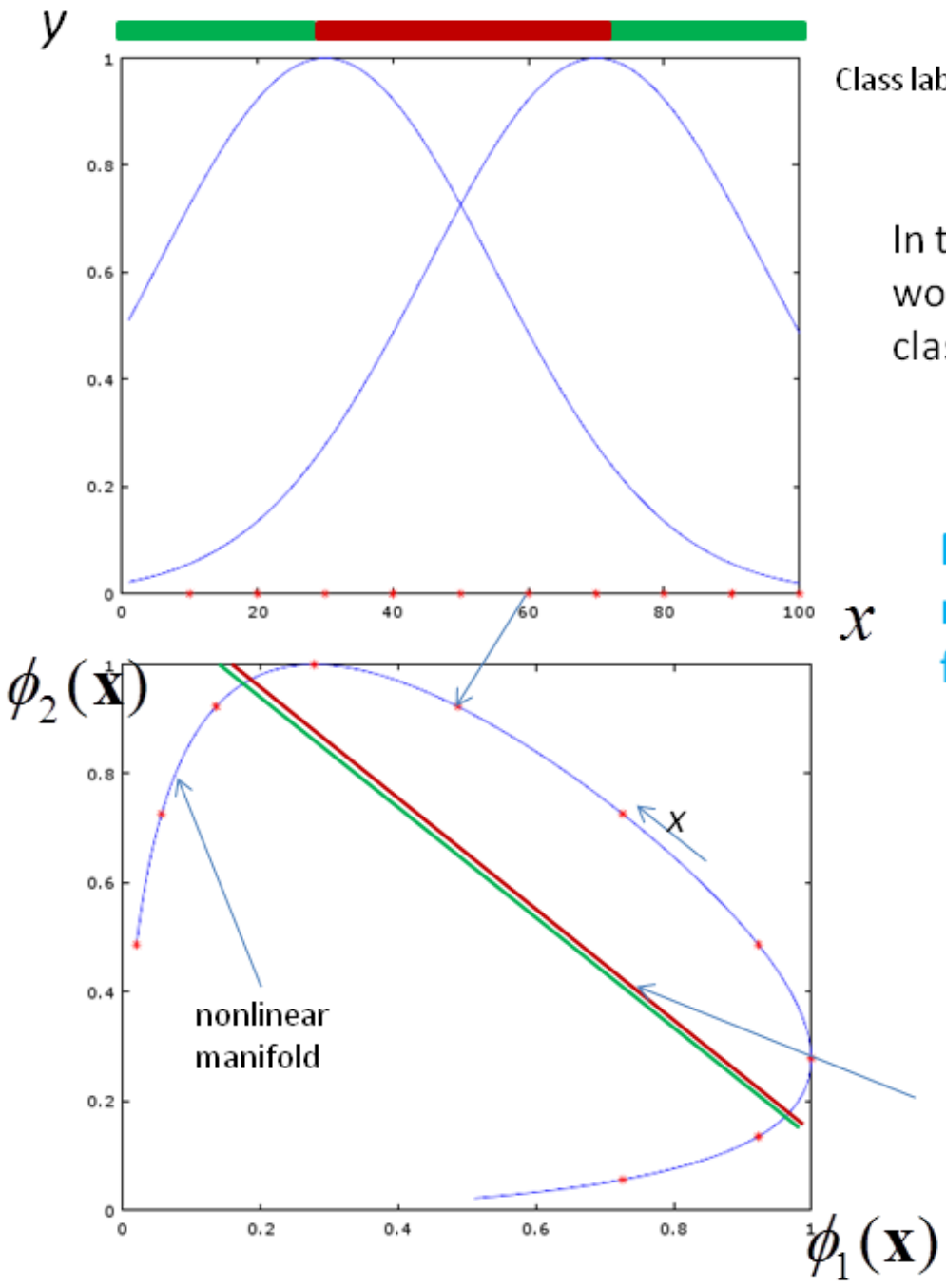
Feature Spaces, Manifolds, and Deep Generative Models

Volker Tresp
Summer 2018

Review: Manifold in Basis Function Space

- In the lecture on basis functions, we discussed the map from input space to basis function space
- In the following figure, data is originally in 1-D space and is mapped via two basis functions in a 2-D space
- In the 2-D basis function space, data lies in a 1-D manifold



In the 1-D input space, a linear classifier would not be able to separate the two classes

From a linear 1-D input space (top) to a nonlinear 1-D manifold in 2-D basis function space (bottom)

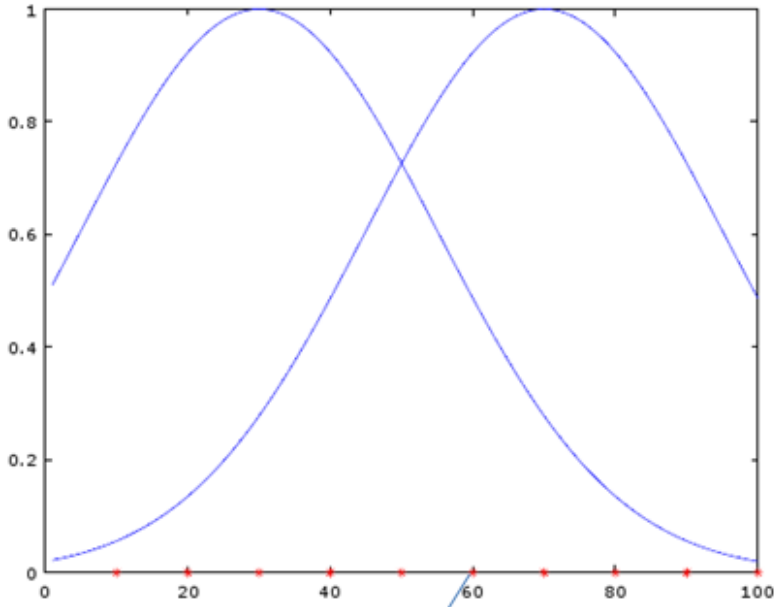
In basis function space, classes can linearly be separated!

The image of the 1-D input data space is a 1-D **nonlinear manifold**

separating hyperplane

Data Represented in Feature Space

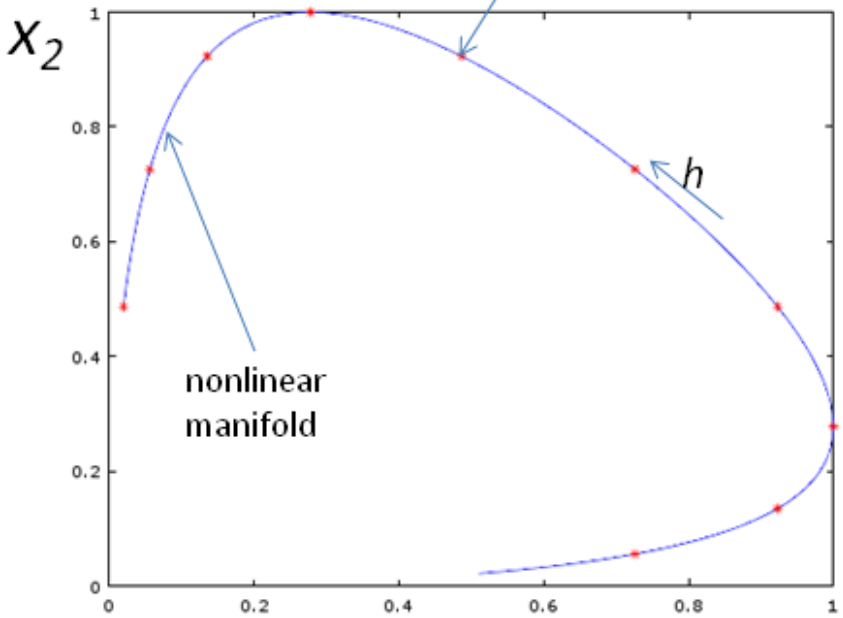
- Often the situation is different: there is a low-dimensional data space with $\mathbf{h} \in \mathbb{R}^{M_h}$ but the observed data are in a high-dimensional feature space $\mathbf{x} \in \mathbb{R}^{M}$
- Note: I keep M for the dimension of the input space with $\mathbf{x} \in \mathbb{R}^{M}$ and $\mathbf{h} \in \mathbb{R}^{M_h}$ for the latent space, which we cannot access
- Thus in the observed input space, the data are distributed on a manifold



feature functions

h *is unobserved*

h



The data is available in feature space ***x***

M: The dimension in feature space is typically large

x_1

Data Represented in Some Feature Space

- We cannot measure data in the data generating space $\mathbf{h} \in \mathbb{R}^{M_h}$
- Instead, we measure data in feature space $\mathbf{x} \in \mathbb{R}^{M}$ either supplied by nature or an application expert (feature engineering)
- Features map

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

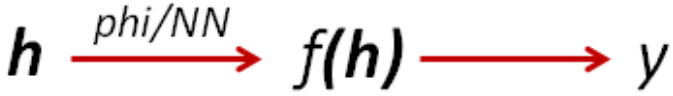
- We say that the data lies in an M_h-dimensional data manifold in $\mathbf{h} \in \mathbb{R}^{M_h}$ but is observed in an M-dimensional feature space $\mathbf{x} \in \mathbb{R}^{M}$
- In the spirit of the discussion in the lecture on the Perceptron: The map from $\mathbf{h}$ to y might be low-dimensional and explainable, but we can only measure $\mathbf{x}$

Feature Engineering

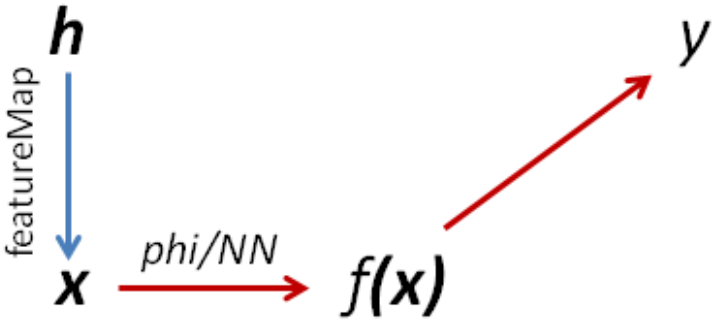
- In a way, features are like basis functions, but supplied by nature or an application expert (feature engineering)
- The definition of suitable features for documents, images, gene sequences, ... is a very active research area: feature engineering
- (One point of Deep Learning is that it does not require feature engineering!)

Overview

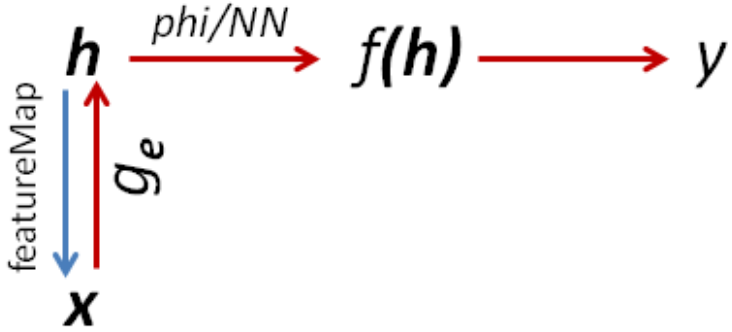
- 1: In (high-dimensional) feature space $\mathbf{x} \in \mathbb{R}^M$ the map might be simple and easy to model; after all, a feature map transforms a typically low-dimensional $\mathbf{h} \in \mathbb{R}^{M_h}$ to a high-dimensional $\mathbf{x} \in \mathbb{R}^M$, where the map to output might have low complexity (B, in next figure)
- 2: Place RBFs on the manifold; putting RBFs on the low dimensional data manifold is better than putting RBfs also in areas with no data
- 3: Learn an **encoder**, i.e., a mapping $\mathbf{x} \rightarrow \mathbf{h}$. This is called dimensionality reduction and might reduce the input to the essentials: input noise reduction. $\mathbf{h}$ can then be the input to the following neural network or basis function model. Sometimes the different dimensions of $\mathbf{h}$ can be interpreted! (C, D in next figure)
- 4: Learn a **generator**, i.e., a mapping $\mathbf{h} \rightarrow \mathbf{x}$. In autoencoders an encoder is learned together with a decoder and the decoder might serve as a generator. In the GANs models, a generator is learned directly. Now we can consider a novel $\mathbf{m}$ which generates a novel $\mathbf{x}$; if $\mathbf{x}$ represents images, the models contain convolutional layers and the generated image can be quite impressive!



