Bayesian deep learning and implicit likelihood inference

Marc Huertas-Company & Hubert Bretonnière

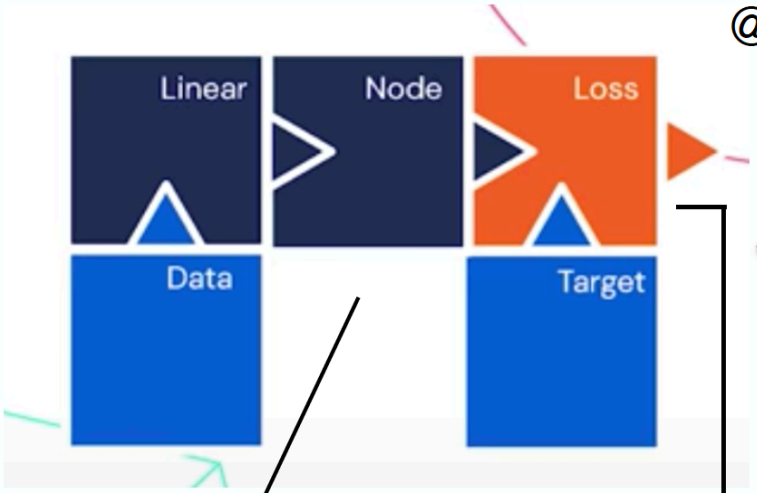
Rodolphe Clédassou School 2025 - Y2

Program

- Basics of bayesian statistics (reminders) and implicit likelihood inference (simulation based inference)
- Neural density estimation techniques for SBI
- Applications to inference in cosmology / astrophysics

NNs as universal approximators (Y1)

(deepmind @Czarnecki)

