Deep learning in GW physics

Second MaNiTou Summer School on Gravitational Waves:
A new window to the Universe

Natalia Korsakova

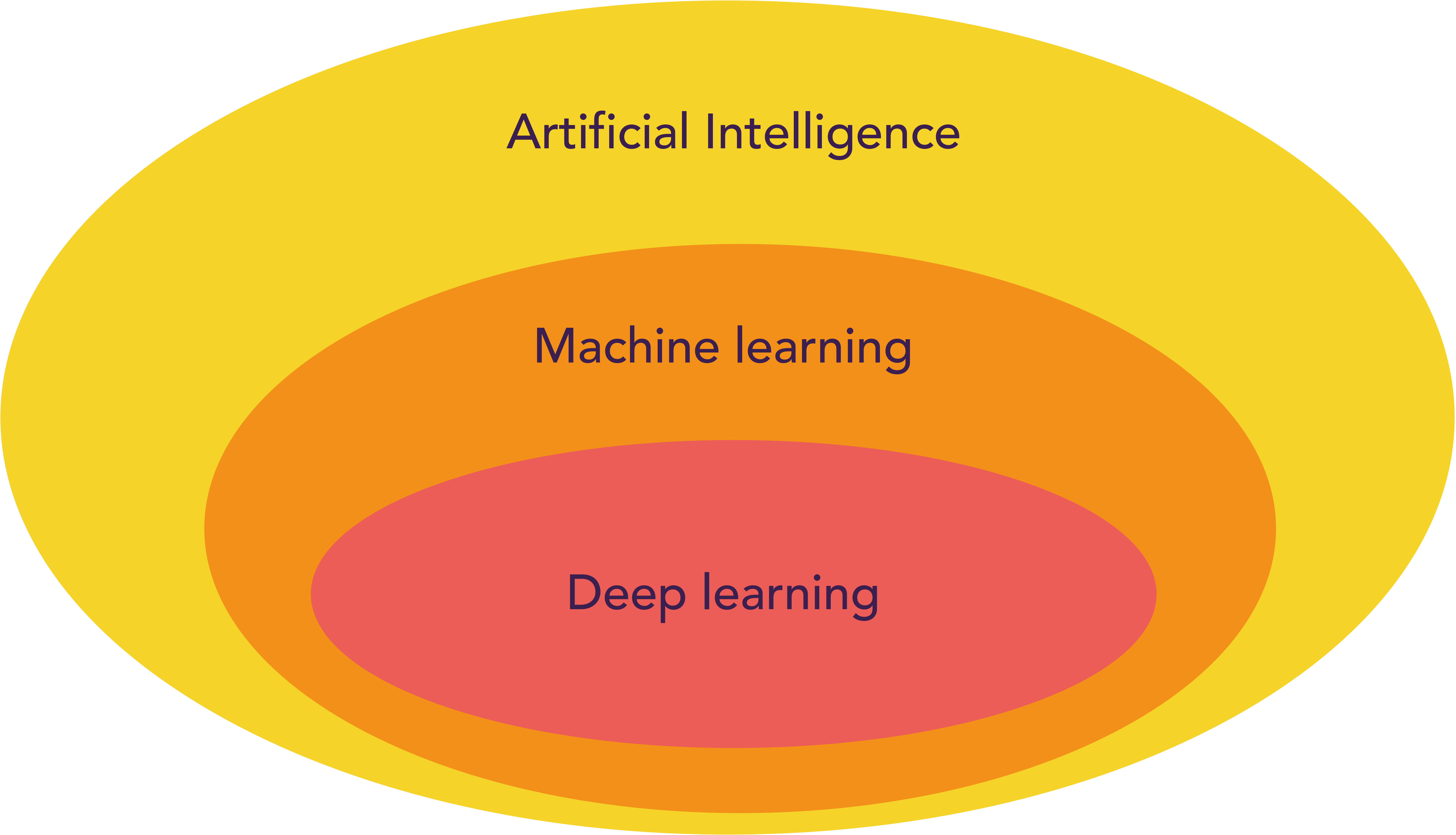
Laboratoire
AstroParticule & Cosmologie

APC, CNRS, FRANCE

Plan of the lecture

1. Basic introduction to Deep learning.
2. Detection and point parameter estimation.
3. Bayesian parameter estimation.
4. Waveform compression.
5. Tutorial

Introduction to deep learning


Artificial Intelligence
Machine learning
Deep learning

Supervised Learning

Regression

Classification

…

Unsupervised Learning

Clustering

Feature extraction

…

Reinforcement Learning

Solving dynamics

Playing games

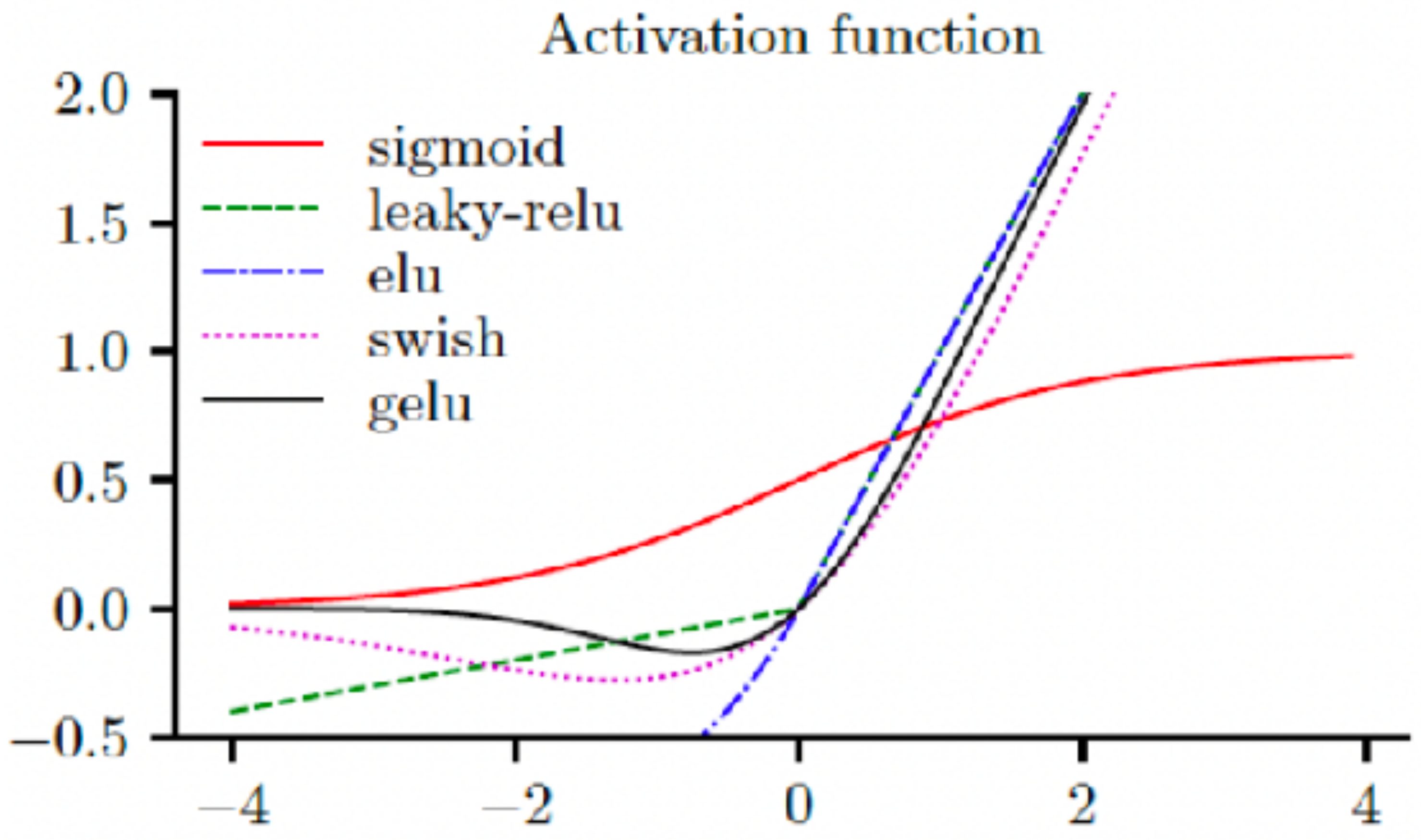
…

Deep learning: introduction

Deep neural network can be viewed as a universal function approximator.

The function is represented as a computational graph, where nodes are primitive operations and edges represent numeric data.

The building block of the network is an artificial **neuron**, which is a real-valued signal **y** computed by multiplying a vector-valued input signal **x** by a weight vector **w**, adding a bias term **b,** and then passing it through an **activation function** ϕ .



$$\phi\left(b + \sum_{i=1}^{n} w_i x_i\right) = \hat{y}$$

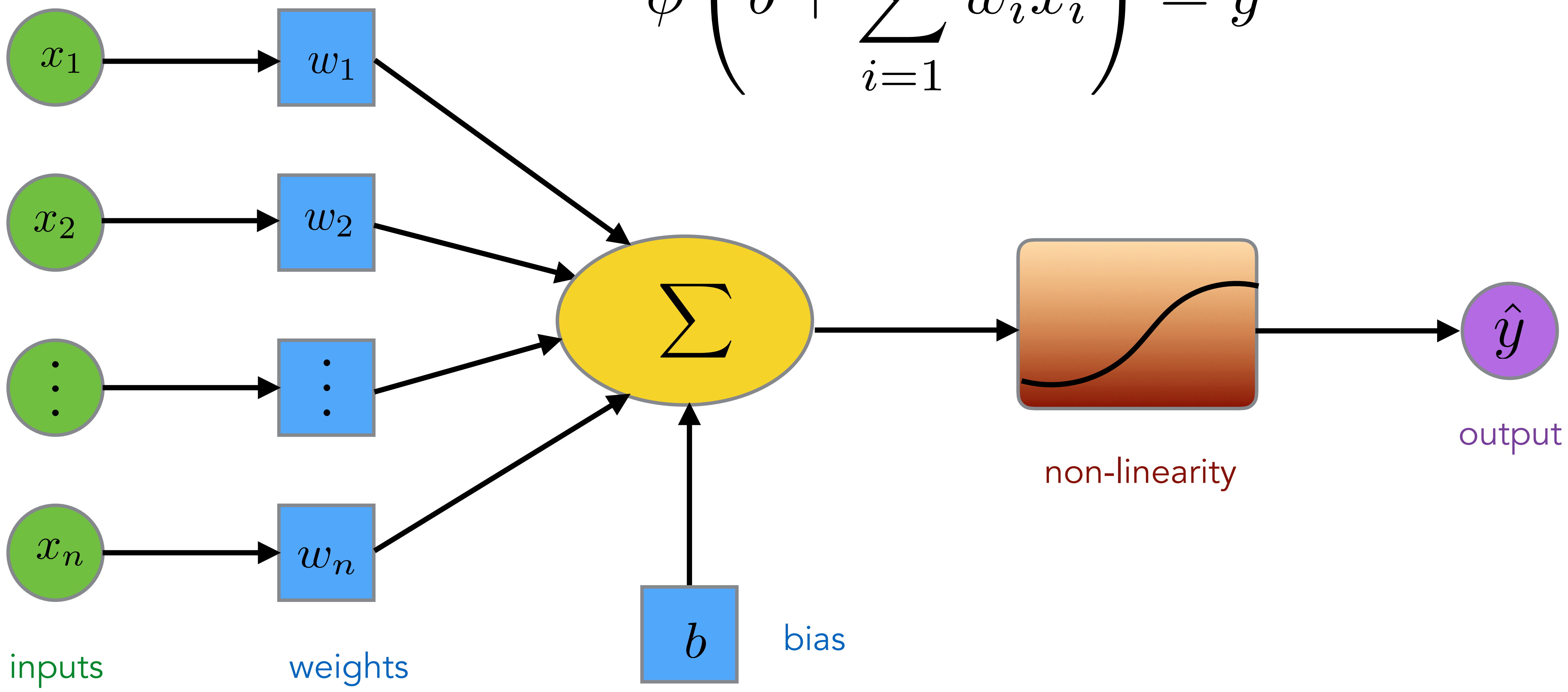
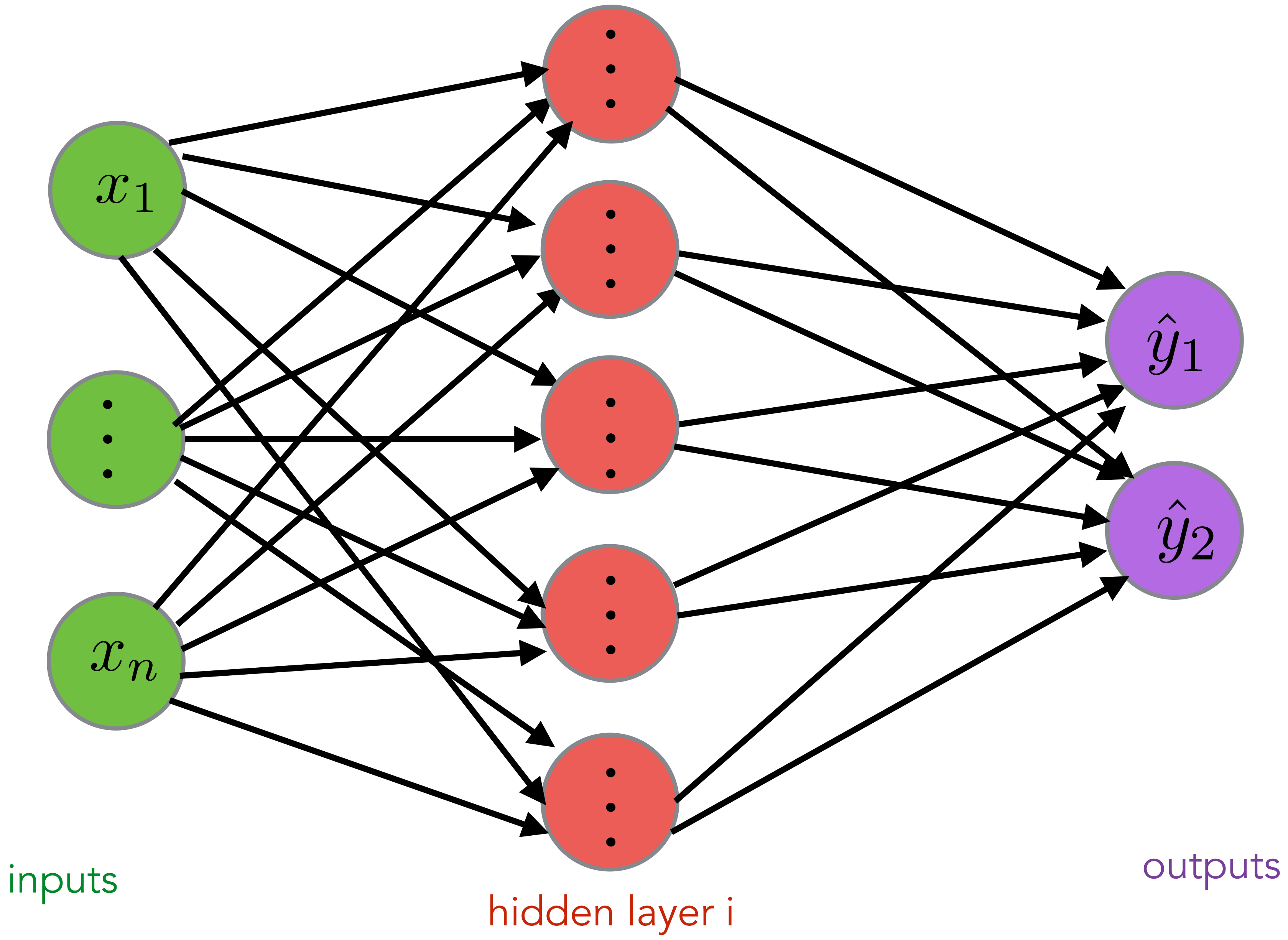


Image: Kevin Patrick Murphy, Probabilistic Machine Learning: An Introduction

Deep learning: introduction



Classical feedforward neural network defines a map

$$y = f(\boldsymbol{x}; \boldsymbol{\theta})$$

which is a composition of a simpler mappings

$$f = f^{(d)} \circ f^{(d-1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}$$

The networks is usually represented by an acyclic graph that describes how the functions are composed together.

Each layer cam be written in the form $f^{(j)}(\boldsymbol{h}) = \sigma_j \left(\boldsymbol{W}_j^\top \boldsymbol{h} + \boldsymbol{b}_j \right)$

The network is parameterised by a set of parameters (weights and biases), which are tuned during training $\boldsymbol{\theta} \equiv \{\boldsymbol{W}_j, \boldsymbol{b}_j\}_{j=1}^{d}$

Deep learning: cost function

In most cases the output of the network can be generalised as a distribution $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$

The network is trained then using maximum likelihood.

We have to find a set of parameters which will optimise the loss function

$$\begin{aligned}
\boldsymbol{\theta}_{\mathrm{ML}} &= \arg\max_{\theta} p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta}) \\
&= \arg\max_{\theta} \prod_{i=1}^{N} p\left(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) \\
&= \arg\max_{\theta} \sum_{i=1}^{N} \log p\left(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right) \\
&= \arg\max_{\theta} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\mathrm{data}}} \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})
\end{aligned}$$

This means that the cost function is simply the negative log-likelihood.

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\mathrm{data}}} \log p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})$$

Deep learning: stochastic gradient descent

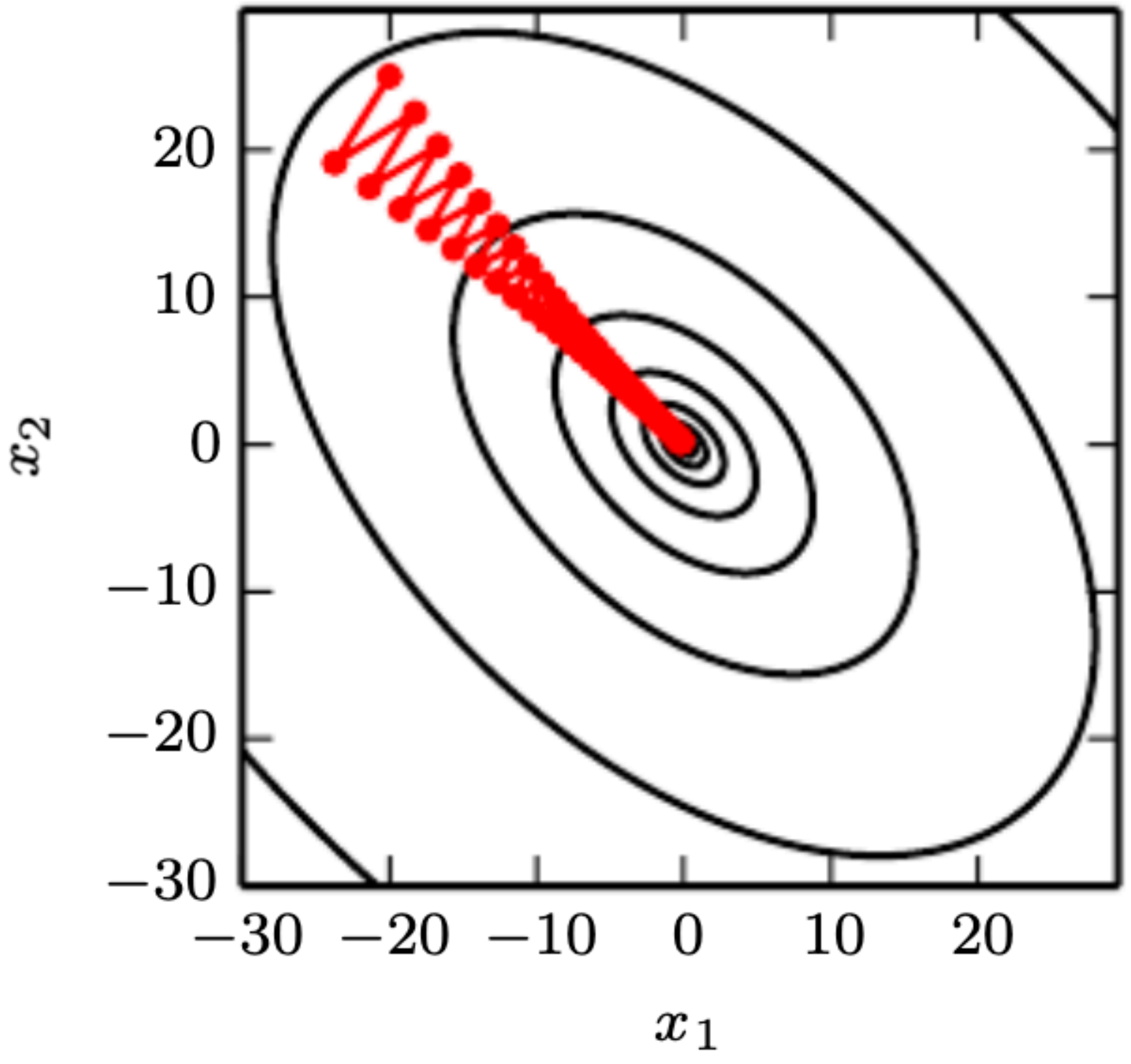


Image: Goodfellow et al (2016)

Stochastic gradient descent is an extension of the gradient descent. When the training dataset is too large we use mini-batches.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J_m(\boldsymbol{\theta})$$

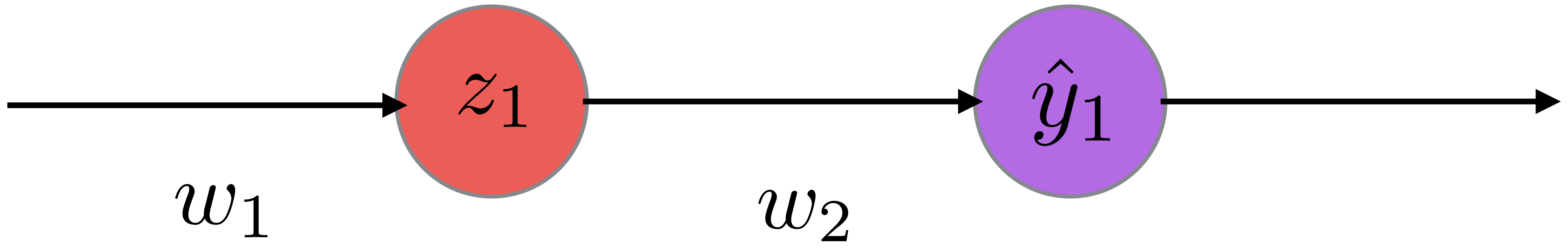
where $$J_m = \frac{1}{m} \sum_{i=1}^{m} J\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

There are different versions of this algorithm.
Plus extensions that involve second (Hessian) or higher order derivatives.