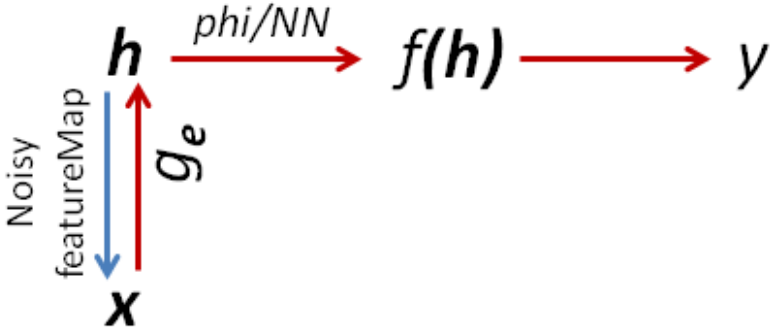
A: Lecture on basis functions and neural networks: we can measure ***h*** (there called ***x***) and perform a map using basis functions or neural networks



B: We can only measure the features ***x*** and not ***h***. We learn a map *f(**x**)*



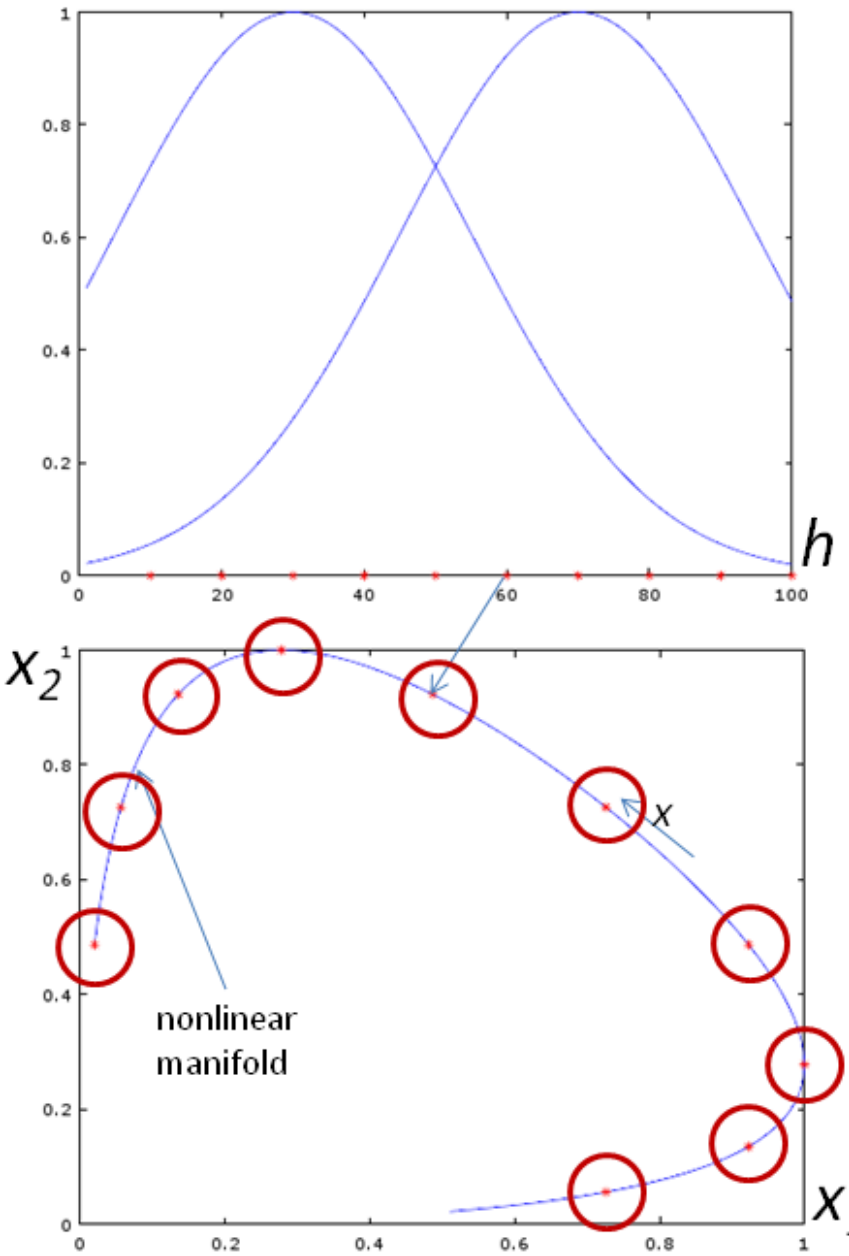
C: We can only measure the features ***x*** and not ***h***. We try to reconstruct ***h*** and then learn a map *f(**h**)*; g_e is the encoder



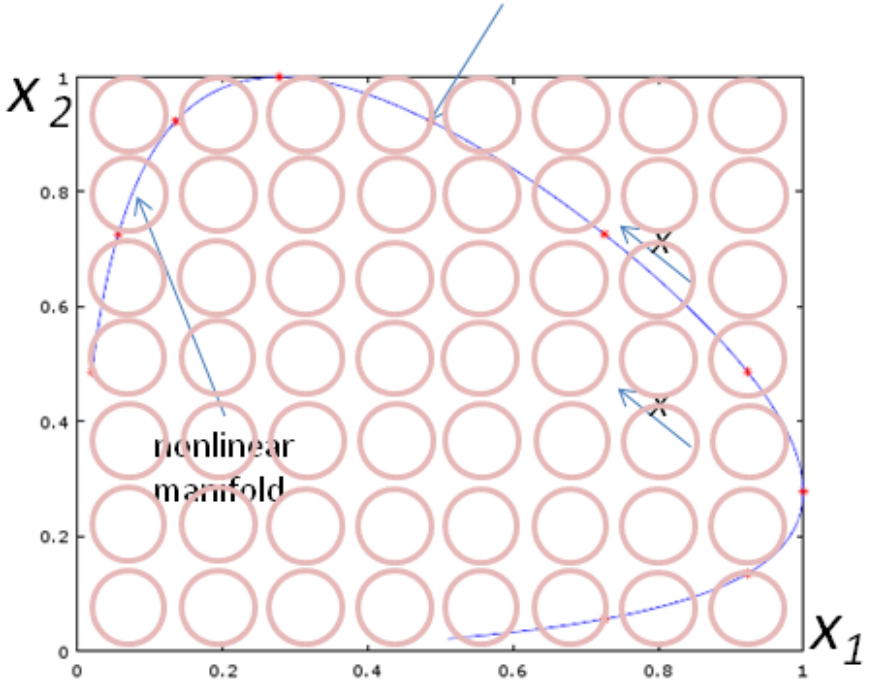
D: Same, but the feature map is noisy

2: Putting Resources on the Data Manifold ("Kernels")

- Assume $M_h < M$
- In this situation, it might work to put the RBFs on the data manifold, where we assume that the manifold can be represented by the training data points
- Use one RBF per data point. The centers of the RBFs are simply the data points themselves and the widths are determined via some heuristics (or via cross validation, see later lecture)



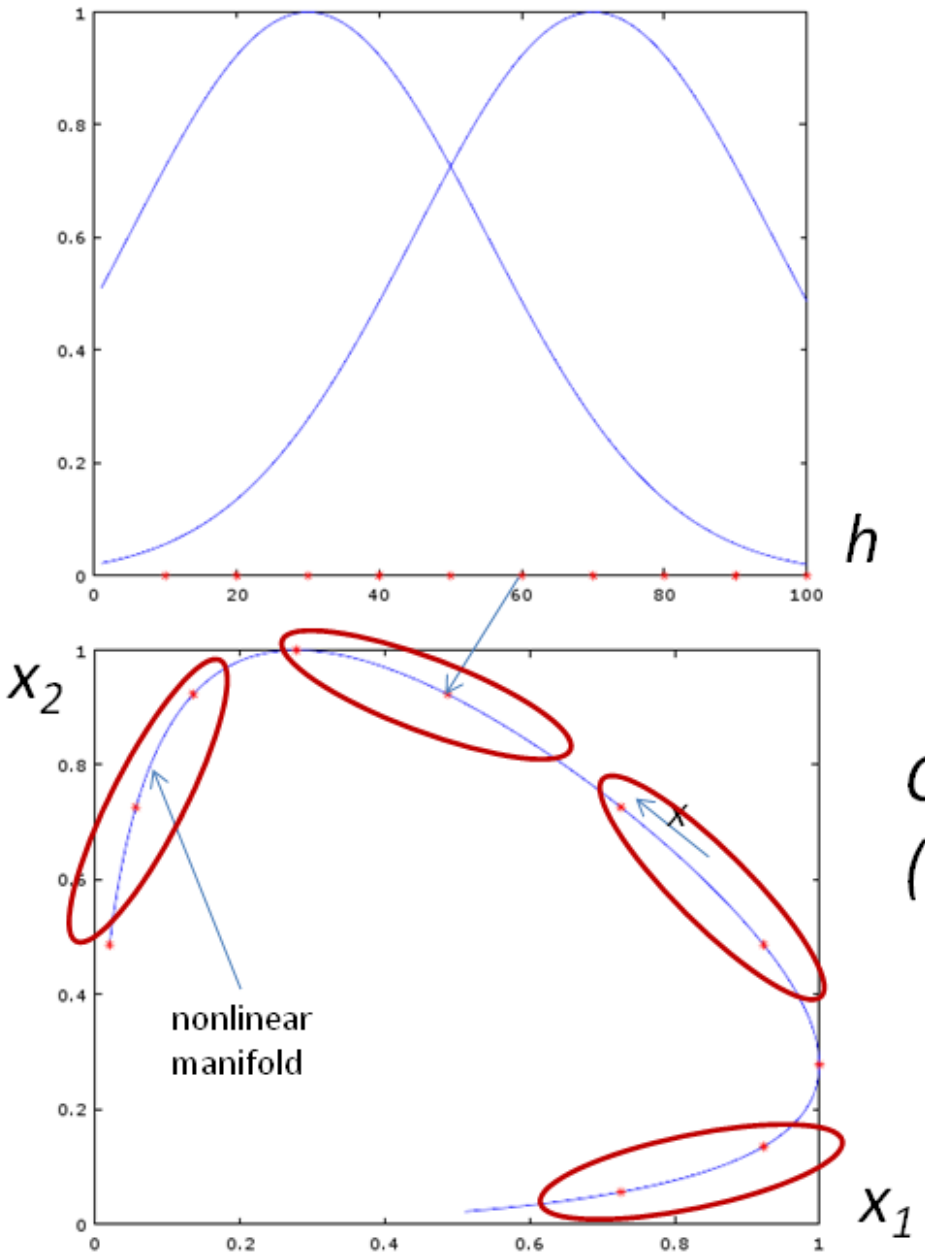
RBFs (kernels) with centers on data points in M-space