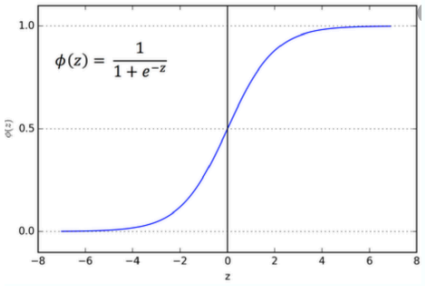


$$f_w(X) = \theta$$

$$\vdots$$

$$g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

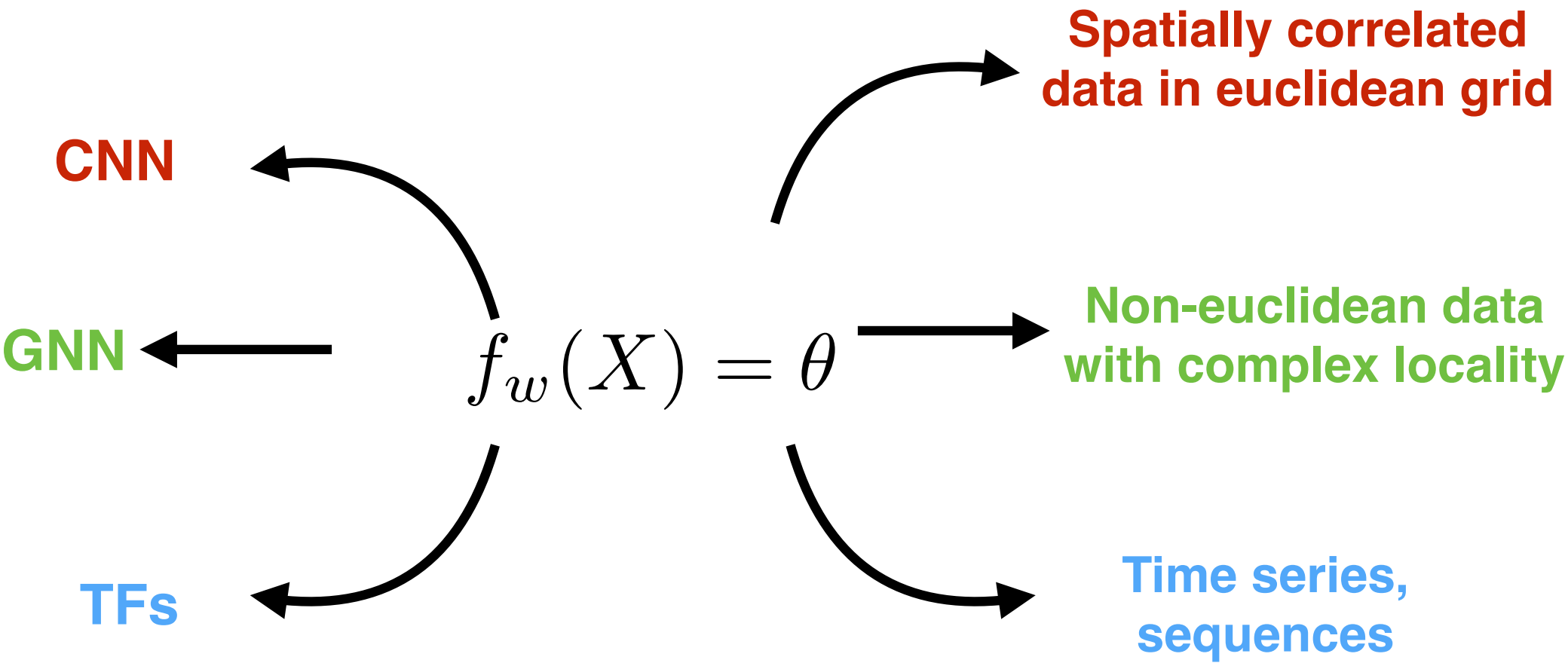
activation function



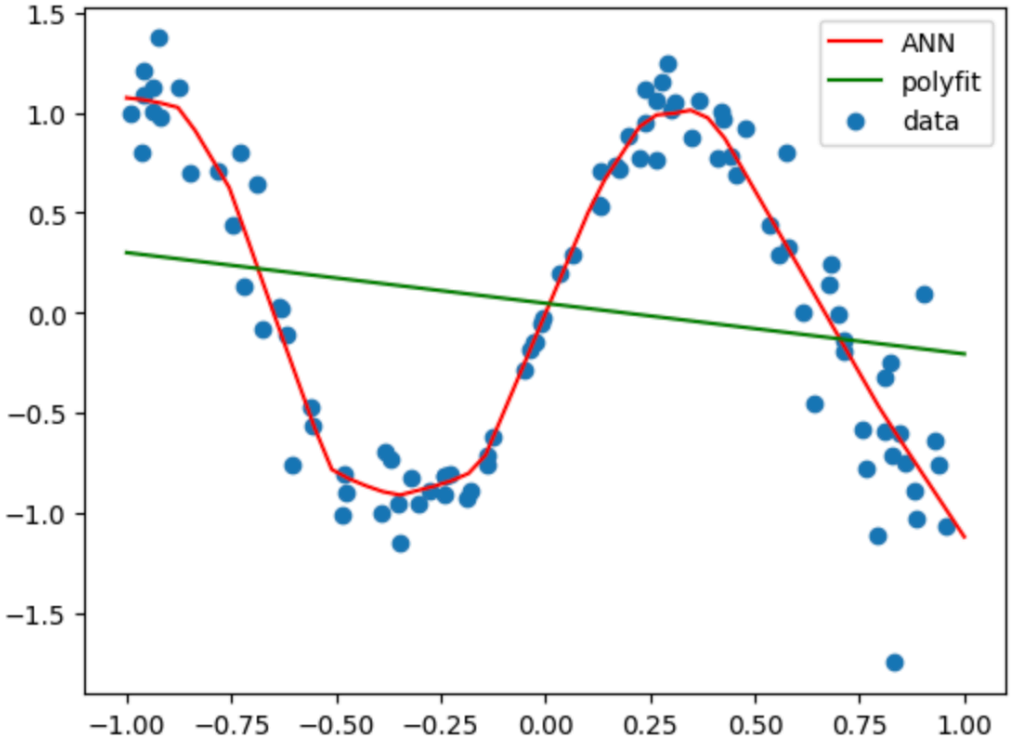
$$\frac{1}{N}\sum_{i=1}^{N}(y_i - f_w(x))^2$$

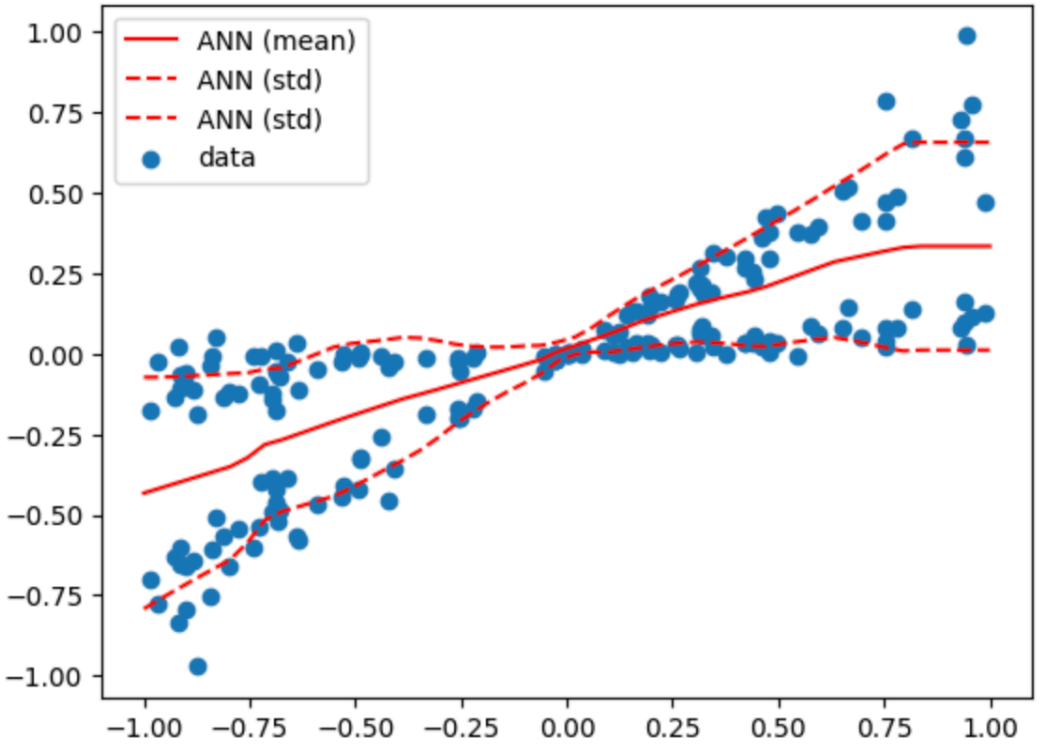
Minimized through gradient descent (backpropagation)