Deep learning: back-propagation

Back-propagation is implemented by a chain rule.

$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2}$$



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

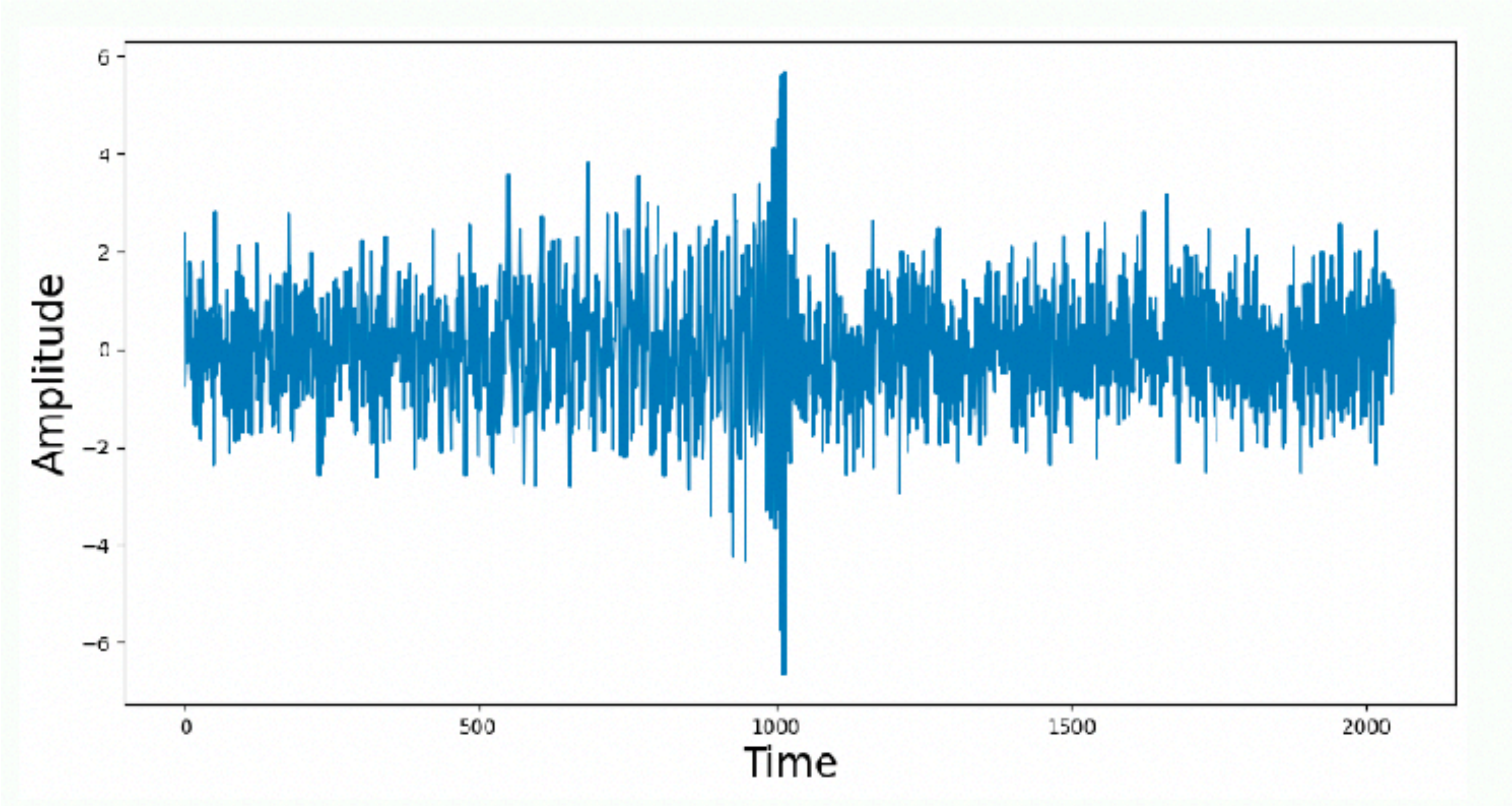
Detection and point parameter estimation

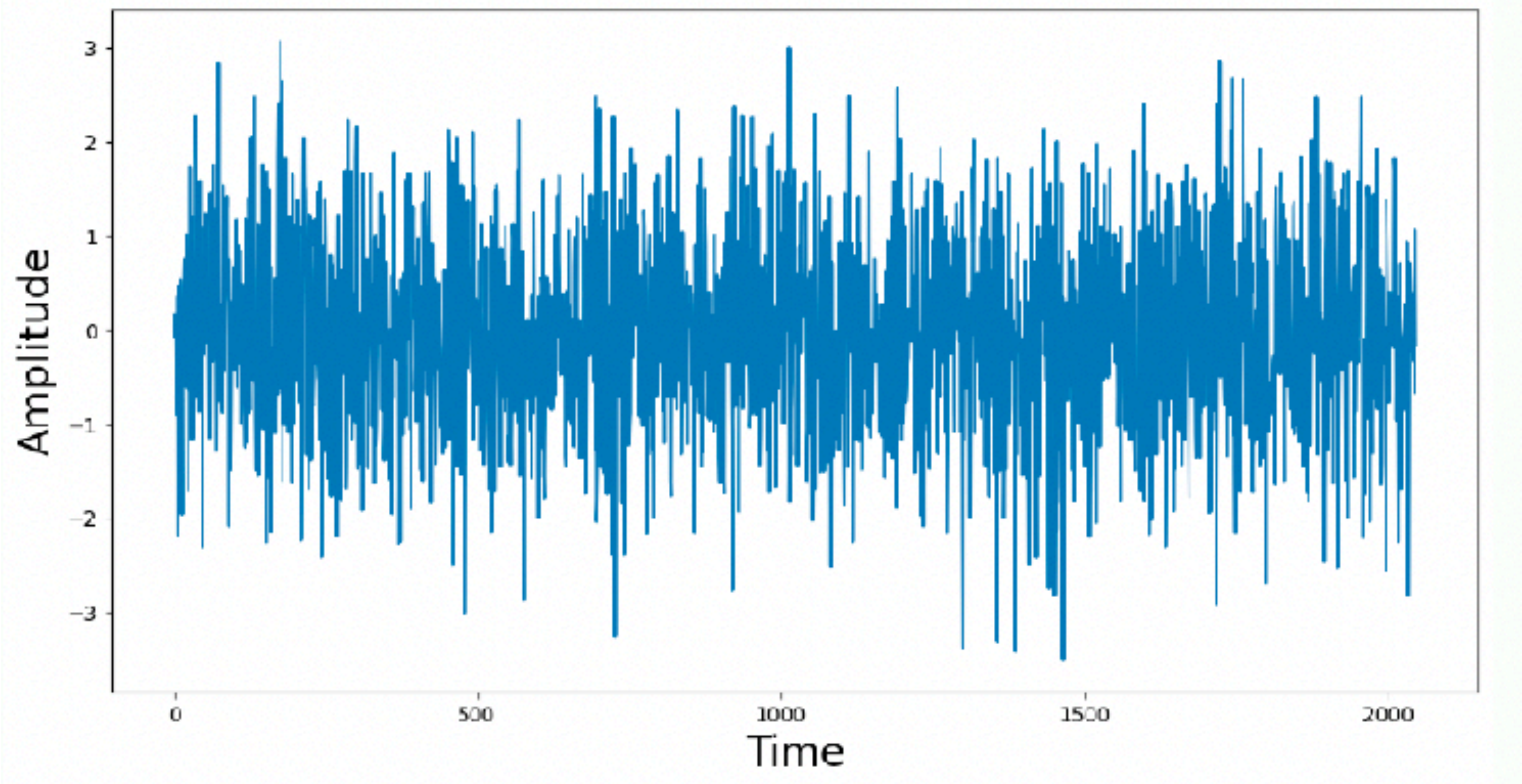
Recall from the classical methods

In the classical approach we use match filtering for the detection and point parameter estimation.

We start with the simplest deep learning approach to solve similar problem.

Detection





Detection answers the question if the signal is present or not in the data.

We have to train a binary classifier.

Training dataset is composed of pairs:

simulated data with signal: $d = h + n$

label: 1

simulated data without signal: $d = n$

label: 0

Detection

We have to choose the activation function and the cost function that will be appropriate for the task.

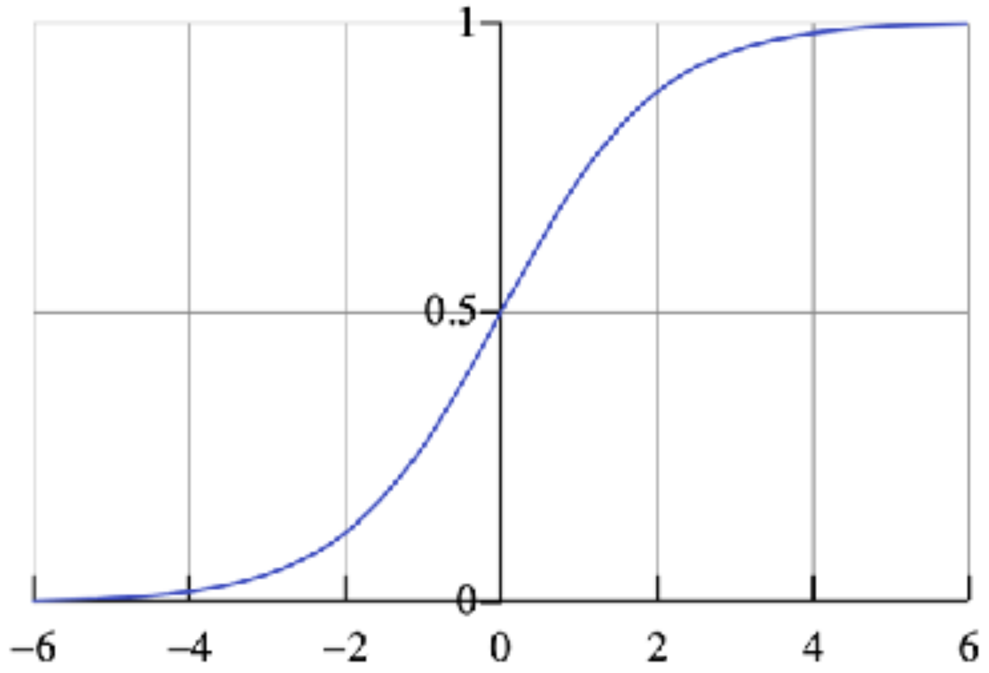
If we have **classification problem** with **two classes** then the appropriate choice for the maximum likelihood approach will be **Bernoulli distribution**.

Bernoulli distribution is the discrete probability distribution which takes the value 1 with probability p and the value 0 with probability 1 – p

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{Ber}\left(y \mid \sigma\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right)\right)$$

where $y \in \{0, 1\}$

last layer activation function: $\sigma(a) \triangleq \dfrac{1}{1+e^{-a}}$ **sigmoid** or logistic



In this case log-likelihood is chosen to be **negative binary cross entropy**:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{y}^{(i)} \log\left(f\left(\mathbf{x}^{(i)}; \boldsymbol{\theta}\right)\right) + \left(1 - \mathbf{y}^{(i)}\right) \log\left(1 - f\left(\mathbf{x}^{(i)}; \boldsymbol{\theta}\right)\right)$$

where y label

$f()$ predicted value

Detection

Thus far we look at the Multilayer Perceptron,
which is the most simple architecture for the feed-forward networks.

We can try different architectures which allow to

- extract features from the data
- capture sequential nature of the data.

Detection: CNN

Convolutional neural networks allow to reduce the size of the network because they use local convolutions instead of metric multiplications.

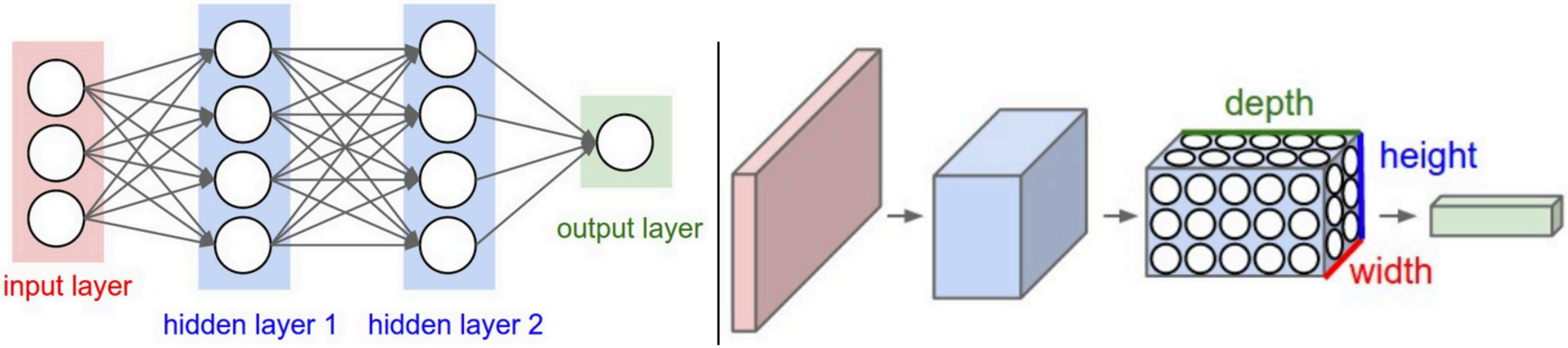


image: https://cs231n.github.io/convolutional-networks/

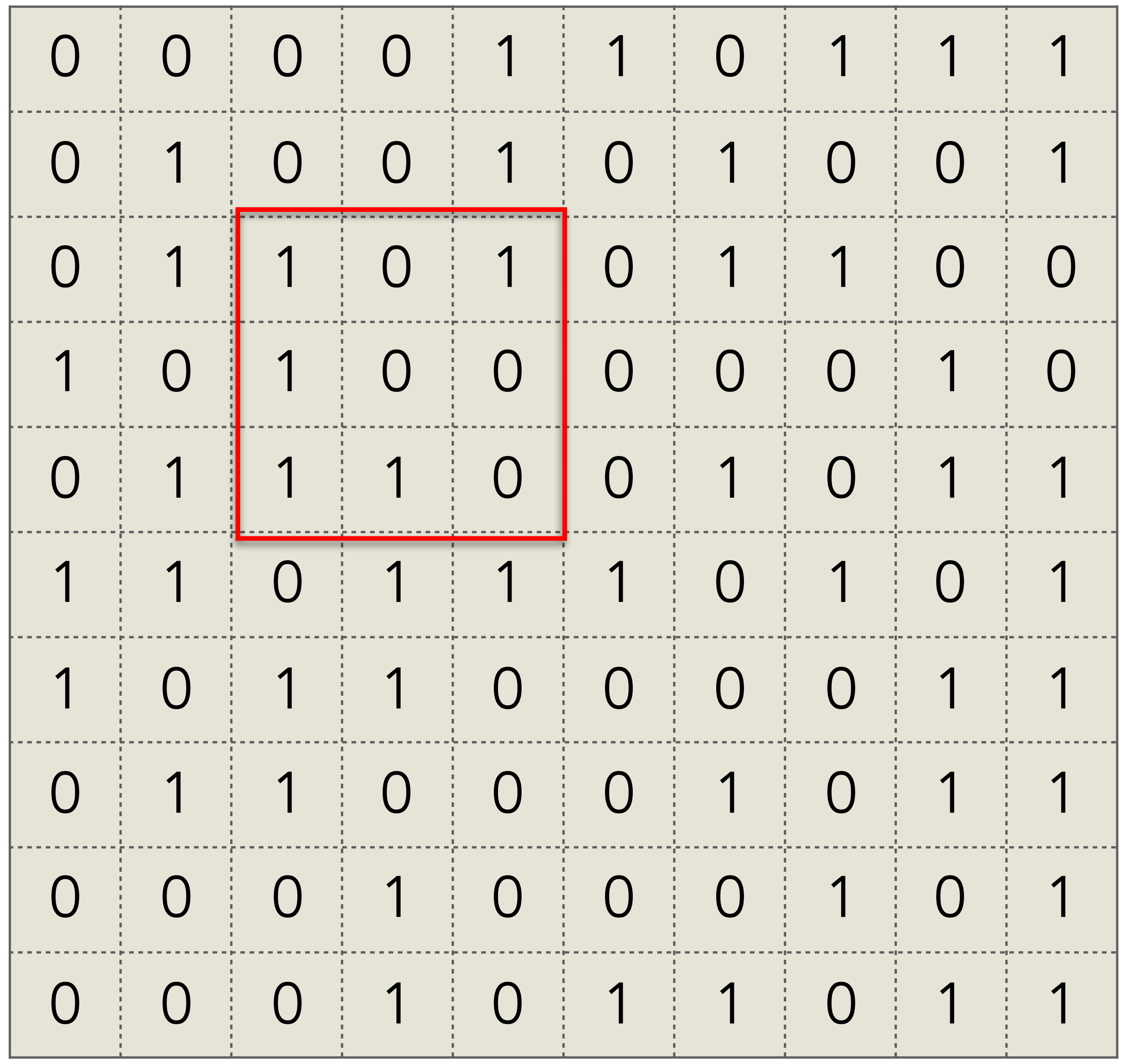
We get a convolution by sliding the weight matrix over the 'image' and adding up the results; in this case the weight matrix is often called a "kernel" or "filter".

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Detection: CNN

Input: array of values

Originally these networks were for images, but can also be 1-dimensional.



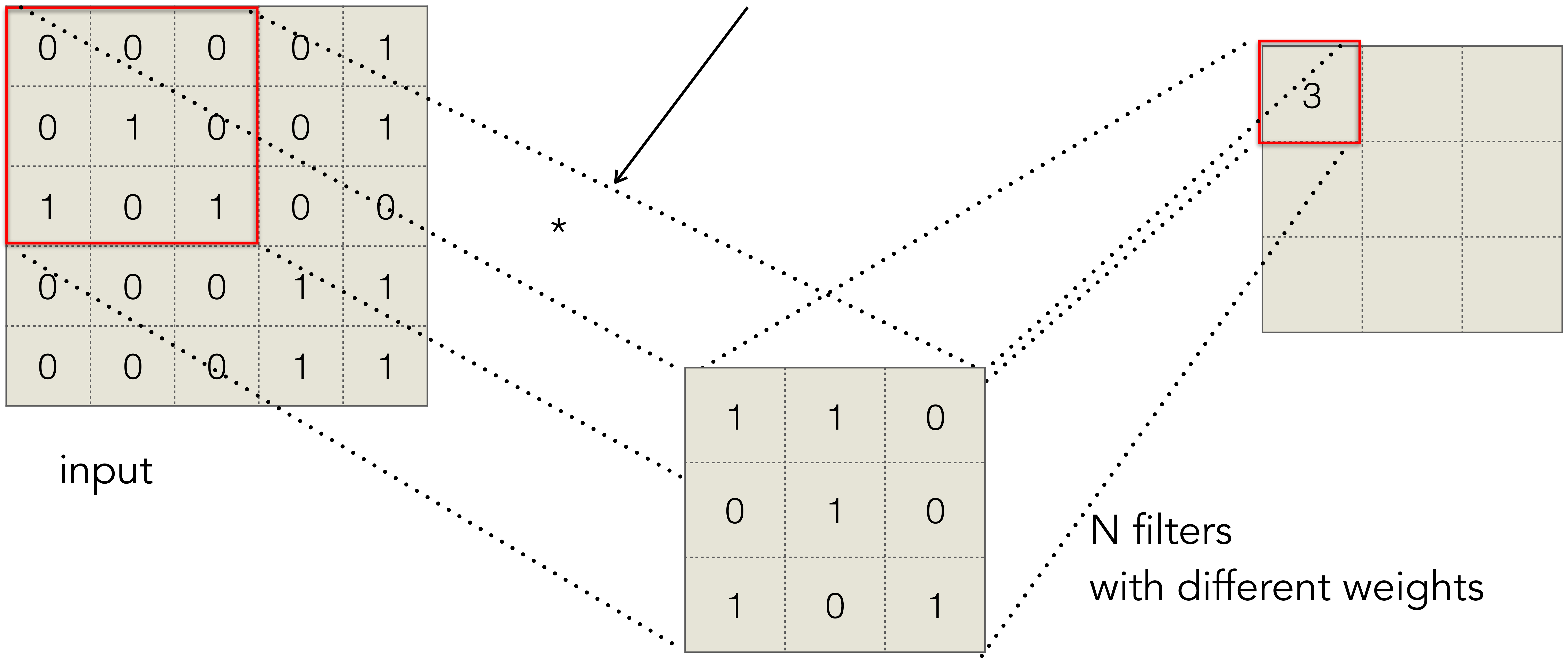
0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	1	0	1	0	1	1	0	0
1	0	1	0	0	0	0	0	1	0
0	1	1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	1	0	1
1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	0	1	0	1	1
0	0	0	1	0	0	0	1	0	1
0	0	0	1	0	1	1	0	1	1

Instead of connecting each neuron to each other one, we apply the filters (or kernels).

To convolve a filter with an image means, to slide the filter with weights over an image and compute the dot products.

Detection: CNN

Filters are sensitive to the features in the data



dot product

*

0	0	0	0	1
0	1	0	0	1
1	0	1	0	0
0	0	0	1	1
0	0	0	1	1

input

1	1	0
0	1	0
1	0	1

weights

3		

N filters
with different weights

Detection: RNN

Recurrent Neural Networks can be useful:

- for analysis of the data sequences
- when the order of the elements matter
- when the size of the data is not fixed
- when we want to connect information on different time scales.

We can express the dependency on the previous time stamps by the concept of the **context**.

$$p\left(x_t \mid x_1, \ldots, x_{t-1}\right) \approx p\left(x_t \mid h\right)$$

RNNs have vanishing gradient problem but there are more modern implementations which address this problem but preserve the concept.

