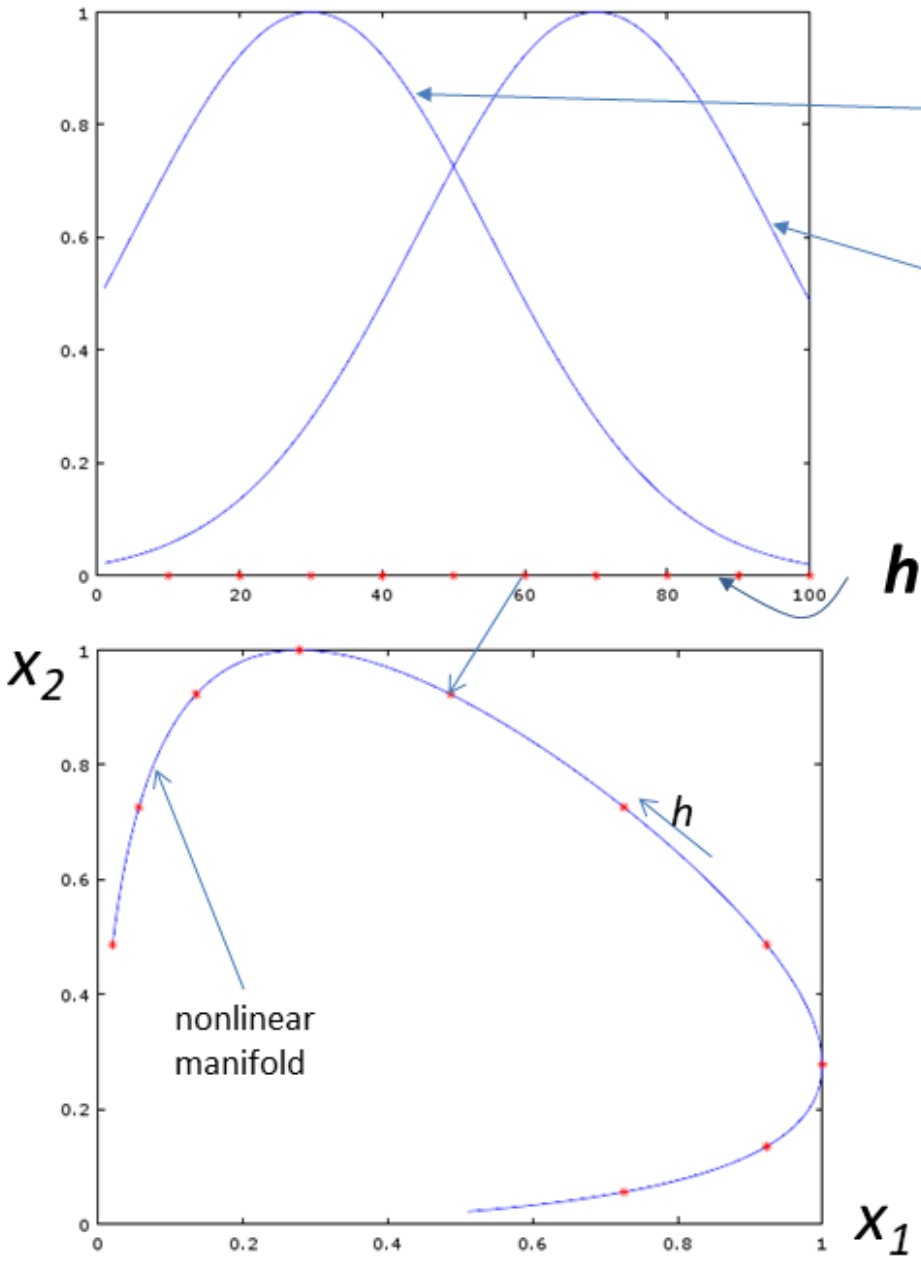
Manifold in Machine Learning

- In Machine Learning: in the observed M-dimensional input space, the data is distributed on an M_h-dimensional manifold

$$\{\mathbf{x} \in \mathbb{R}^M : \exists \mathbf{h} \in \mathbb{R}^{M_h} \ \ s.th. \ \ \mathbf{x} = g^{gen}(\mathbf{h})\}$$

  where $g^{gen}(\cdot)$ is smooth

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



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

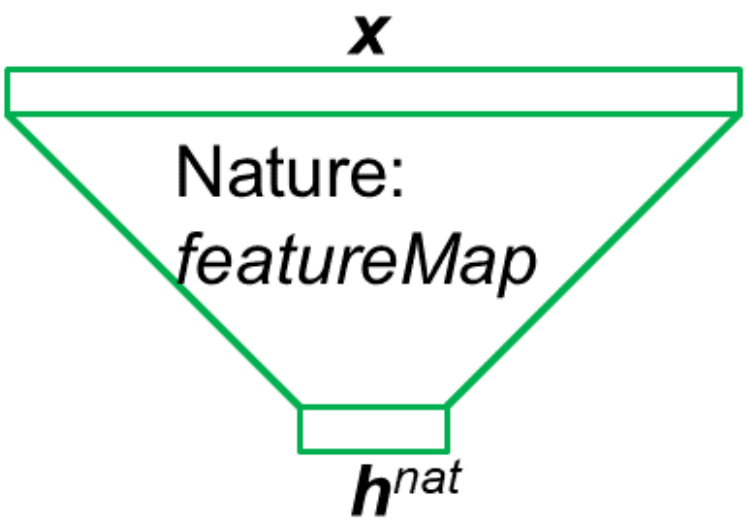
$x_2 = featureMap_2(\boldsymbol{h}^{nat})$

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

The data is available in feature space $\boldsymbol{x}$
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

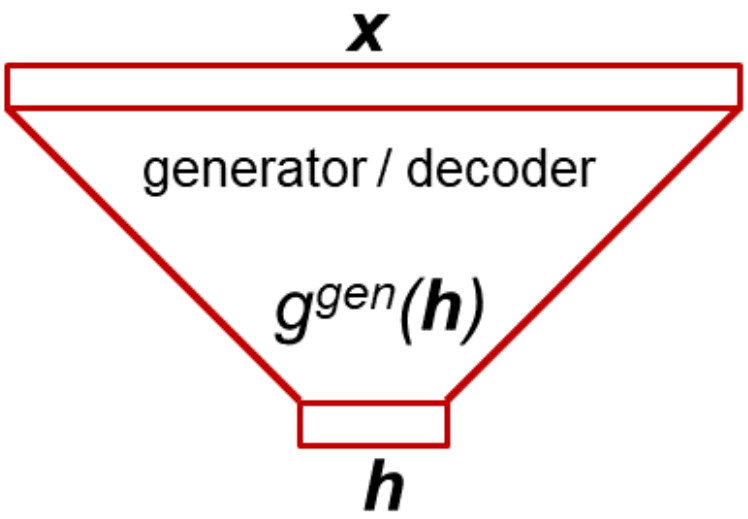
Learning a Generator / Decoder

- Example: $\mathbf{x}$ represents a face; let's assume nature selects $\mathbf{h}^{nat}$ from an M_h-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

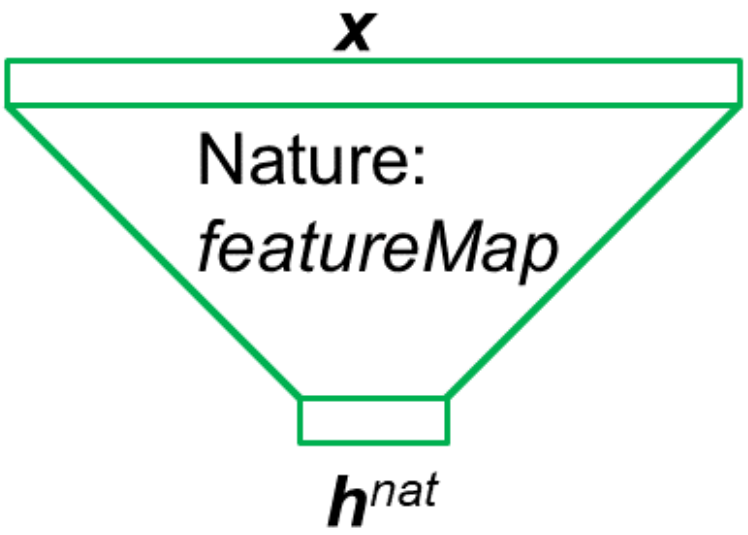
$$\mathbf{x} = g^{gen}(\mathbf{h})$$

(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 called a decoder $g_d(\cdot)$



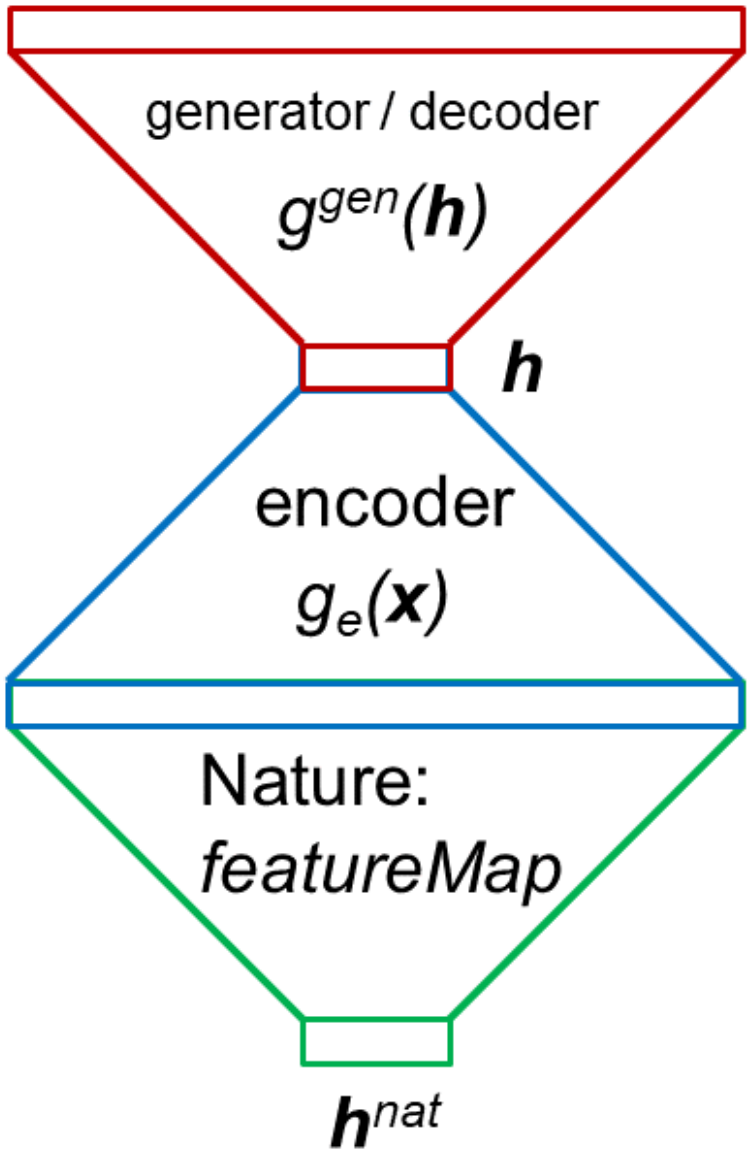
- A generator tries to copy this generative process
- If the ***x*** represent images, one can now generate new natural looking images



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 network

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



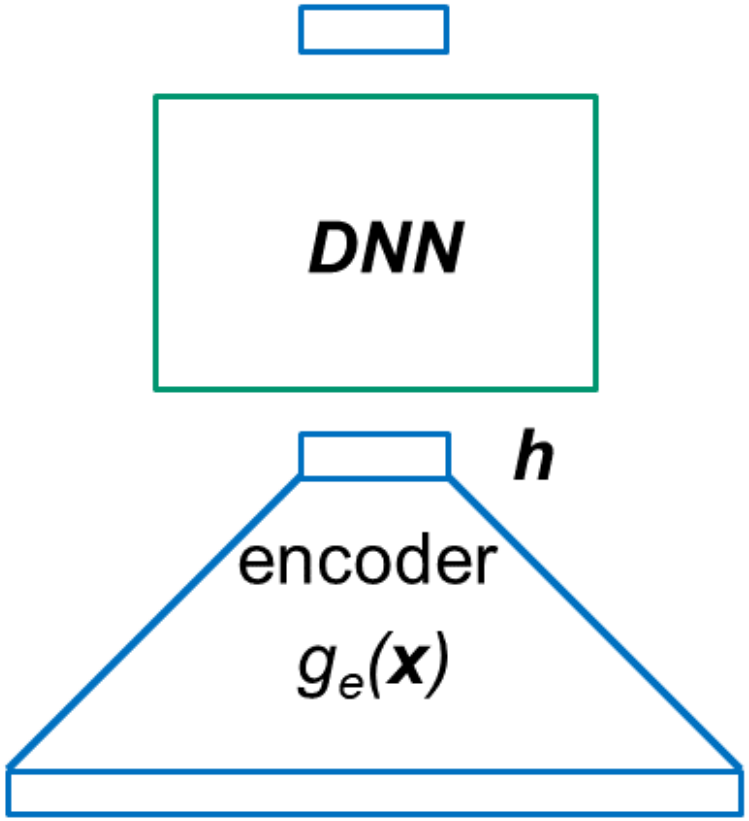
$\boldsymbol{x}$

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)

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:

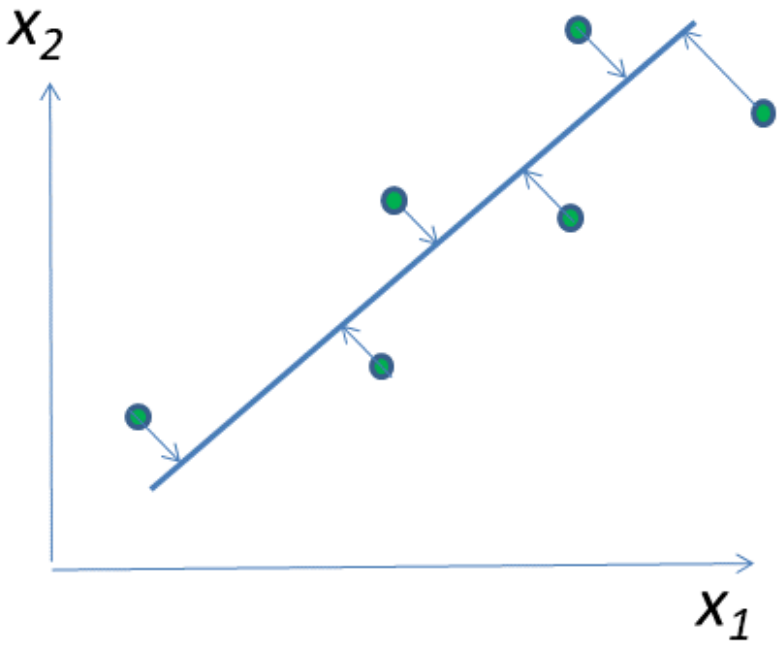
$$\mathbf{h} = g_e(\mathbf{x}) = \mathbf{V}_h^T \mathbf{x}$$

  Decoder:

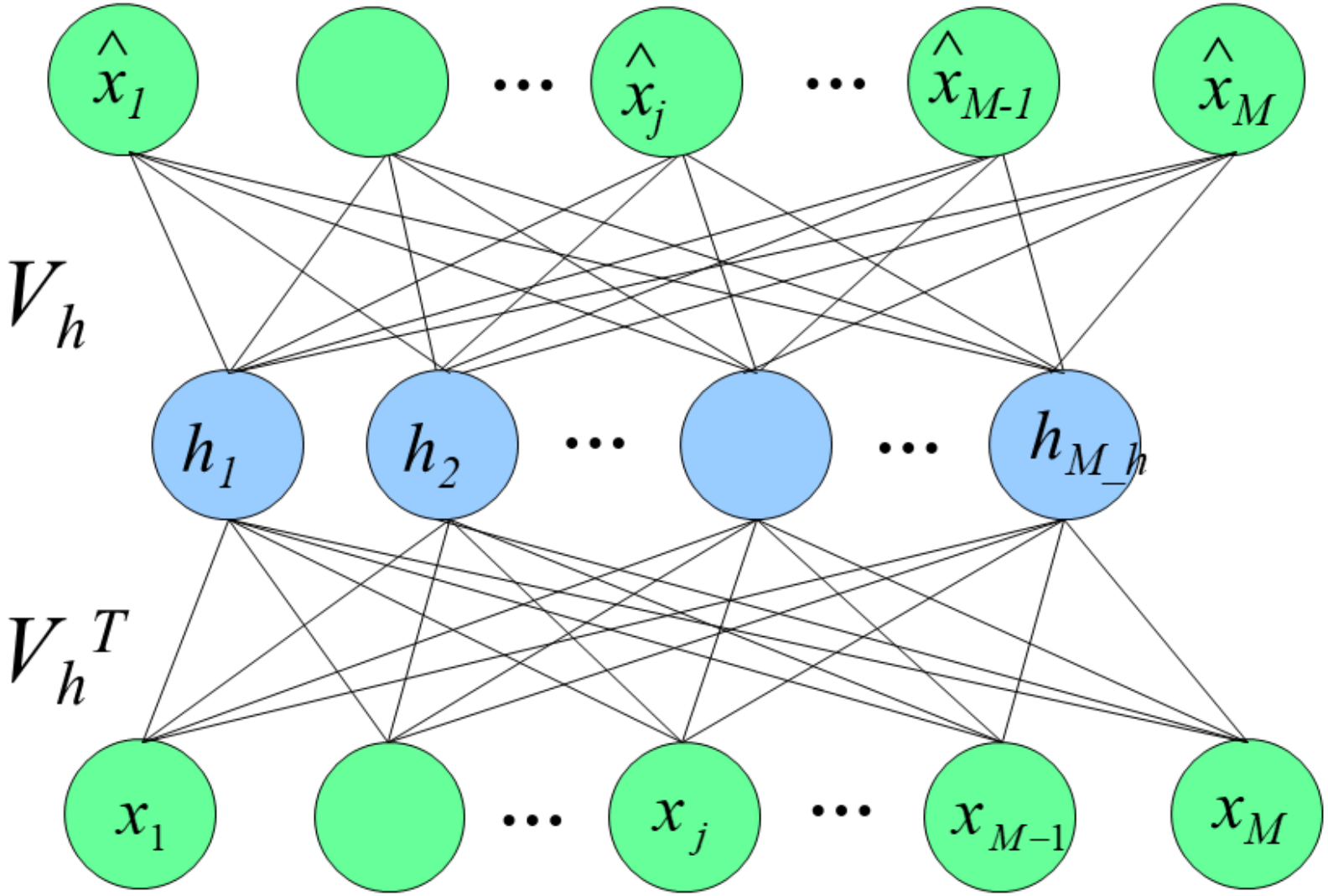
$$\hat{\mathbf{x}} = g_d(\mathbf{h}) = \mathbf{V}_h \mathbf{h} = \mathbf{V}_h \mathbf{V}_h^T \mathbf{x}$$

- The $\mathbf{V}_h$ are the first M_h columns of $\mathbf{V}$, where $\mathbf{V}$ is obtained from the SVD

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



The manifold is a linear subspace

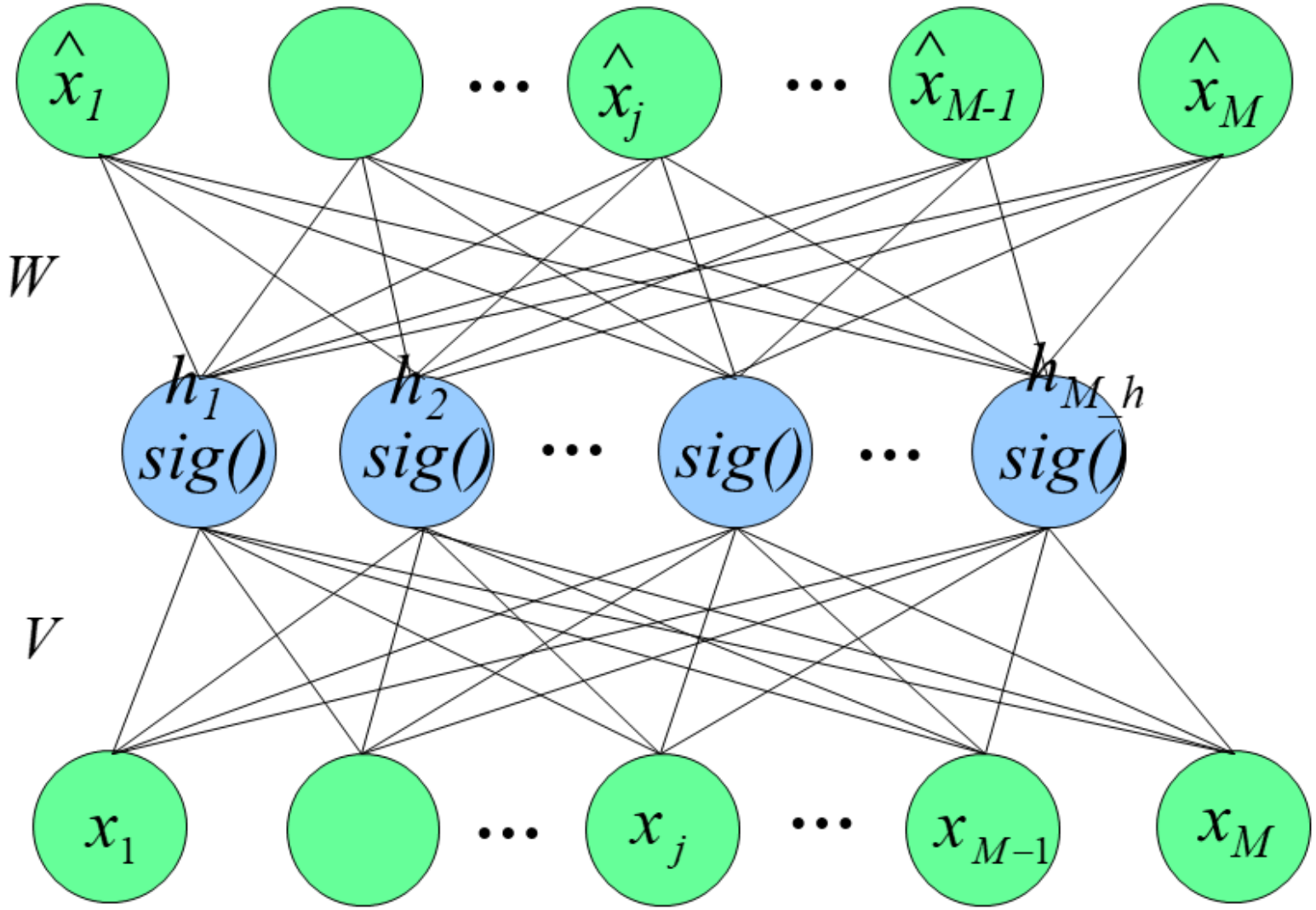

$\hat{x}_1$
$\hat{x}_j$
$\hat{x}_{M-1}$
$\hat{x}_M$
V_h
h_1
h_2
h_{M_h}
V_h^T
x_1
x_j
x_{M-1}
x_M

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

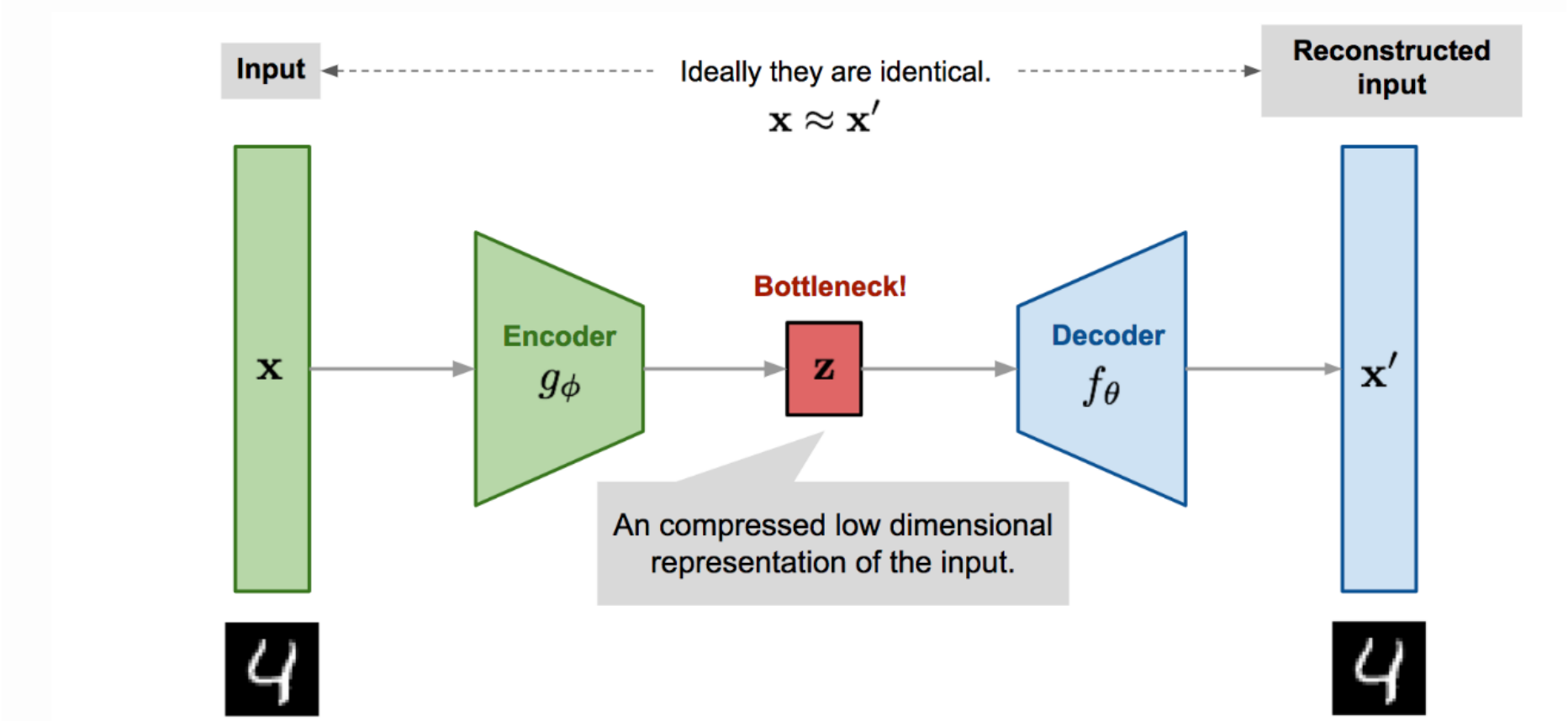
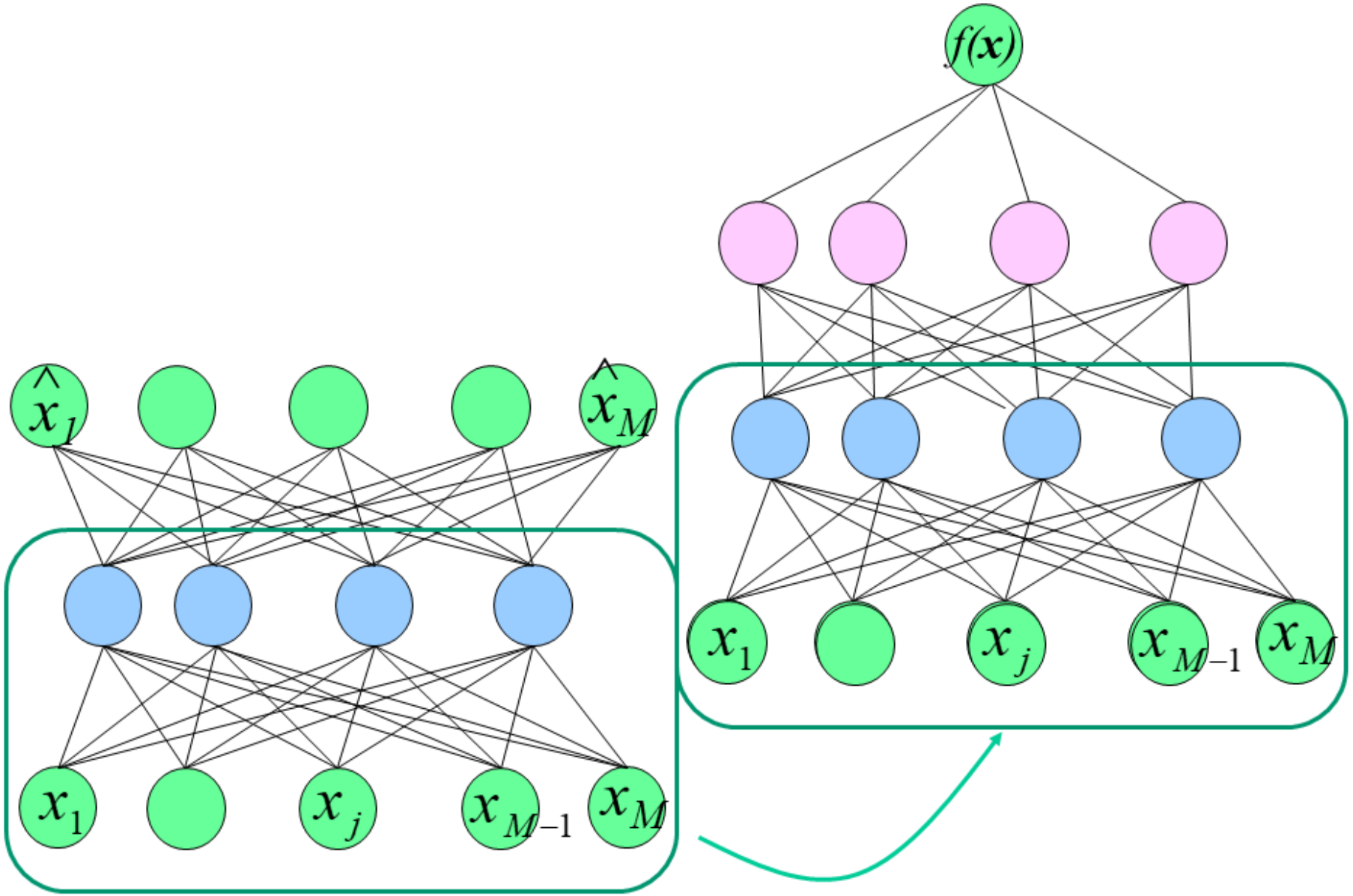