RBFs (kernels) uniformly placed

Clustering to Finding the Data Manifold

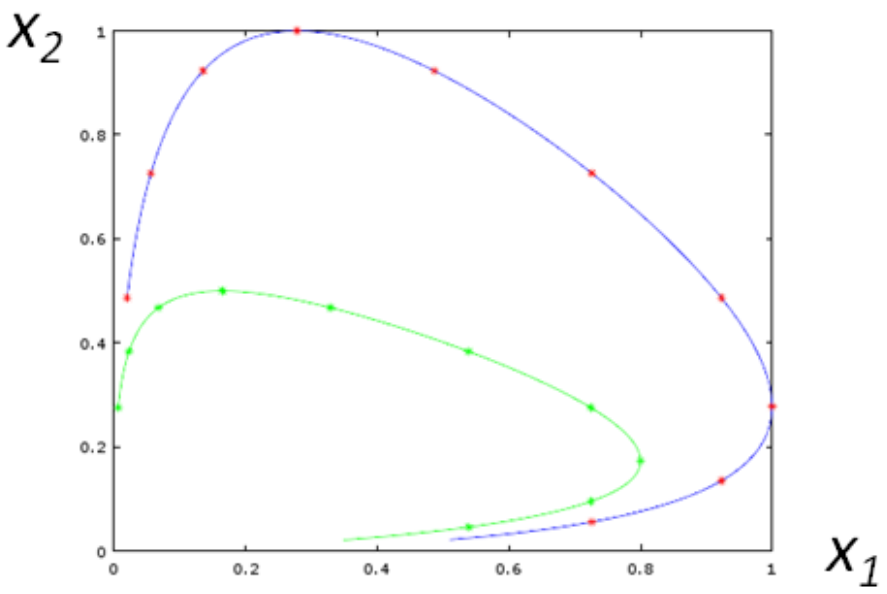
- It might be sensible to first group (cluster) data in feature space and to then use the cluster centers as positions for the Gaussian basis functions; the widths of the Gaussian basis functions might be derived from the sample covariance of the data in the cluster
- Even simpler: we assign a unique output label to each cluster
- Sometimes, data are clustered to start with



Clusters in x-space
(M-dimensional)

Class Specific Data Manifold

- There might be different manifolds for each class
- In latent space, the different classes might occupy different subspaces
- This assumption is the basis for tangent distance algoritm and tangent prop (beyond the scope of this lecture)



The manifold might be class specific with

$phi_m(h, class)$ or $phi_{m,\ class}(h)$

3: Learning an Encoder

- If we have,

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

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

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

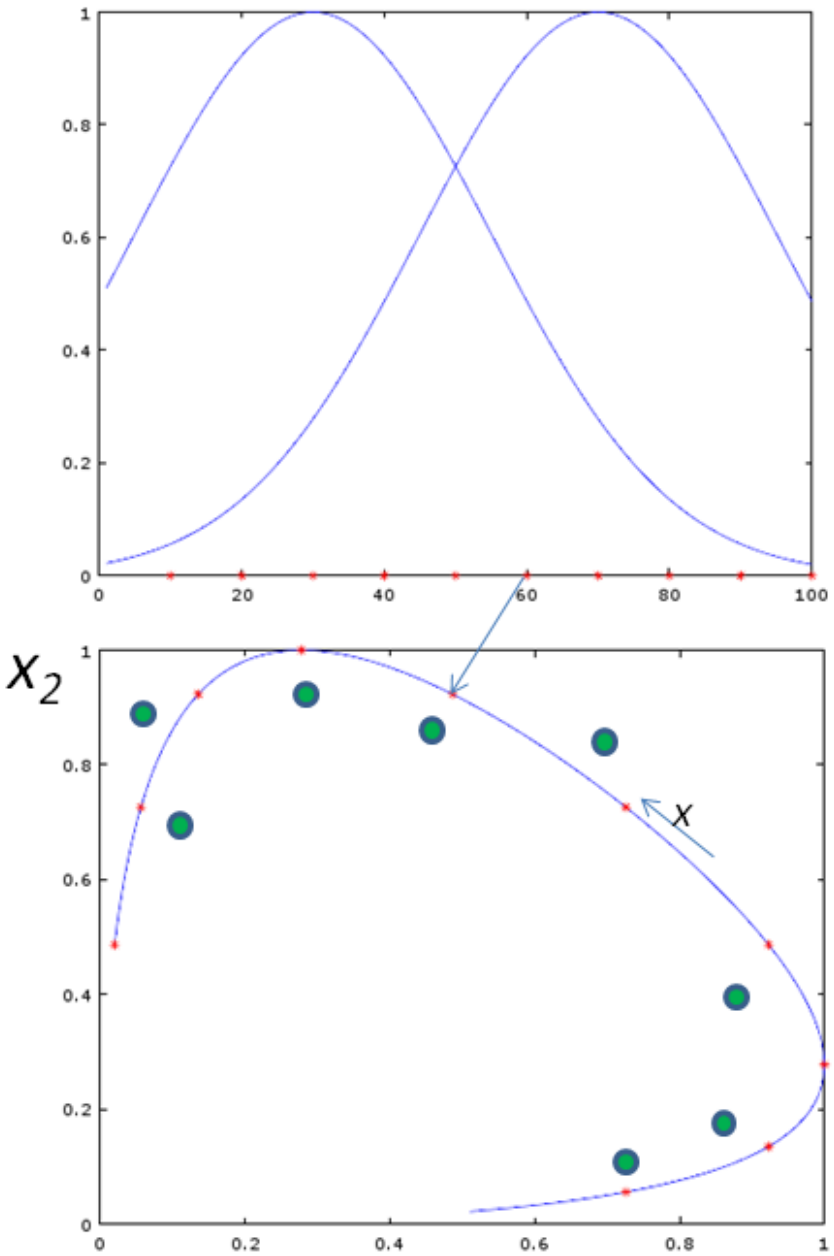
- $g_e(\mathbf{x})$ is called an encoder

Data Represented in a Noisy Feature Space

- Learning an encoder is especially important, when the feature map included some noise,

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

where $\vec{\delta}$ and $\vec{\epsilon}$ are noise vectors



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

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:

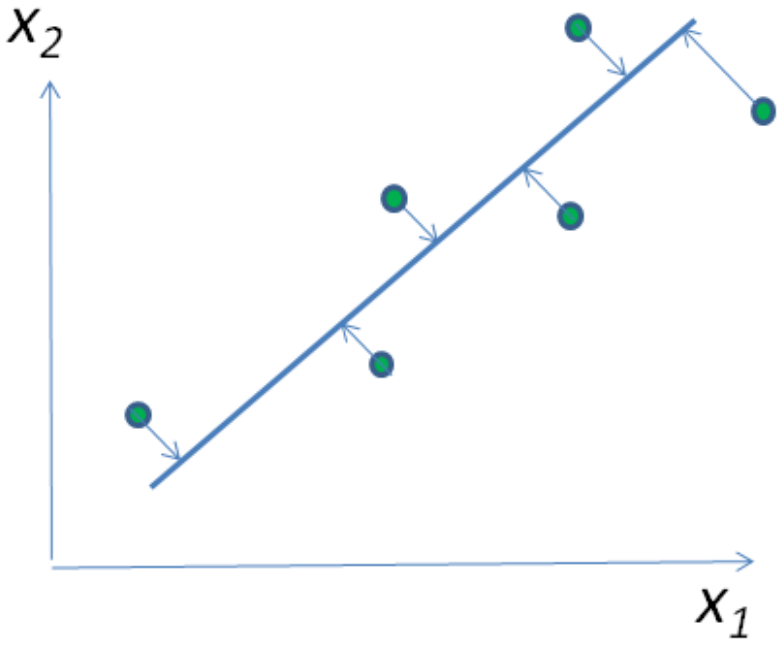
$$\mathbf{h} = g_e(\mathbf{x}) = \mathbf{V}_{M_h}^T \mathbf{x}$$

  Decoder:

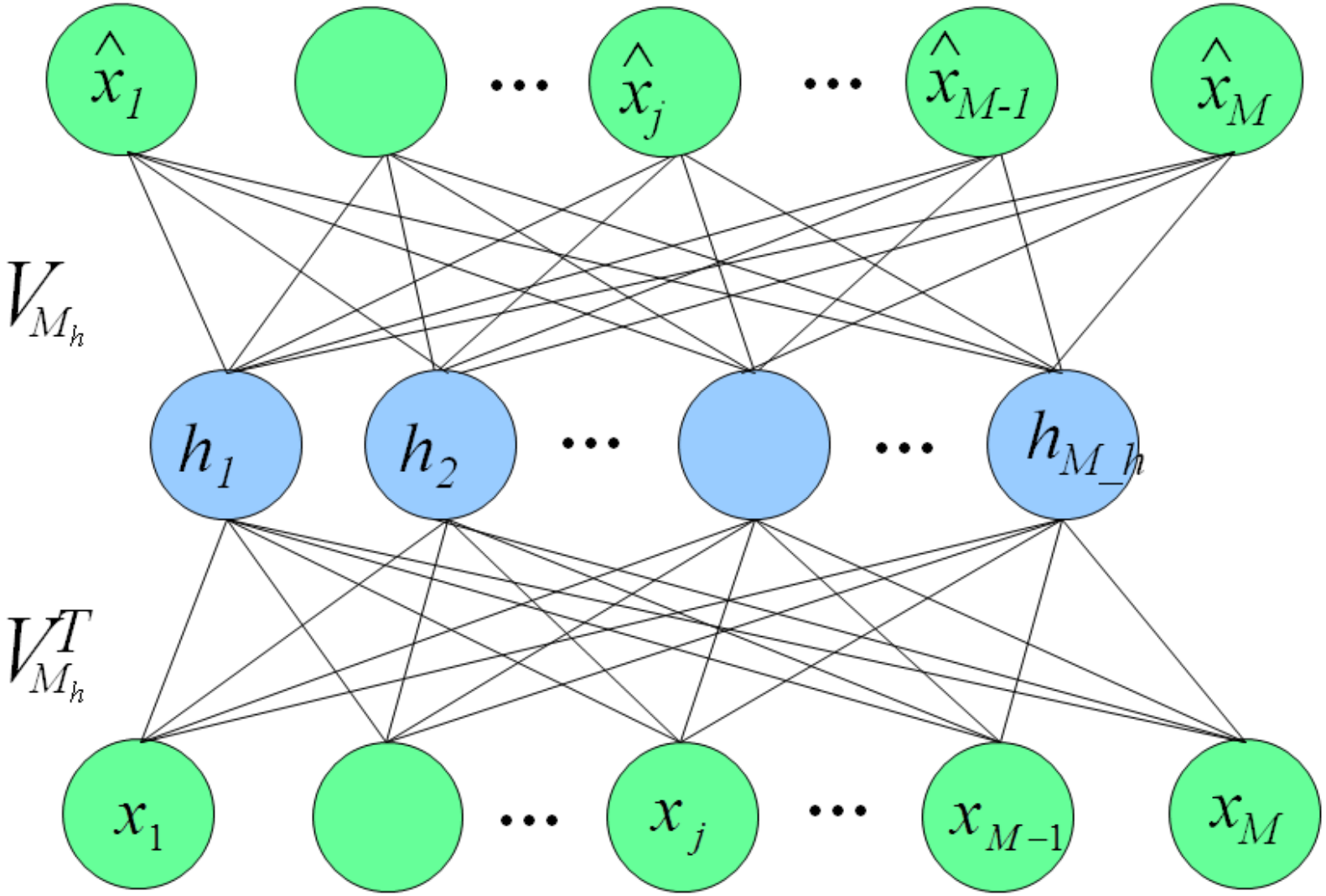
$$\hat{\mathbf{x}} = g_d(\mathbf{h}) = \mathbf{V}_{M_h} \mathbf{h} = \mathbf{V}_{M_h} \mathbf{V}_{M_h}^T \mathbf{x}$$