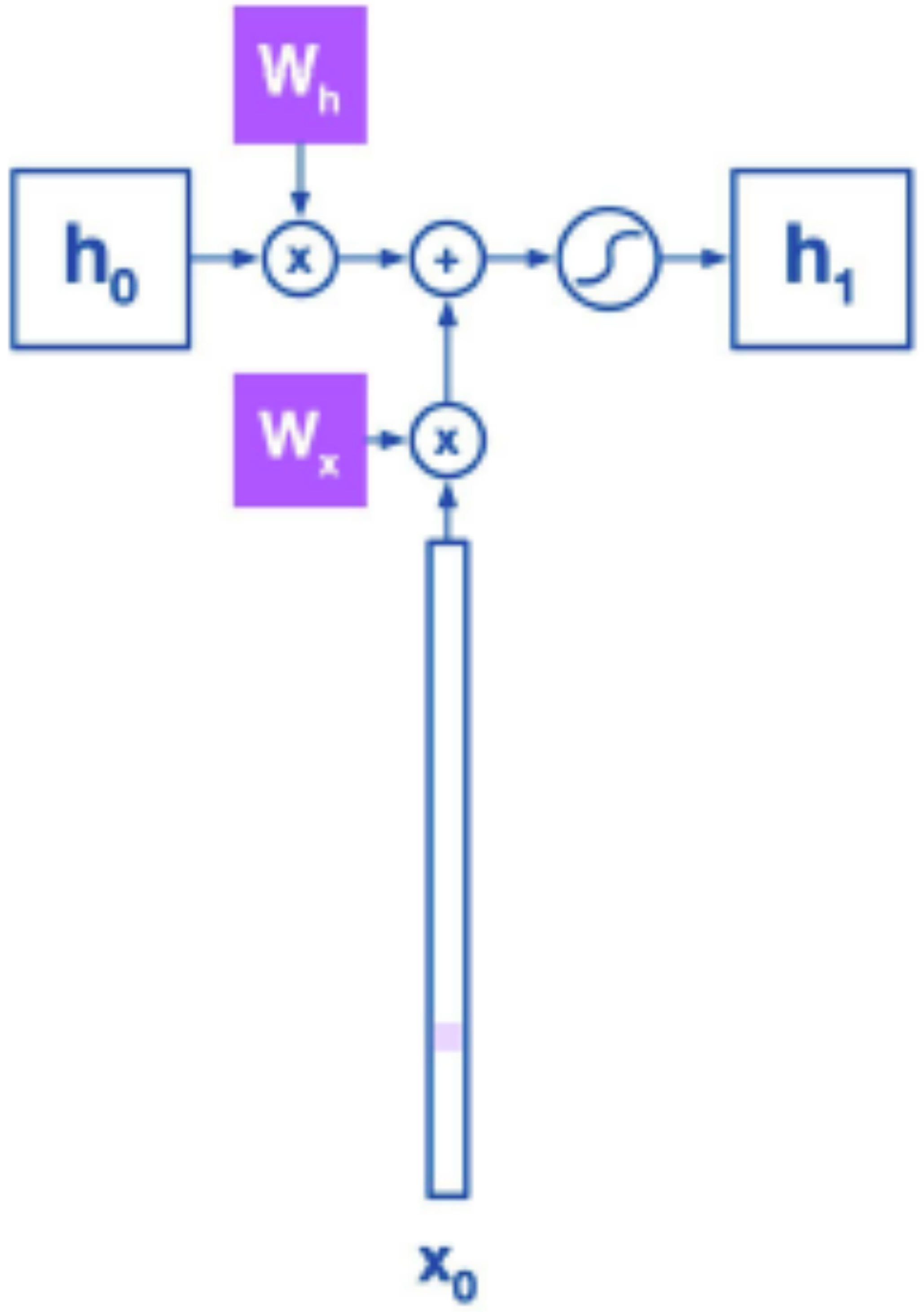


image: Deepmind lectures

Detection: ROC curve

Very important to correctly access the performance.

ROC (receiver operating characteristic) curve is one of the methods that can be used.

True alarm probability plotted versus false alarm probability.

How to construct ROC curve:
1) Choose y_th which will be a detection threshold.
2) Predicted y values greater than y_th will be considered as an alarm.
3) At a given y, FAP and TAP are constructed as the fraction of noise-only and signal plus noise samples, respectively, that are reported as an alarm.
4) Vary y, obtain pairs of TAP/FAP

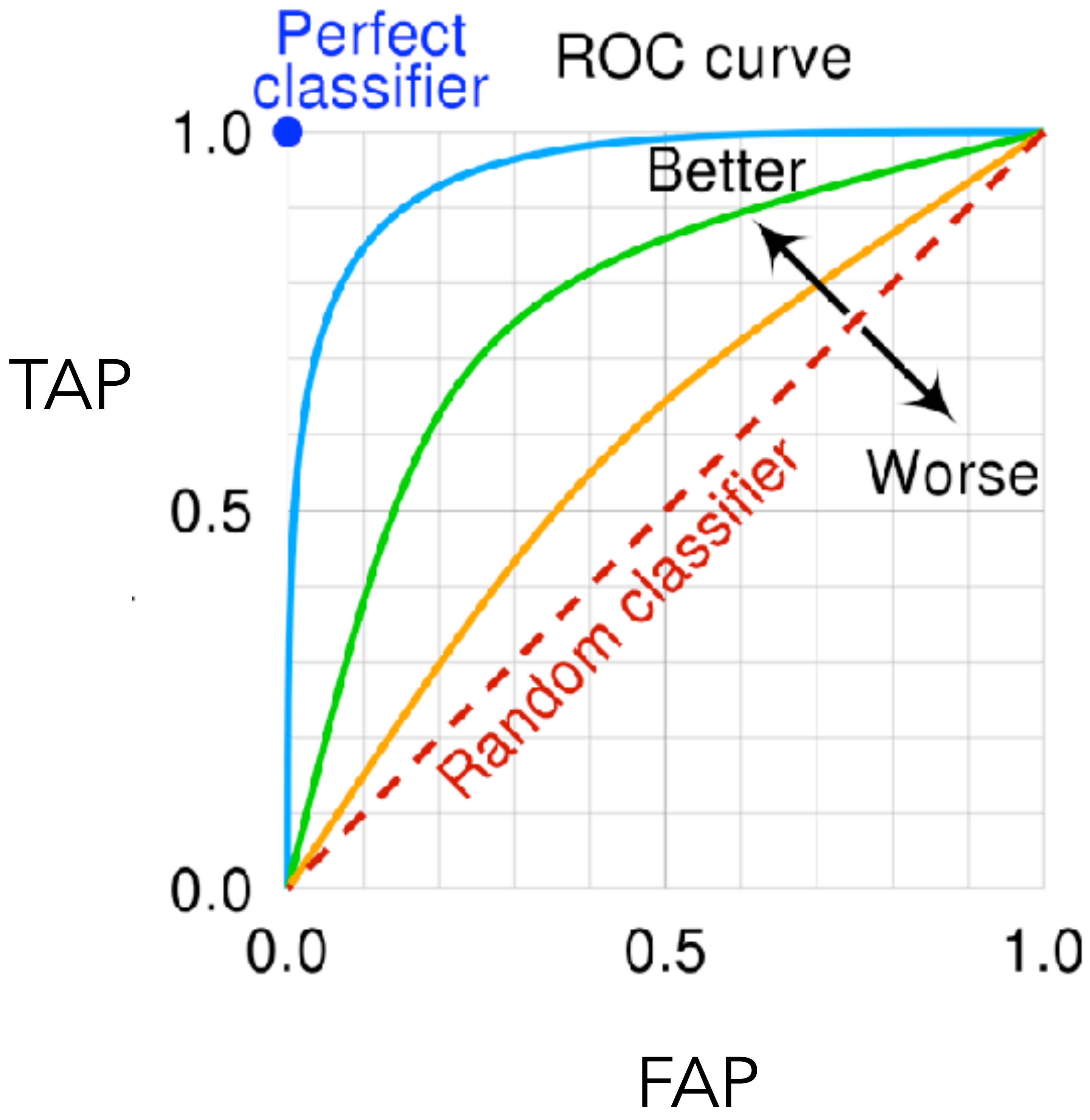


image: wikipedia

Point parameter estimation

Instead of detection we can perform parameter estimation by using known parameters that corresponded to the signal injected in the data.

The architecture can be similar as for the detection problem except last activation function and cost function.

simulated data: $d = h(\theta) + n$

labels: θ

In this case the appropriate cost function will be mean square error:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{y}^{(i)} - f\left(\mathbf{x}^{(i)};\boldsymbol{\theta}\right)\right)$$

The last activation layer have to be a liner function, therefore we can choose a ReLU (Rectified Linear Unit):

$$\sigma(a) = a^{+} = \max(0, a)$$

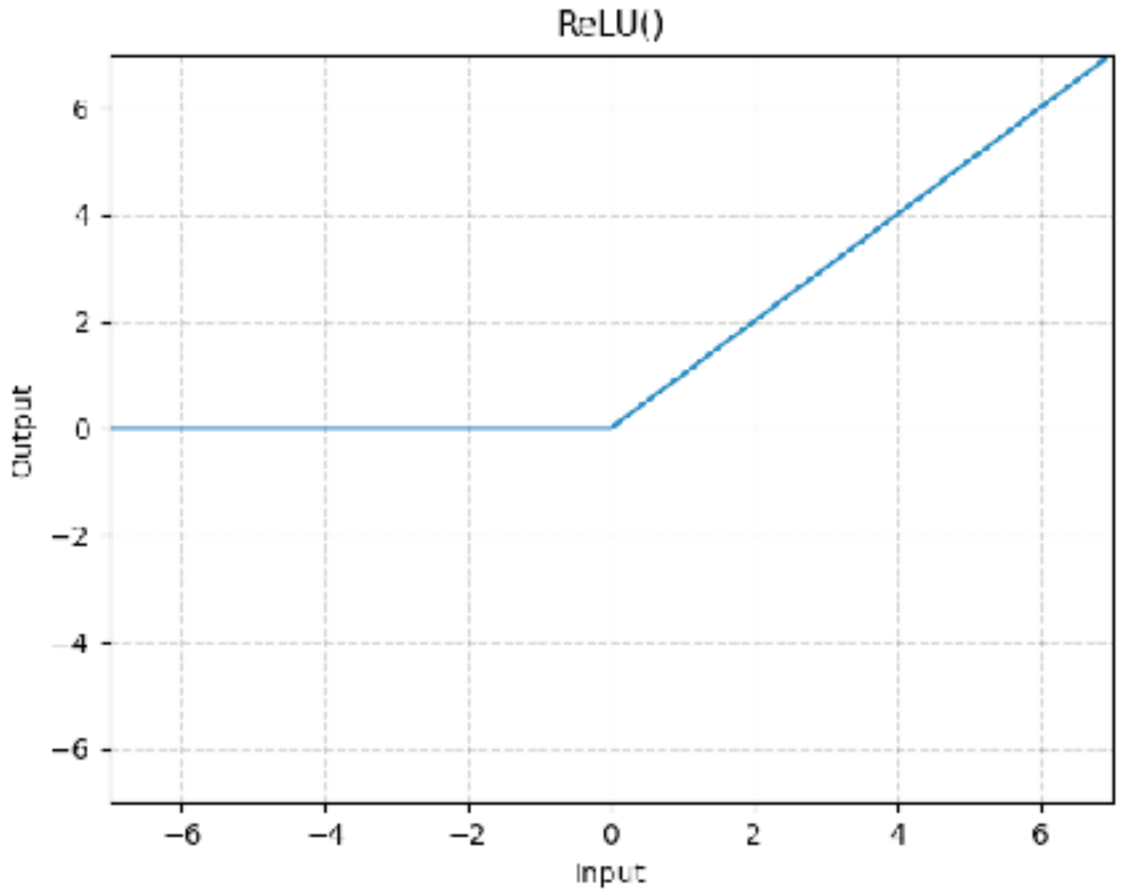


image: pytorch documentation

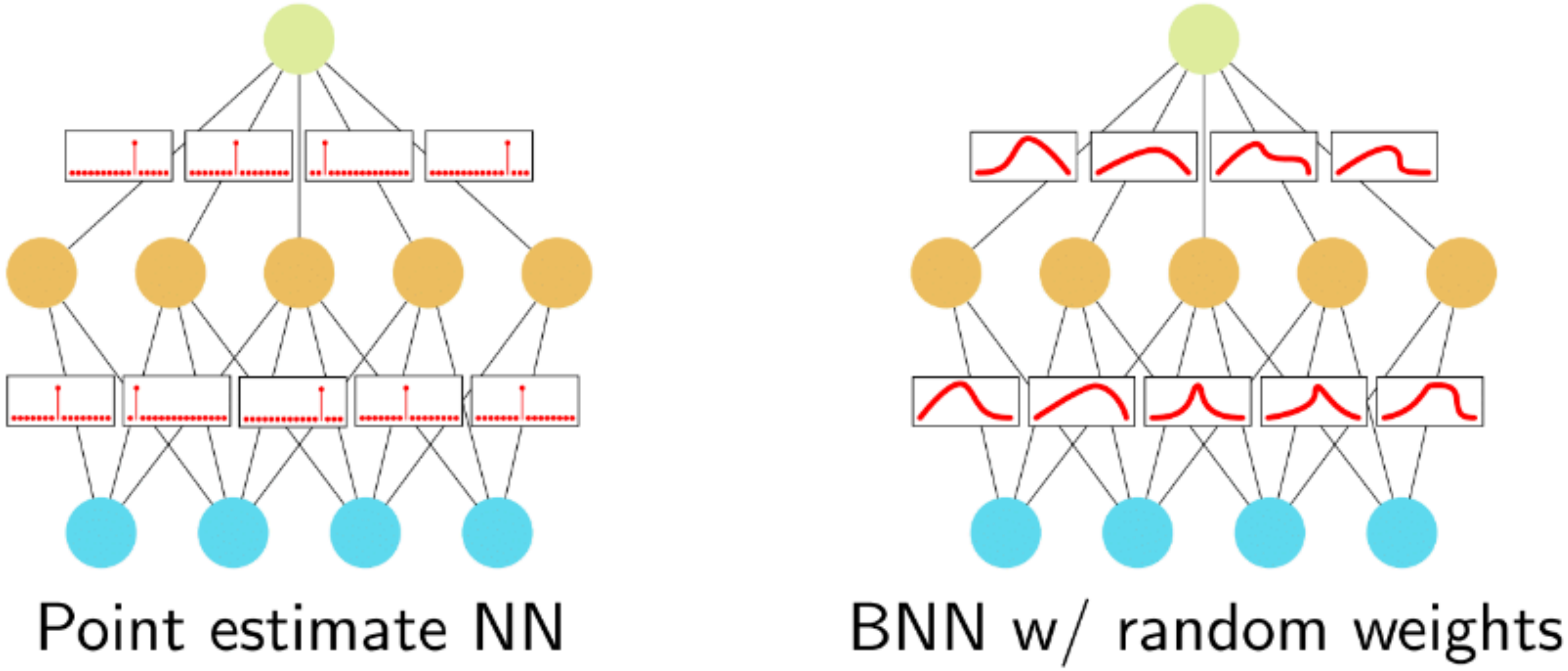
Different type of errors

This was the point parameter estimation.

In addition to that the error on the estimation of the weights is incorporated in the estimation of parameter.

How can we solve this problem?

Bayesian NN



Point estimate NN

BNN w/ random weights

Replace weights with the probability distributions.

image: https://statisfaction.wordpress.com/2020/01/20/course-on-bayesian-machine-learning-in-paris/

Bayesian Parameter Estimation

Parameter estimation

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\boxed{p(x)}}$$

problem:

marginal likelihood
has no exact solution

$$p(x) = \int p(x \mid \theta)p(\theta)\mathrm{d}\theta$$

Parameter estimation

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\boxed{p(x)}}$$

solutions:

- approximate inference:
  - MCMC/Nested sampling

    requires likelihood evaluation

    we can do it, but it is slow

Parameter estimation

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\boxed{p(x)}}$$

solutions:

- approximate inference:
  - MCMC/Nested sampling

    requires likelihood evaluation

    we can do it, but it is slow

  - Variational inference

    approximate the posterior distribution

    with a tractable distribution

Parameter estimation

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\boxed{p(x)}}$$

solutions:

- simplification to the model:
  - Gaussian mixture models

    too simple

Parameter estimation

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\boxed{p(x)}}$$

solutions:

- simplification to the model:
  - Gaussian mixture models

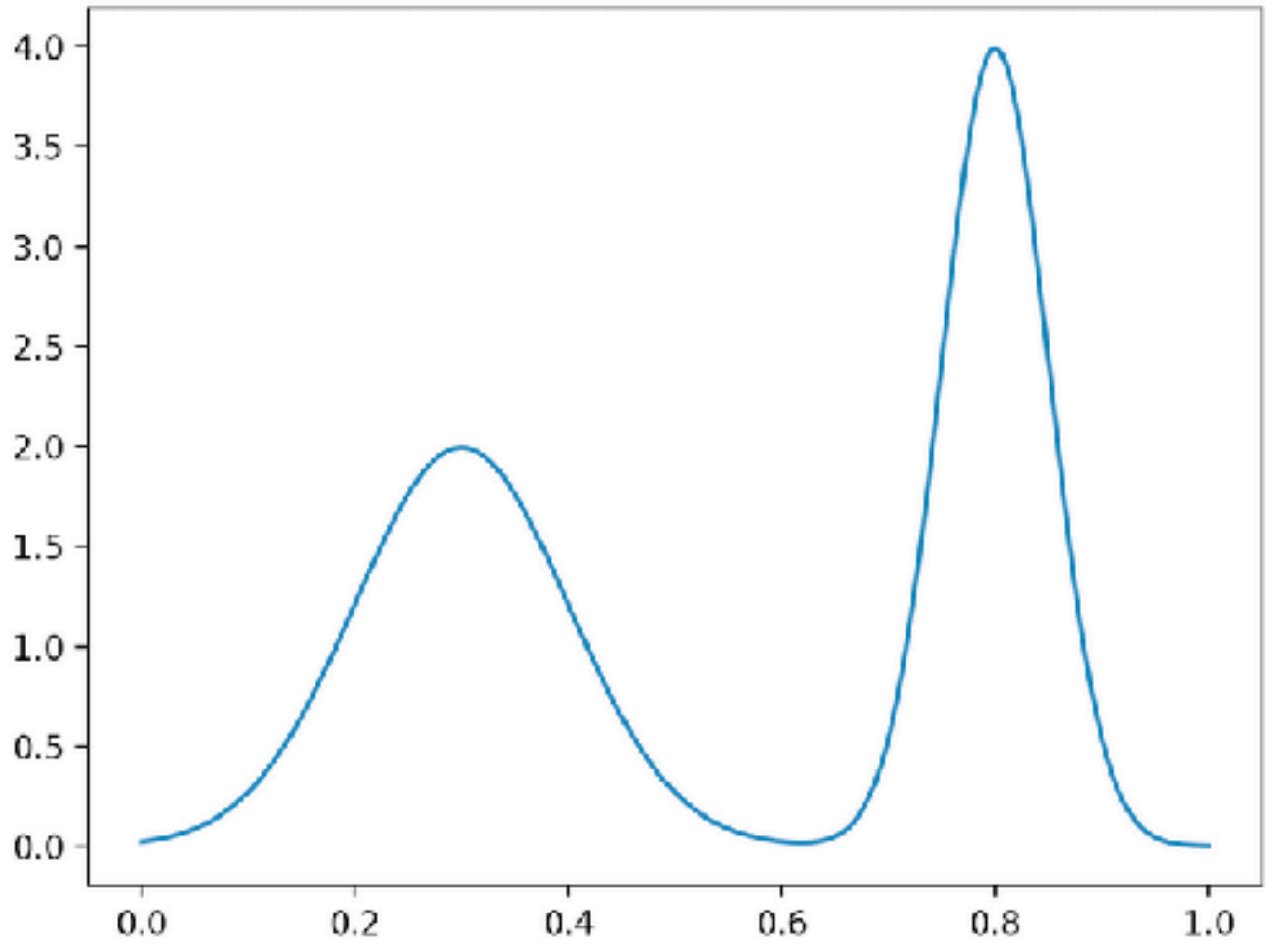
    too simple

  - Invertible models

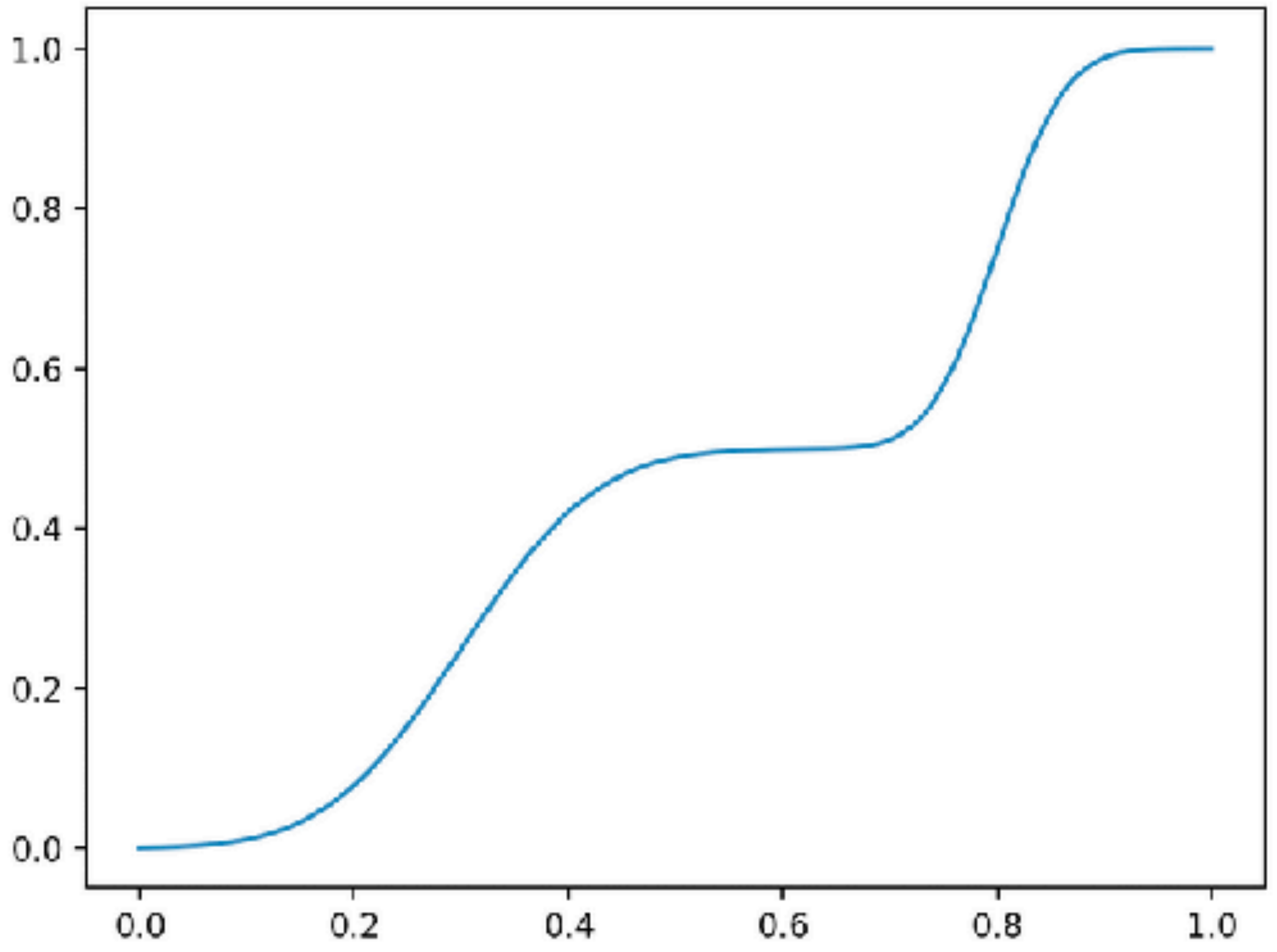
    will talk about them today

Invertible transform

If x is a random variable with the CDF f(x),
then the random variable y = f(x) has a uniform distribution on [0,1].