Fig. 1. Illustration of autoencoder model architecture.

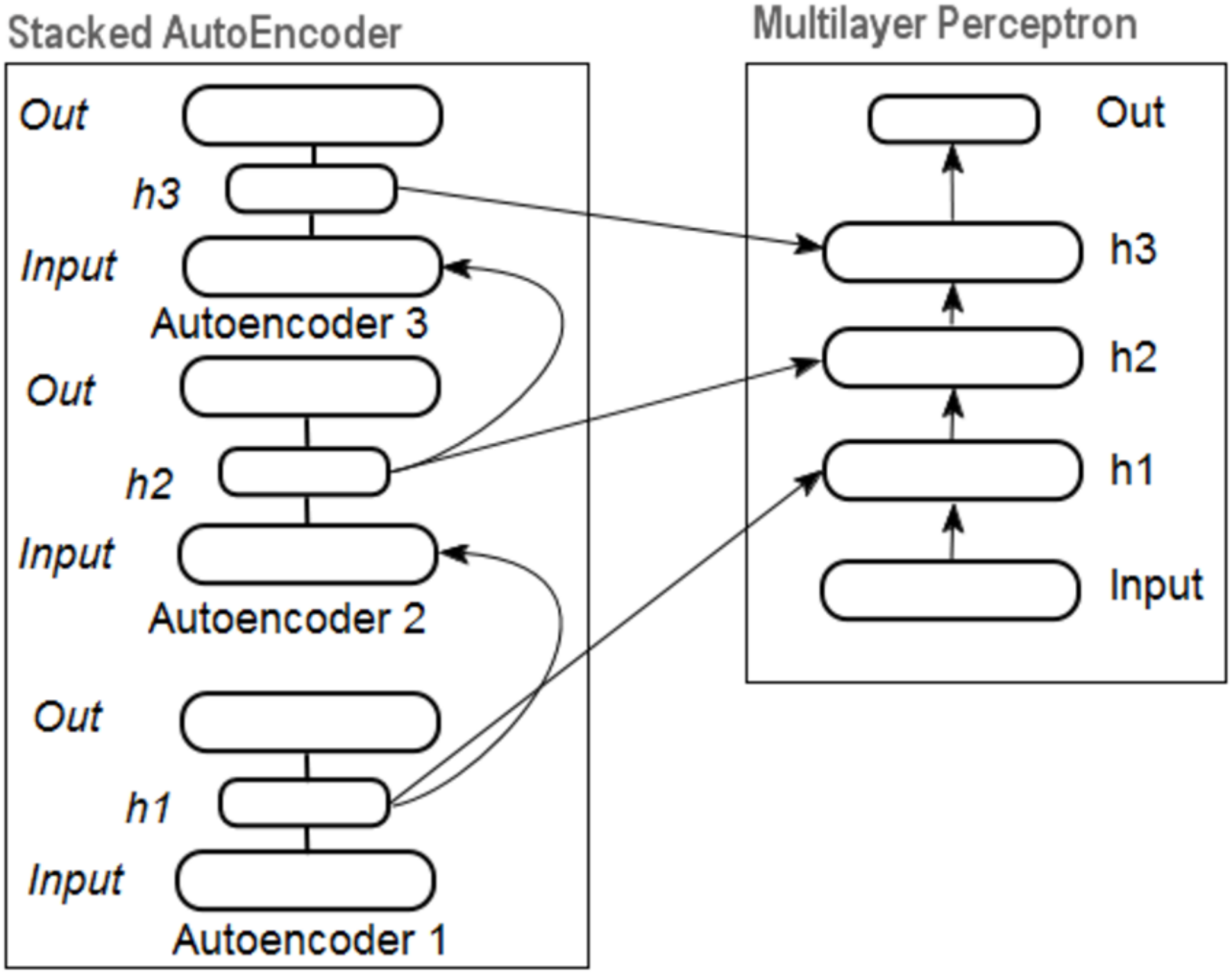
Comments and Applications

- 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

Stacked Autoencoder (SAE)

- The figure represents the idea of a stacked autoencoder: a deep neural network with weights initialized by a stacked autoencoder


Stacked AutoEncoder
Out
h3
Input
Autoencoder 3
Out
h2
Input
Autoencoder 2
Out
h1
Input
Autoencoder 1
Multilayer Perceptron
Out
h3
h2
h1
Input

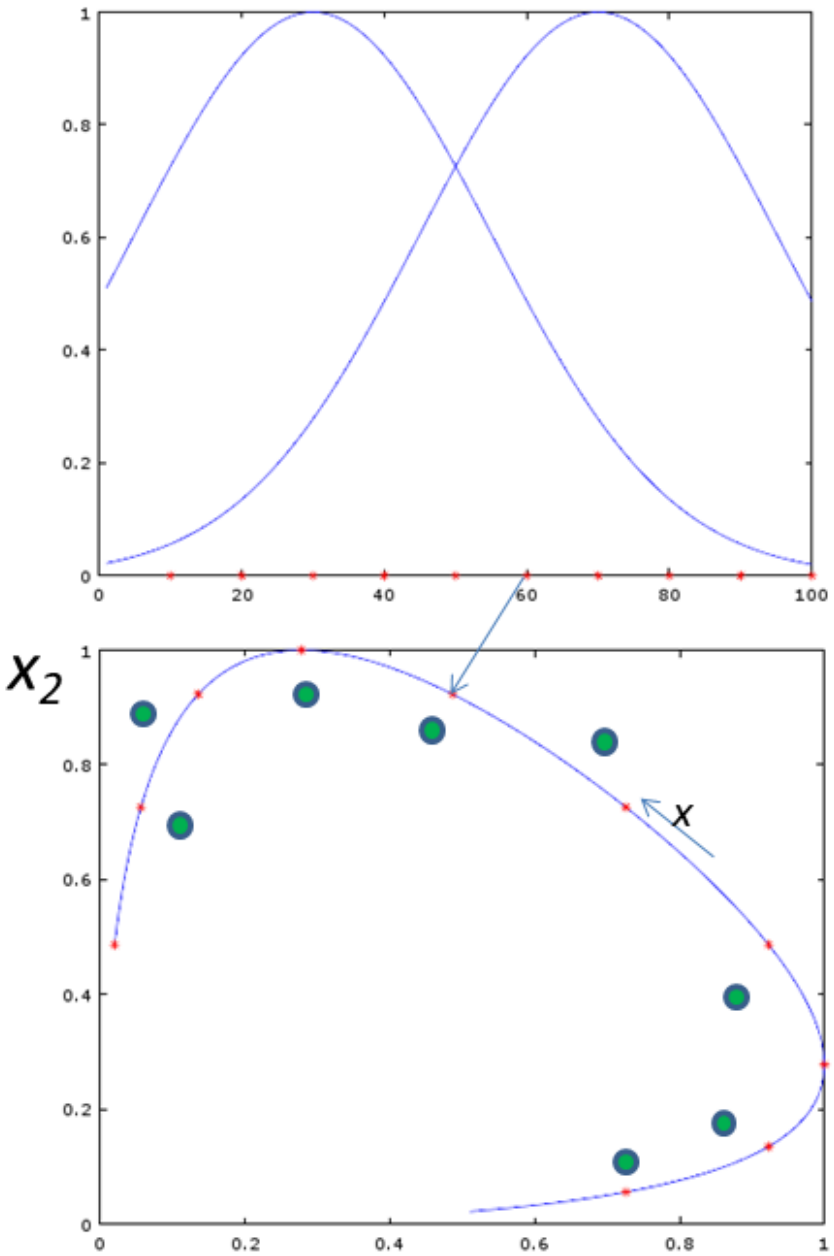
Data Represented in a Noisy Feature Space

- 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

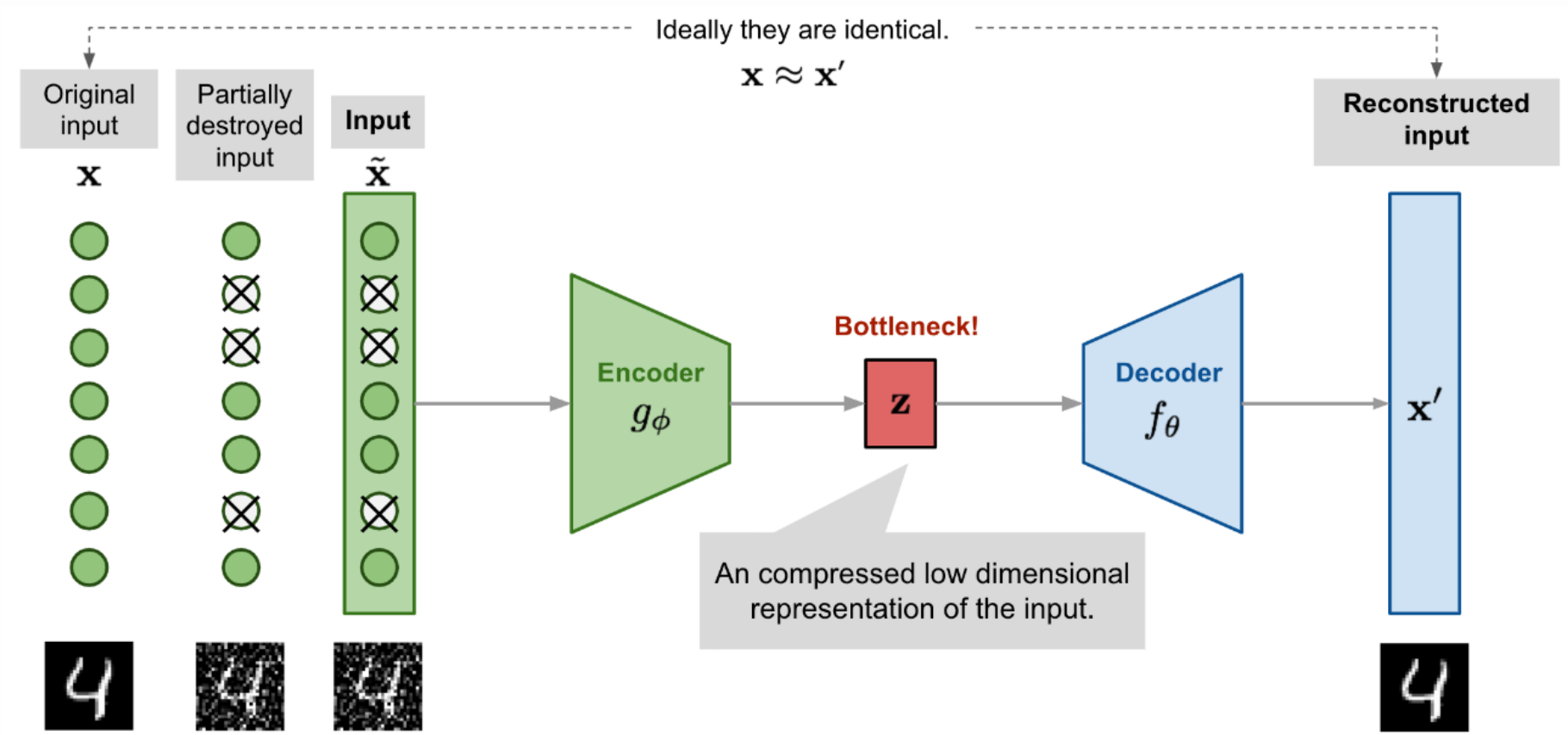


Fig. 2. Illustration of denoising autoencoder model architecture.