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

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

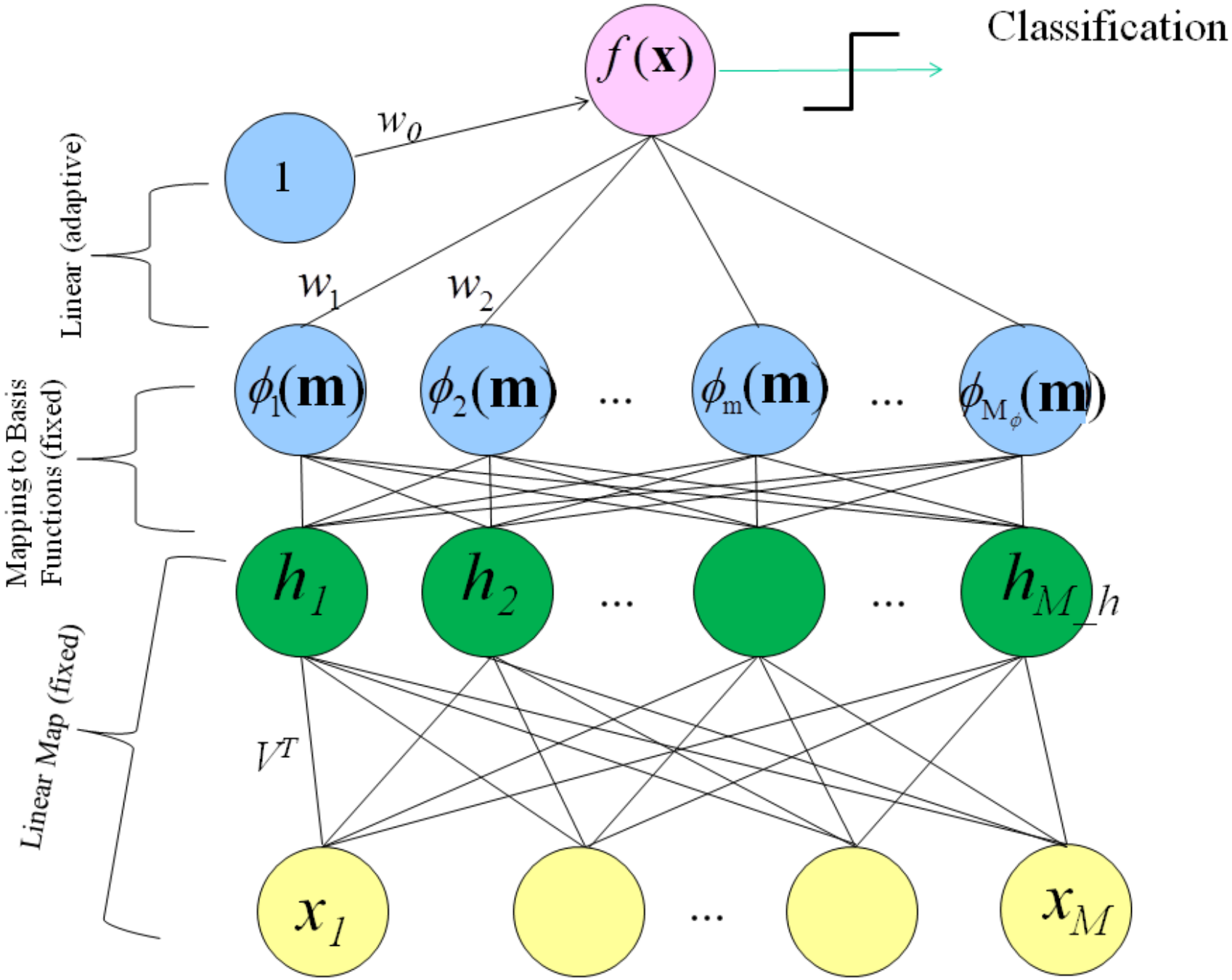


The manifold is a linear subspace


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

PCA as Dimensionality Reduction

- The PCA is often used as dimensionality reduction; the figure shows basis functions which are applied to the latent representation

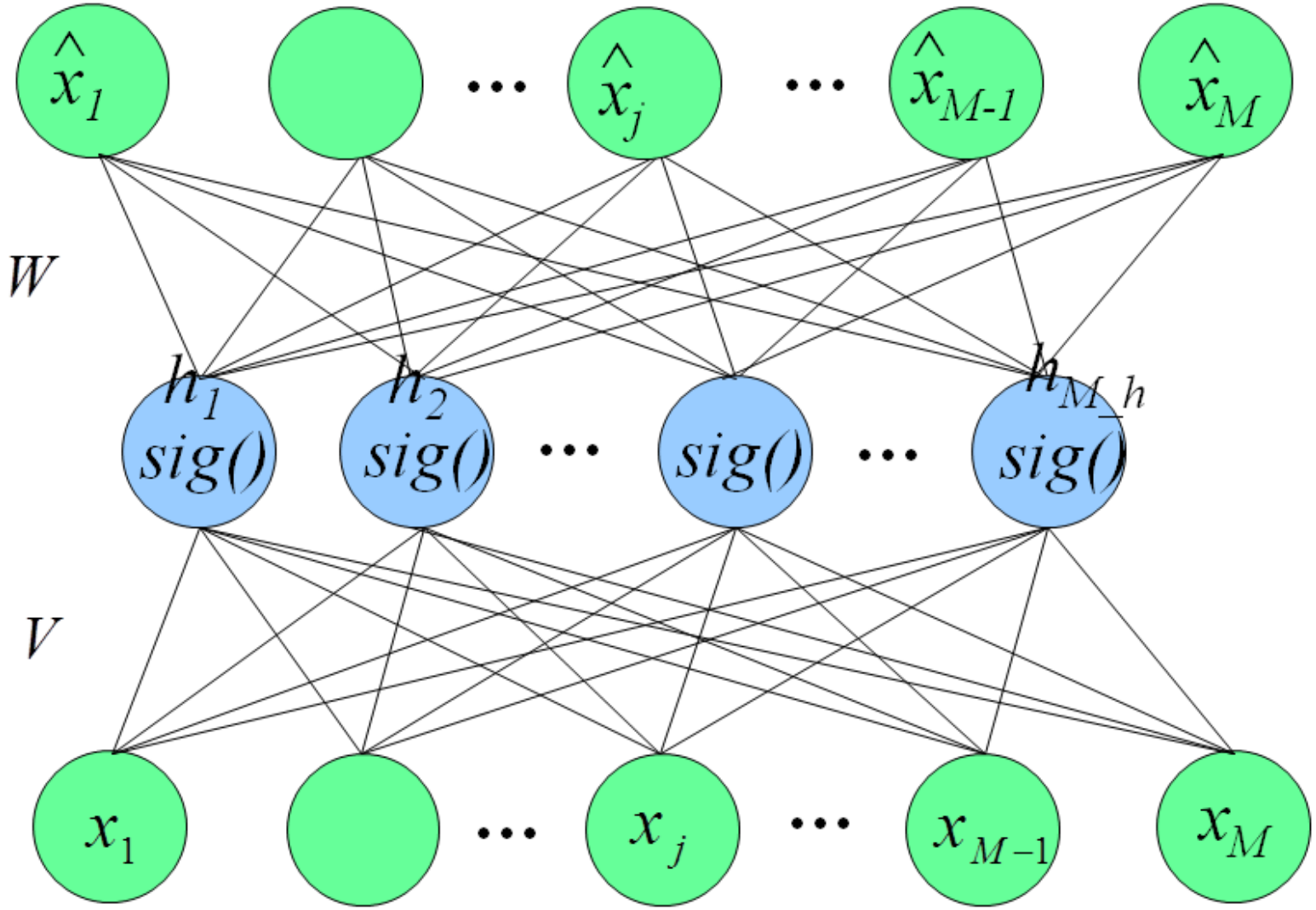

Classification
$f(\mathbf{x})$
w_0
1
Linear (adaptive)
w_1
w_2
$\phi_1(\mathbf{m})$
$\phi_2(\mathbf{m})$
...
$\phi_m(\mathbf{m})$
...
$\phi_{M_\phi}(\mathbf{m})$
Mapping to Basis Functions (fixed)
h_1
h_2
...
...
h_{M_h}
Linear Map (fixed)
V^T
x_1
...
x_M

Neural Network Autoencoder

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

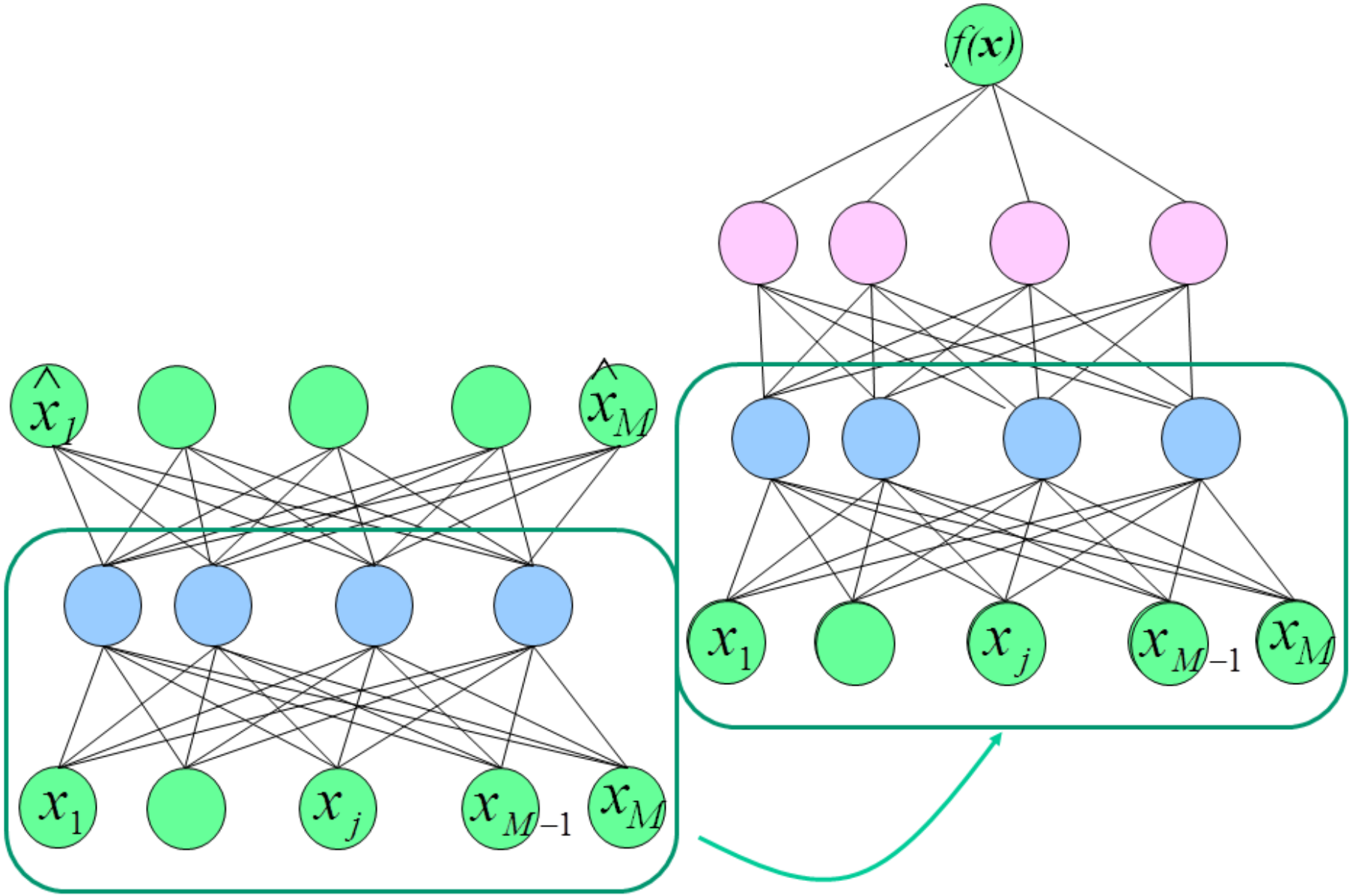
$$\text{cost}(W, V) = \sum_{i=1}^{N} \sum_{j=1}^{M} (x_{i,j} - \widehat{x}_{i,j})^2$$