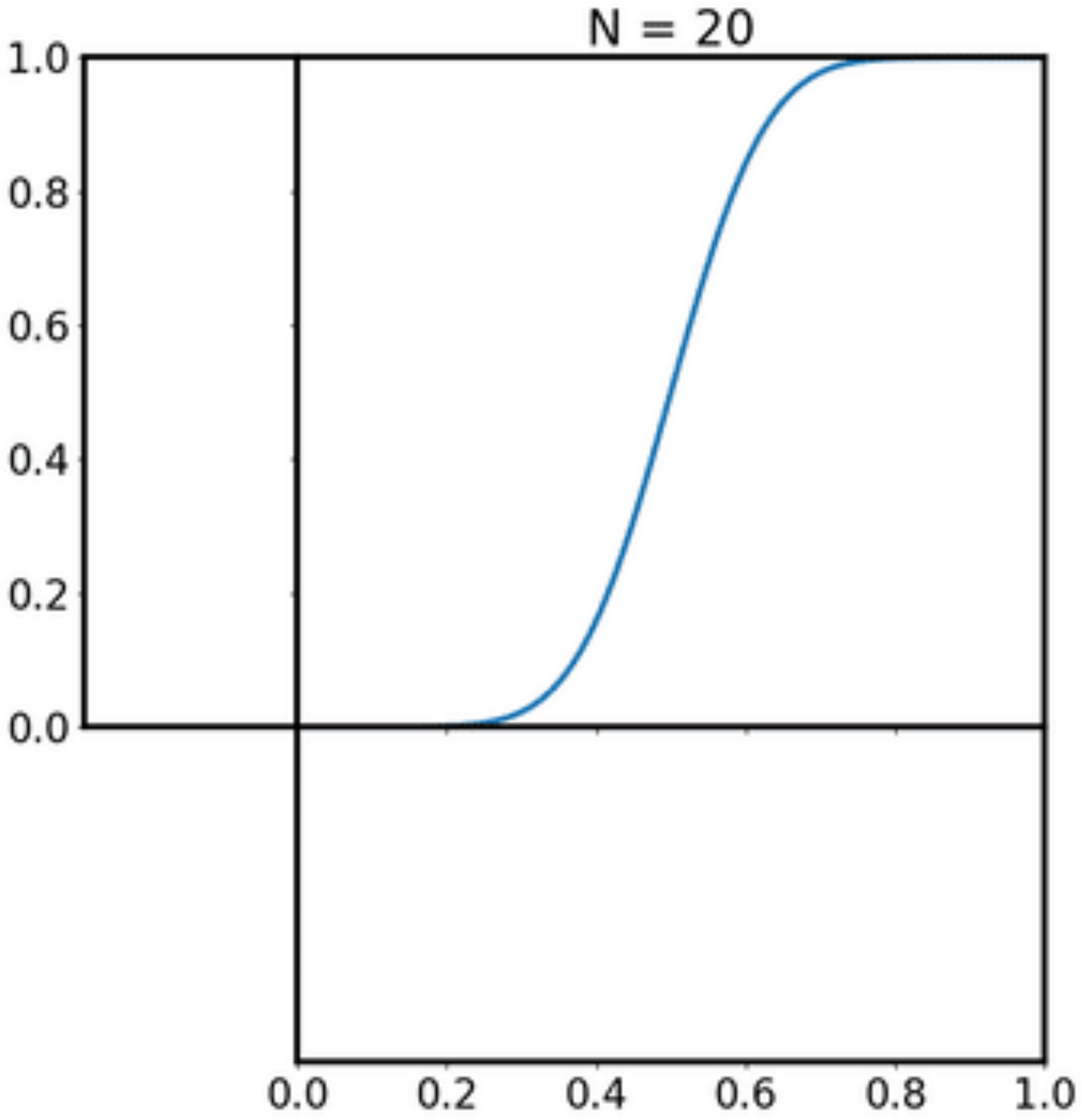


$p_\theta(x)$



$$f_\theta(x) = \int_{-\infty}^{x} p_\theta(t)\, dt$$

Invertible transform



Invertible transform

Change of variables for probability density function

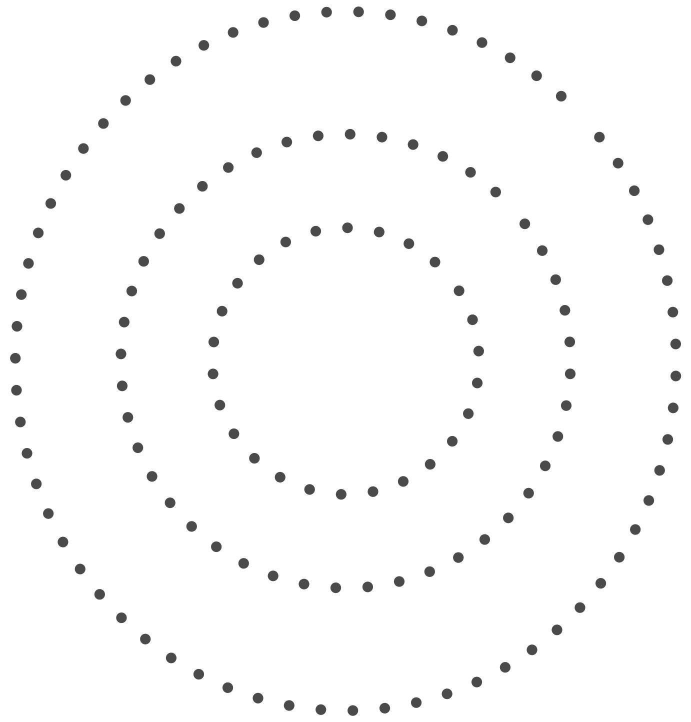
$$f_Y(y) = \frac{d}{dy} F_Y(y) = \frac{d}{dy} F_Z\left(g^{-1}(y)\right)$$

Apply chain rule

$$= f_Z\left(g^{-1}(y)\right)\left|\frac{d}{dy} g^{-1}(y)\right|$$

Normalising flows

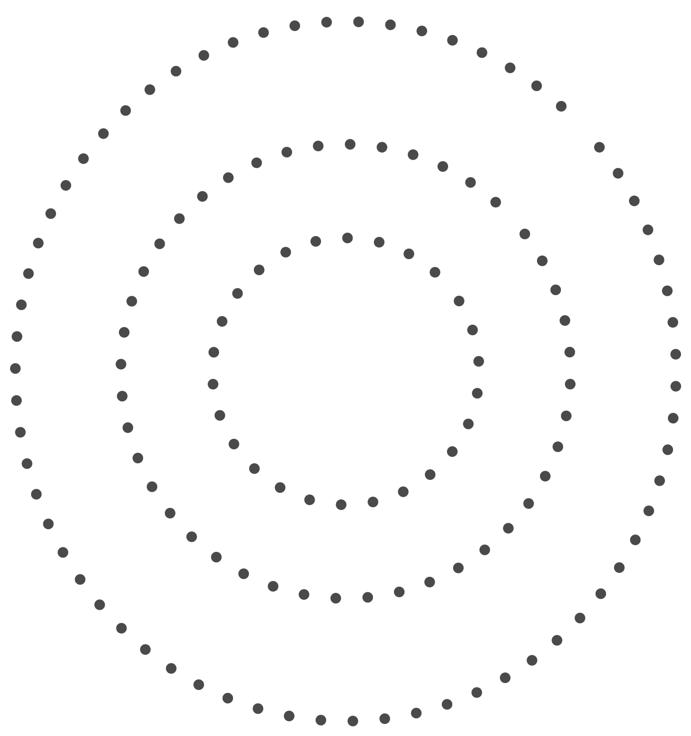
1. We have simple random generator



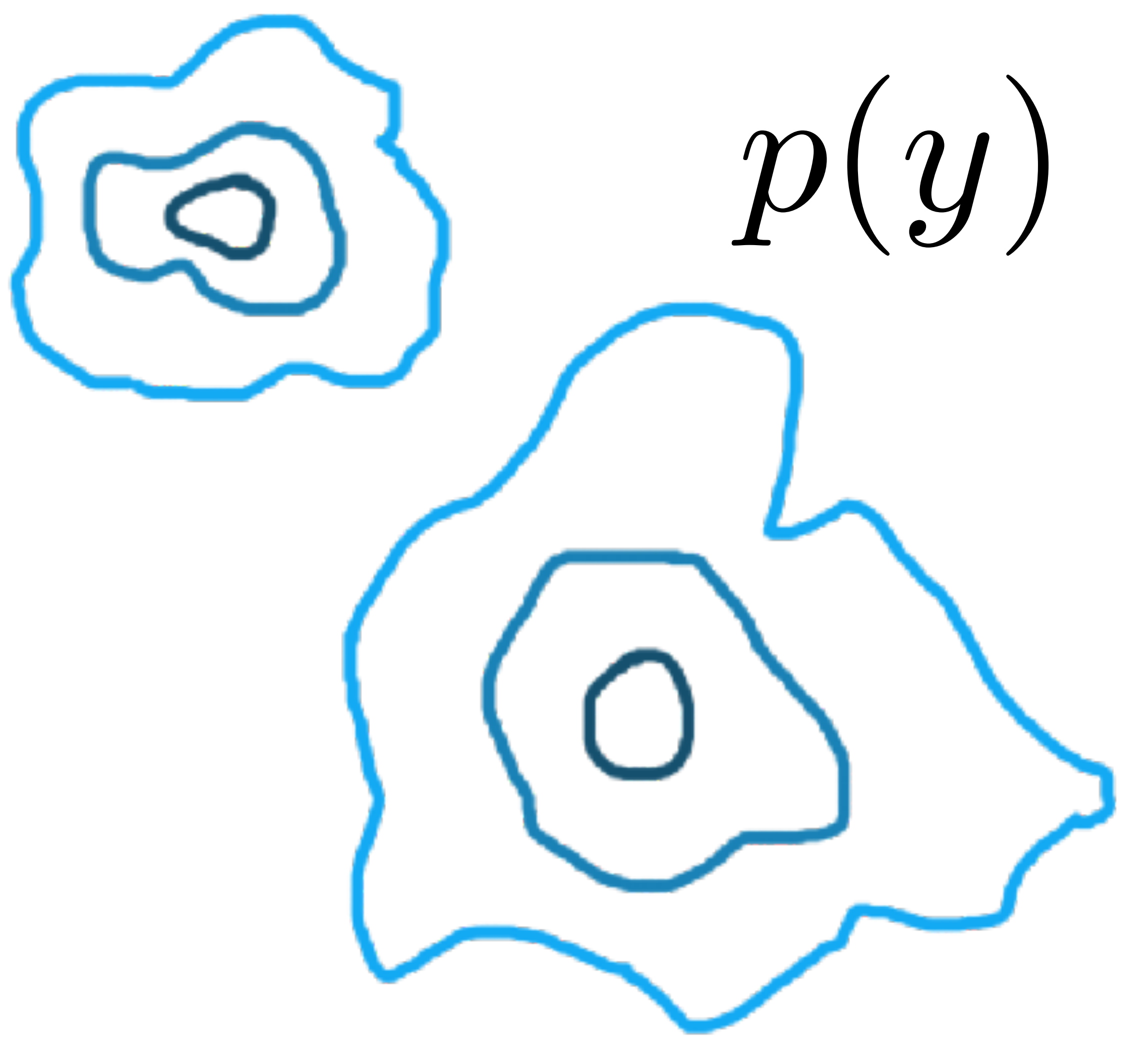
$$q(z) = \mathcal{N}(0, 1)$$

Normalising flows

1. We have simple random generator
2. We want to sample from a more complex distribution



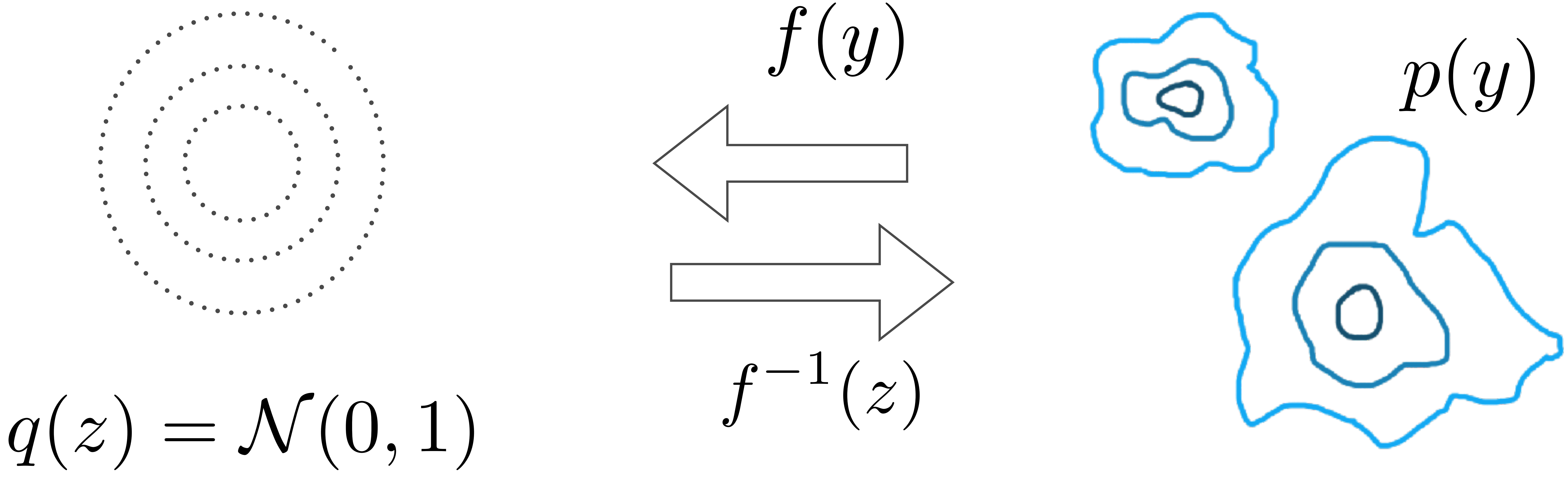
$$q(z) = \mathcal{N}(0, 1)$$



$p(y)$

Normalising flows

1. We have simple random generator
2. We want to sample from a more complex distribution
3. We can estimate a bijective transformation which will allow us to do that



Change of variables equation

$$p(y) = q(f^{-1}(y)) \left|\det\left(J_{f^{-1}}(y)\right)\right|$$

Change of variables equation

$$p(y) = \boxed{q(f^{-1}(y))} \left| \det \left(J_{f^{-1}}(y) \right) \right|$$

- f has to be a bijection

Change of variables equation

$$p(y) = \boxed{q(f^{-1}(y))}\,\boxed{\left|\det\left(J_{f^{-1}}(y)\right)\right|}$$

- f has to be a bijection
- f and f^{-1} have to be differentiable
- Jacobian determinant has to be tractably invertable

Optimisation

The flow is trained by maximising the total log likelihood of the data with respect to the parameters of the transformation

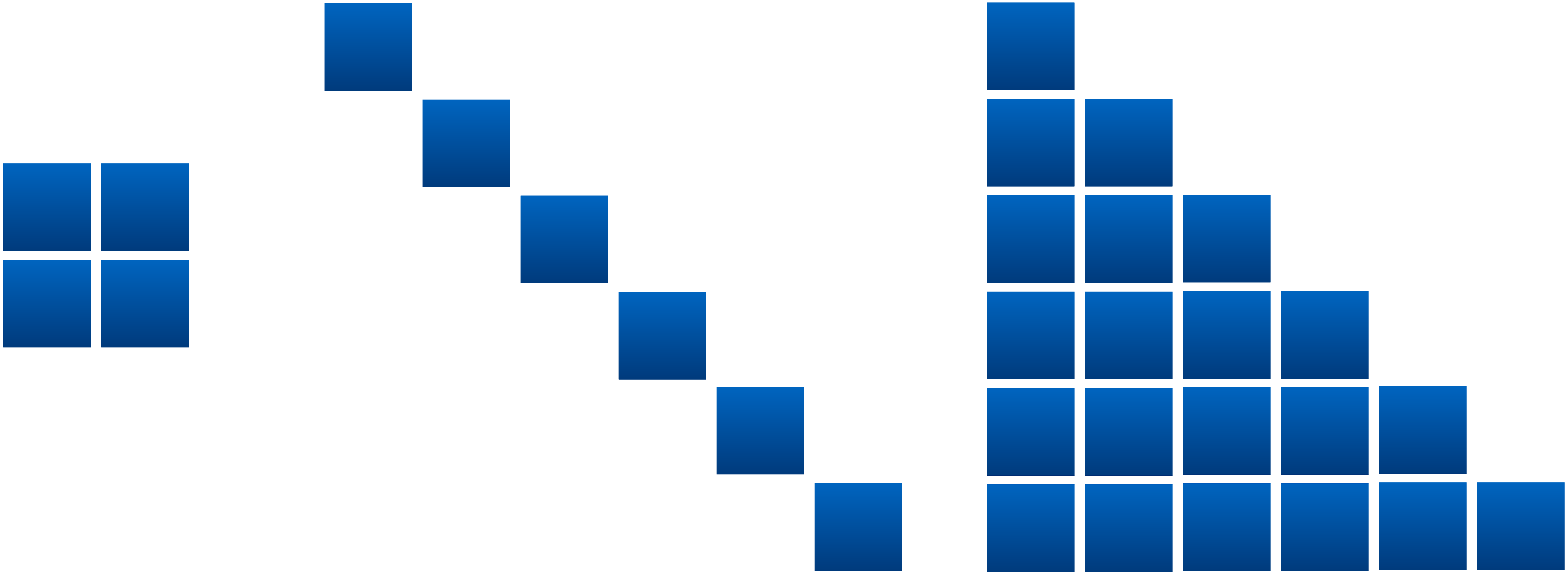
$$\arg\min_{\theta} \mathbb{E}_{\mathbf{x}} \left[-\log p_{\theta}(\mathbf{x}) \right] = \mathbb{E}_{\mathbf{x}} \left[-\log p(f_{\theta}(\mathbf{x})) - \log \det \left| \frac{\partial f_{\theta}(\mathbf{x})}{\partial \mathbf{x}} \right| \right]$$

Jacobian

$$J_f(\mathbf{z}) = \begin{bmatrix} \frac{\partial f_1}{\partial z_1} & \cdots & \frac{\partial f_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial z_1} & \cdots & \frac{\partial f_n}{\partial z_n} \end{bmatrix}$$

The calculation of determinant Jacobian will take O(n^3)
To make it faster we have to ensure that the Jacobian is triangular
The determinant of the triangular matrix is a product of the diagonal elements

Jacobian



Affine transformation

location-scale transformation

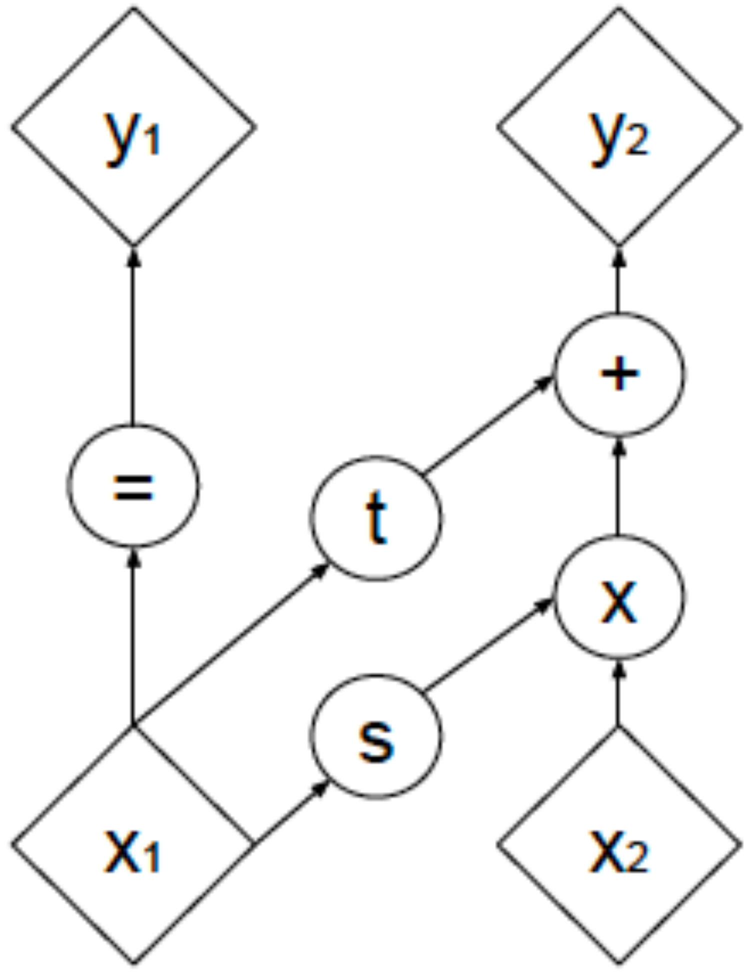
$$\tau(z_i; \mathbf{h}_i) = \alpha_i z_i + \beta_i \qquad \mathbf{h}_i = \{\alpha_i, \beta_i\}$$

Invertibility for $\alpha_i \neq 0$

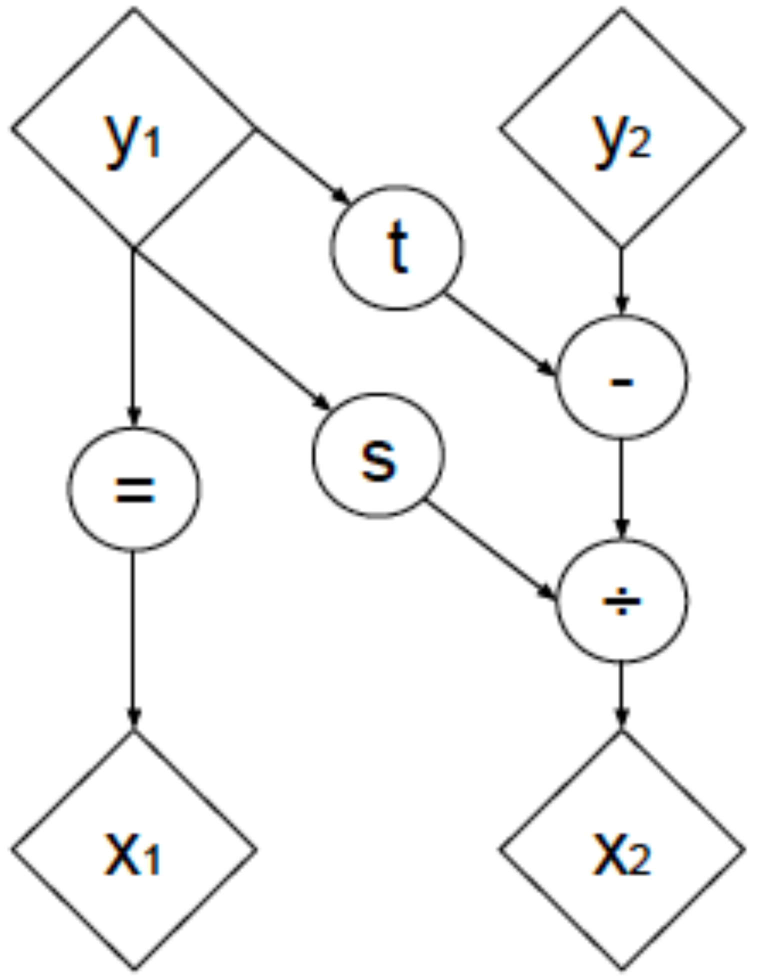
log-Jacobian becomes

$$\log|\det J_f(\mathbf{z})| = \sum_{i=1}^{N} \log|\alpha_i|$$

Coupling transform



(a) Forward propagation



(b) Inverse propagation

In each simple bijection,
part of the input vector
is updated using a function
which is simple to invert,
but which depends on the
remainder of the input vector
in a complex way.
The other part is left unchanged.