More

- **Sparse autoencoders**: add a penalty, e.g, $\sum_{k=1}^{M_h} |h_k|$, to encourage sparsity of the latent representation
- **Contractive autoencoder** (CAE): add a penalty, e.g,

$$\sum_{k=1}^{M_h} \sum_{j=0}^{M} \left(\frac{\partial h_k}{\partial x_j} \right)^2$$

(squared Frobenius norm of the Jacobian); encourages mapping to a low-dimensional manifold: small changes in $\mathbf{x}$ should lead to small changes in $\mathbf{h}$; the CAE has similar advantages as the denoising autoencoder; CAE even works when $M_h \geq M$

Learning a Generator

Variational Autoencoder

Learning a Generator

- If I knew for each data point $\mathbf{h}$ and $\mathbf{x}$, I could learn a generator $g^{gen}(\cdot)$ 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

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

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

- Thus generating a new $\mathbf{x}$ is trivial, but how do I learn the $g^{gen}(\cdot)$?

Training the VAE

- Training with a maximum likelihood approach is infeasible; one uses a mean field approximation, a special case of a variational approximation; details can be found in the Appendix
- For each data point $\mathbf{x}$, one learns the variational parameters, mean $\mu_{\mathbf{u}}(\mathbf{x})$ and covariance matrix $\Sigma_{\mathbf{u}}(\mathbf{x})$ as a function of $\mathbf{x}$ (e.g., with a deep neural network with weights $\mathbf{u}$); this is the **encoder**
- A sample $\mathbf{h}_s$ is generated from $\mathcal{N}(\mu_{\mathbf{u}}(\mathbf{x}), \Sigma_{\mathbf{u}}(\mathbf{x}))$ (which is an approximation to $P(\mathbf{h}|\mathbf{x})$); thus instead of transmitting the prediction of the encoder $\mu_{\mathbf{u}}(\mathbf{x})$ to the decoder, we transmit a noisy version of $\mu_{\mathbf{u}}(\mathbf{x})$
- Based on this sample all parameters ($\mathbf{v}$, $\mathbf{u}$) are adapted with backpropagation
- Then one proceeds with the next data point
- Details can be found in the Appendix

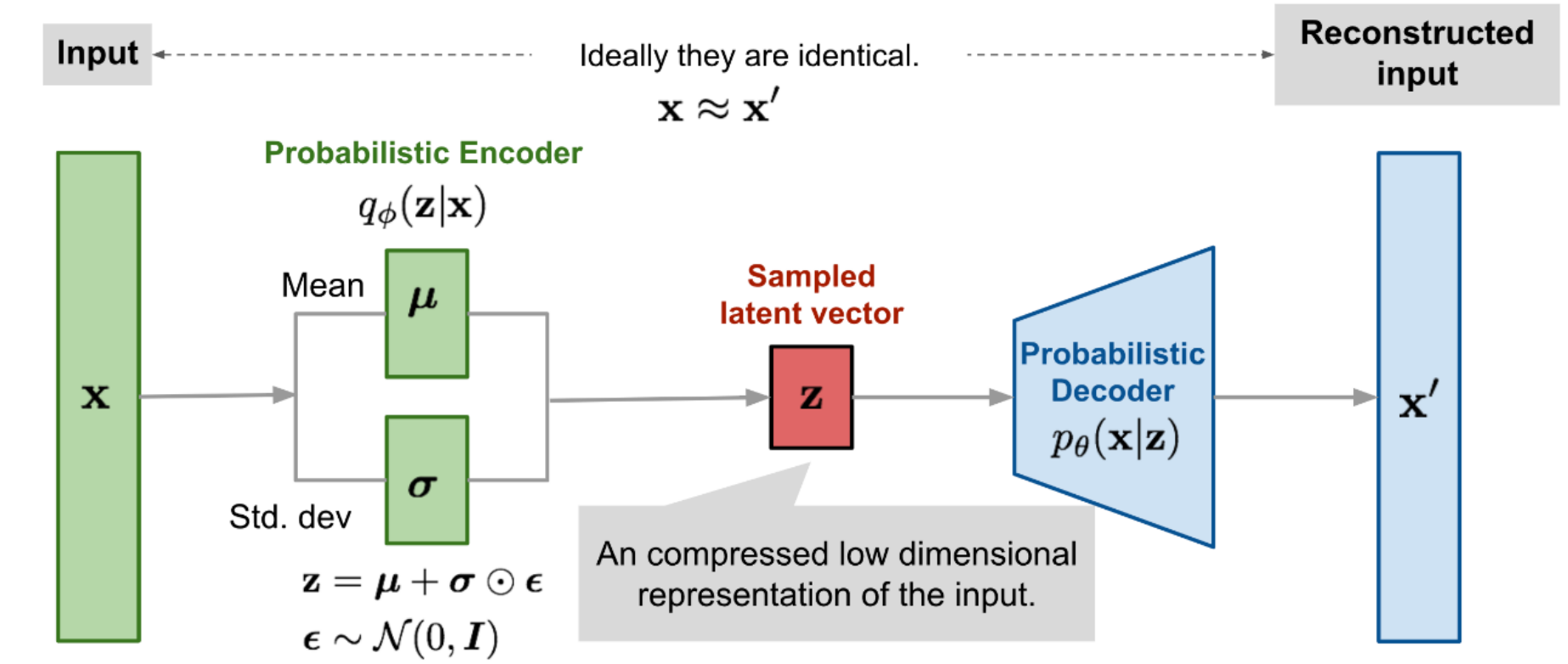
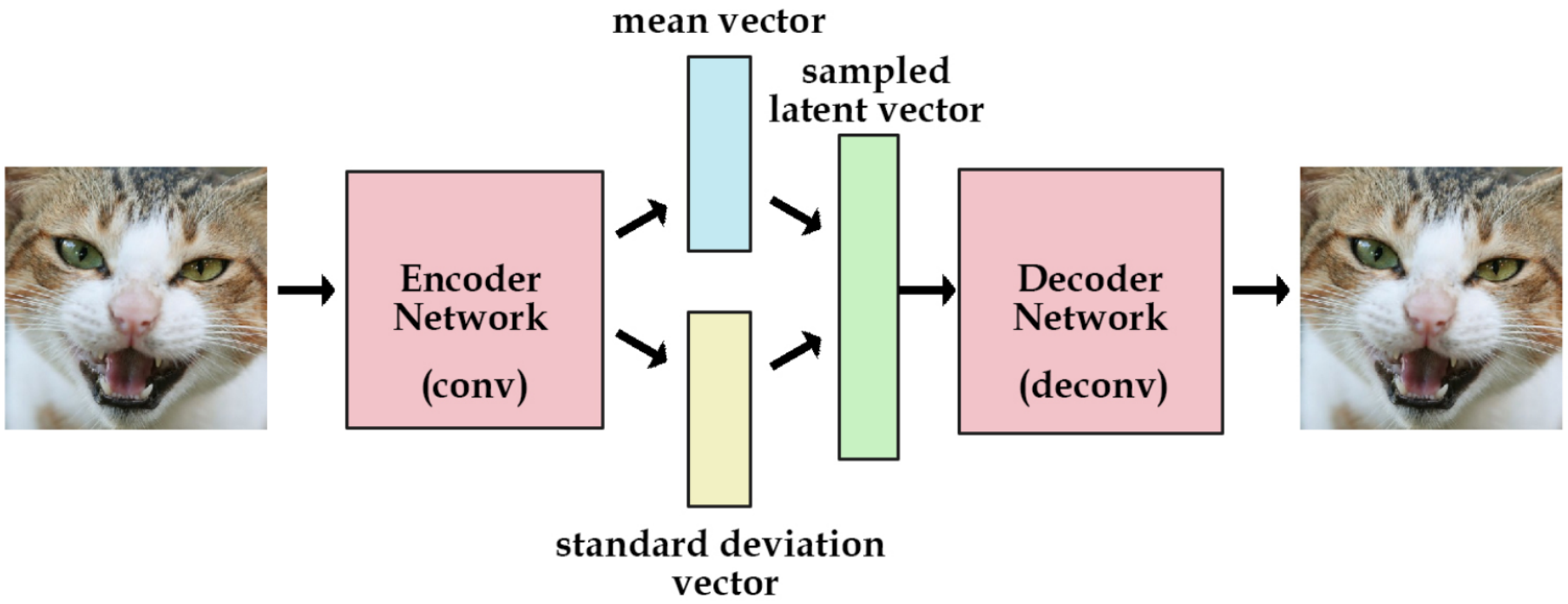


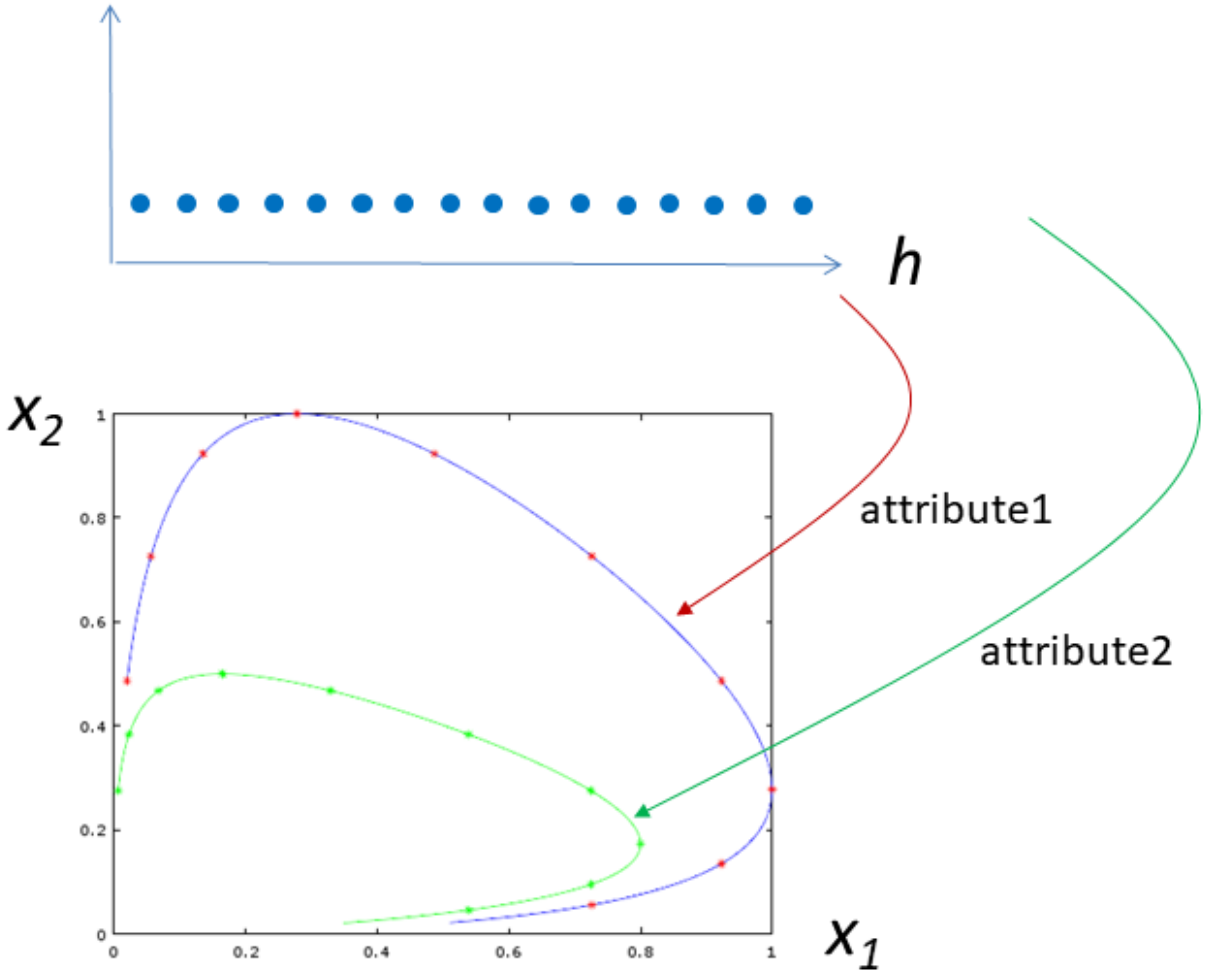
Fig. 9. Illustration of variational autoencoder model with the multivariate Gaussian assumption.