NNs as universal approximators (Y1)



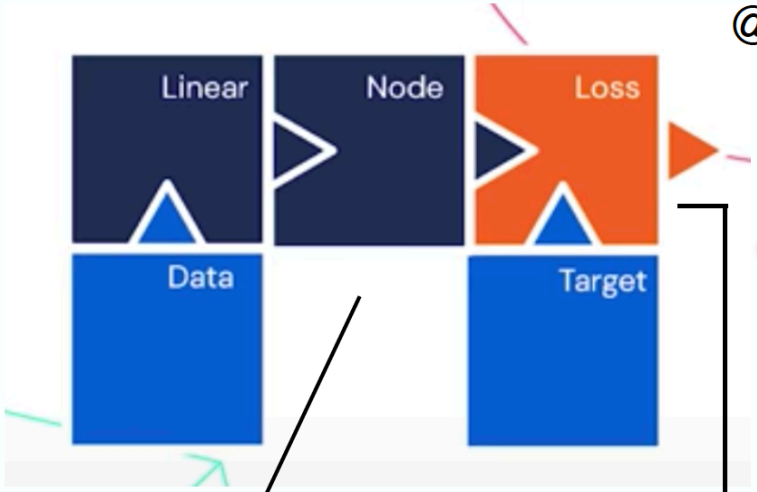
relatively useful…





NNs as statistical models (Y2)

(deepmind @Czarnecki)

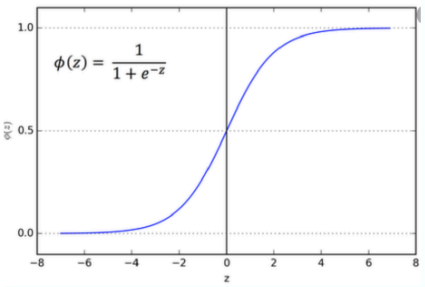


$$p_w(\theta|X)$$

$$f_w(X) = \theta$$

$$g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

activation function



$$\frac{1}{N}\sum_{i=1}^{N}(y_i - f_w(x))^2$$

Minimized through gradient descent (backpropagation)

"standard" bayesian inference

goal of inference

$$\underbrace{p(\theta \mid X)}_{\textit{posterior}} = \frac{\overbrace{p(X \mid \theta)}^{\textit{likelihood}}\overbrace{p(\theta)}^{\textit{prior}}}{p(X)}$$

"standard" bayesian inference

goal of inference

posterior $p(\theta \mid X)$ $=$ $\dfrac{\textit{likelihood}\ p(X \mid \theta)\ \ \textit{prior}\ p(\theta)}{p(X)}$

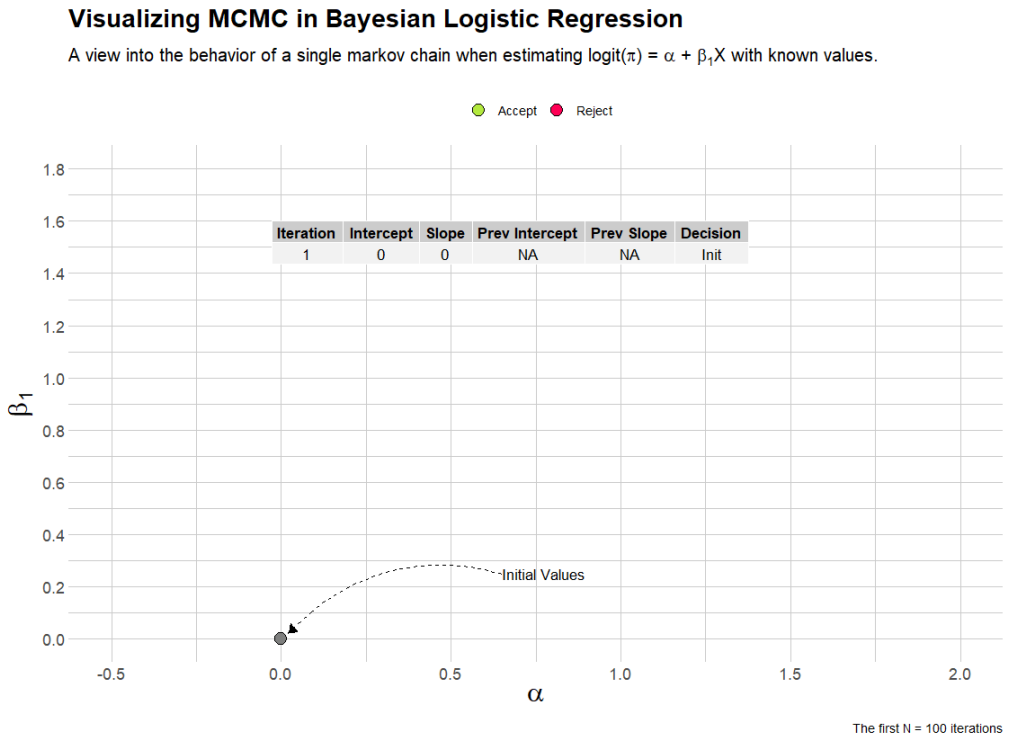
$$\ln p(X \mid \theta) = \ln \mathscr{L} = \left(X - m(\theta)\right)^T \mathbf{C}^{-1} \left(X - m(\theta)\right)$$

covariance matrix

model

"standard" bayesian inference

$$\ln p(\theta \mid X) = \ln p(X \mid \theta) + \ln p(\theta) + \text{const}.$$



sample $p(\theta \mid X)$ using montecarlo sampling to estimate posterior (*e.g. MCMC, HMC, nested sampling)*