Real NVP

Coupling transformation combined with affine transformation and its invention

$$
\begin{cases} y_{1:d} & = x_{1:d} \\ y_{d+1:D} & = x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}) \end{cases}
$$

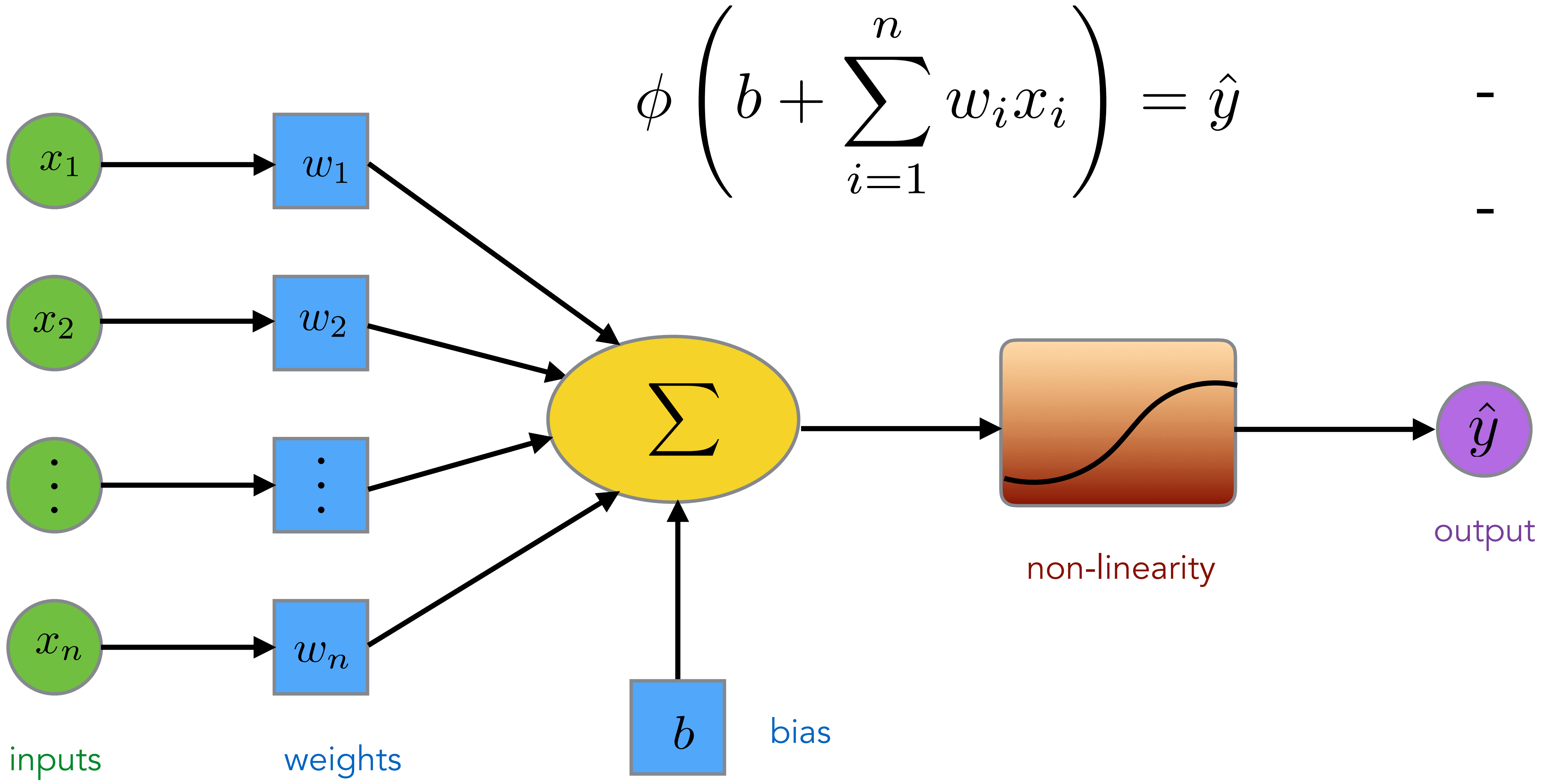
$$
\Leftrightarrow \begin{cases} x_{1:d} & = y_{1:d} \\ x_{d+1:D} & = \big(y_{d+1:D} - t(y_{1:d})\big) \odot \exp\big(-s(y_{1:d})\big), \end{cases}
$$

It is easy to do the inversion.
We can keep the same parameters of the function which we used on the forward pass.

What is ***t*** and ***s***?

https://arxiv.org/abs/1605.08803

Function approximation



can be parameterised by any NN:

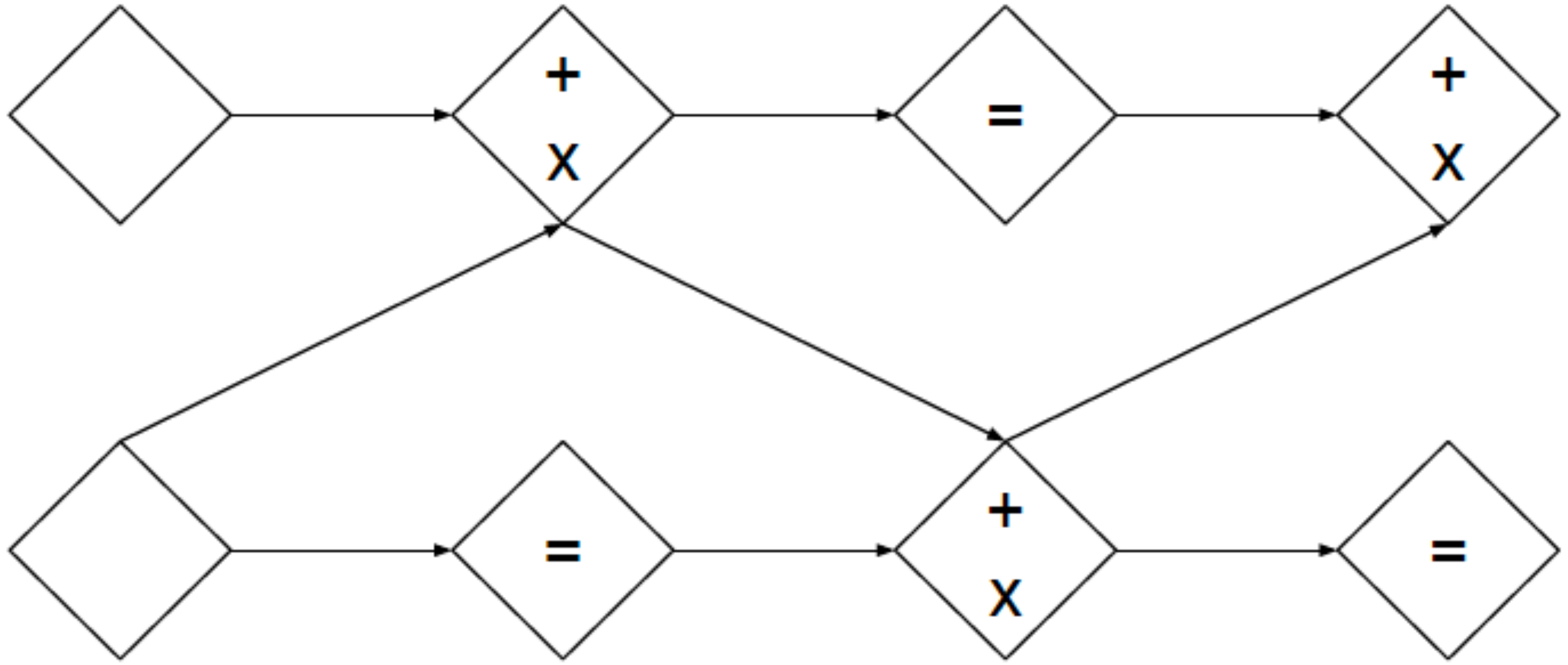
- Fully connected
- Residual
- CNN
- …

Jacobian

The form of Jacobian is such that we can only need the components on the diagonal.

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{bmatrix}$$

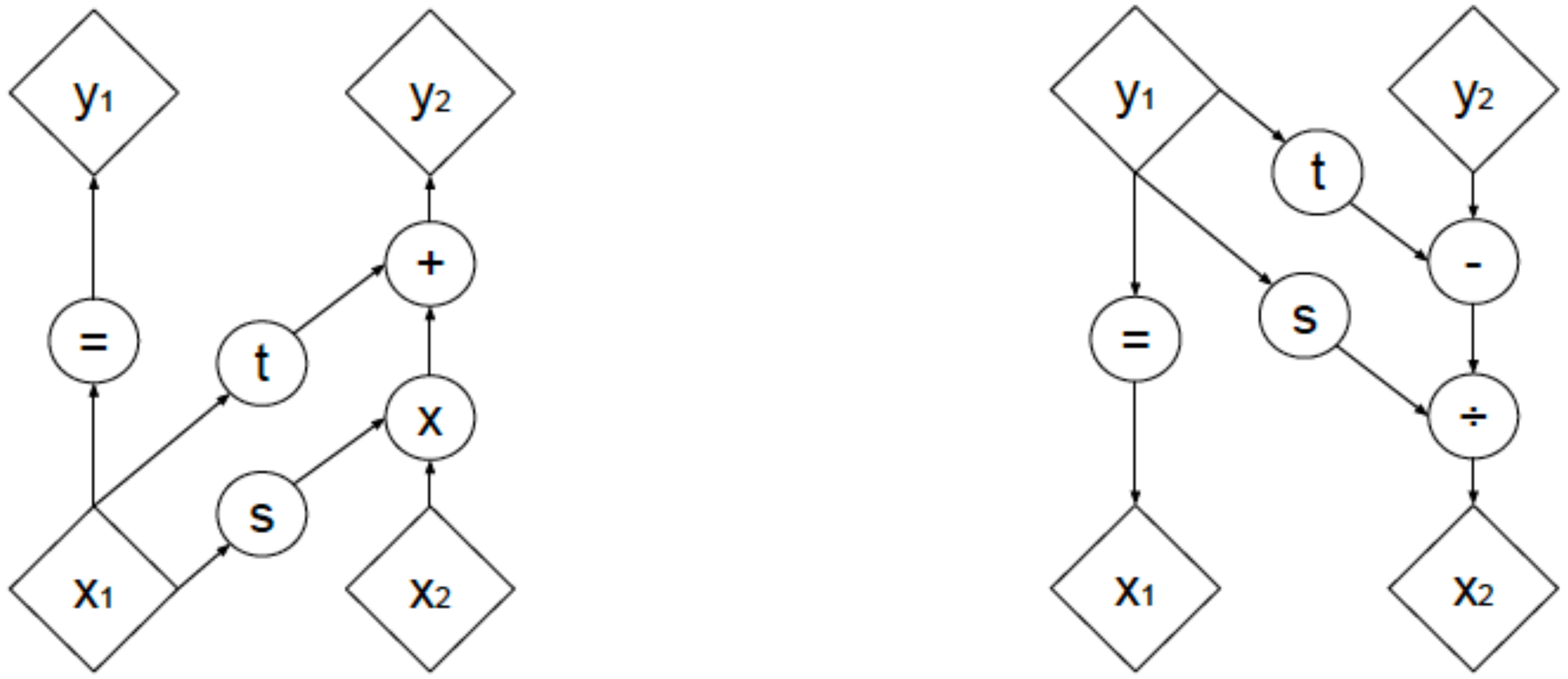
Function approximation



A sequence of the flows which fix one dimension at one step and vary
the other dimension and repeat this process for a given number of times.

Neural Spline Flows

- Coupling transform

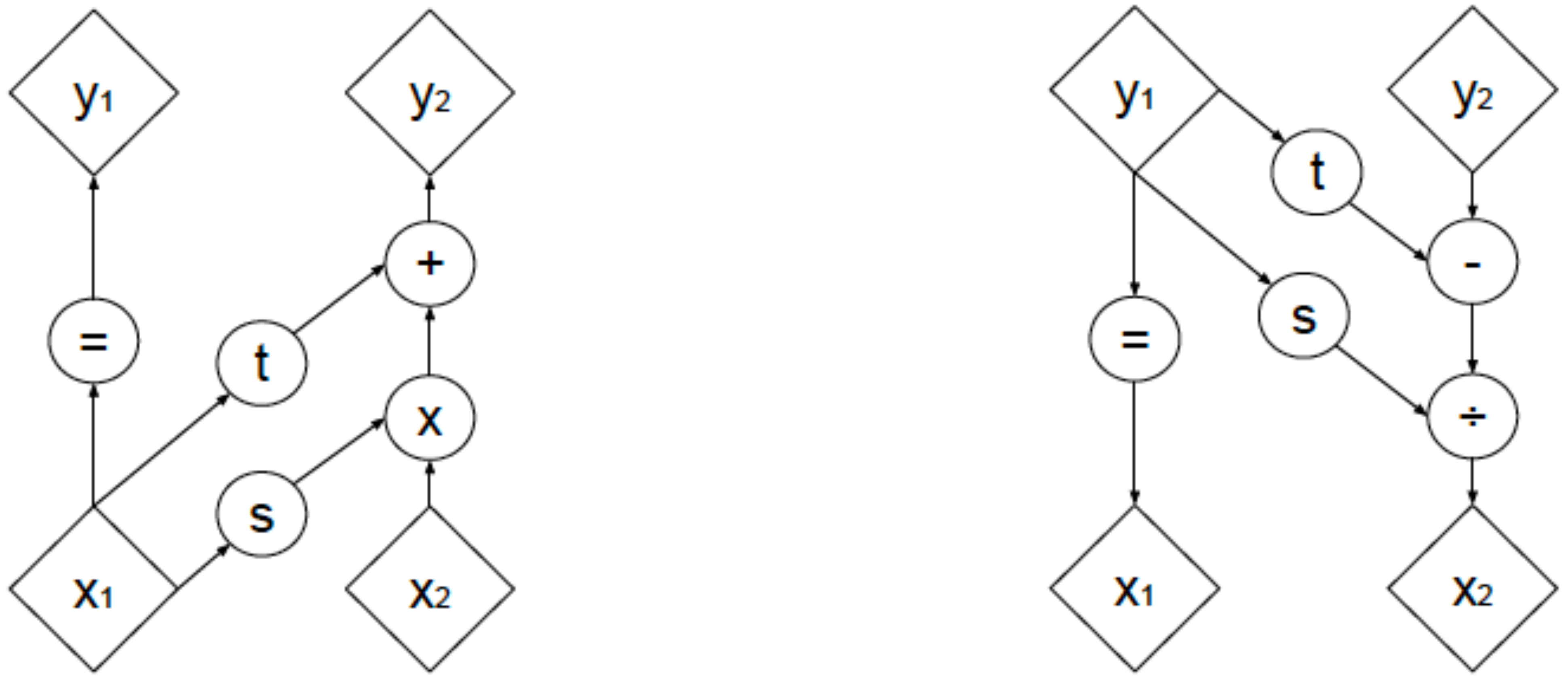


(a) Forward propagation

(b) Inverse propagation

Neural Spline Flows

- Coupling transform



(a) Forward propagation

(b) Inverse propagation

- Monotonic rational-quadratic spline transform

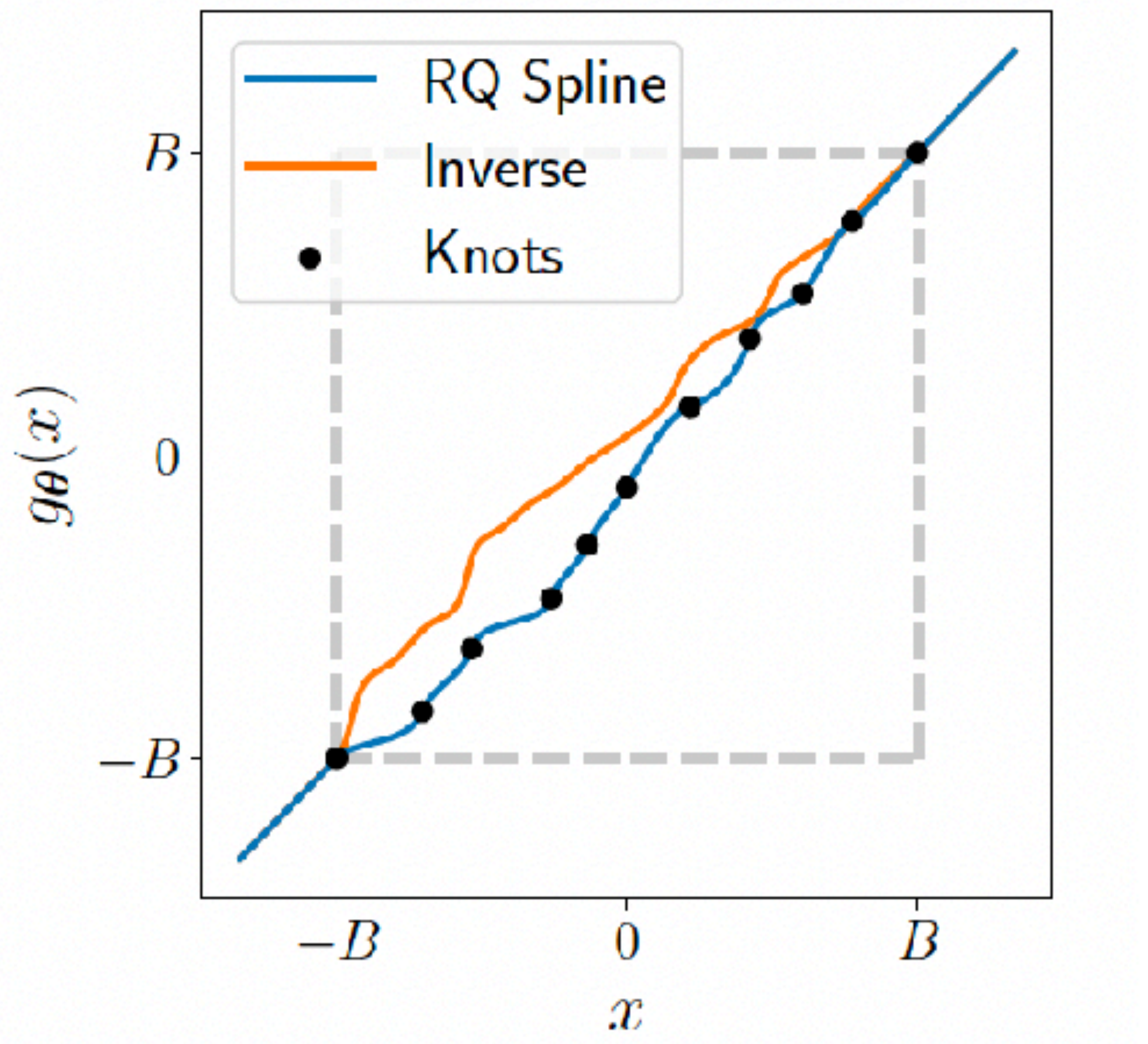
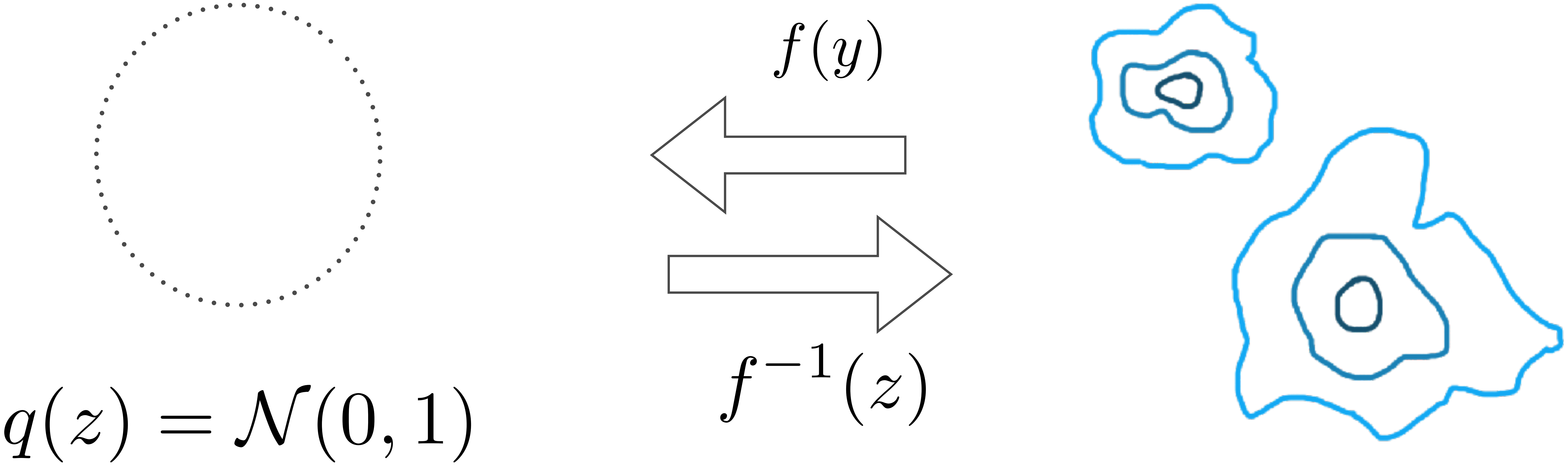