mean vector
sampled
latent vector
Encoder
Network
(conv)
Decoder
Network
(deconv)
standard deviation
vector

Attribute-specific Data Manifold

- There might be different manifolds for each attribute

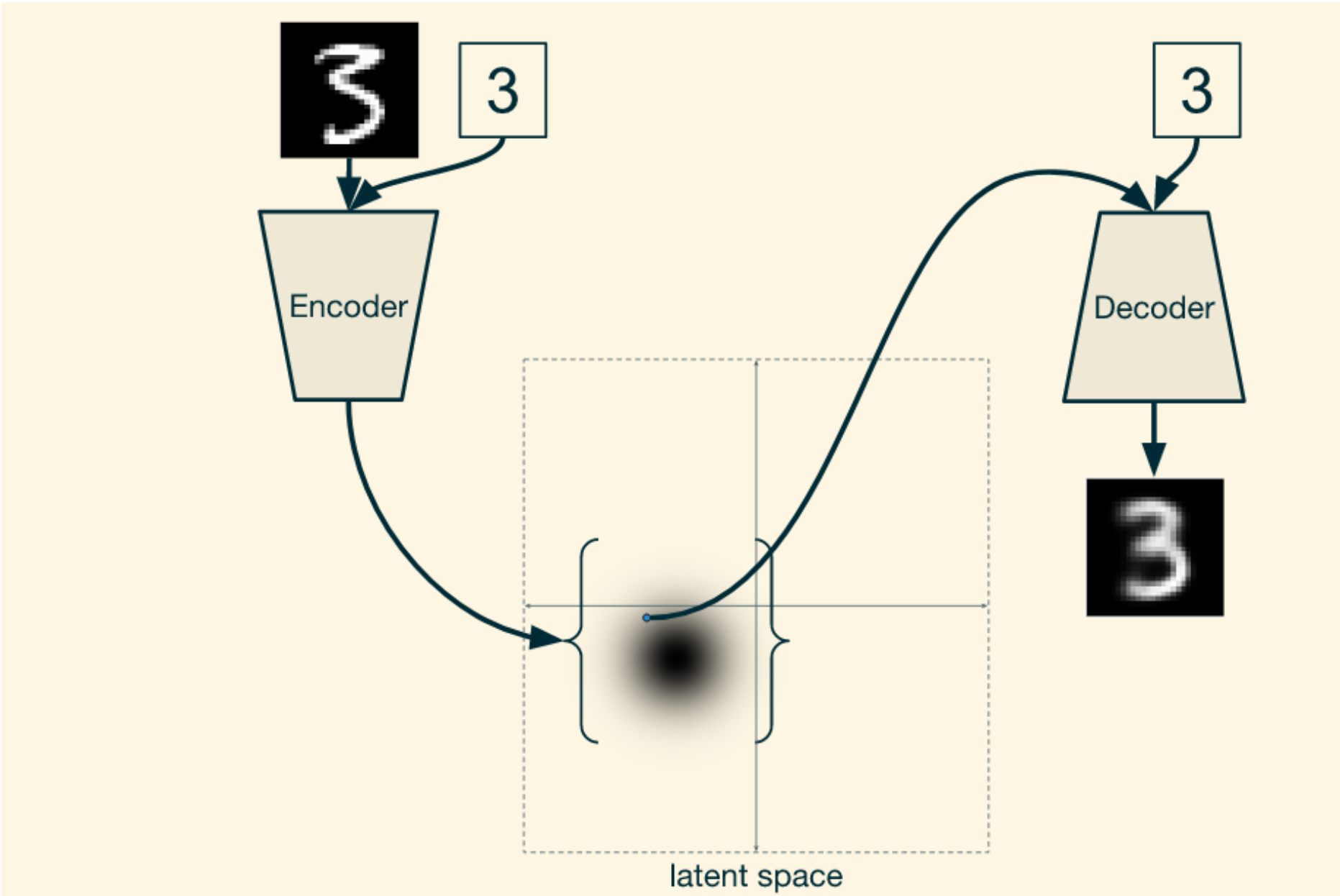
Attribute-specific distributions



Conditional Variational Autoencoders

- Conditional Variational Autoencoders
- The generator becomes attribute sensitive, $\mu_{\mathbf{u}}(\mathbf{x})$, $\Sigma_{\mathbf{u}}(\mathbf{x}) \rightarrow \mu_{\mathbf{u}}(\mathbf{x}, Attr)$, $\Sigma_{\mathbf{u}}(\mathbf{x}, Attr)$
- For generation,

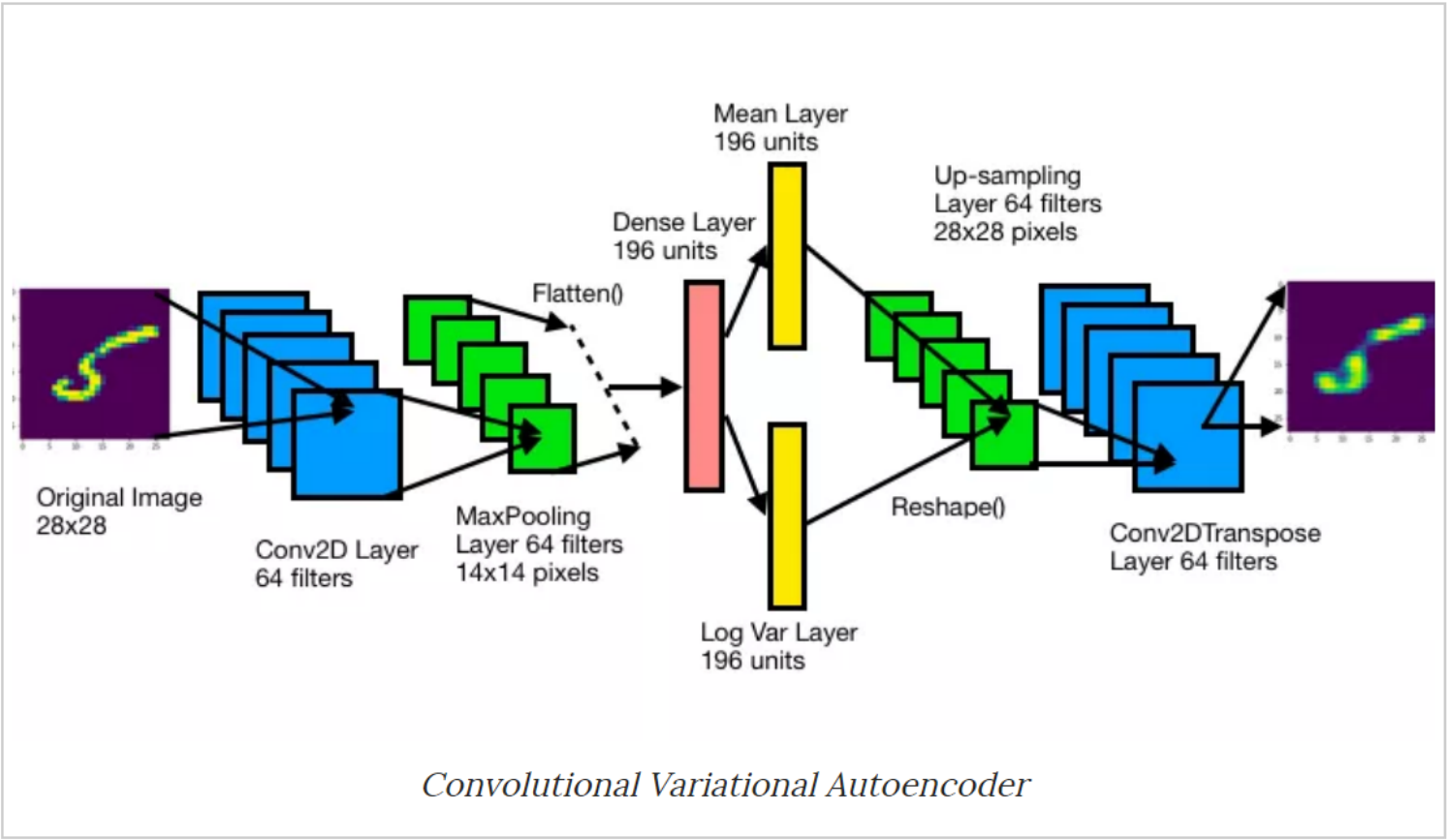
$$\mathbf{x} = g_{\mathbf{v}}^{gen}(\mathbf{h}, Attr)$$



A conditional variational autoencoder

Convolutional Variational Autoencoders

- The convolutional variational autoencoder uses convolutional layers



Convolutional Variational Autoencoder

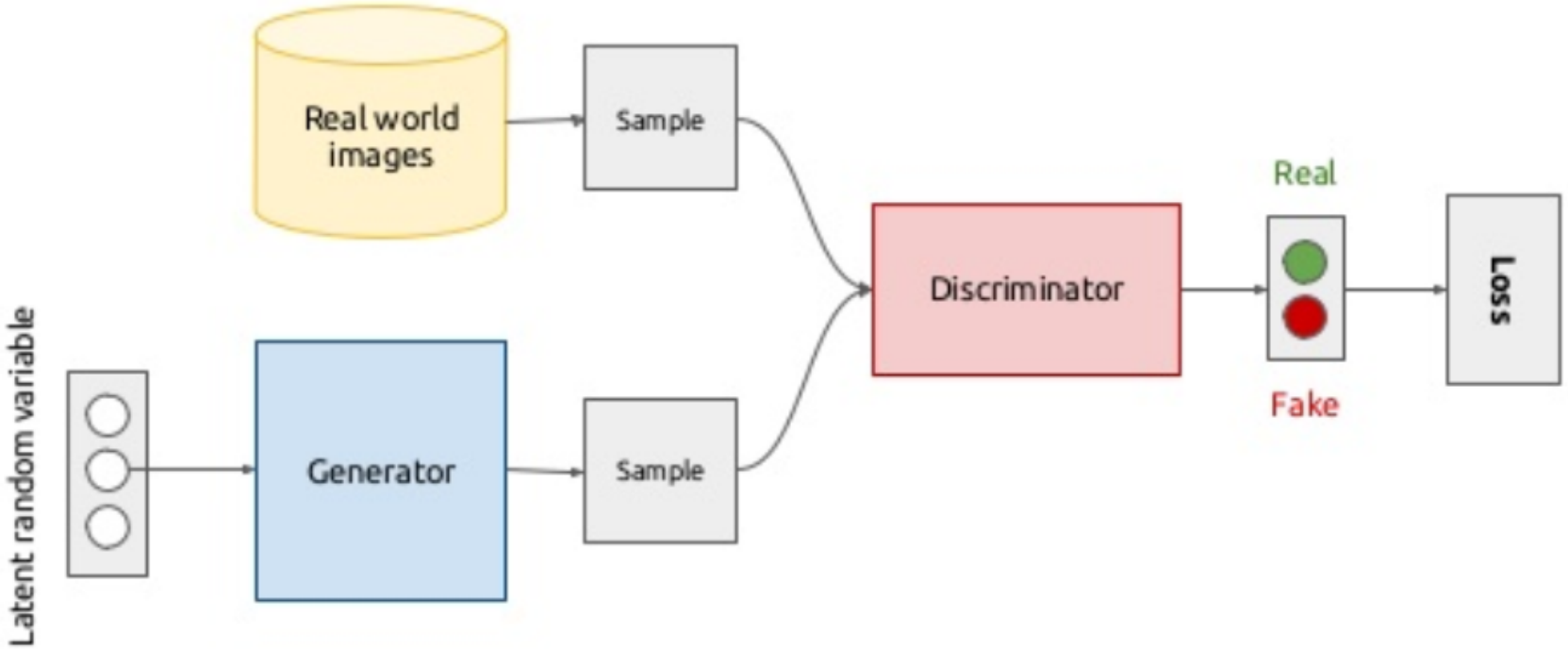
Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs)