"standard" bayesian inference

Gaussian likelihood is often **an incorrect assumption (not enough data)**

$$\ln p(X \mid \theta) = \ln \mathscr{L} \neq \left(X - m(\theta)\right)^T \mathbf{C}^{-1} \left(X - m(\theta)\right)$$

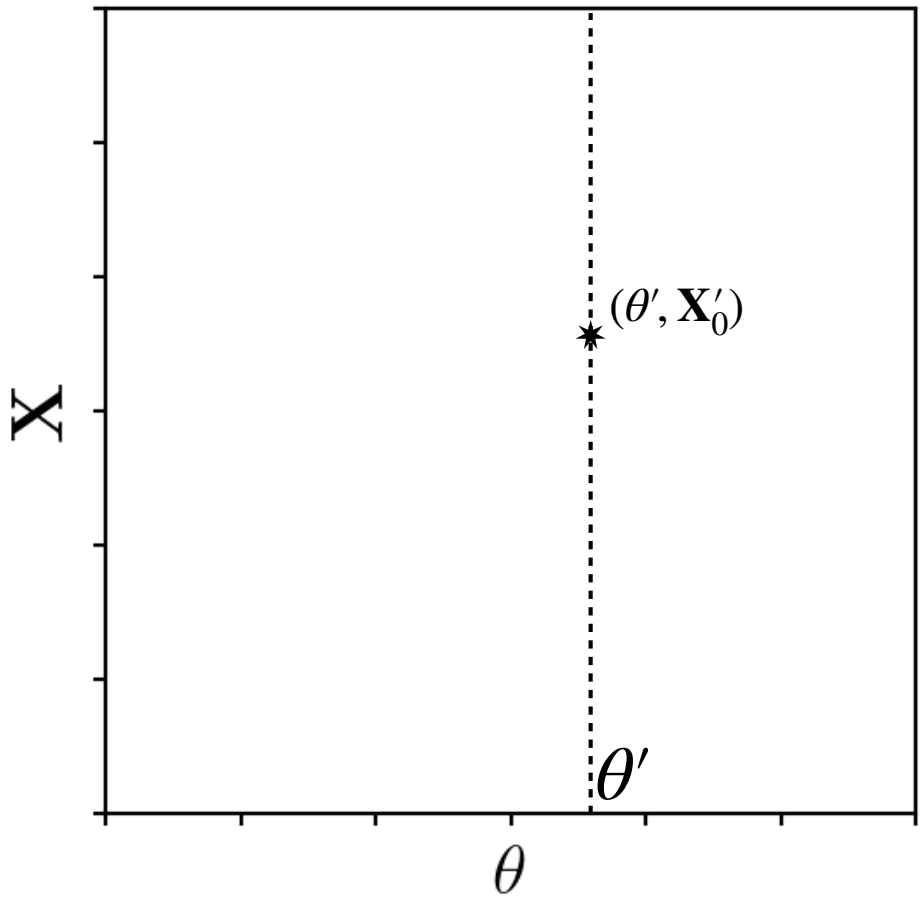
Sometimes the likelihood is intractable. e.g. Baryonic properties of DM haloes?

MCMC is slow! Impractical to infer *billions* of observations and/or on large parameter spaces.

references: Hogg+(2010), Hogg (2012)

forward model F that can simulate observations:

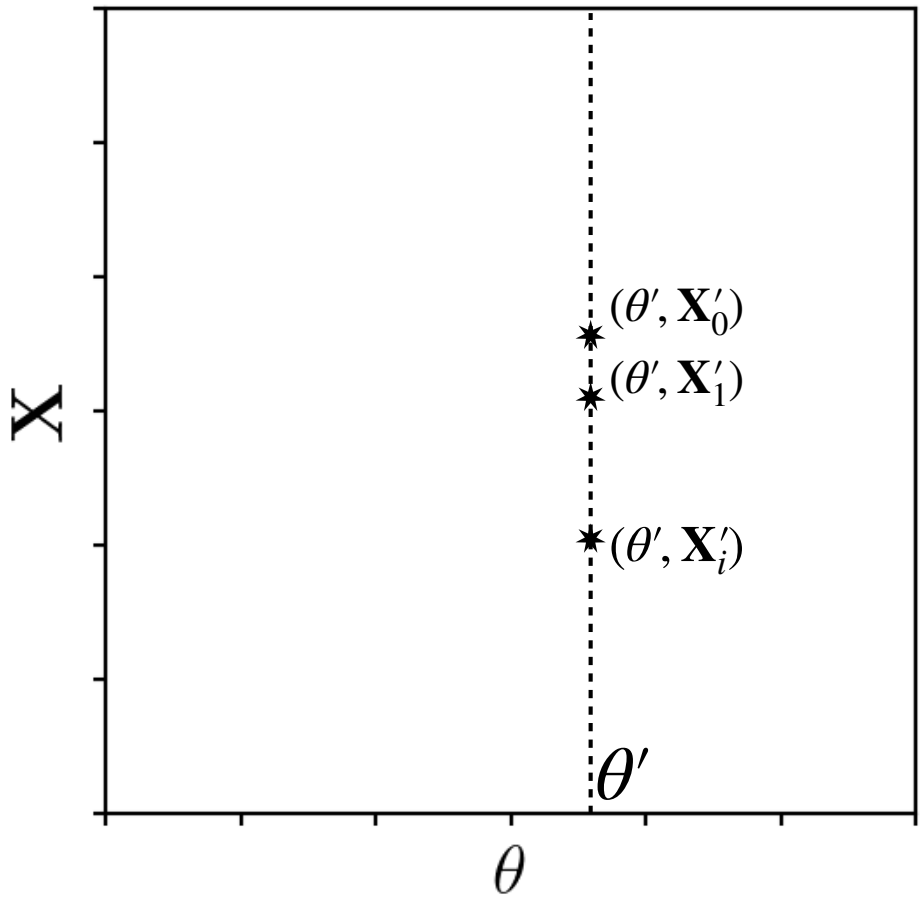
$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