image: Duncan C. et al, Neural Spline Flows

Conditioning

- Do not have access to samples from posterior



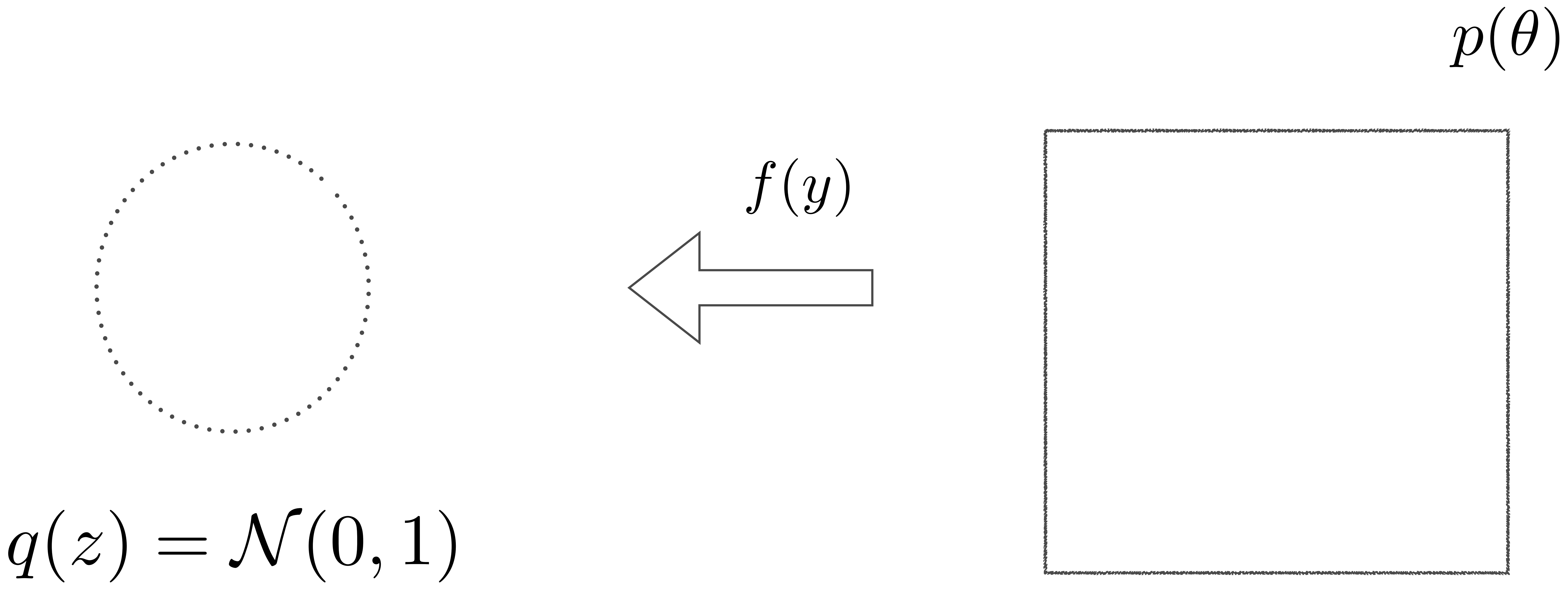
$f(y)$

$f^{-1}(z)$

$q(z) = \mathcal{N}(0, 1)$

Conditioning

- Do not have access to samples from posterior
- Have access to samples from prior +



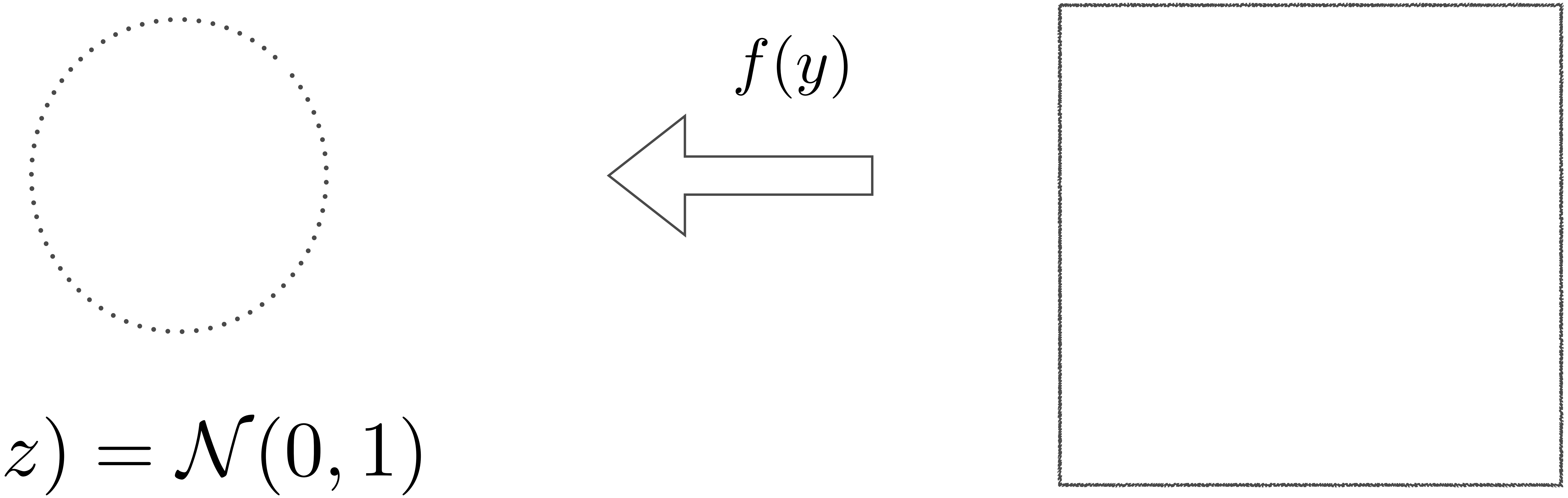
$p(\theta)$

$f(y)$

$q(z) = \mathcal{N}(0,1)$

Conditioning

- Do not have access to samples from posterior
- Have access to samples from prior +
- Can generated simulated data $x = h(\theta) + n$



$p(\theta)$

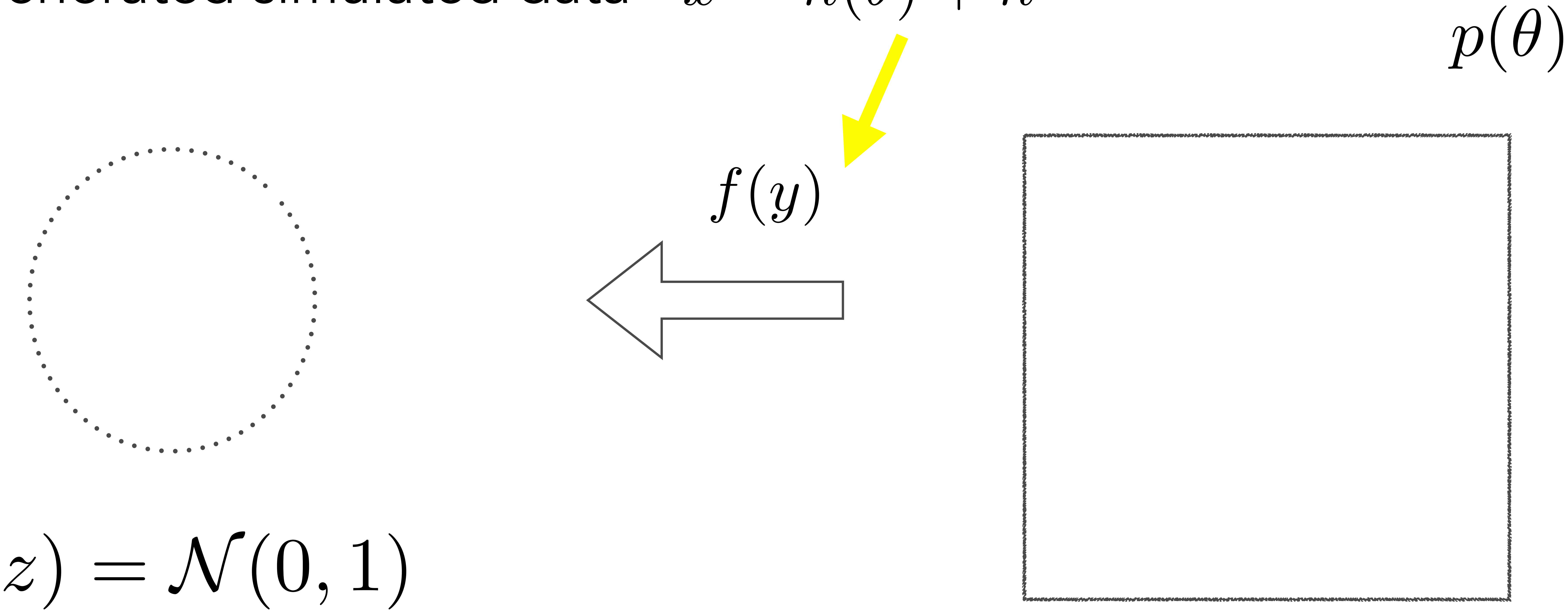
$f(y)$

$q(z) = \mathcal{N}(0, 1)$

Conditioning

Condition inverted map on real data

- Do not have access to samples from posterior
- Have access to samples from prior +
- Can generated simulated data $x = h(\theta) + n$



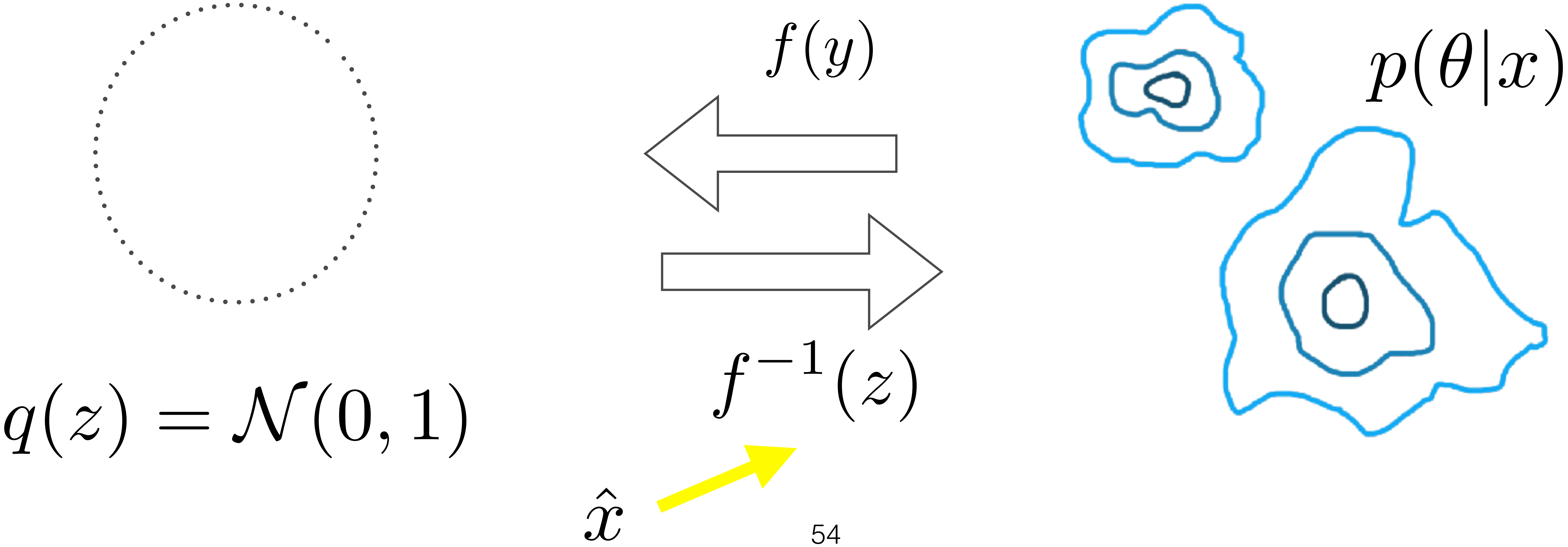
$$q(z) = \mathcal{N}(0, 1)$$

Therefore have access to the joint sample $p(x, \theta) = p(x \mid \theta)p(\theta)$

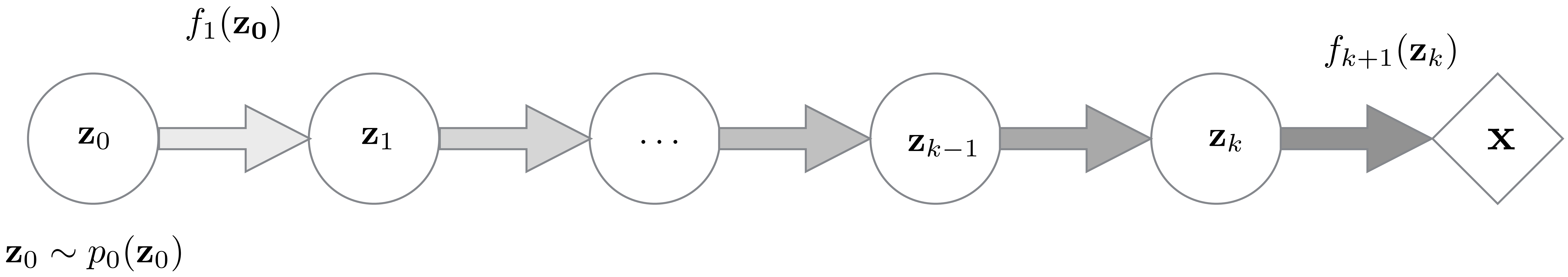
Conditioning

Condition inverted map on real data

- Do not have access to samples from posterior
- Have access to samples from prior +
- Can generated simulated data $x = h(\theta) + n$



Composing flows



PCA and Autoencoders

Waveform embedding

- Low frequency sensitivity -> long waveforms
- Construct reduced orthogonal basis
- Use coefficients of the waveform projection on a new basis

Waveform embedding

Decompose a matrix constructed of the set of waveforms

$$\mathbf{H} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^{\mathbf{T}}$$

Waveform embedding

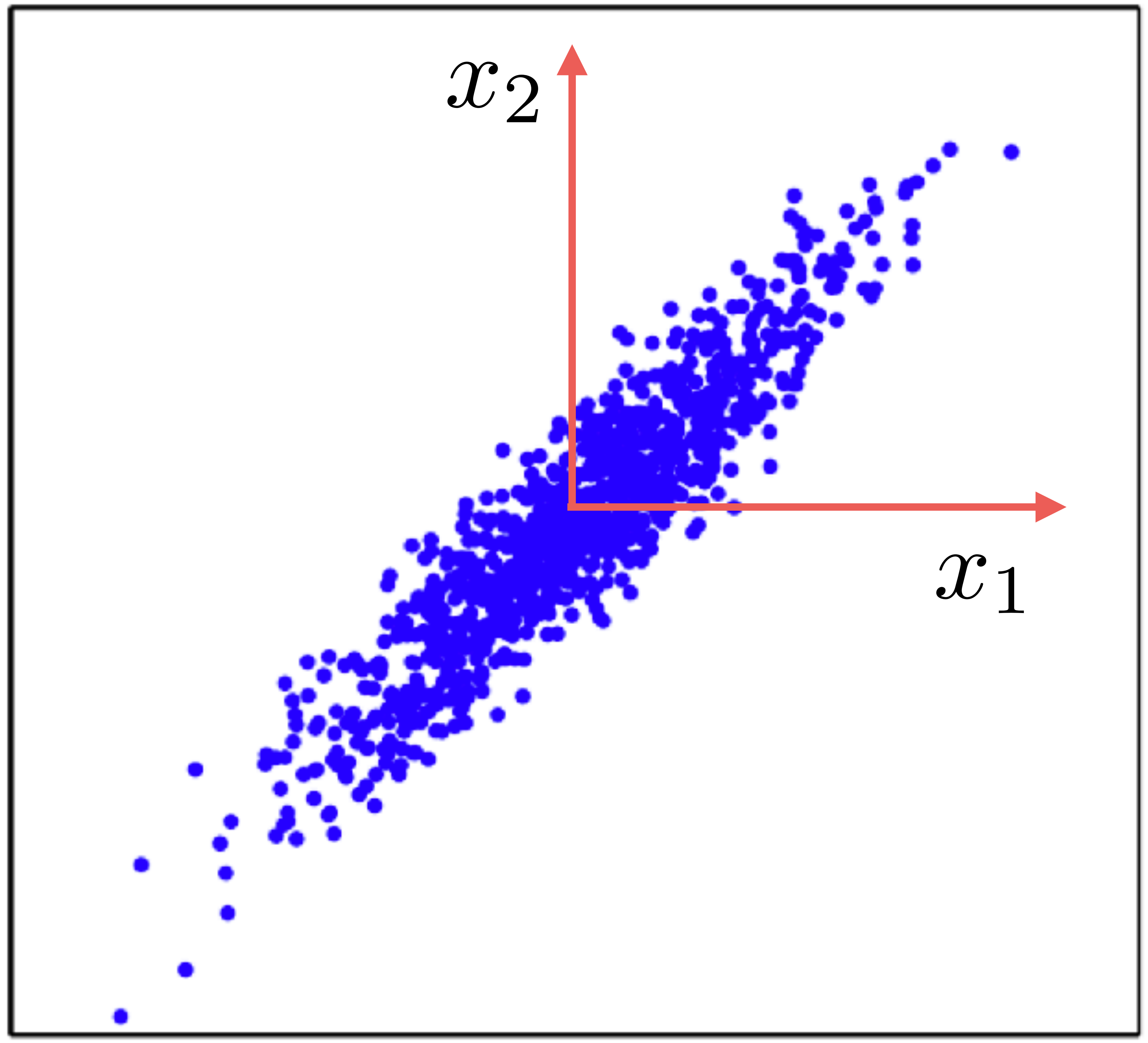
Decompose a matrix constructed of the set of waveforms

$$\mathbf{H} = \mathbf{V\Sigma U^{T}}$$

Project sample simulated data on this basis

$$v'_{\alpha\mu} = \frac{1}{\sigma_\mu}\sum_{j=1}^{N} h_{\alpha j} u_{\mu j}$$

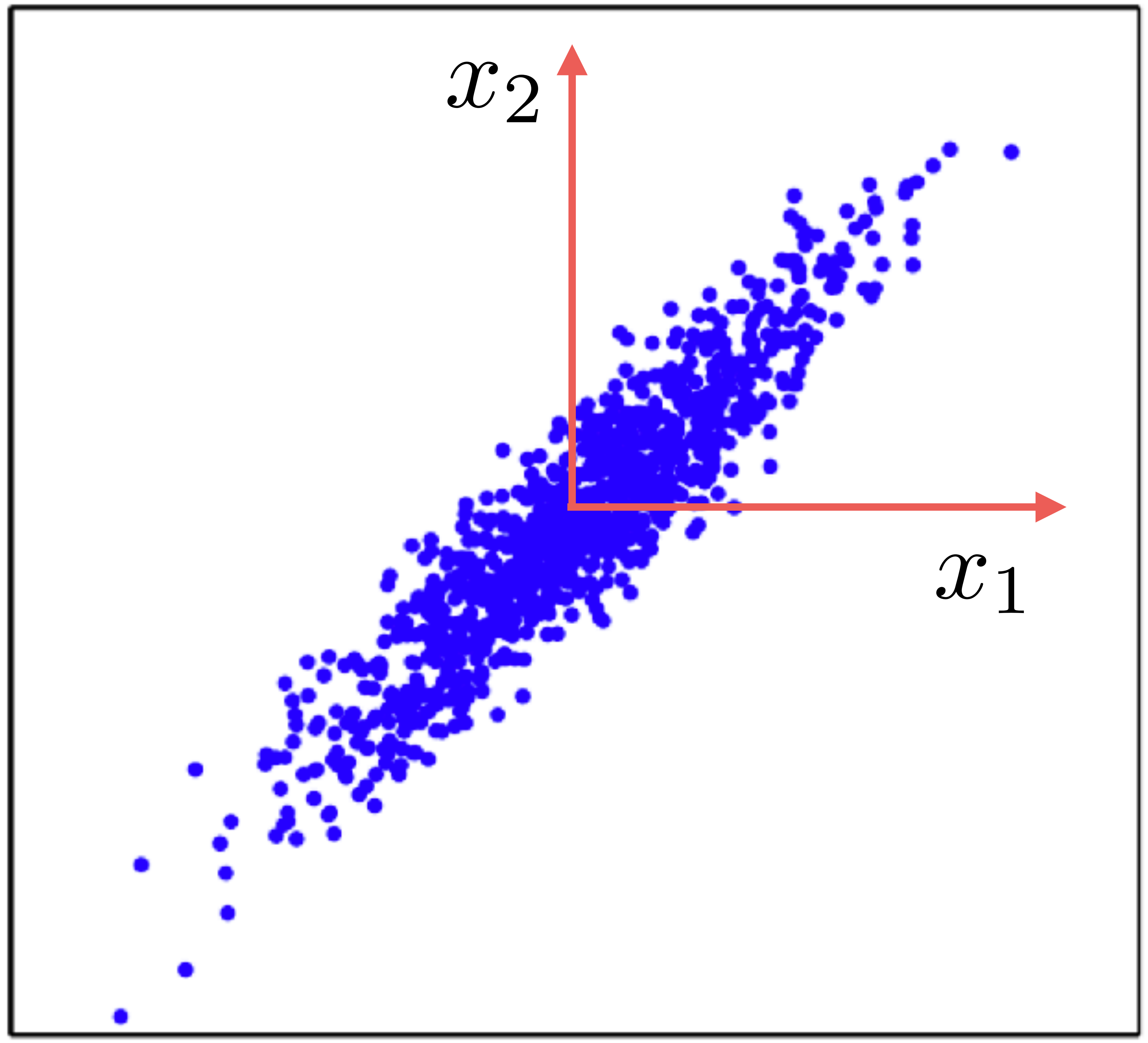
Principle Component Analysis



PCA maps original data
into a new coordinate system
which maximises variance of the data