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

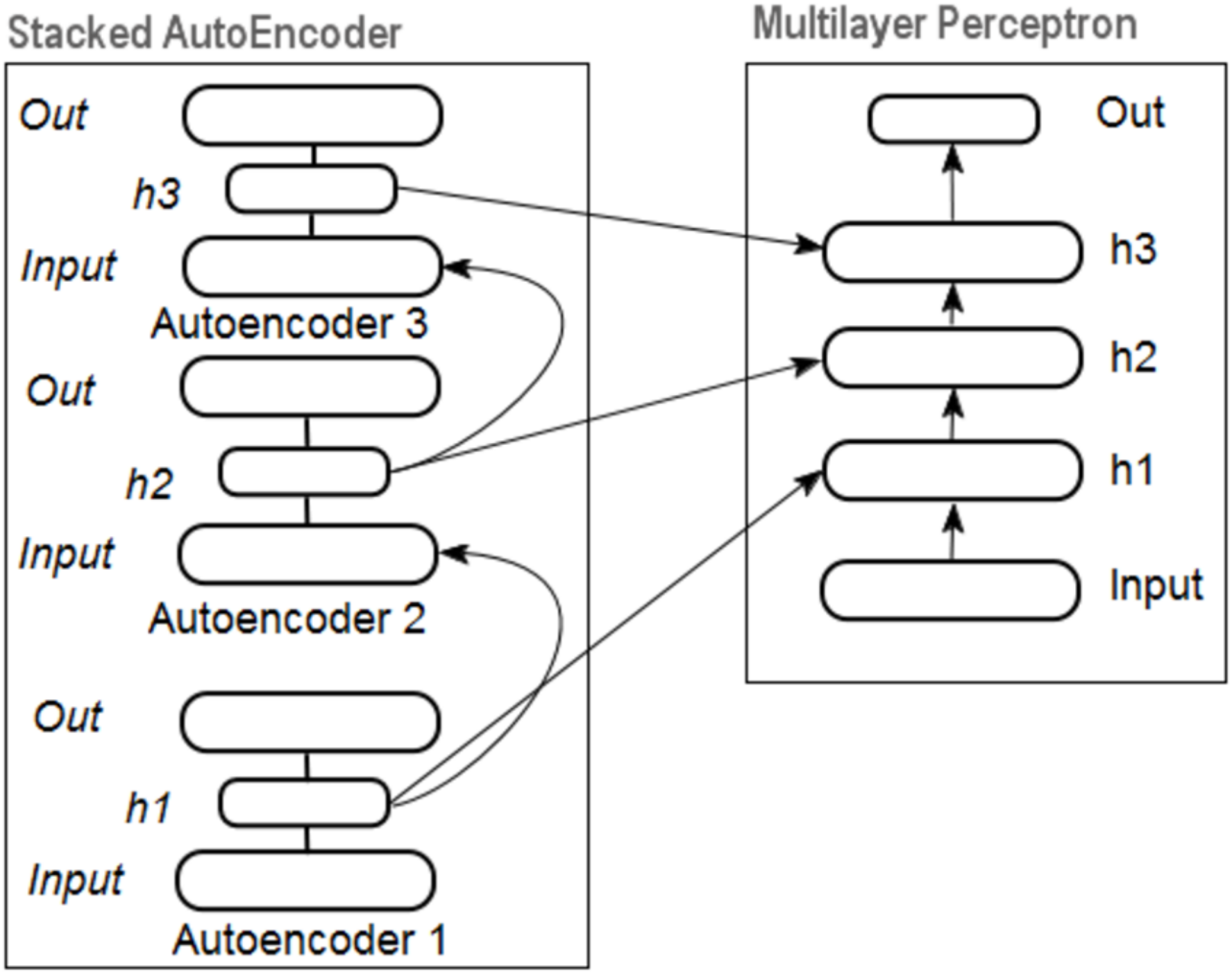
Comments and Applications

- 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

Denoising Autoencoder (DAE)

- Denoising autoencoder,

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

  where ϵ_i is random noise added to the input!

- Prevents an autoencoder from learning an identity function

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; the denoising autoencoder seems to have similar properties

4: Learning a Generator

- Note that the decoder part of an autoencoder can be used as a generator: simply select a new $\mathbf{h}$ and generate the corresponding data point $\mathbf{x}$
- This idea works better, when it is enforced that the latent representations for the training data are approximately Gaussian distributed: each dimension is independently Gaussian distributed, with unit variance
- This is the idea of the Variational Autoencoder (VAE)

VAE

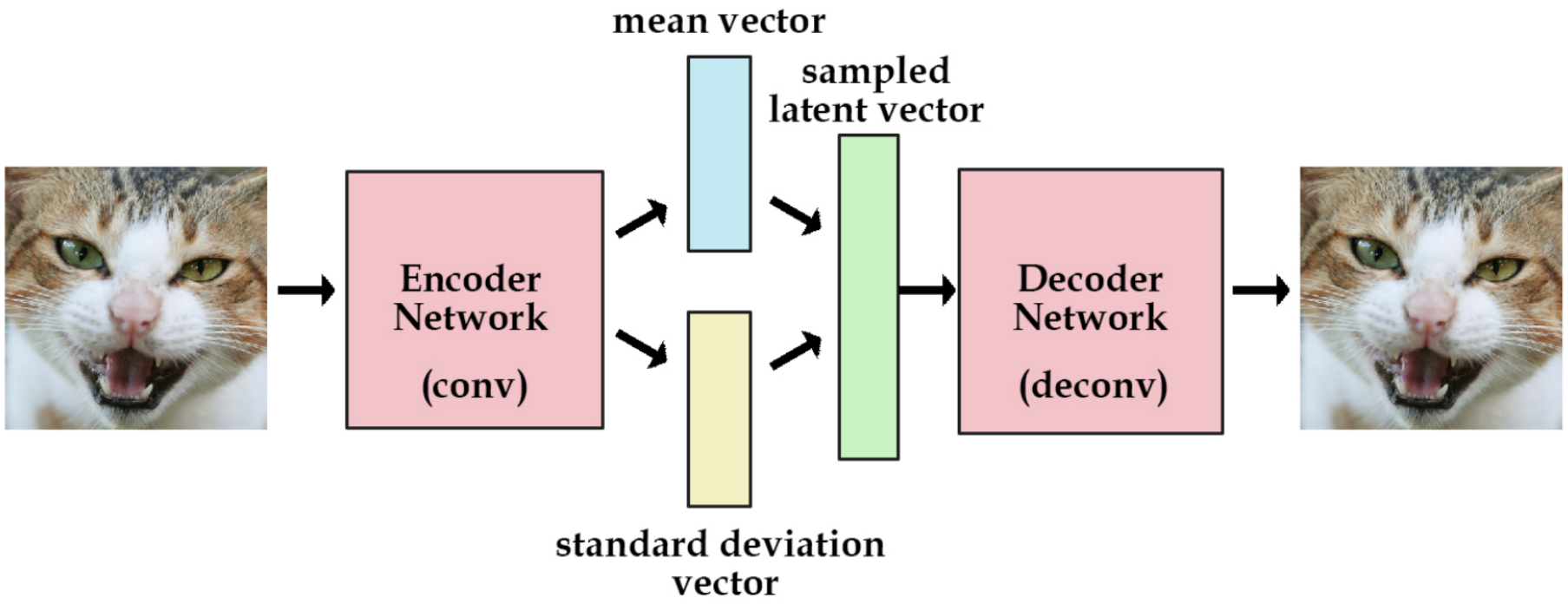
- After training (the details are beyond the scope of the lecture), we get approximately

$$h_k = \mathcal{N}(h_k; mean_k, \sigma_k^2)$$

  and we can easily generate samples from this distribution

Convolutional VAE

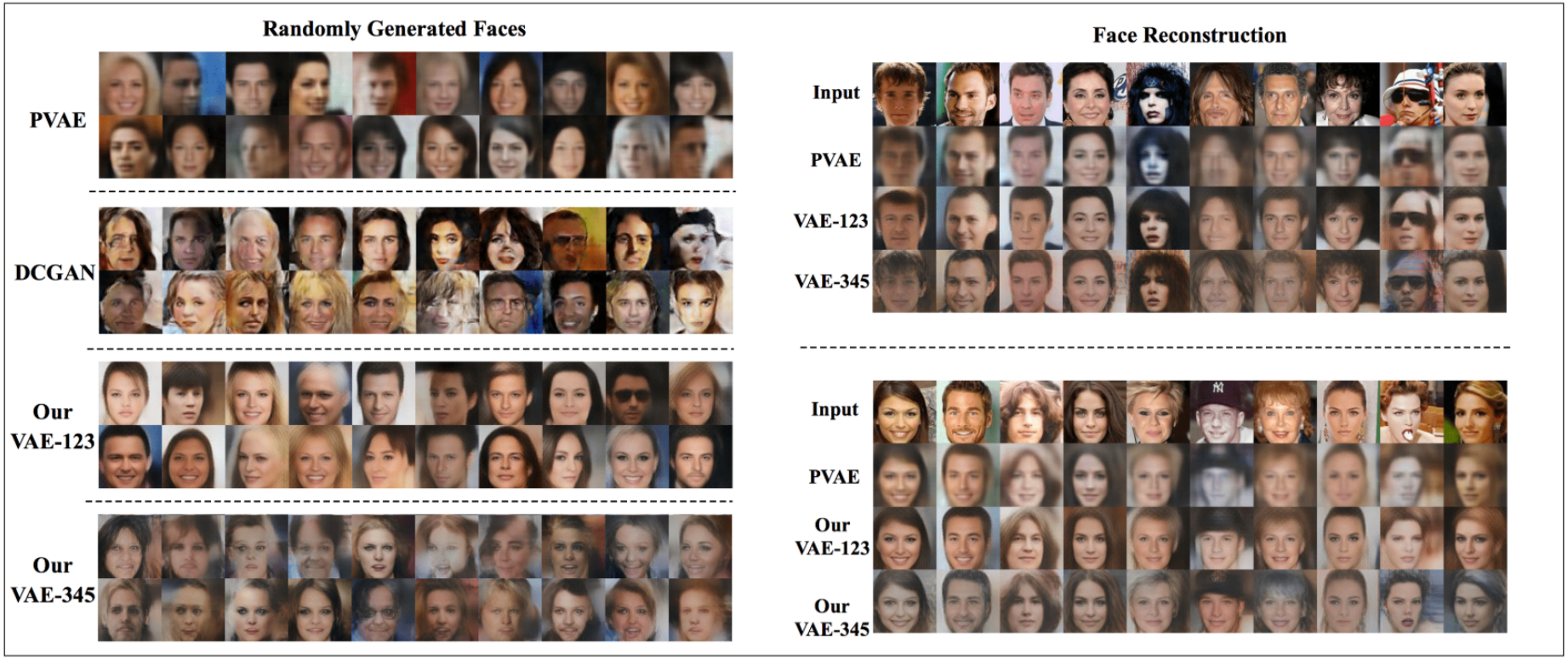
- The VAE contains convolutional layers in the encoder and deconvolutional layers in the decoder
- Deconvolution layer is a very unfortunate name and should rather be called a transposed convolutional layer
- A convolutional VAE can generate quite realistically looking images
- Often the different latent dimensions have a real world interpretation