F must **include noise model** and **observational systematics**

forward model F that can simulate observations:

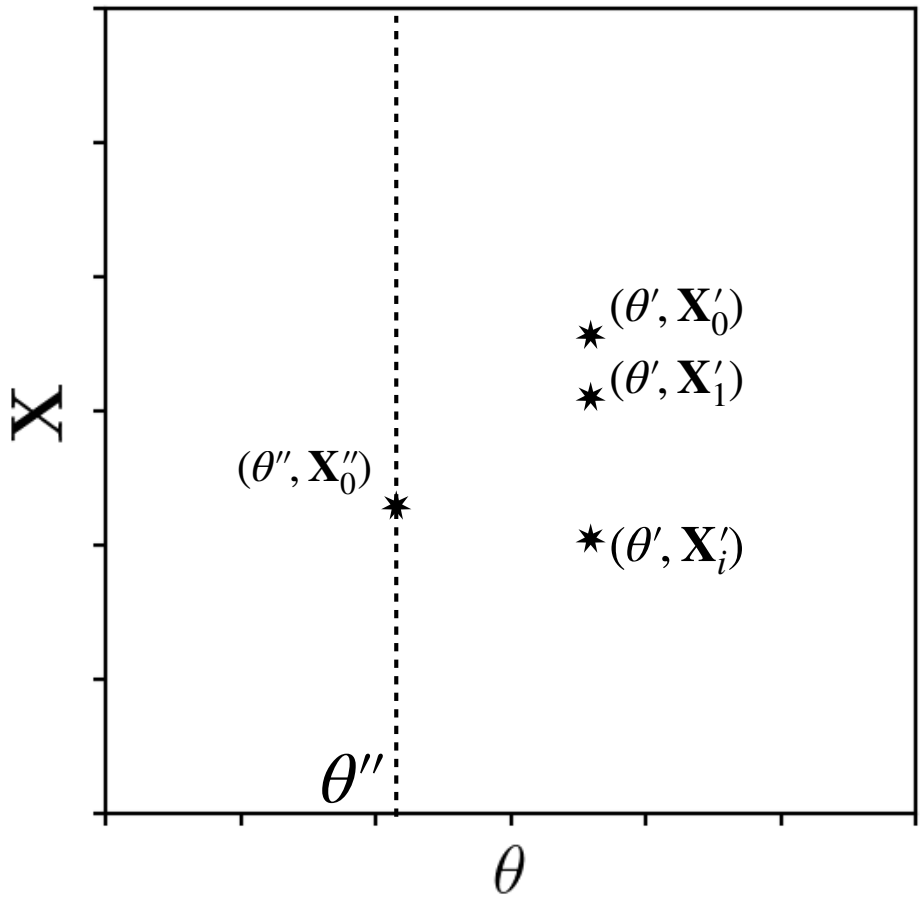
$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