- 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

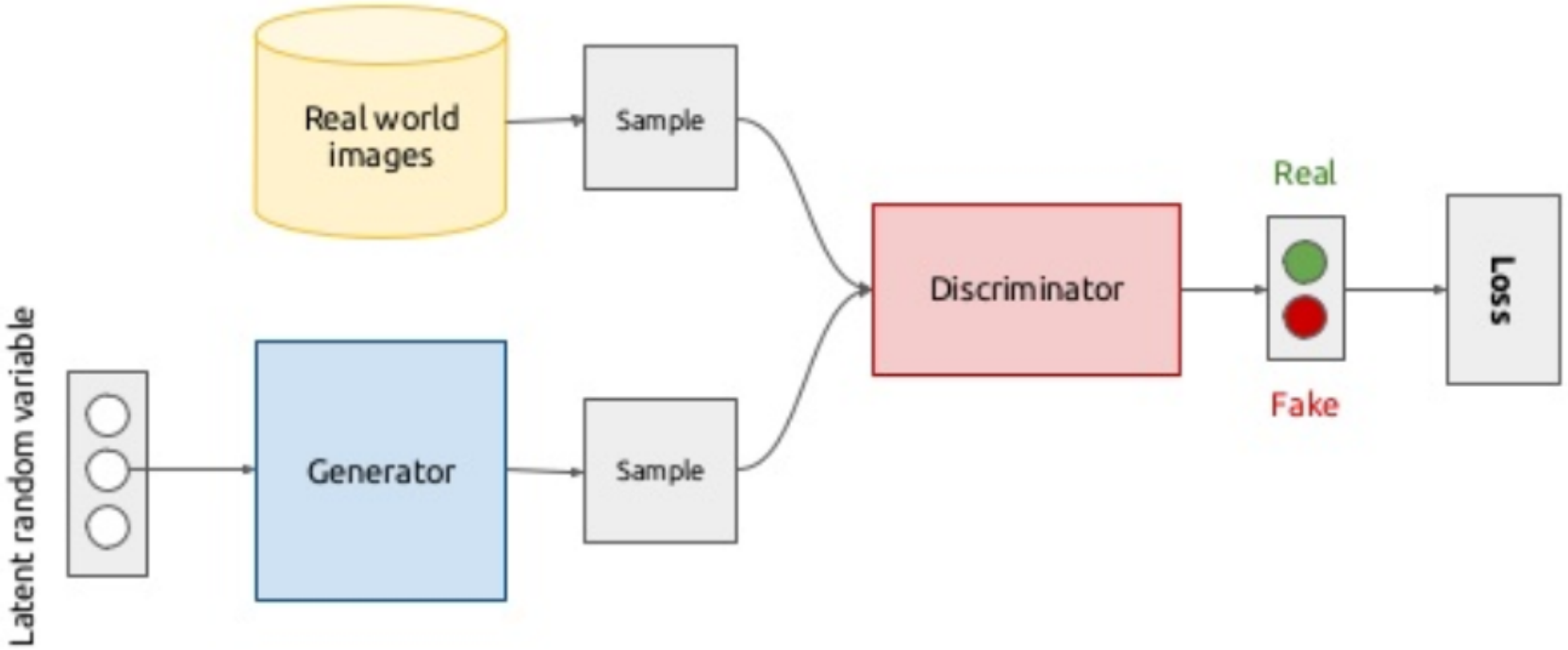
$$P_{train}(\mathbf{x}) \approx P_{gen}(\mathbf{x})$$

- 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

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

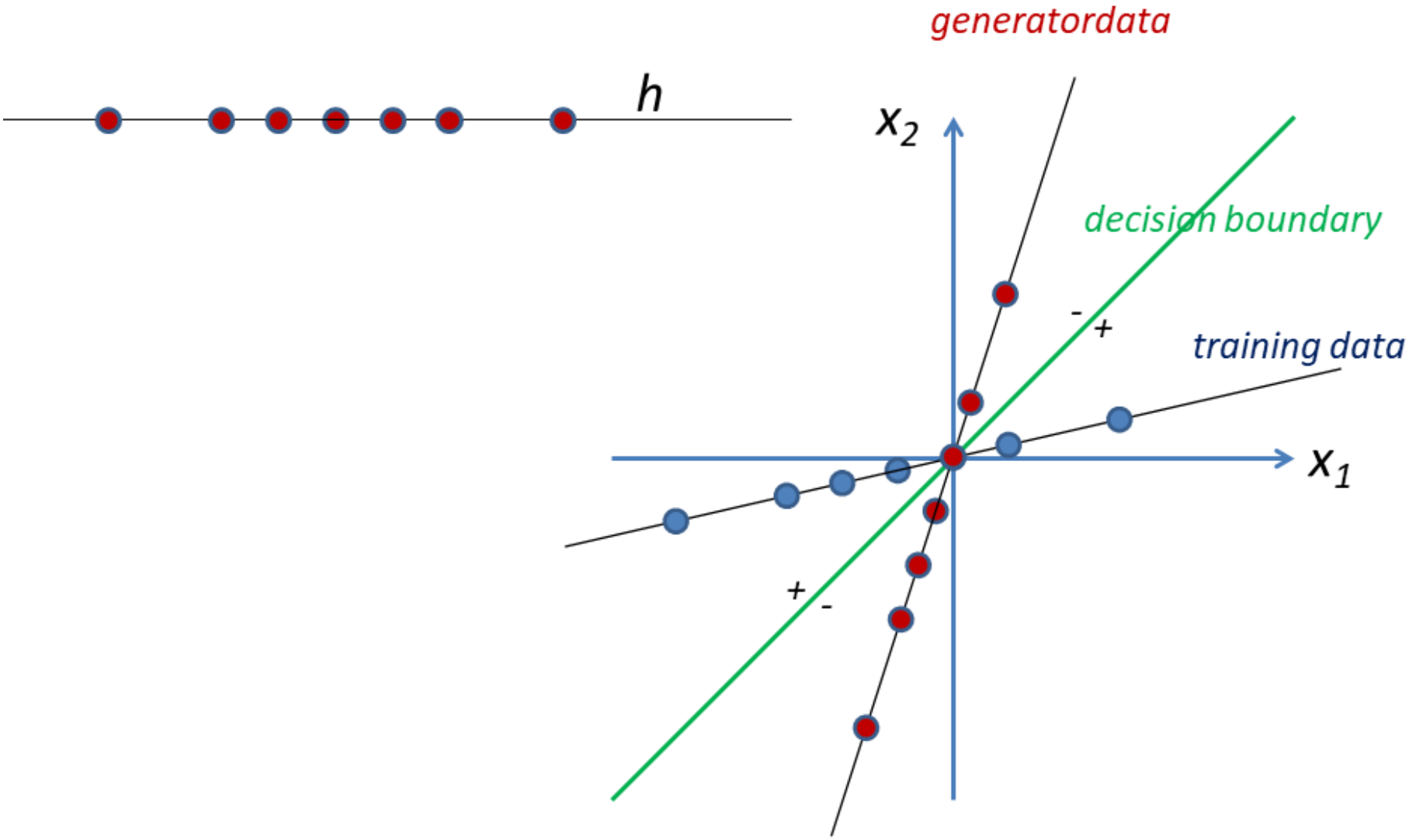
  where

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

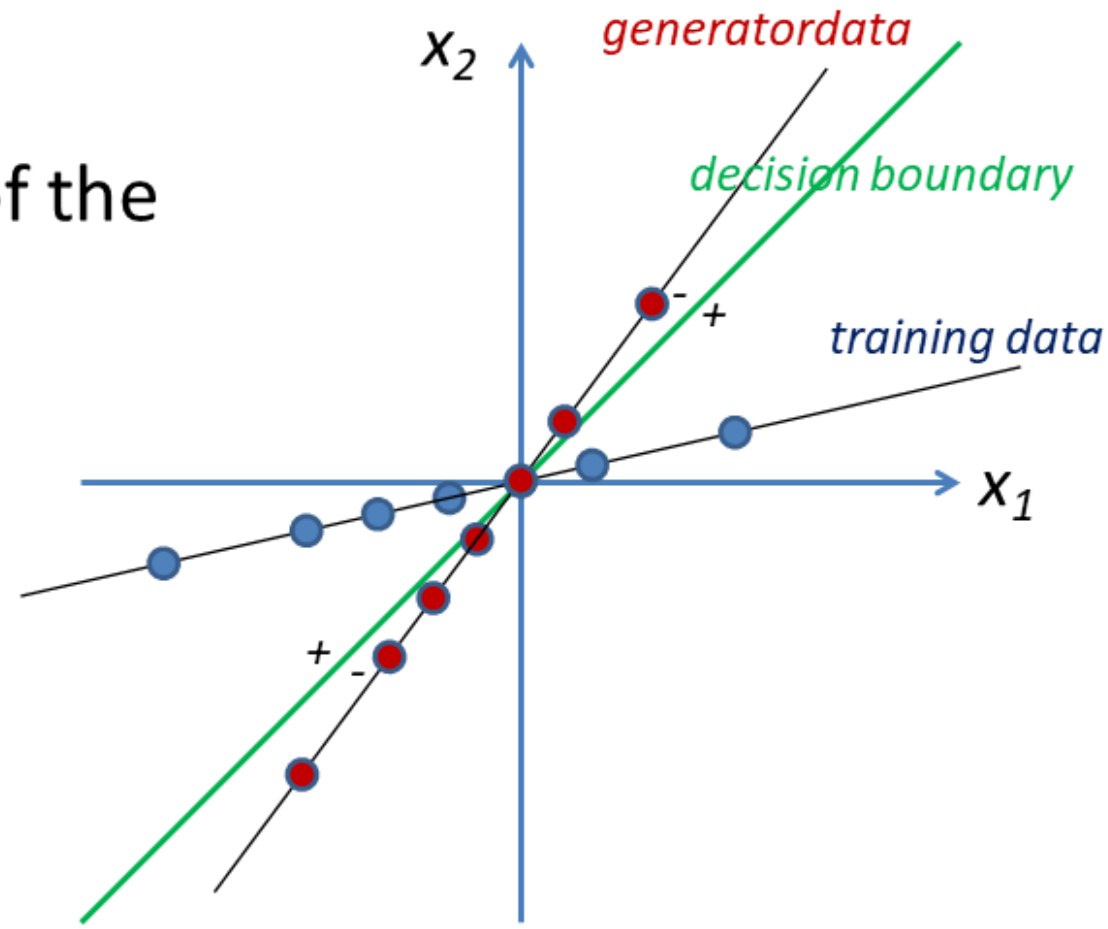
- This can be related to game theory: The solution for a zero-sum game is called a minimax solution, where the goal is to minimize the maximum cost for the player, here the generator. The **generator** wants to find the lowest cost, without knowing the actions of the **discriminator** (in two-player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

Illustration

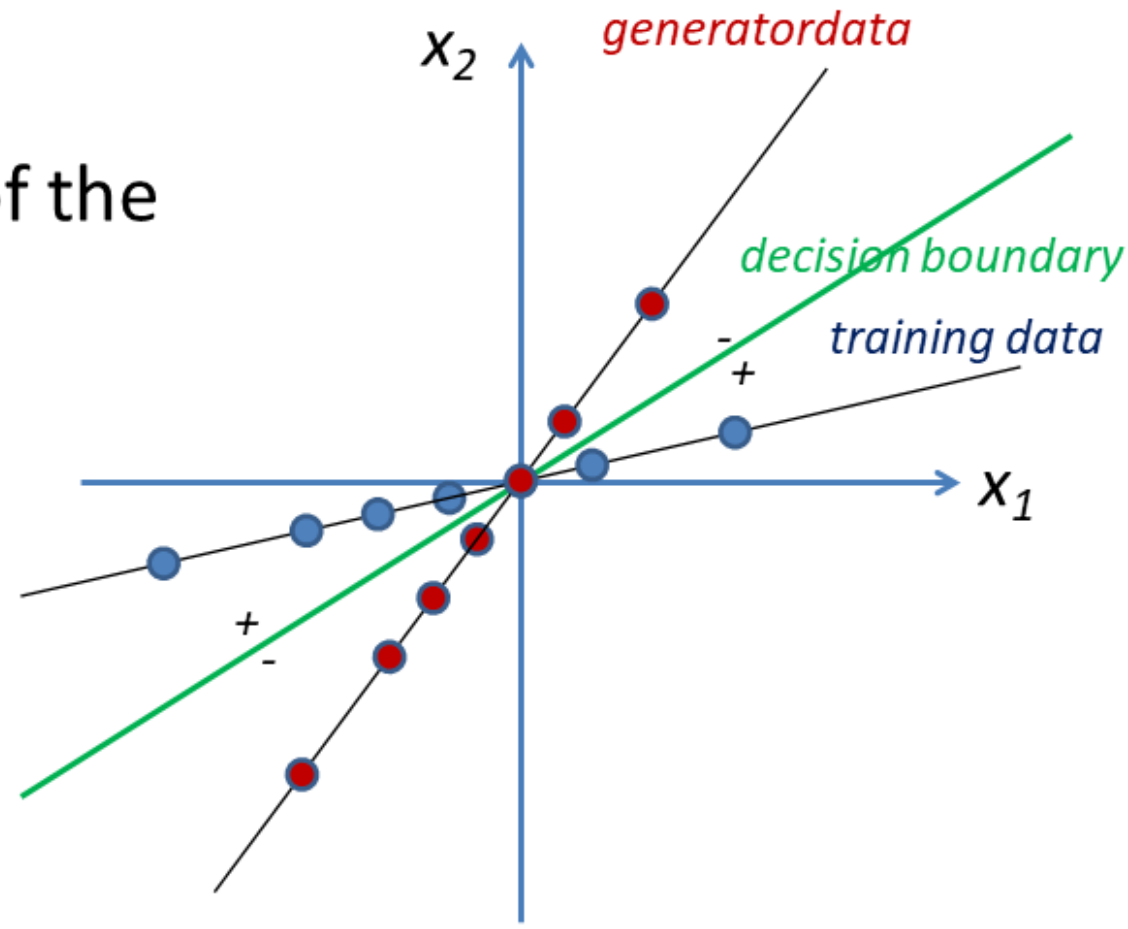
- Consider the following figure; h is one-dimensional Gaussian distributed: $P(h) = \mathcal{N}(h; 0, 1)$, $M_h = 1$
- The generator is $\mathbf{x} = h\mathbf{v}$, where $M = 2$; the data points are on a 1-D manifold in 2-D space; here: $v_1 = 0.2$, $v_2 = 0.98$
- The training data are generated similarly, but with $\mathbf{x} = h\mathbf{w}$ and $w_1 = 0.98$, $w_2 = 0.2$
- The discriminator is $y = \mathsf{sig}(|x_1|w_1 + |x_2|w_2)$, 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
generatordata
x_2
decision boundary
training data
x_1
-
+
+
-