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

left: 1st epoch, middle: 9th epoch, right: original

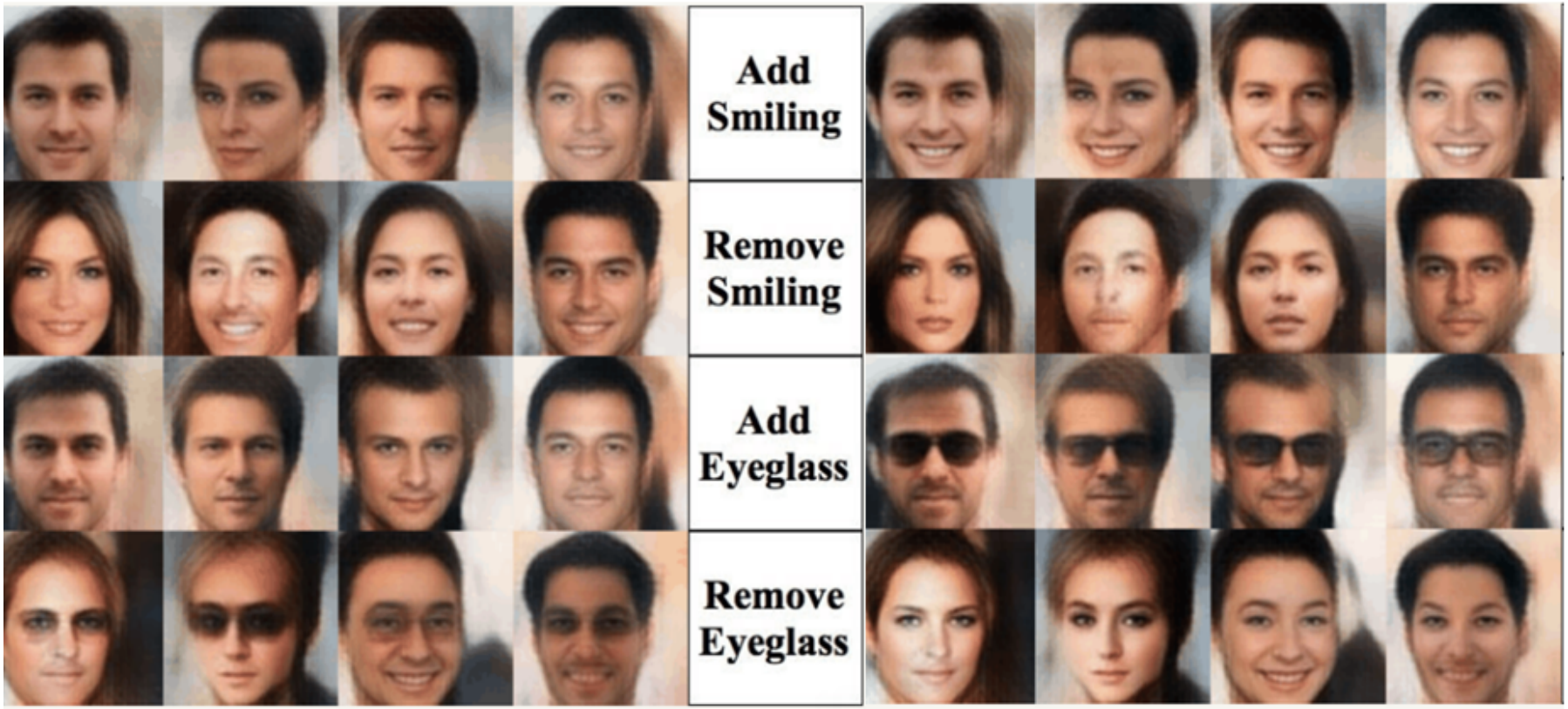
Generating Faces

- The next figures show applications to face images
- PVAE: Plain Variational Autoencoder trained with pixel-by-pixel loss in the image space
- DCGAN: Deep Convolutional Generative Adversarial Networks
- VAE 123, VAE 345: Instead of using pixel-by-pixel loss, deep feature consistency between the input and the output of a VAE is enforced (by using layers relu1_1, relu2_1, relu3_1 and relu3_1, relu4_1, relu5_1 respectively.)
- Source: Deep Feature Consistent Variational Autoencoder Xianxu Hou, Linlin Shen, Ke Sun, Guoping Qiu


Randomly Generated Faces
PVAE
DCGAN
Our VAE-123
Our VAE-345
Face Reconstruction
Input
PVAE
VAE-123
VAE-345
Input
PVAE
Our VAE-123
Our VAE-345

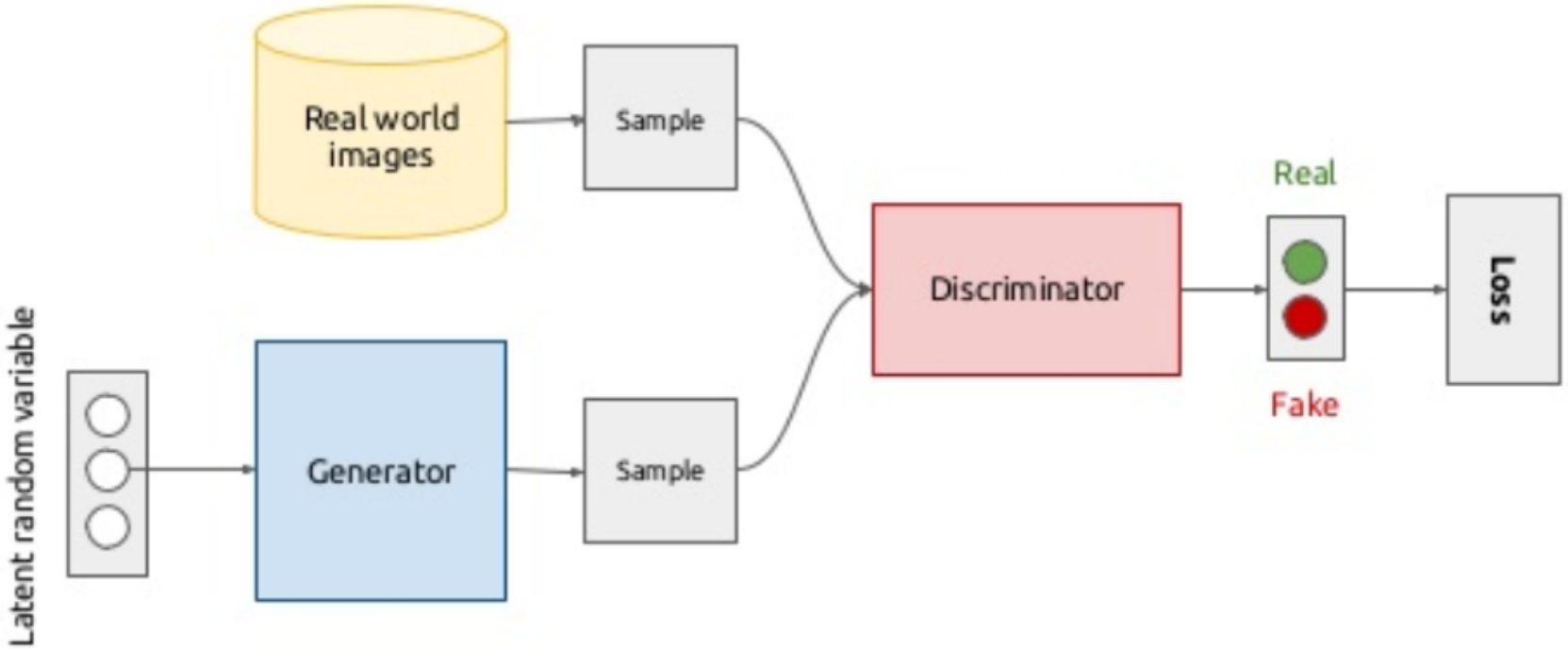
Manipulating Faces

- We randomly choose 1,000 face images with eyeglass and 1,000 without eyeglass respectively from the CelebA dataset
- The two types of images are fed to our encoder network to compute the latent vectors, and the mean latent vectors are calculated for each type respectively, denoted as $z_{pos\_eyeglass}$ and $z_{neg\_eyeglass}$
- We then define the difference $z_{pos\_eyeglass} - z_{neg\_eyeglass}$ as eyeglass-specific latent vector $z_{eyeglass}$. In the same way, we calculate the smiling-specific latent vector $z_{smiling}$
- Then we apply the two attribute-specific vectors to different latent vectors z by simple vector arithmetic like $z + \alpha z_{smiling}$ or $z + \alpha z_{eyeglass}$


Add Smiling
Remove Smiling
Add Eyeglass
Remove Eyeglass

Generative Adversarial Networks (GANs)

- Can we train a generator without an autoencoder, i.e., without an encoder?
- Let's assume we have a larger number of generators available; which one is the best one? Let's assume that each generator generates a data set
- The best generator might be the one where a discriminator (i.e., a binary neural network classifier) trained to separate training data from the data from a particular generator, cannot separate both. Then one might say that approximately $P_{train}(\mathbf{x}) = P_{gen}(\mathbf{x})$
- 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

- The discriminator is simply trained to **maximize** the negative cross entropy cost function; the targets for the training data are 1 and for the generated data 0
- The generator is simply trained to **minimize** the negative cross entropy cost function, where backpropagation is performed via the discriminator (zero-sum game)
- Optimal parameters are

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

where

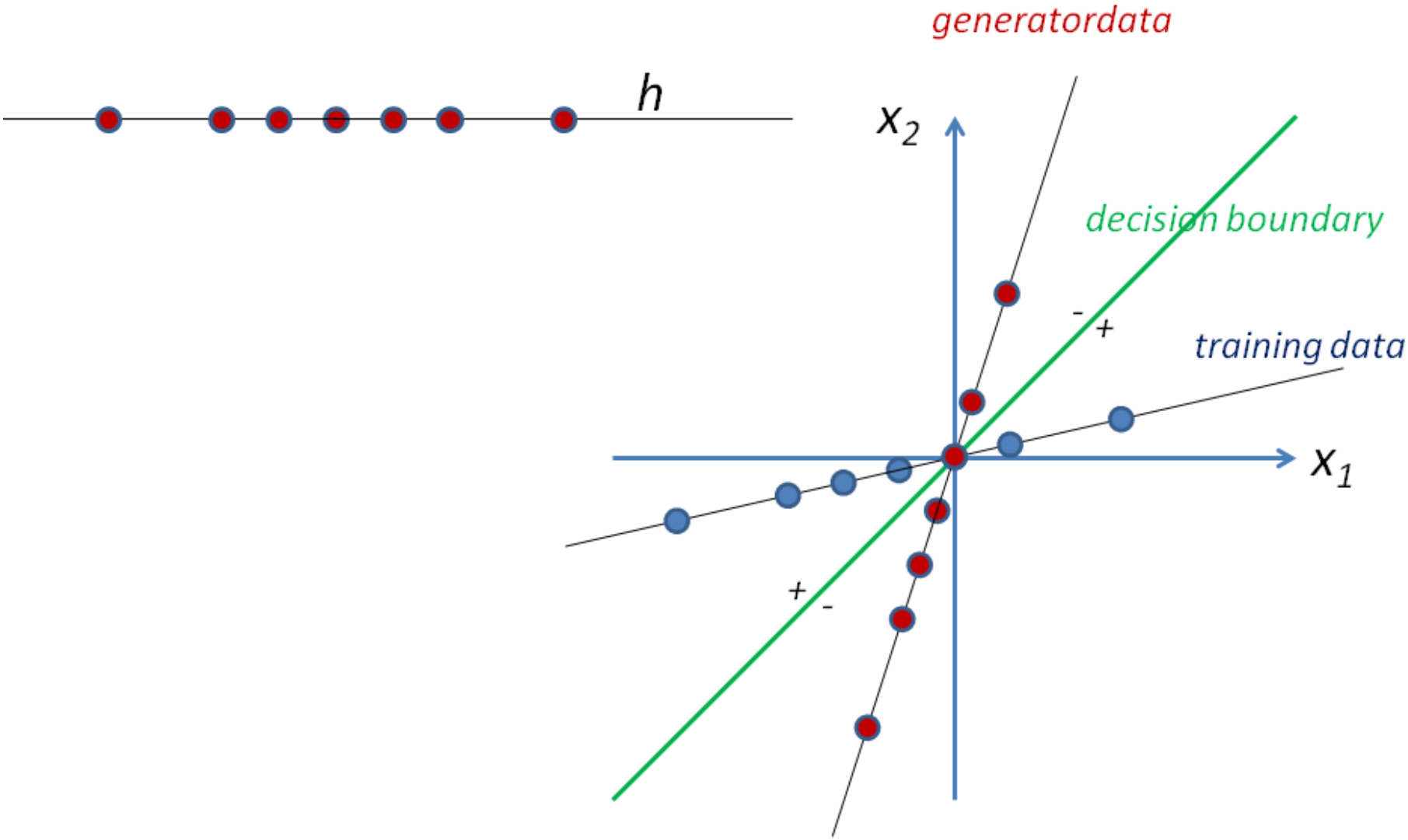
$$\text{cost}(\mathbf{w}, \mathbf{v}) = \sum_{\mathbf{x}_i \in \text{train}} \log g_{dis}(\mathbf{x}_i, \mathbf{w}) + \sum_{\mathbf{x}_i \in \text{gen}} \log[1 - g_{dis}(g_{gen}(\mathbf{h}_i, \mathbf{v}), \mathbf{w})]$$