$$y_1 = \sum_{k=1}^{n} w_{k1} x_k$$

Principle Component Analysis



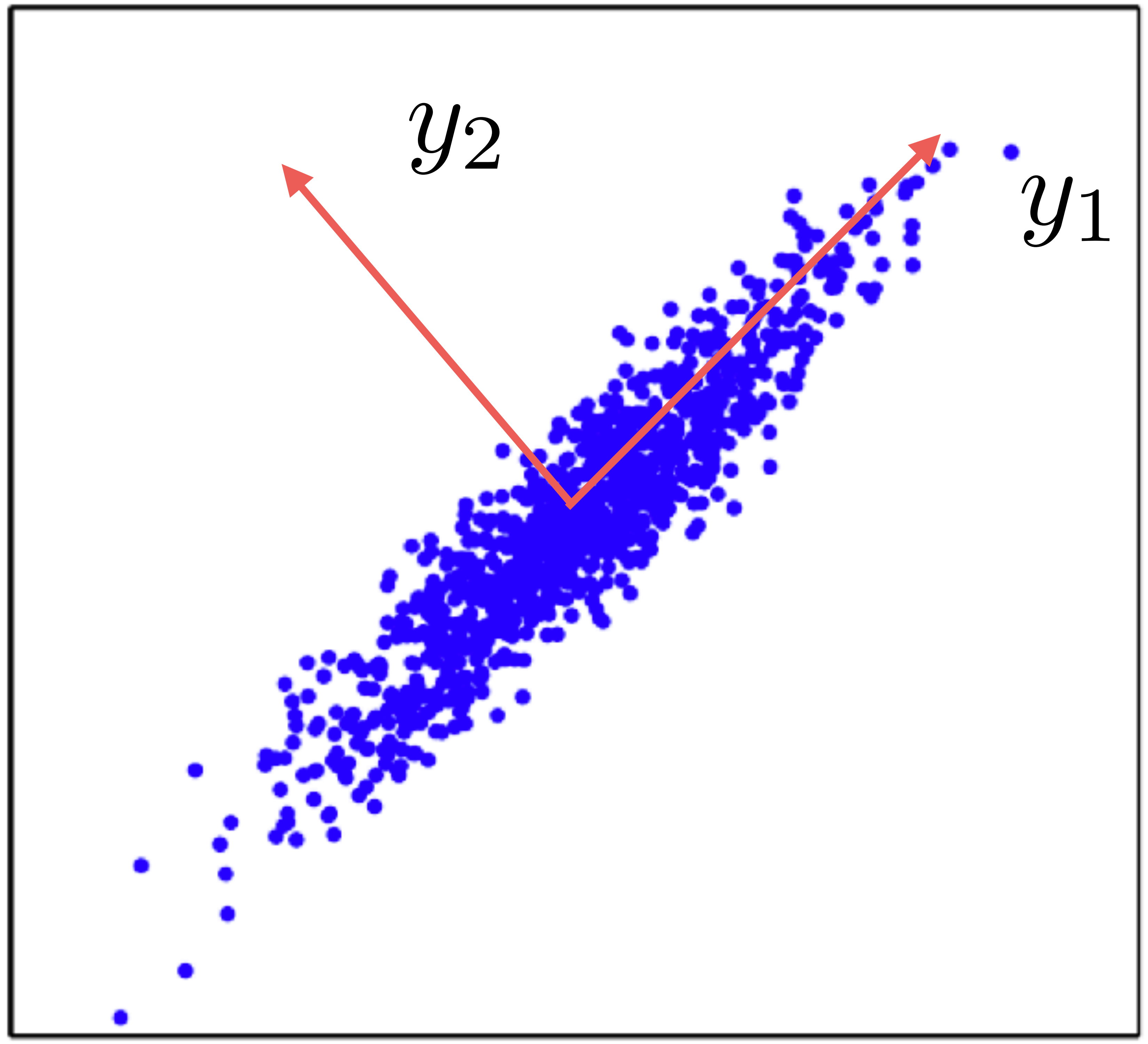
The mapping to the new basis can be expressed using the eigenvectors of the Covariance matrix

$$C = E\{\mathbf{x}\mathbf{x}^T\}$$

Eigenvalue decomposition

$$\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{U}^T$$

Principle Component Analysis



The vector of principle components will be

$$\mathbf{y} = \mathbf{U}^T \mathbf{x}$$

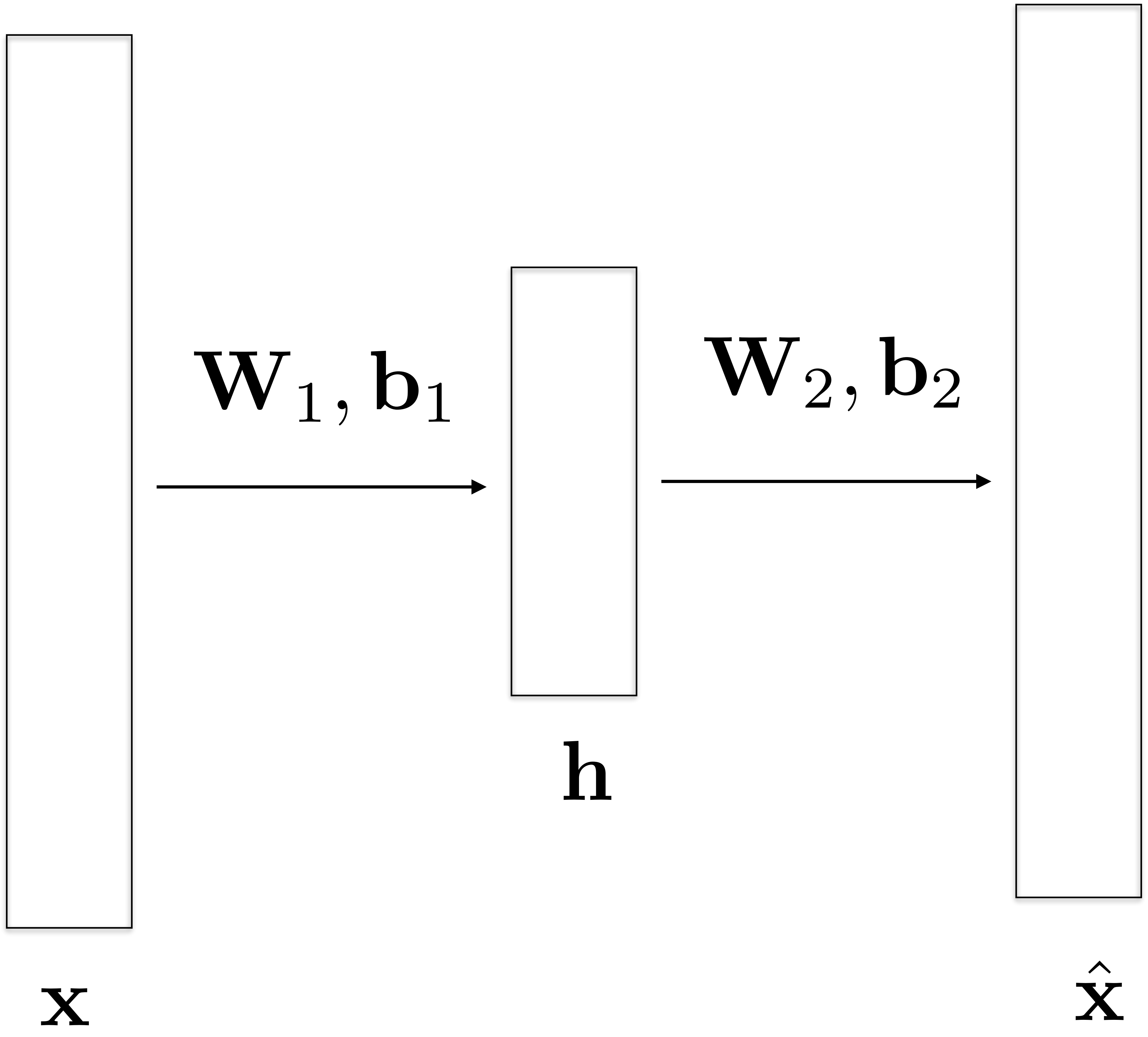
Principle Component Analysis

It has been shown that it is possible to formulate PCA in terms of Neural Networks

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

$$J_{MSE} = \frac{1}{T}\sum_{j=1}^{T} ||\hat{\mathbf{x}}(j) - \mathbf{W}\mathbf{W}^T\mathbf{x}(j)||^2$$

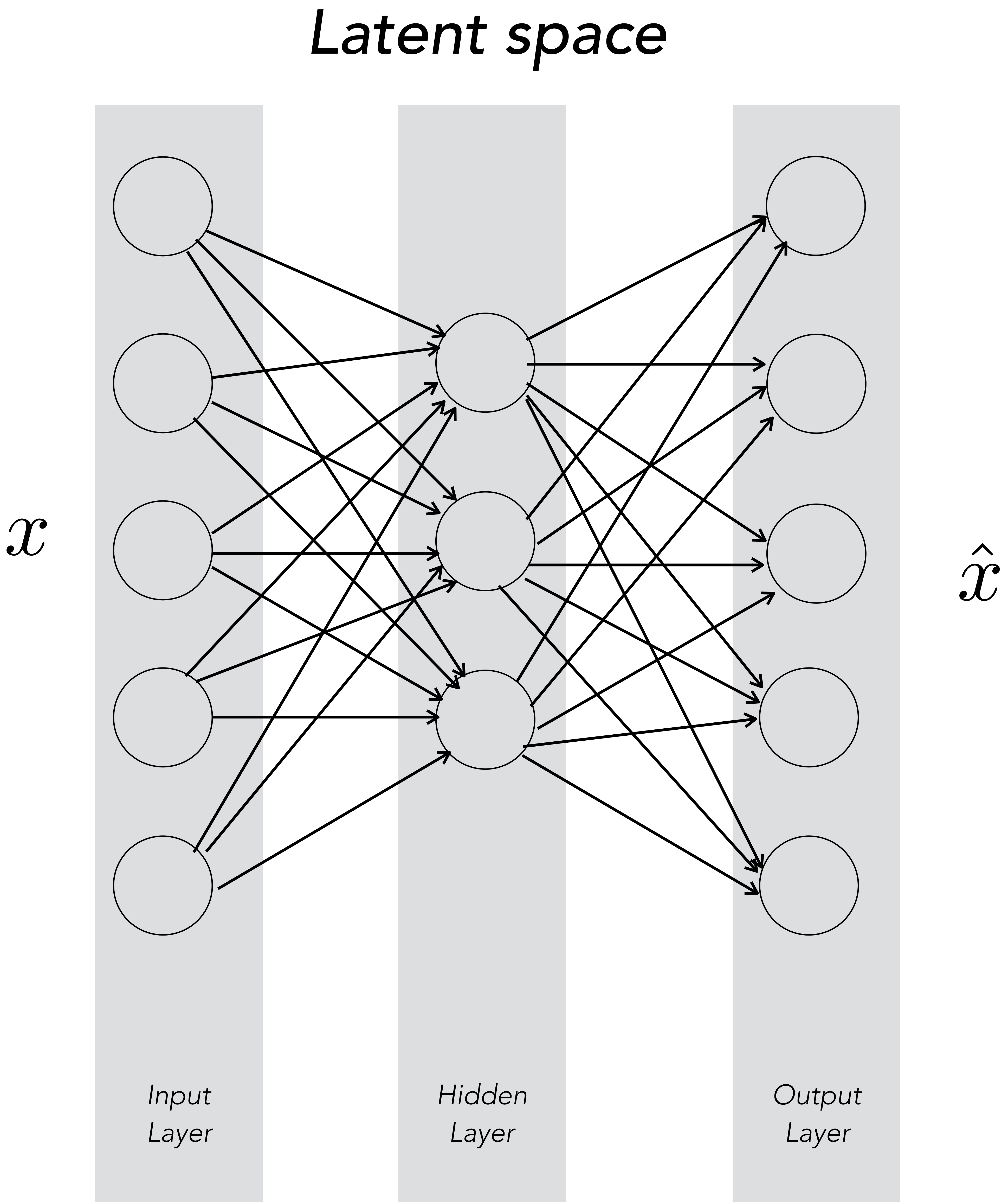
Principle Component Analysis



$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{y} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

Autoencoder

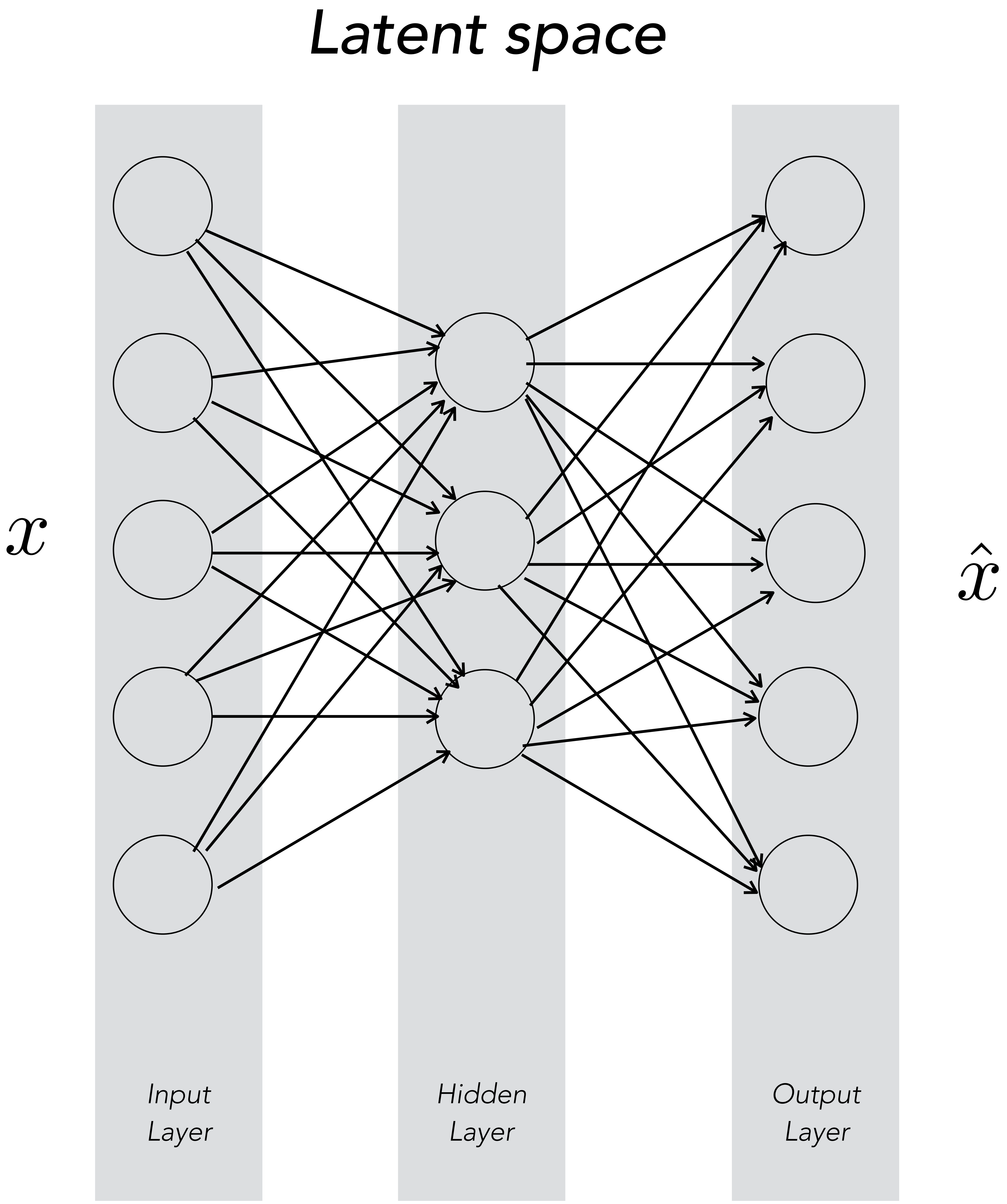


Autoencoders is unsupervised learning technique, which solves the task of representational learning.

Learning is done by comparing reconstruction to original input.

$$\mathcal{L}\left(x, \hat{x}\right)$$

Autoencoder



Variations:

- Denoising autoencoders
- Contractive autoencoders
- Undercomplete autoencoders

Autoencoder

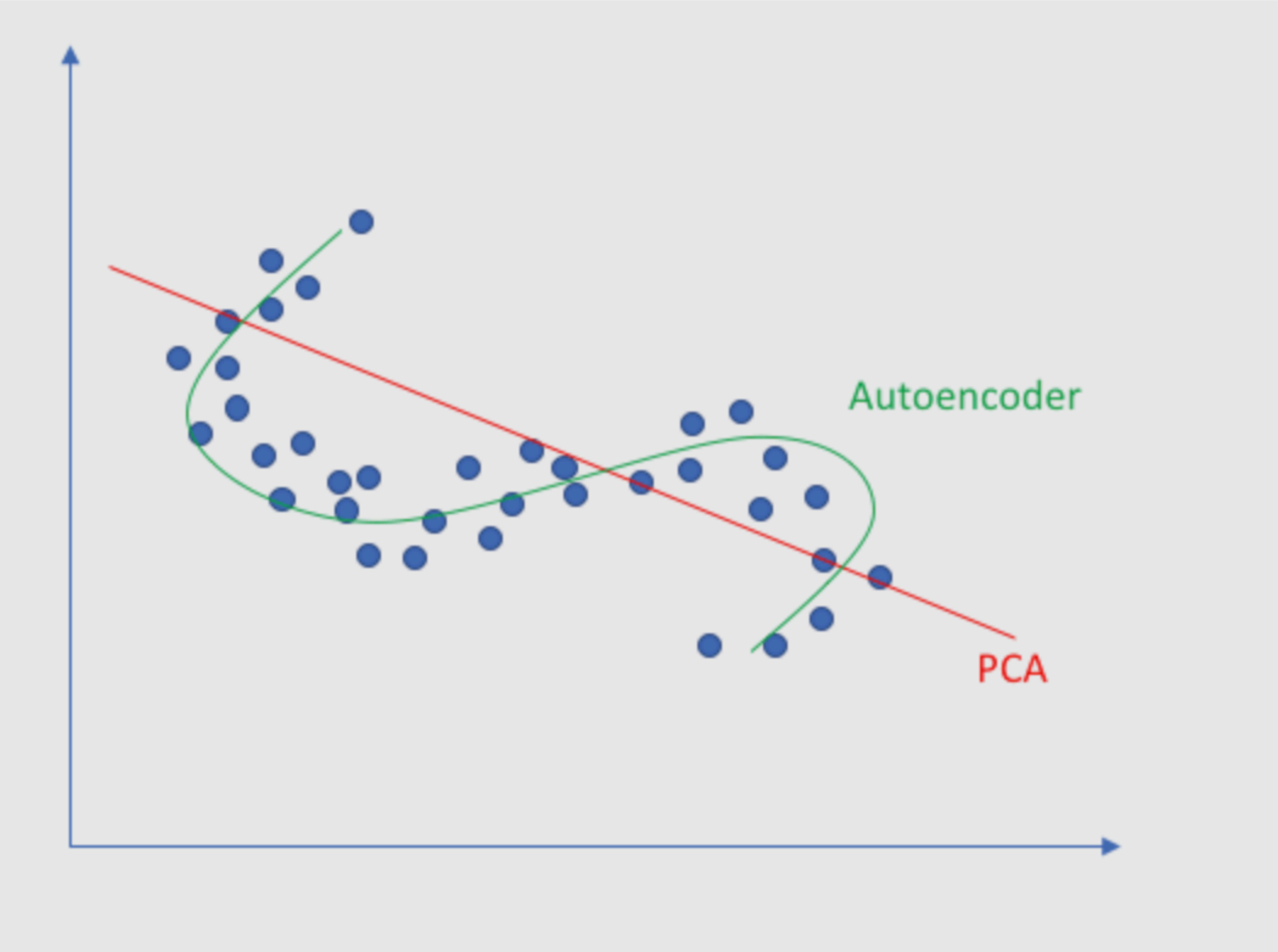
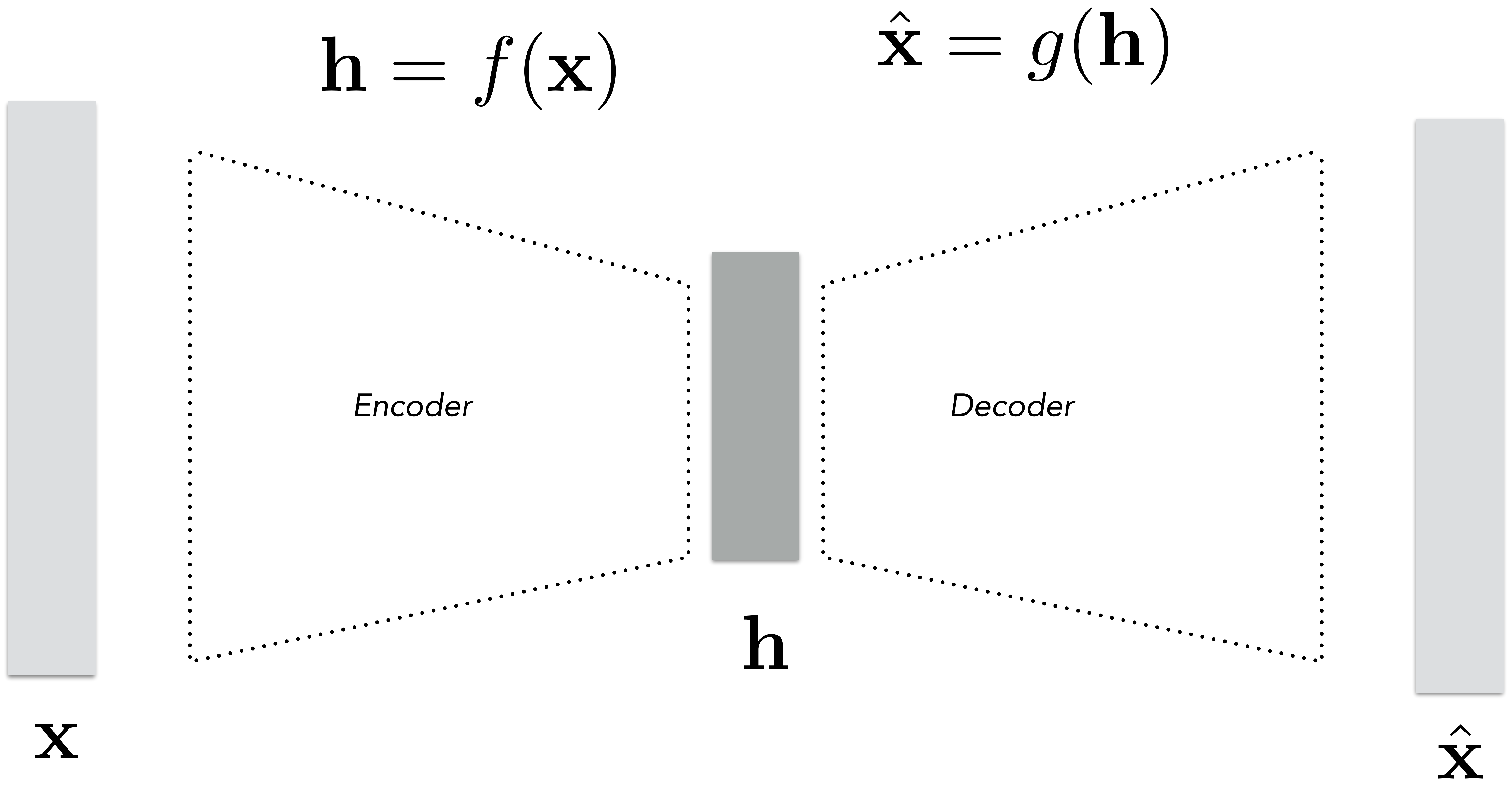