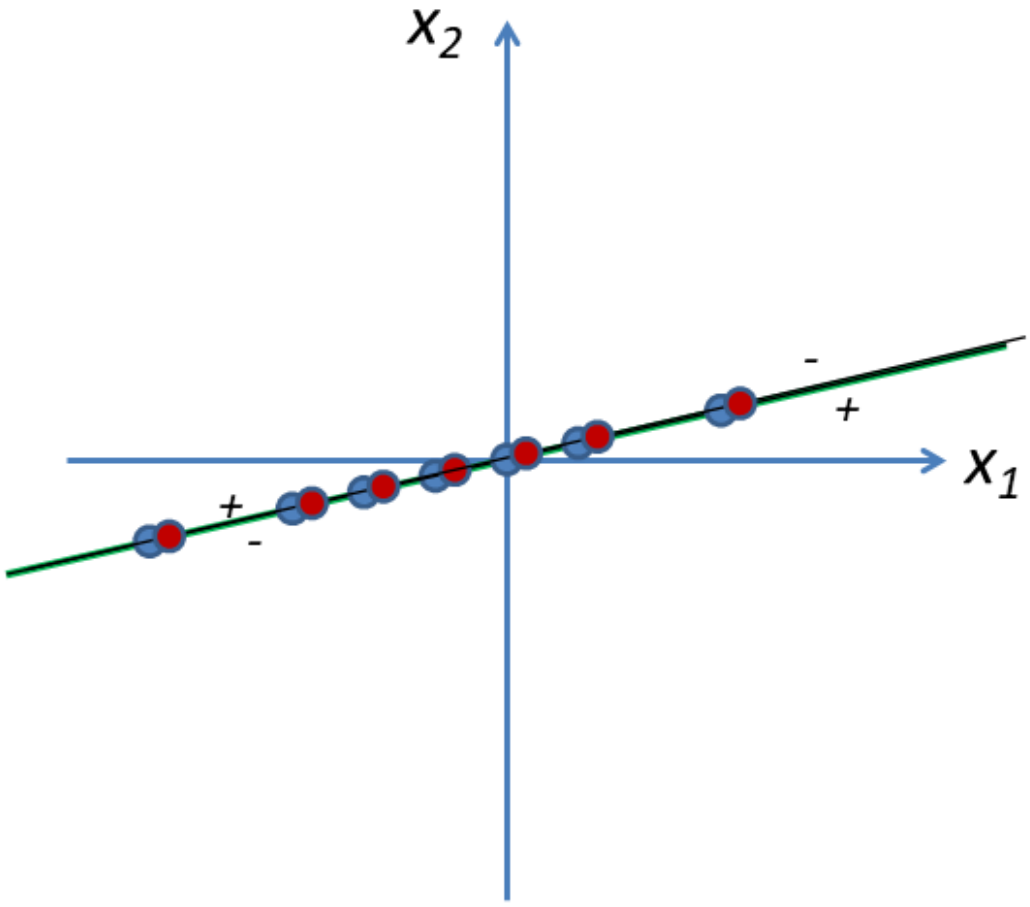
After update of the generator



After update of the discriminator



Convergence



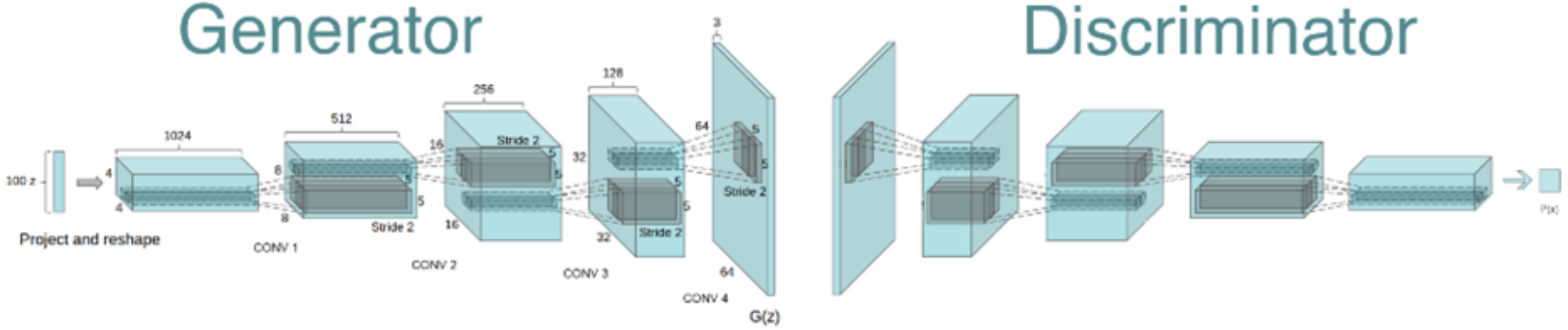
Applications

- 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

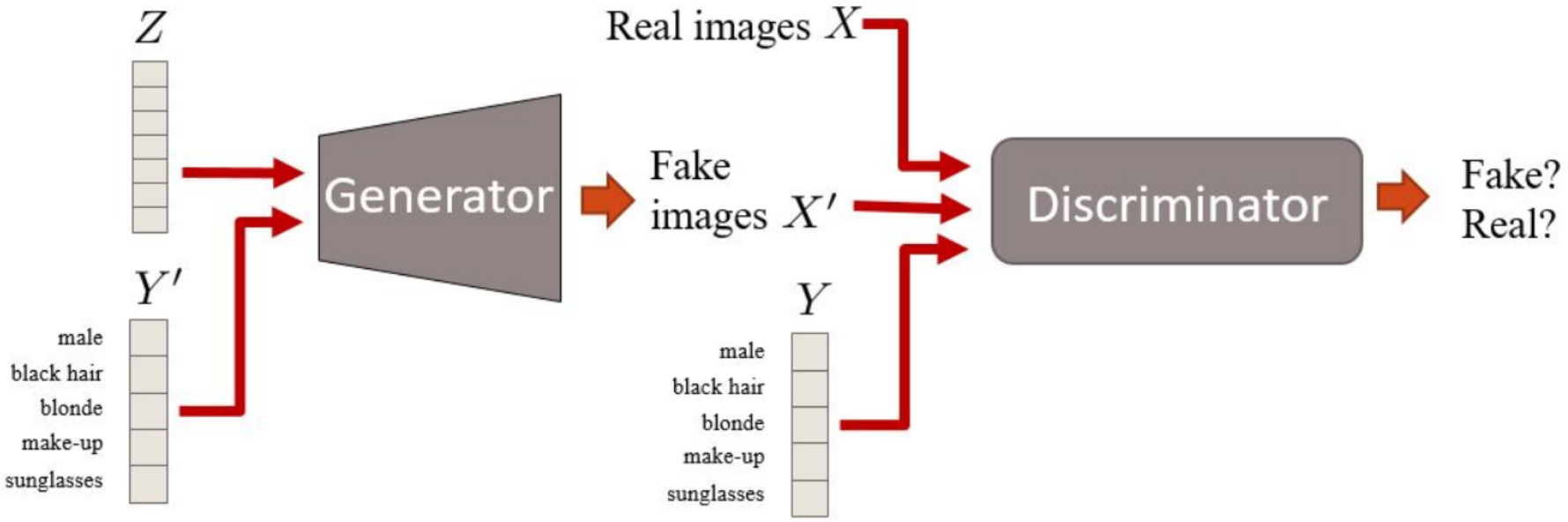
- If the data consists of images, the discriminator is a binary image classifier and the generator needs to generate images
- Deep Convolutional GAN (DCGAN): the generator and the discriminator contain convolutional layers and deconvolutional layers
- (Deconvolution layer is a very unfortunate name and should rather be called a transposed convolutional layer)
- Radford et al. (shown below). $M_h = 100$; 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)

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 attribites 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

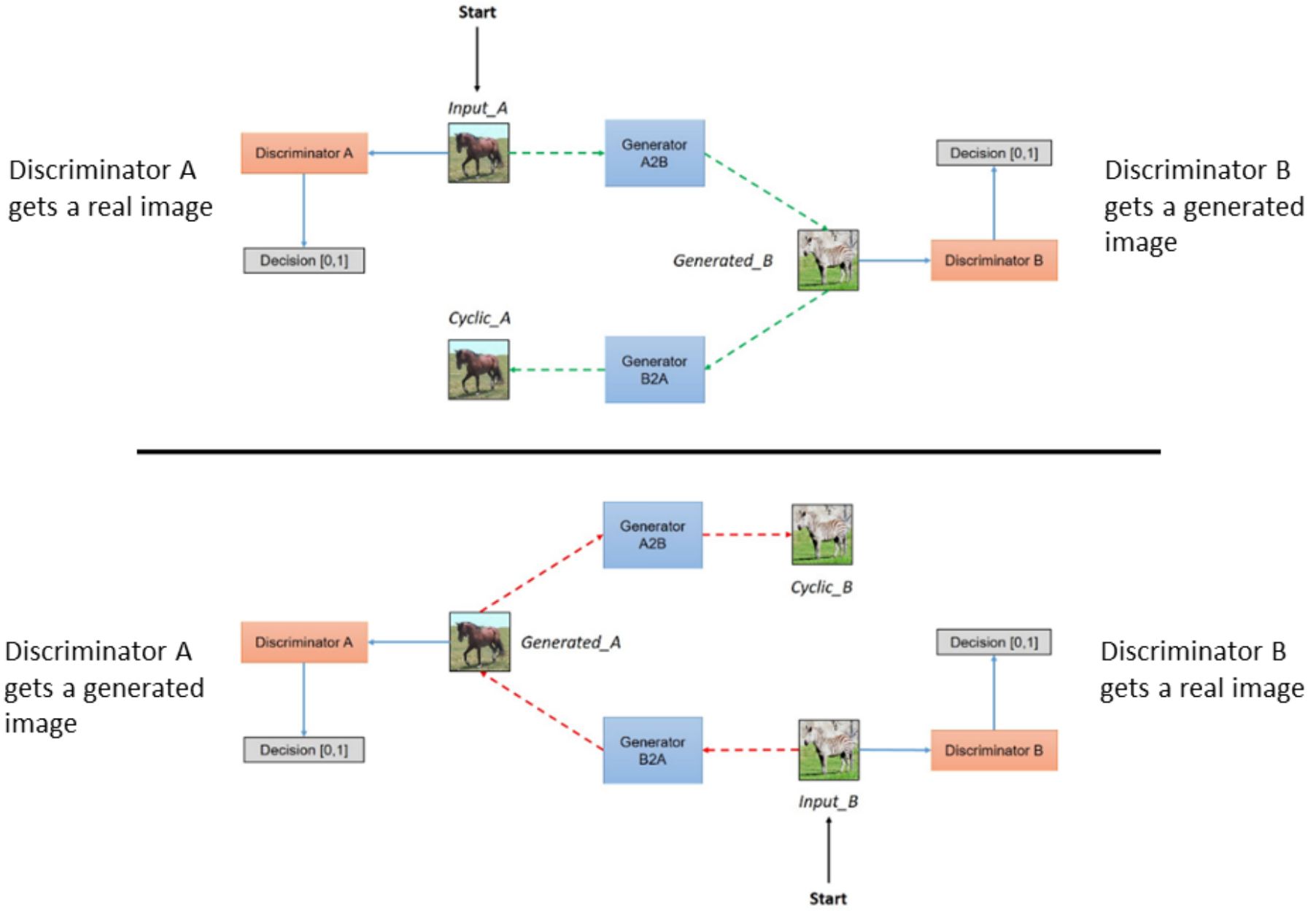
$$\hat{\mathbf{x}}_{horse} = g_{B2A}(g_{A2B}(\mathbf{x}_{horse}))$$

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

- These constraints are enforced using the *cycle consistency loss*

CycleGAN

- CycleGAN does exactly that
- CycleGAN adds two discriminators, trained with the *adversarial loss*
- *discriminator*$_A$ tries to classify real horses from generated horses
- *discriminator*$_B$ tries to classify 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

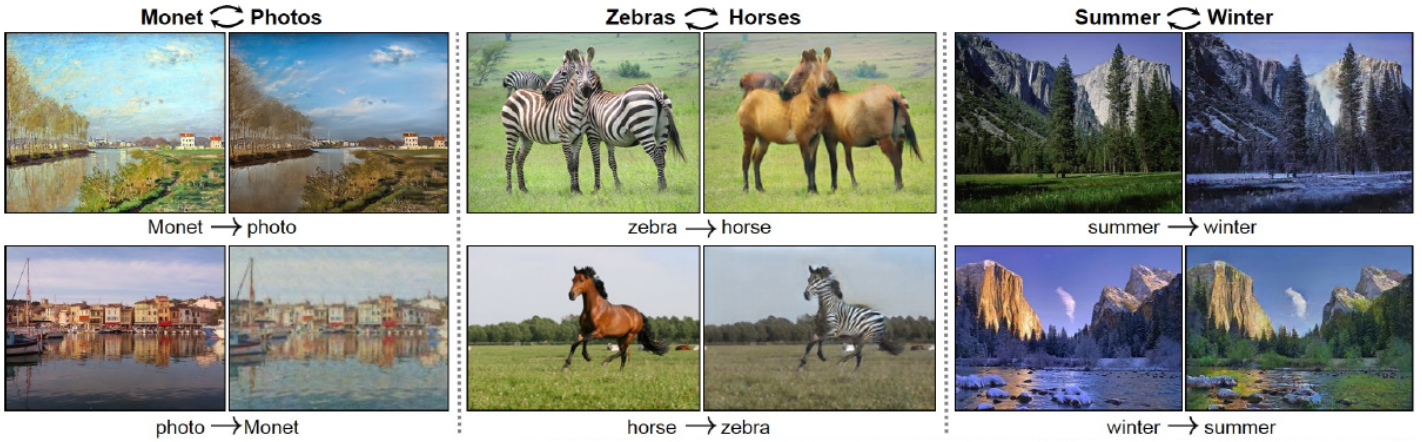


Fig. 8. CycleGAN model learns image to image translations between two unordered image collections. Shown here are the examples of bi-directional image mappings: Monet paintings to landscape photos, zebras to horses, and summer to winter photos in Yosemite park. Figure reproduced from [4].