Uninformative or Informative Information

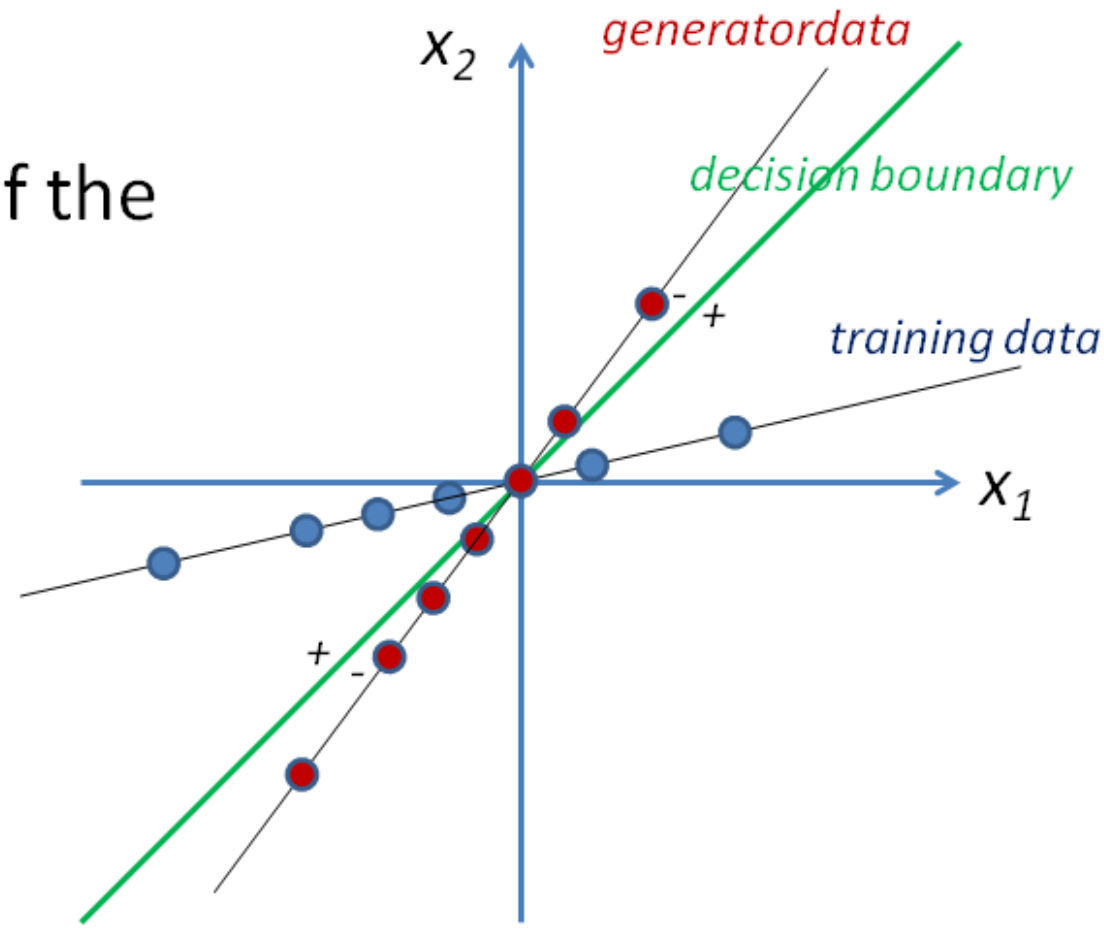
- If the generator is close to perfect, the discriminator is not able to separate the two classes
- The generator obtains rich gradient information

Illustration

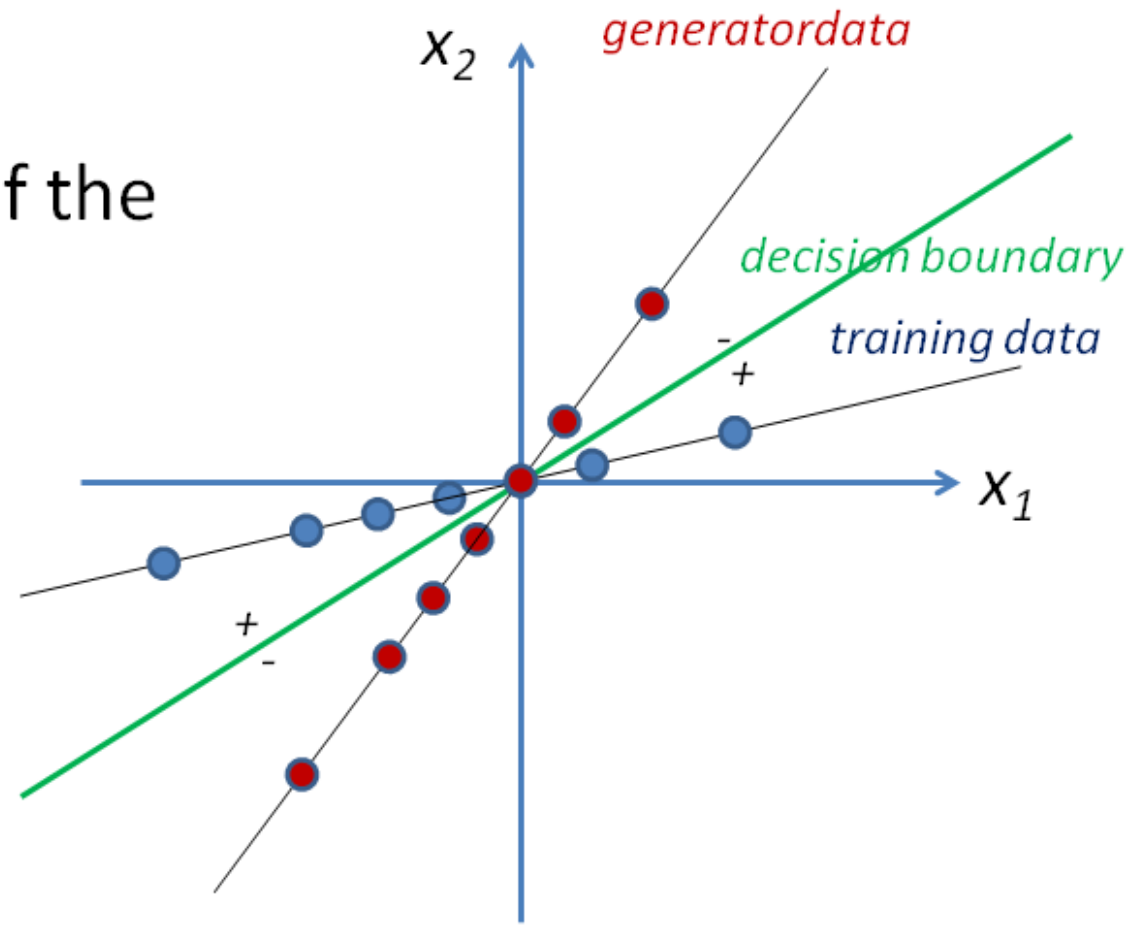
- Consider the following figure; h is one-dimensional Gaussian distributed: $P(h) = N(h; 0, 1)$, $M_h = 1$
- The generator is $\mathbf{x} = h\mathbf{v}$, where $M = 2$; the data points are on a 1-D manifold in 2-D space; here: $v_1 = 0.2$, $v_2 = 0.98$
- The training data are generated similarly, but 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
-
+
+
-

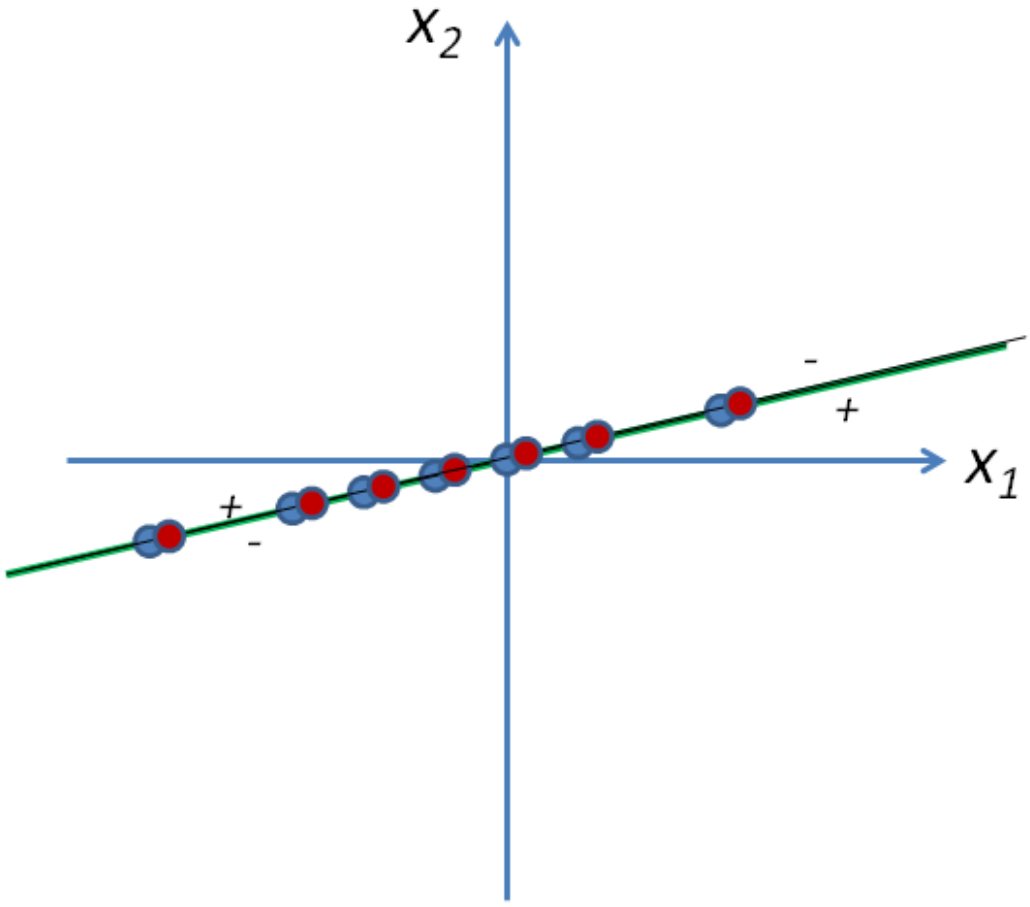
After update of the generator



After update of the discriminator



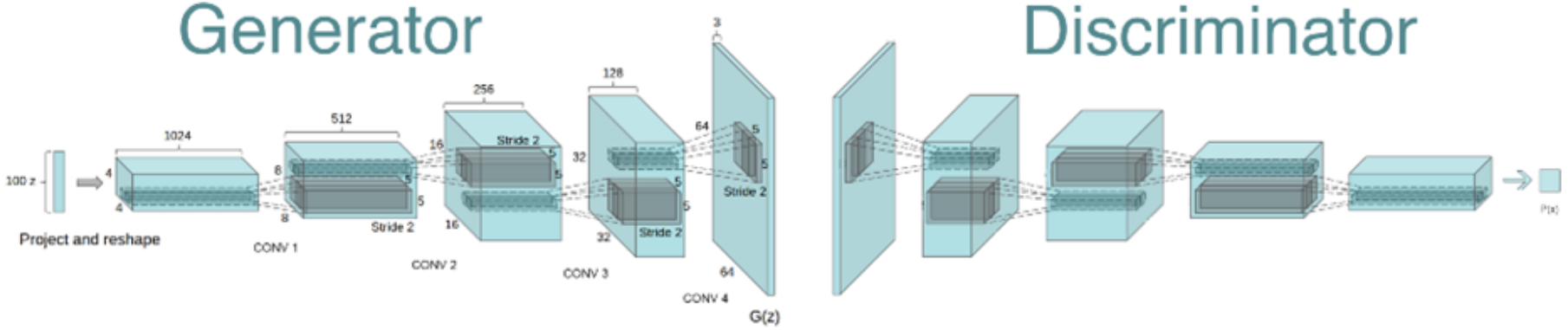
Convergence



DCGAN

- Deep Convolutional GAN (DCGAN): the generator and the discriminator contain convolutional layers