Image: https://www.jeremyjordan.me/autoencoders/

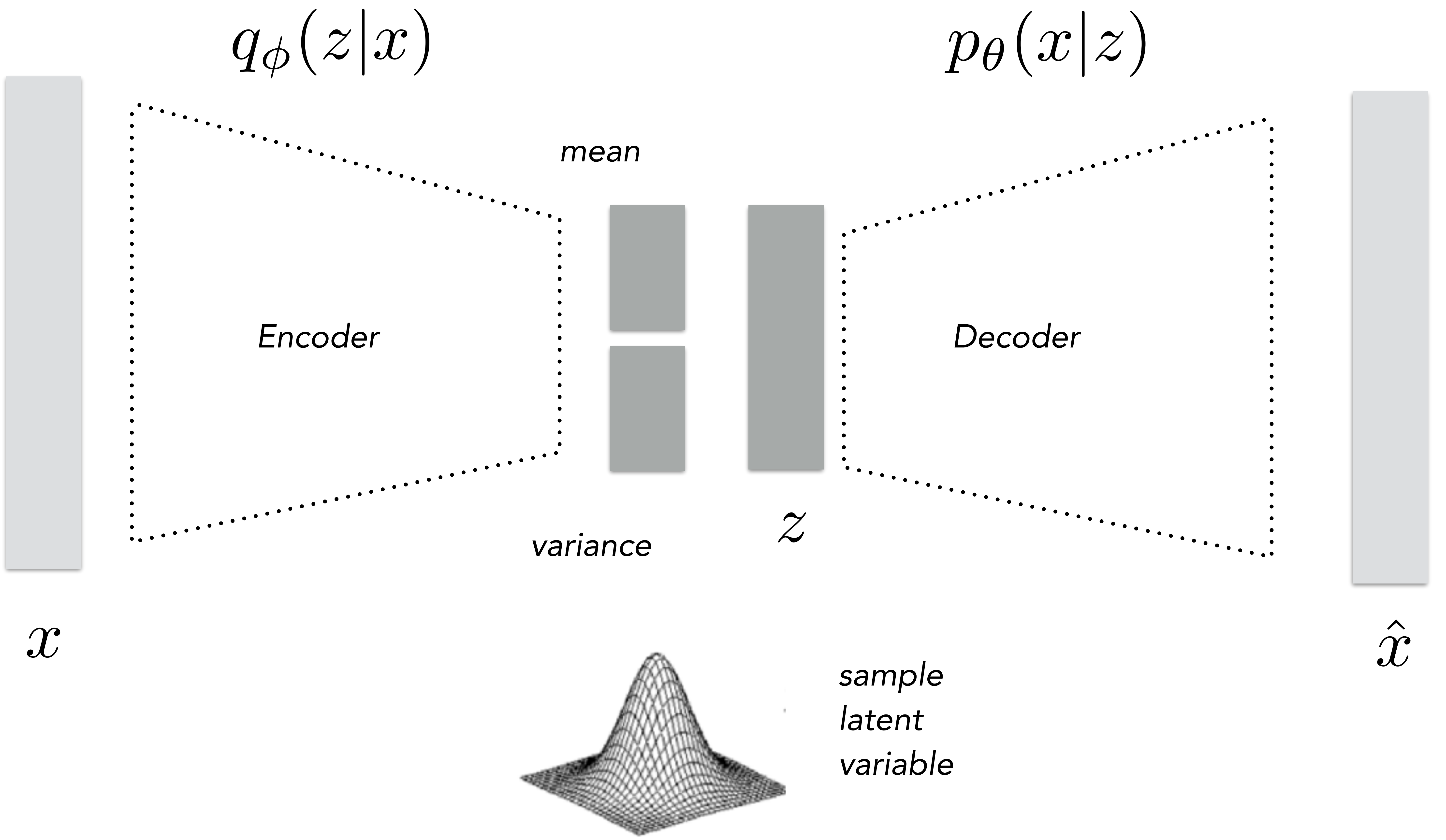
Autoencoder



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

$$\hat{\mathbf{x}} = g(\mathbf{h})$$

Variational Autoencoder

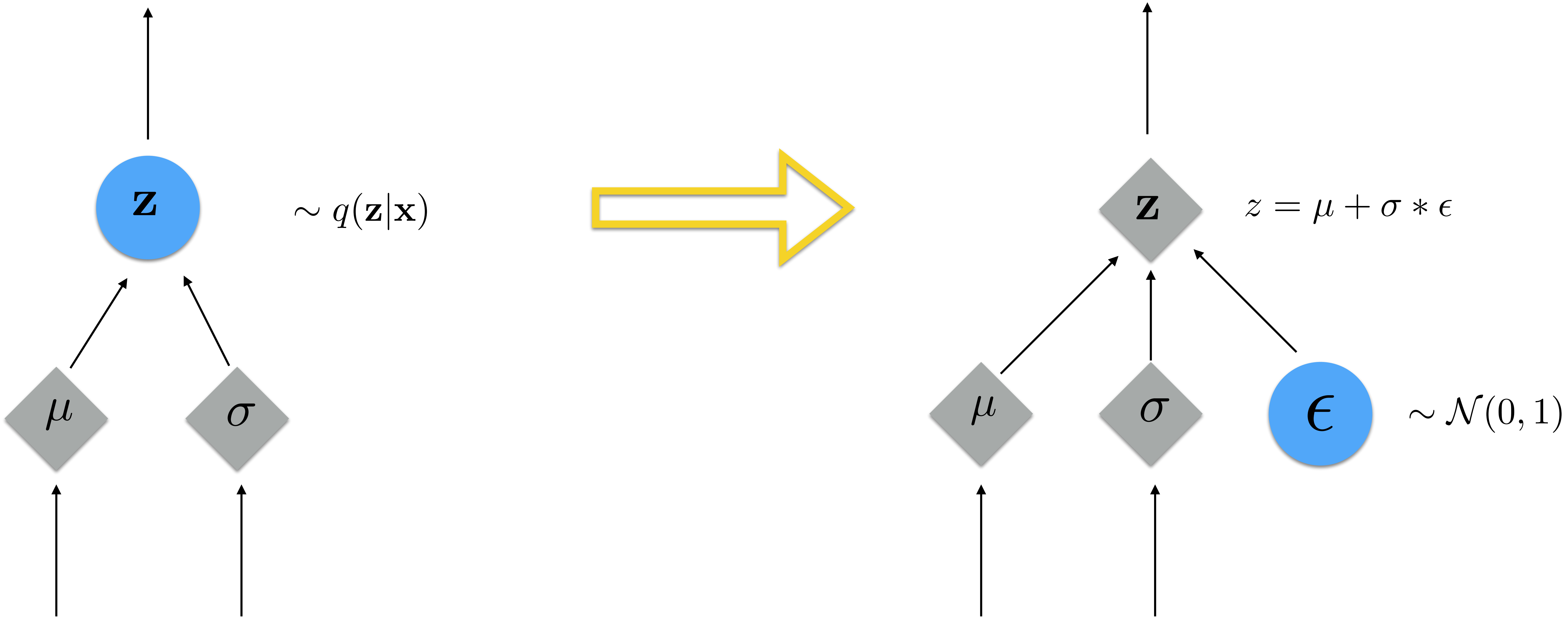


Probabilistic representation of latent space.

Auto-Encoding Variational Bayes, Diederik P Kingma, Max Welling [*https://arxiv.org/abs/1312.6114*](https://arxiv.org/abs/1312.6114)

Reparameterisation trick

Reparemeterization trick: used to propagate back the error



Variational Autoencoder

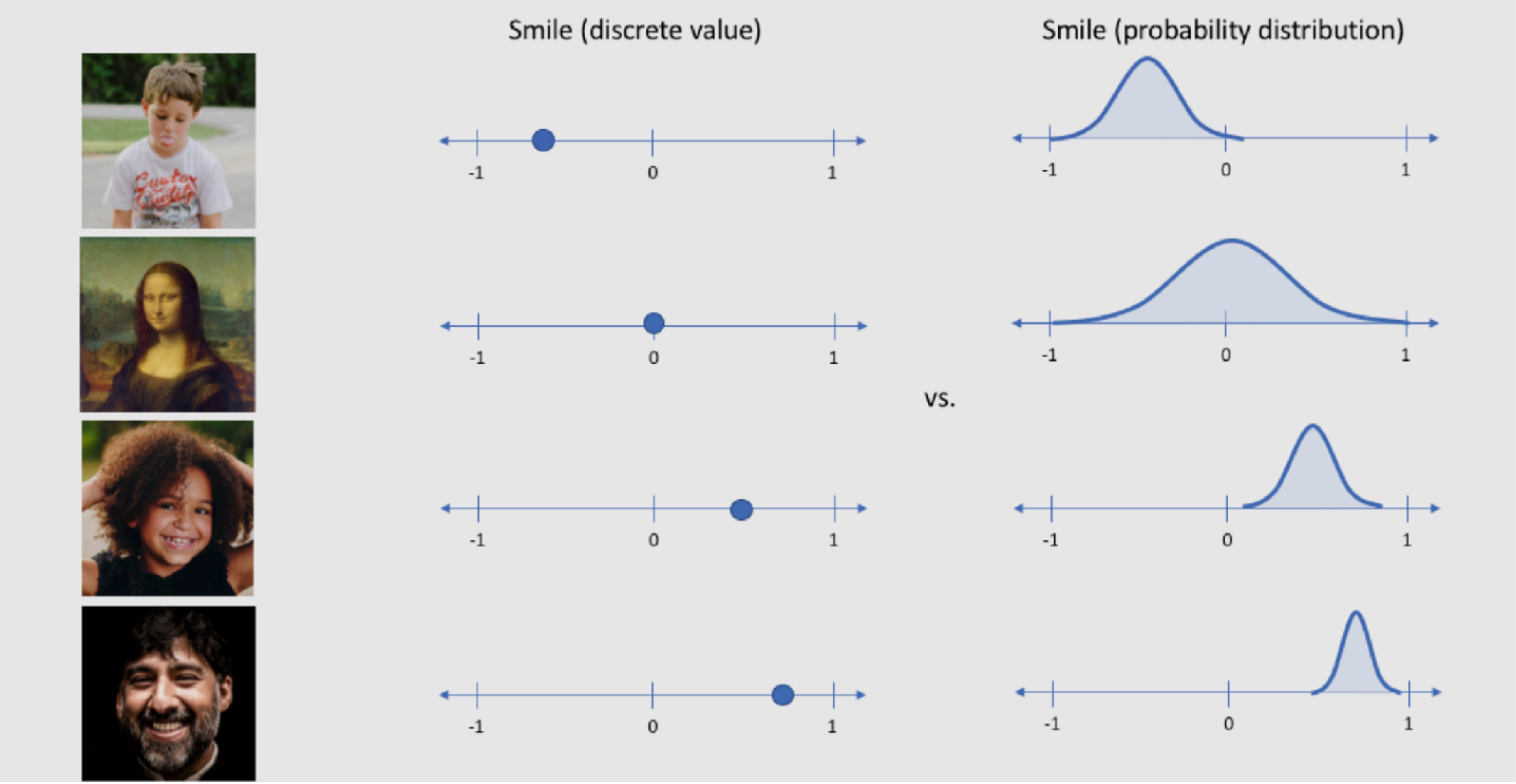
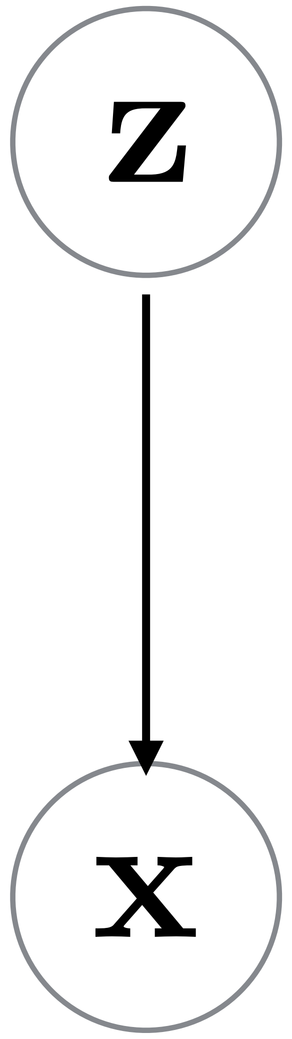


Image: https://www.jeremyjordan.me/variational-autoencoders/

Variational Autoencoder



$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

<— *we want to estimate the latent variables given the data*

$$p(x) = \int p(x|z)p(z)dz = E_{p(z)}\left[p(x|z)\right]$$

<— intractable but
we apply
variational inference
to estimate this value

Variational Autoencoder

$$\log p(x) = \log \int p(x,z) \mathrm{d}z$$

Introduce tractable $q(z|x)$

$$= \log \int p(x,z) \frac{q(z|x)}{q(z|x)} \mathrm{d}z \qquad \geq \mathbb{E}_{q(z|x)} \log \frac{p(x,z)}{q(z|x)}$$

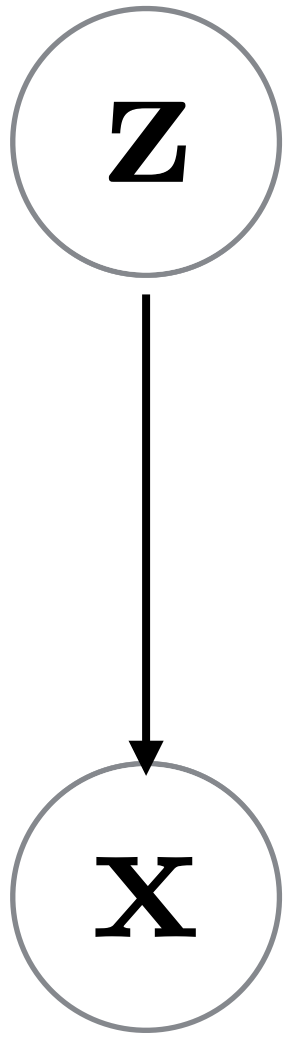
Jensen inequality

$$= \mathbb{E}_{q(z|x)} \log \frac{p(x|z)p(z)}{q(z|x)} \qquad = \mathbb{E}_{q(z|x)} \log p(x|z) + \mathbb{E}_{q(z|x)} \log \frac{p(z)}{q(z|x)}$$

$$= \text{likelihood} - D_{KL}[q(z|x)||p(z)]$$

ELBO — evidence lower bound

Variational Autoencoder



observe

Lets approximate $p(z|x)$ *with* $q(z|x)$

such that we set a condition that they are close to each other as possible.

We can enforce this condition by minimising Kullback–Leibler divergence

Optimisation

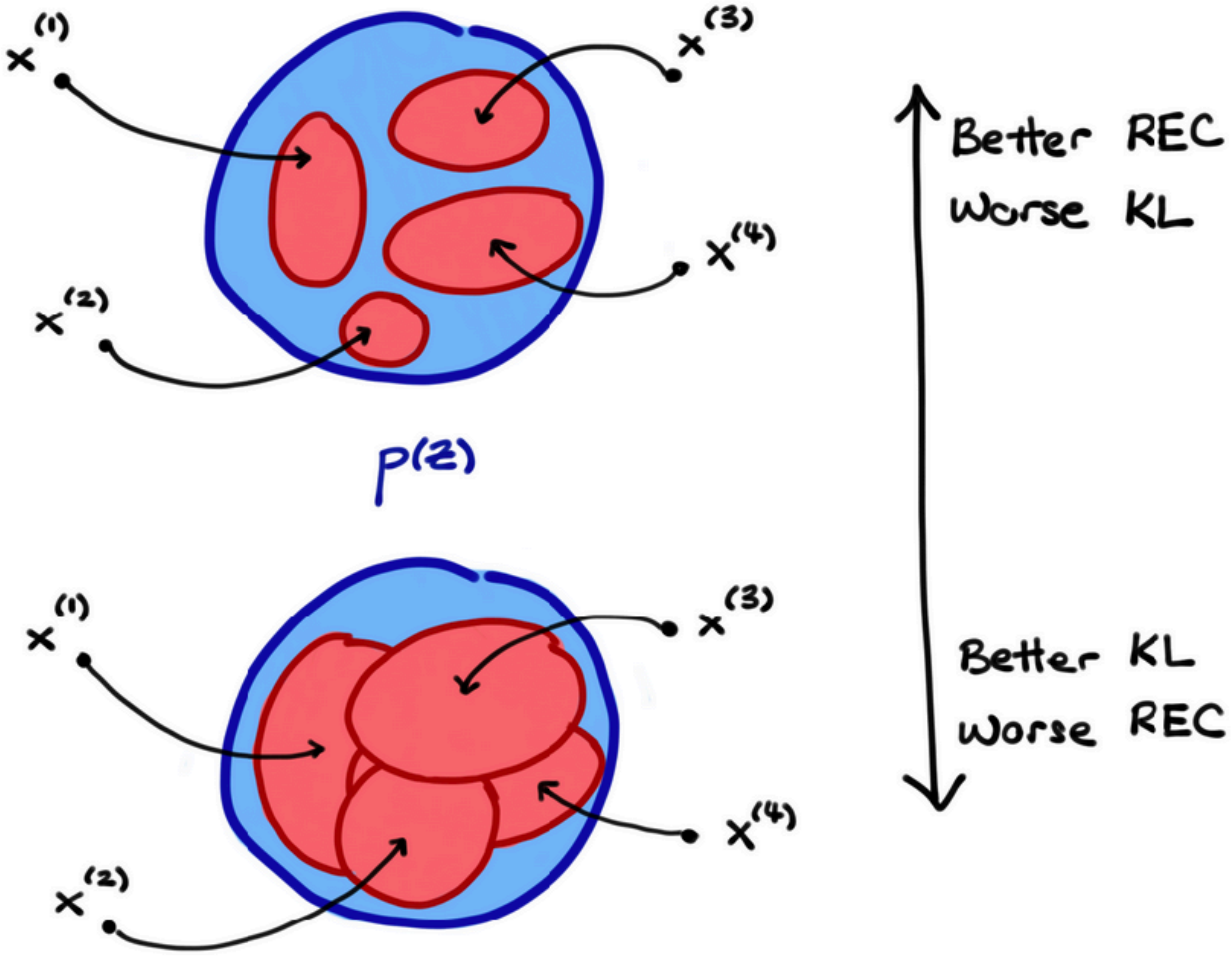
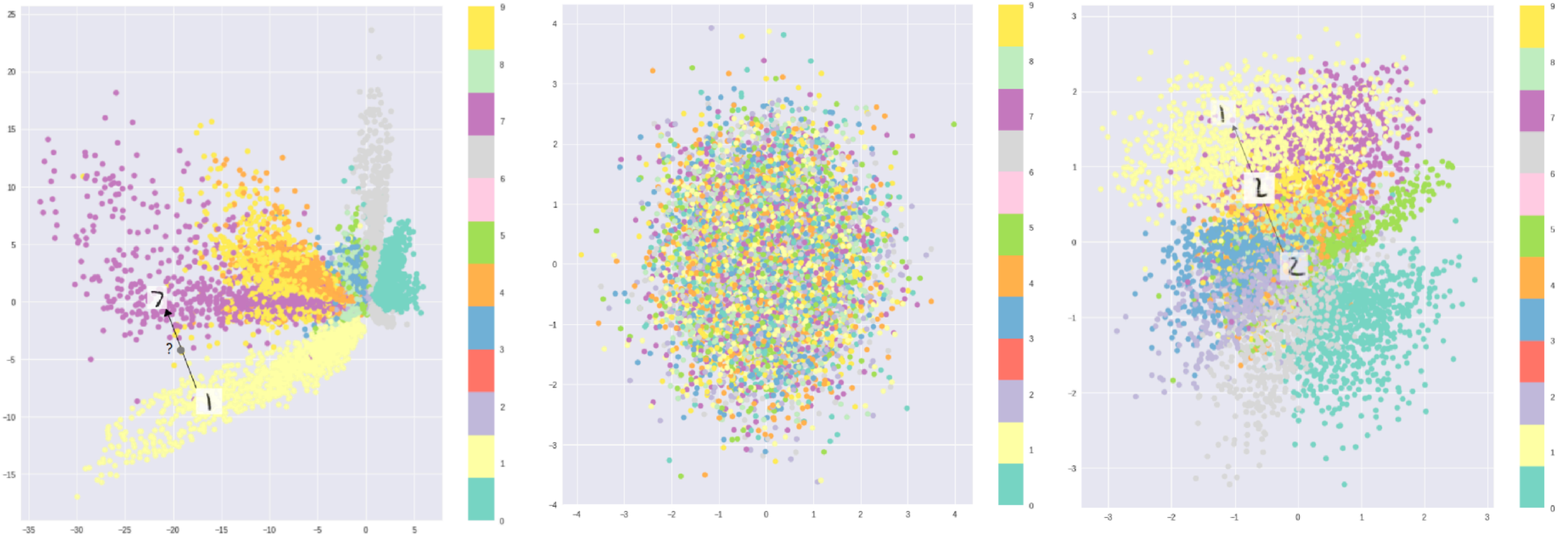


Image: http://ruishu.io/2018/03/14/vae/

Latent space



https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

Practical tutorial

https://github.com/NataliaKor/tutorial/blob/main/tutorial-ML-for-GWPE.ipynb