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

Related Approaches

- 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

Appendix*

Distribution*

- Both the VAE decoder and a GAN generator produce samples from probability distributions

$$P(\mathbf{x}) = \int \mathcal{N}(\mathbf{x}; g(\mathbf{h}), \epsilon^2 I)\mathcal{N}(\mathbf{h}; 0, I))d\mathbf{h}$$

  where latent features are generated from $\mathcal{N}(\mathbf{h}; 0, I))$ and where we added a a tiny noise with variance ϵ^2 to the generator

- With $\epsilon^2 \to 0$, points outside the manifold will get zero probability density

- Obtaining the probability value of $P(\mathbf{x})$ by the last formula is not straight forward

Variational Autoencoder*

- We apply mean field theory, which is a special case of a variational method
- We consider one data point with measured $\mathbf{x}$
- The contribution to the log likelihood of that data point is

$$l(\mathbf{x}) = \log P(\mathbf{x}) = \log \int P(\mathbf{h})P(\mathbf{x}|\mathbf{h})d\mathbf{h}$$

  The VAE assumes that $P(\mathbf{h}) = \mathcal{N}(\mathbf{h}; 0, I)$

- With approximating density $Q(\mathbf{h})$, we write

$$l(\mathbf{x}) = \log \int Q(\mathbf{h}) \frac{P(\mathbf{h})P(\mathbf{x}|\mathbf{h})}{Q(\mathbf{h})} d\mathbf{h}$$

  Using Jensen's inequality

$$l(\mathbf{x}) \geq \int Q(\mathbf{h})[\log P(\mathbf{h}) + \log P(\mathbf{x}|\mathbf{h}) - \log Q(\mathbf{h})]d\mathbf{h}$$

Variational Autoencoder (cont'd)

- The last expression can be written as

$$\mathbb{E}_Q[\log(P(\mathbf{h})P(\mathbf{x}|\mathbf{h}))] - \mathbb{E}_Q[\log Q(\mathbf{h})]$$

  or as

$$\mathbb{E}_Q[\log P(\mathbf{x}|\mathbf{h})] - \mathbb{E}_Q[\log Q(\mathbf{h}) - \log P(\mathbf{h})]$$

  The second term in the last equation is the same as (Kullback-Leibler divergence) $KL(Q(\mathbf{h})\|P(\mathbf{h}))$, so we maximize

$$\mathbb{E}_Q[\log P(\mathbf{x}|\mathbf{h})] - KL(Q(\mathbf{h})\|P(\mathbf{h}))$$

- With a least squares cost function, the first term becomes, with $\mathbf{h}_s$ being a sample from $P(\mathbf{h})$,

$$\int Q(\mathbf{h}) \log P(\mathbf{x}|\mathbf{h}) d\mathbf{h} \approx const - \frac{\|\mathbf{x} - g_{\mathbf{v}}^{gen}(\mathbf{h}_s)\|^2}{2\sigma_\epsilon^2}$$

Variational Autoencoder (cont'd)

- The VAE assumes for the approximating distribution, $Q(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mu_{\mathbf{u}}(\mathbf{x}), \Sigma_{\mathbf{u}}(\mathbf{x}))$. Then, (see Wikipedia),

$$KL(Q(\mathbf{h})\|P(\mathbf{h}) = \frac{1}{2}\left(\text{tr}\,(\Sigma_{\mathbf{u}}(\mathbf{x})) + \mu_{\mathbf{u}}(\mathbf{x})^{\mathsf{T}}(\mu_{\mathbf{u}}(\mathbf{x}) - M_h - \ln\det\Sigma_{\mathbf{u}}(\mathbf{x})\right)$$

- Reparameterization trick: since we cannot do backpropagation through the sampling, we sample ϵ_s from $\mathcal{N}(0, I)$ and multiply the sample by $\sqrt{\Sigma_{\mathbf{u}}(\mathbf{x})}$ and add $\mathbf{u}$.

Variational Autoencoder (cont'd)

- The overall cross entropy cost function (negative log likelihood), with a diagonal $\Sigma_{\mathbf{u}}(\mathbf{x})$, and which we minimize w.r.t $\mathbf{w}$ and $\mathbf{u}$, is for data point $\mathbf{x}$

$$-l(\mathbf{x}) = \frac{1}{2\sigma_\epsilon^2}\|\mathbf{x} - g_{\mathbf{v}}^{gen}(\mu_{\mathbf{u}}(\mathbf{x}) + \sqrt{\Sigma_{\mathbf{u}}(\mathbf{x})} \circ \epsilon_s)\|^2$$

$$+\frac{1}{2}\left(\mu(\mathbf{x})^{\mathsf{T}}\mu(\mathbf{x}) + \sum_j \left[\Sigma_{\mathbf{u},j,j}(\mathbf{x}) - \log \Sigma_{\mathbf{u},j,j}(\mathbf{x})\right]\right)$$

- The first term in the large bracket encourages $\mu_{\mathbf{u}}(\mathbf{x} \to 0$; the second term is minimum if $\Sigma_{\mathbf{u},j,j} = 1$, $\forall j$

Variational Autoencoder (cont'd)

- Encoder: An encoder is only used in training outputs $\mu_{\mathbf{u}}(\mathbf{x})$ and $\Sigma_{\mathbf{u}}(\mathbf{x})$.
- Decoder/Generator: To generate a new data point we sample $\mathbf{h}_s$ from $\mathcal{N}(0, I)$ and calculate $g^{gen}_{\mathbf{v}}(\mathbf{h}_s)$.
- See: Tutorial on Variational Autoencoders, Carl Doersch

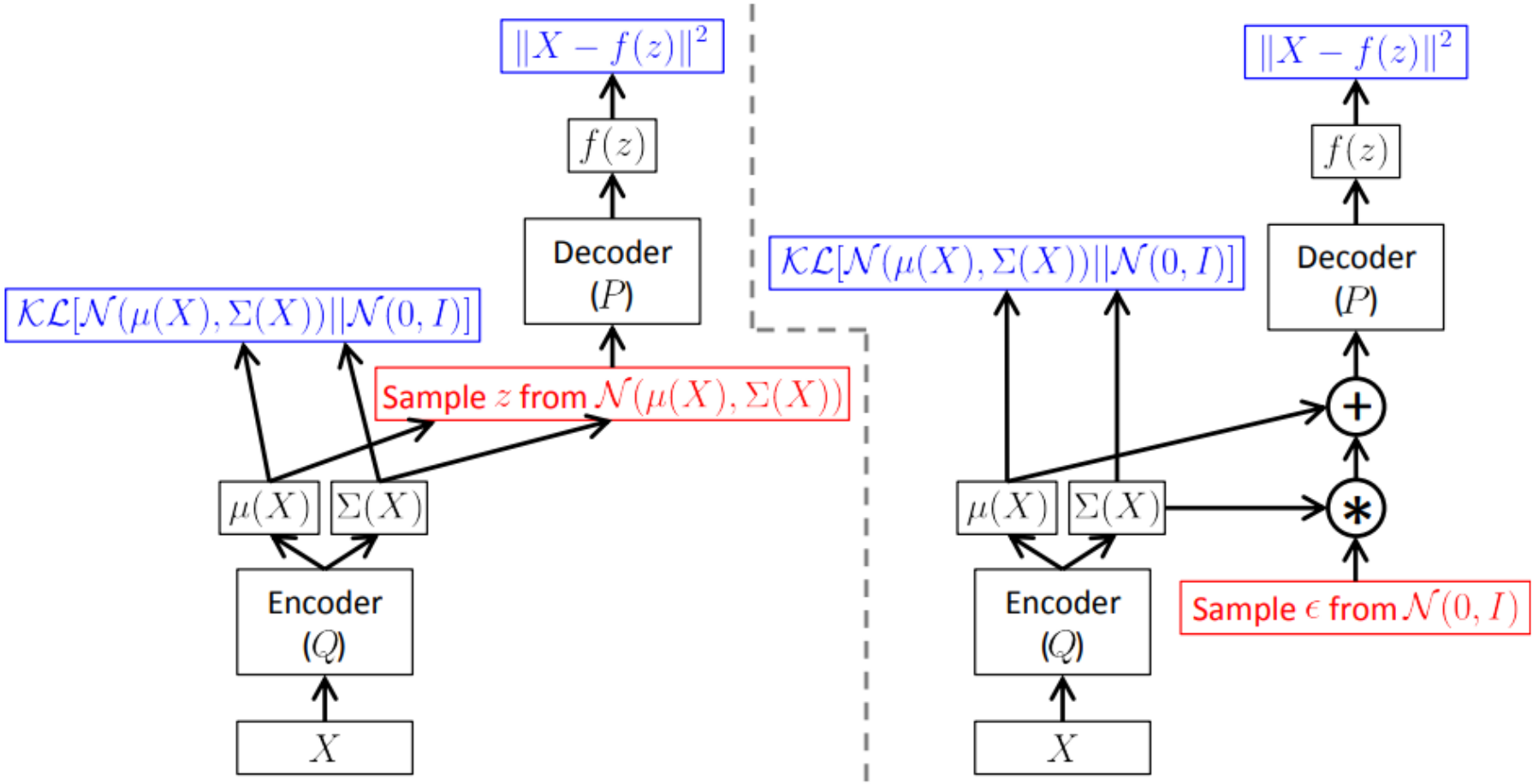


Figure 4: A training-time variational autoencoder implemented as a feedforward neural network, where $P(X|z)$ is Gaussian. Left is without the "reparameterization trick", and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward

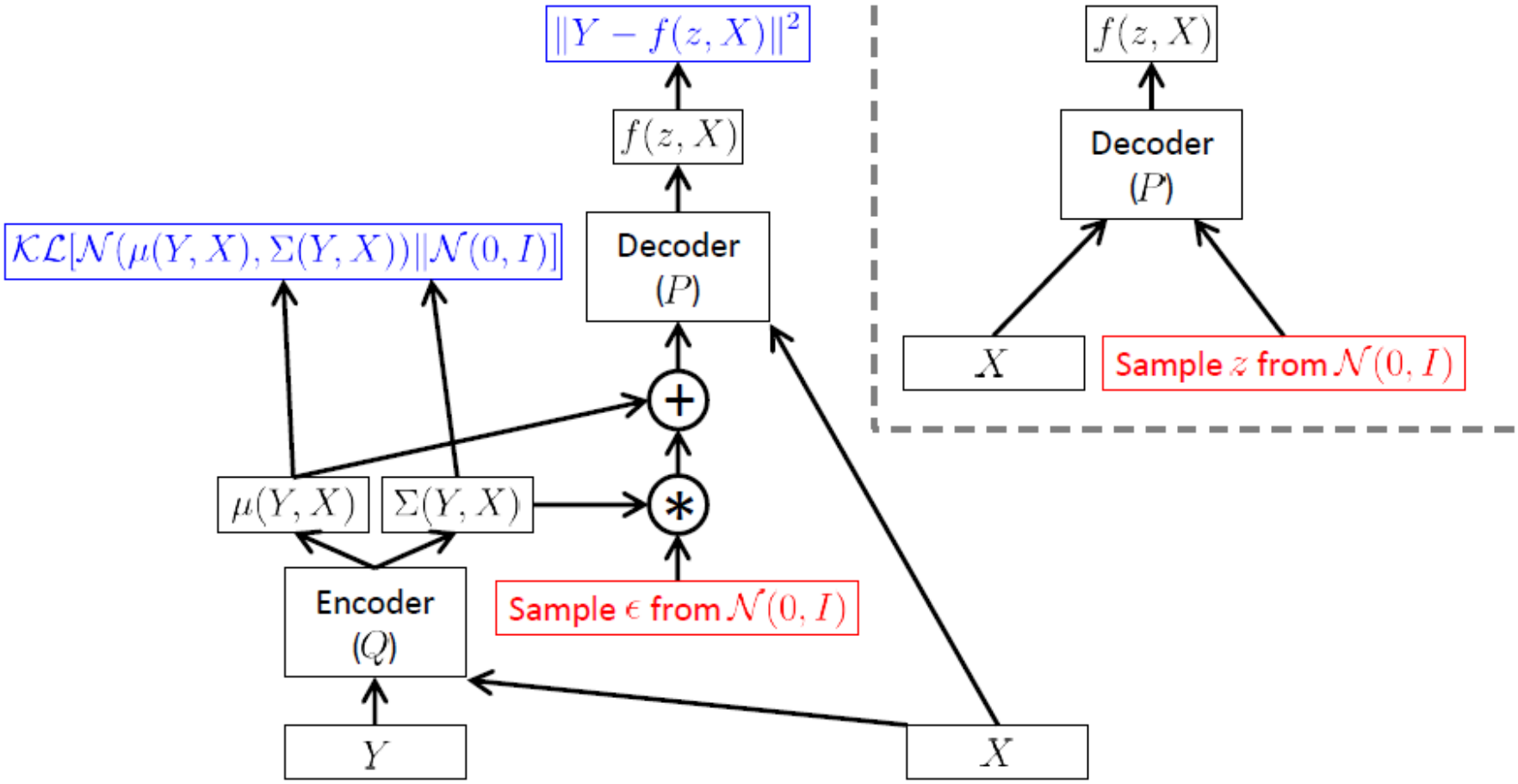


Figure 6: Left: a training-time conditional variational autoencoder implemented as a feedforward neural network, following the same notation as Figure 4. Right: the same model at test time, when we want to sample from $P(Y|X)$.