DCGAN


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

cGAN and InfoGan

- Conditional GAN (cGAN): An additional input to the generator and the discriminator is the class label
- InfoGan: An additional input to the generator is the class label. The dicriminator also predicts the class label

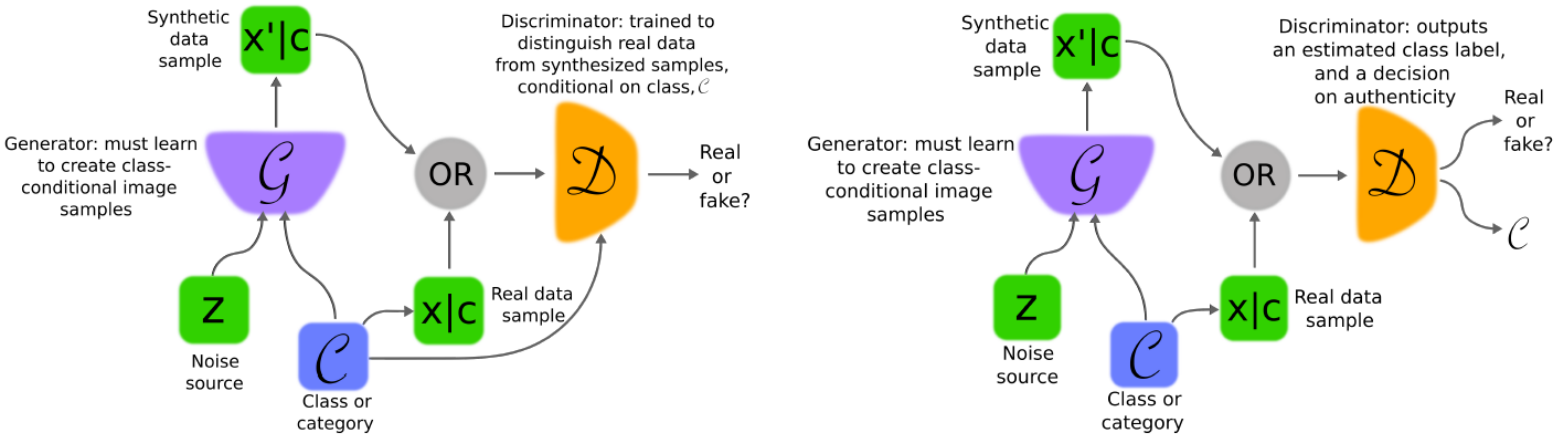
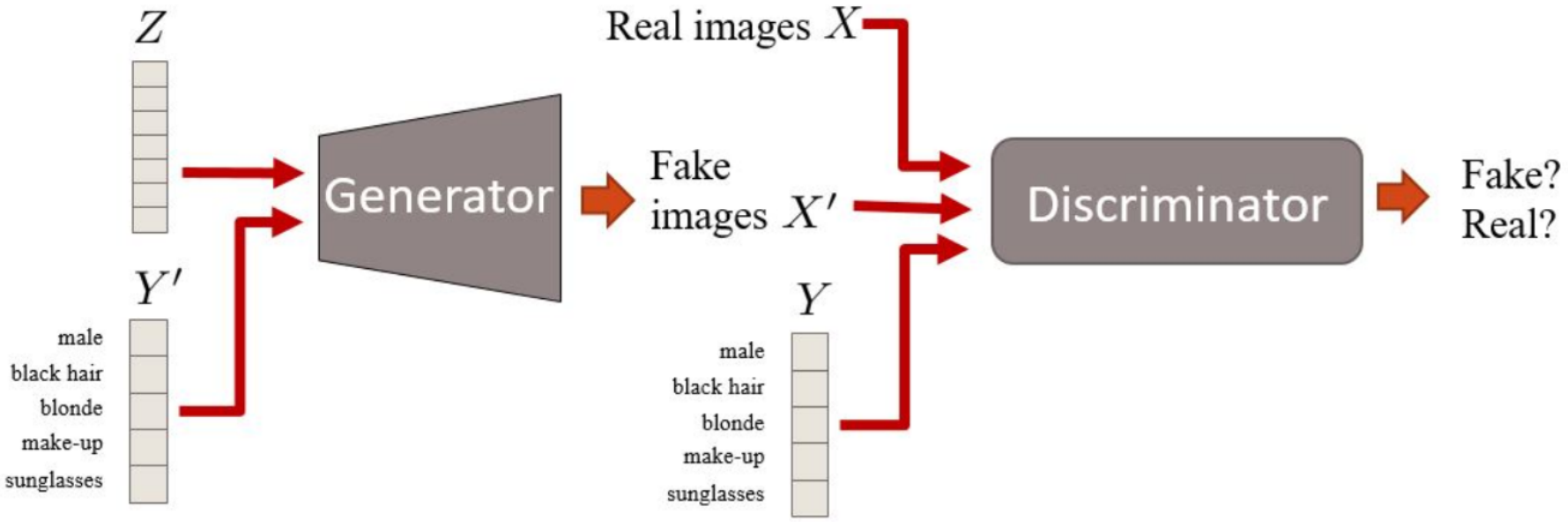


Fig. 3. Left, the Conditional GAN, proposed by Mirza et al. [15] performs class-conditional image synthesis; the discriminator performs class-conditional discrimination of real from fake images. The InfoGAN (right) [16], on the other hand, has a discriminator network that also estimates the class label.



Overview of a conditional GAN with face attributes information.

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

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

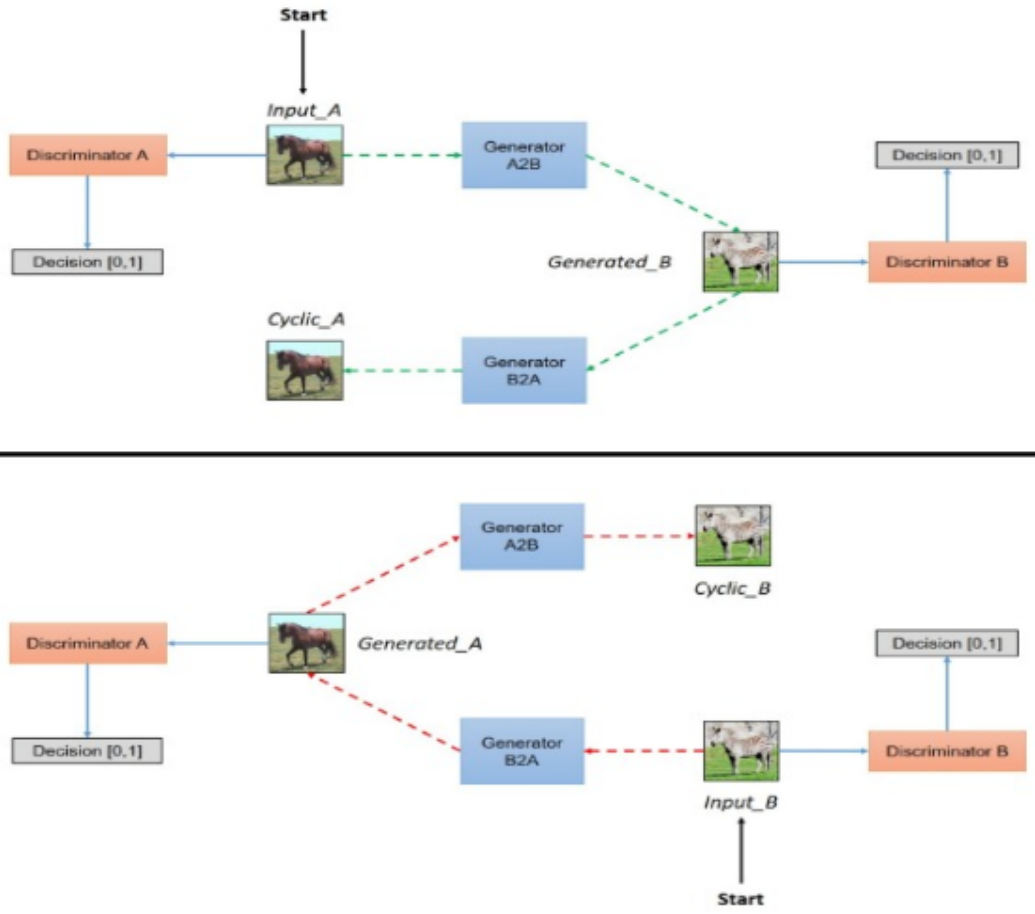
$$\hat{\mathbf{x}}_{zebra} = g_{A2B}(g_{B2A}(\mathbf{x}_{zebra}))$$

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

CycleGAN

- CycleGAN does exactly that
- CycleGAN adds two discriminators, trained with the *adversial 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 adversial loss are used in training

Understanding and Implementing CycleGAN



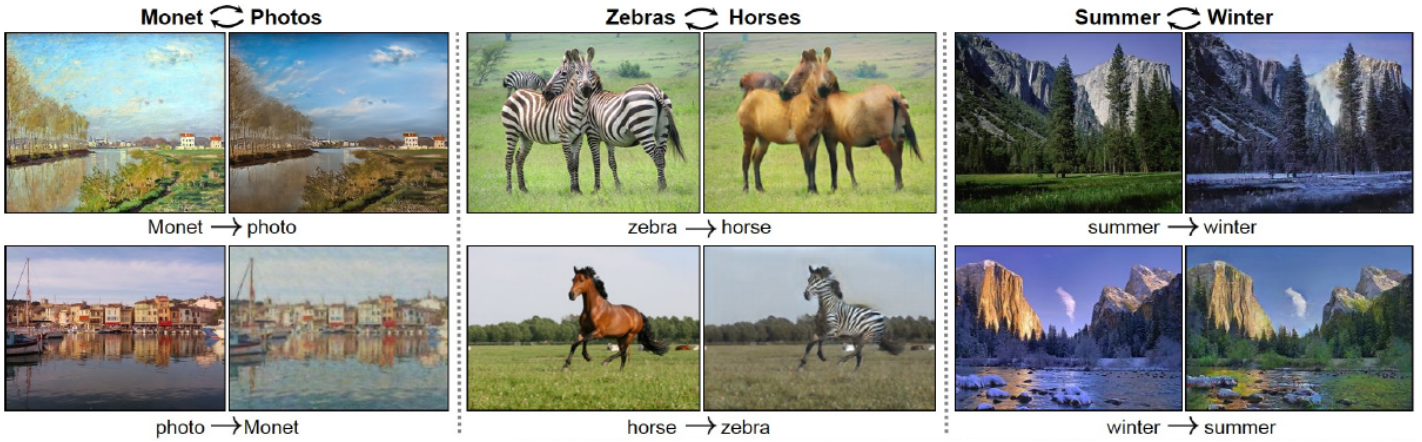


Fig. 8. CycleGAN model learns image to image translations between two unordered image collections. Shown here are the examples of bi-directional image mappings: Monet paintings to landscape photos, zebras to horses, and summer to winter photos in Yosemite park. Figure reproduced from [4].

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
- 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

General Comment

- In general, models for $P(\mathbf{x})$ can be used to generate new data (by sampling from the distribution) and to evaluate the likelihood of a new data point
- Gaussian mixtures, other mixture models
- Kalman filters, Markov models, hidden Markov models, Bayesian networks, Markov networks
- Deep Learning: Deep Boltzmann machines
- At this stage, GAN and VAE excel in generating realistic image samples

Appendix*

Distribution*

- Both the VAE decoder and a GAN generator produce 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 and points on the manifold get infinite density