F must **include noise model**

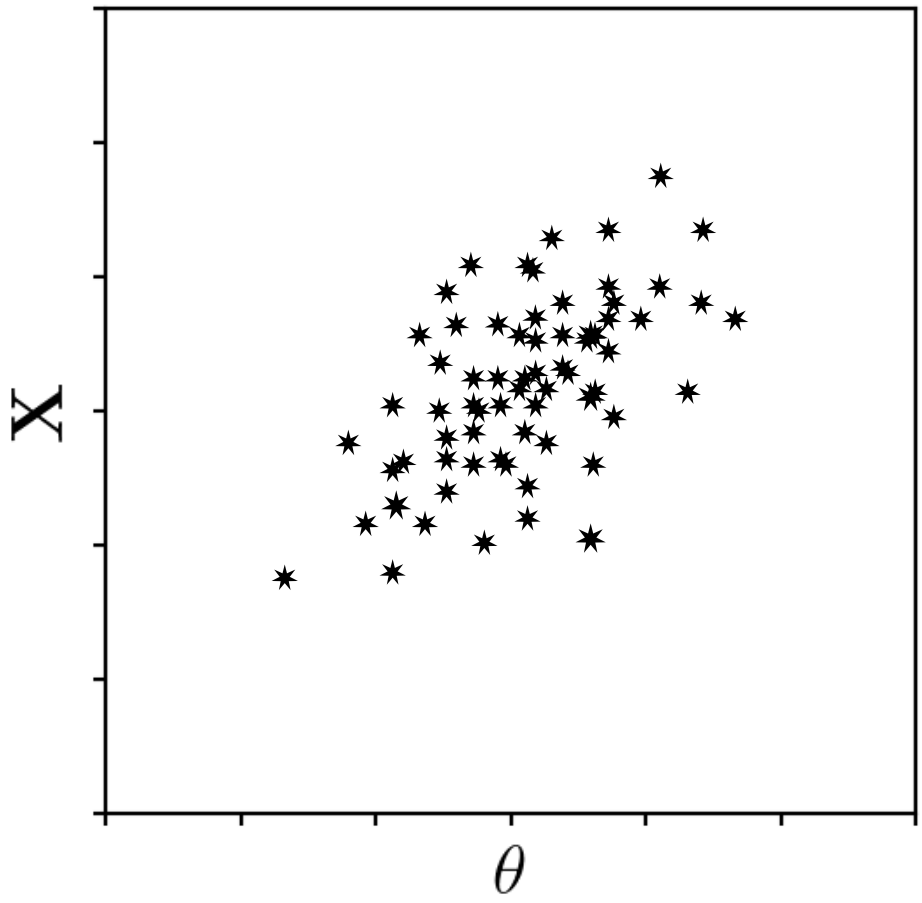
forward model F that can simulate observations:

$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



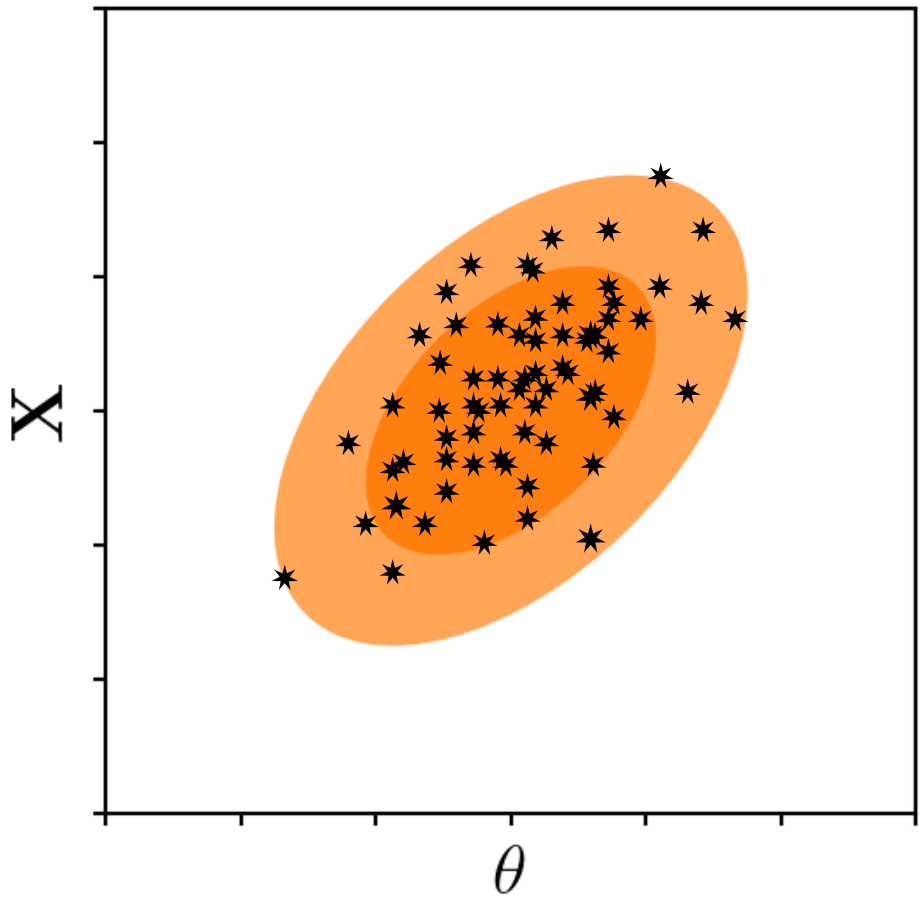
forward model F that can simulate observations:

$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



forward model F that can simulate observations:

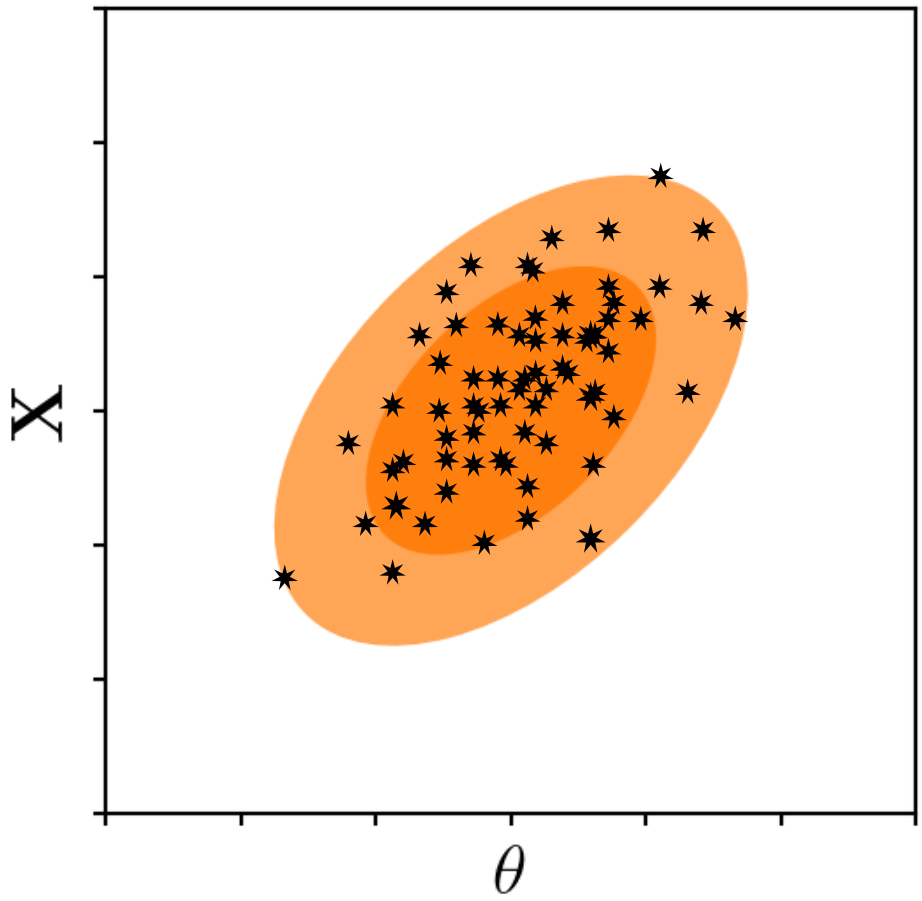
$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



F allows us to sample $(\theta', X') \sim p(\theta, X)$ ⟵ Unknown

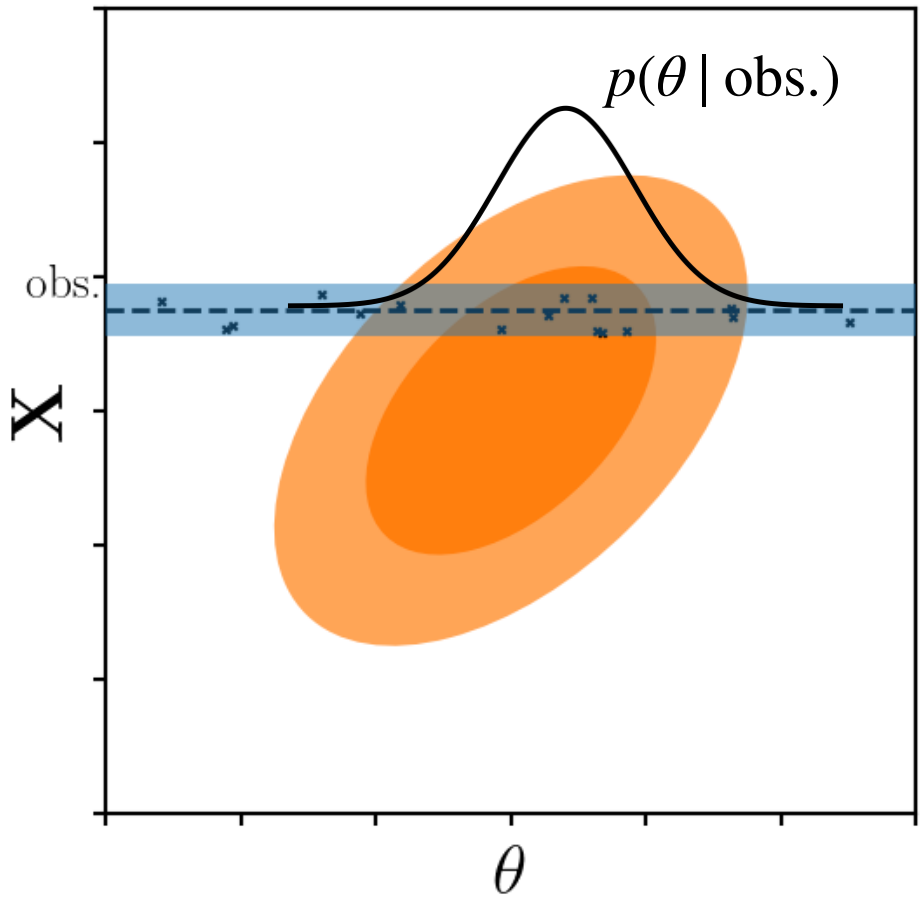
forward model F that can simulate observations:

$$\mathbf{X}'_0, \mathbf{X}'_1 \ldots \sim F(\theta')$$



F allows us to sample $(\theta', X') \sim p(\theta, X)$ ⟵ Unknown

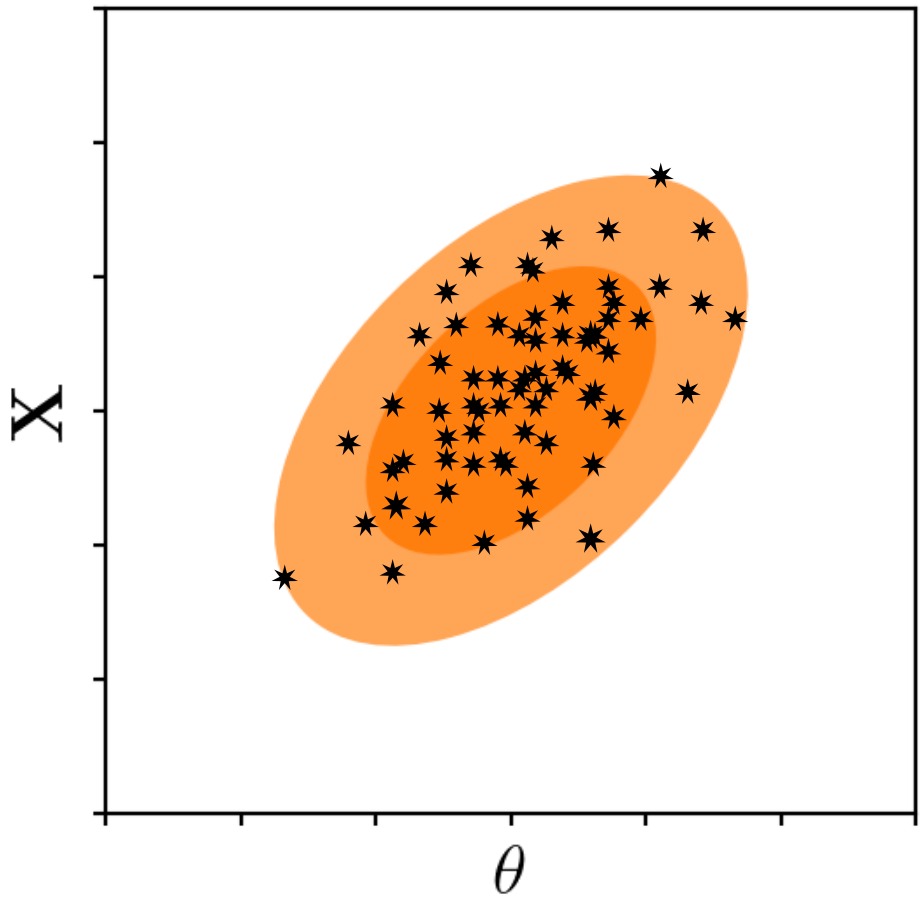
with F one way to infer the posterior is through *brute force* — **approximate bayesian computation** (ABC)



only keep the simulations “close” to the observation

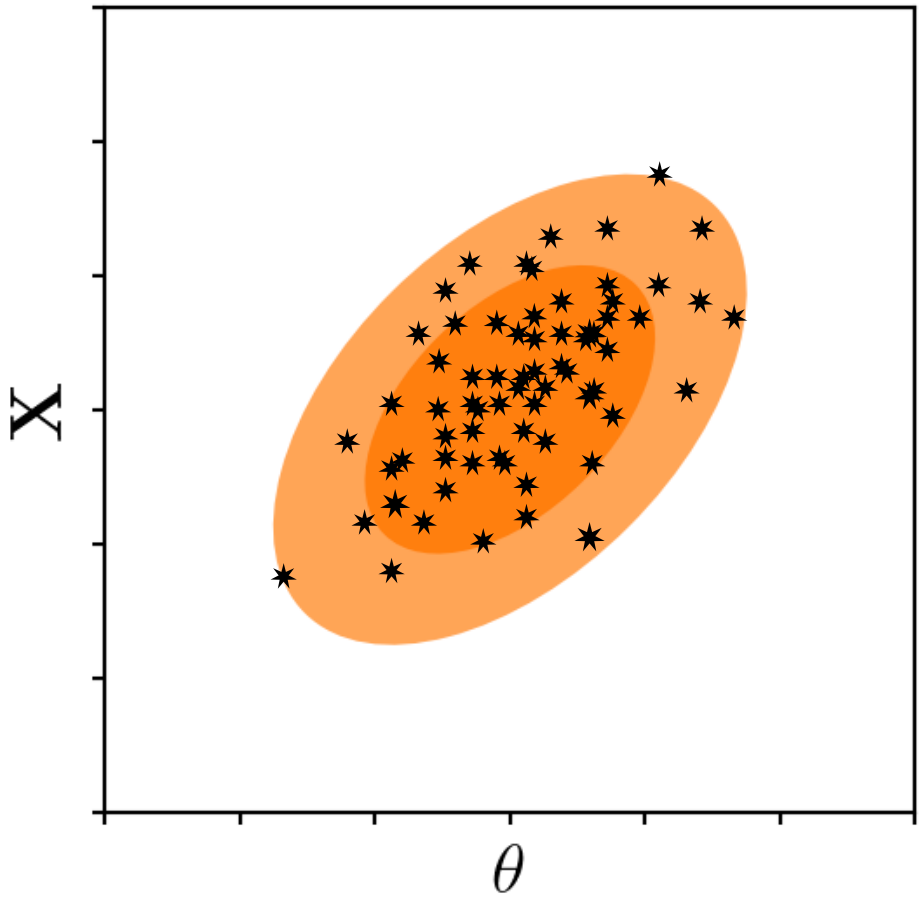
(This is called simulation-based inference, likelihood-free inference or implicit inference)

simulation-based inference is a **density estimation problem!**



F allows us to sample $(\theta', X') \sim p(\theta, X)$ ← Unknown

Density estimation



$$p(x, \theta)$$

Given this dataset, the goal of density estimation is to fit a model to the data distribution such that we can synthesize new data points at will by sampling from the distribution.

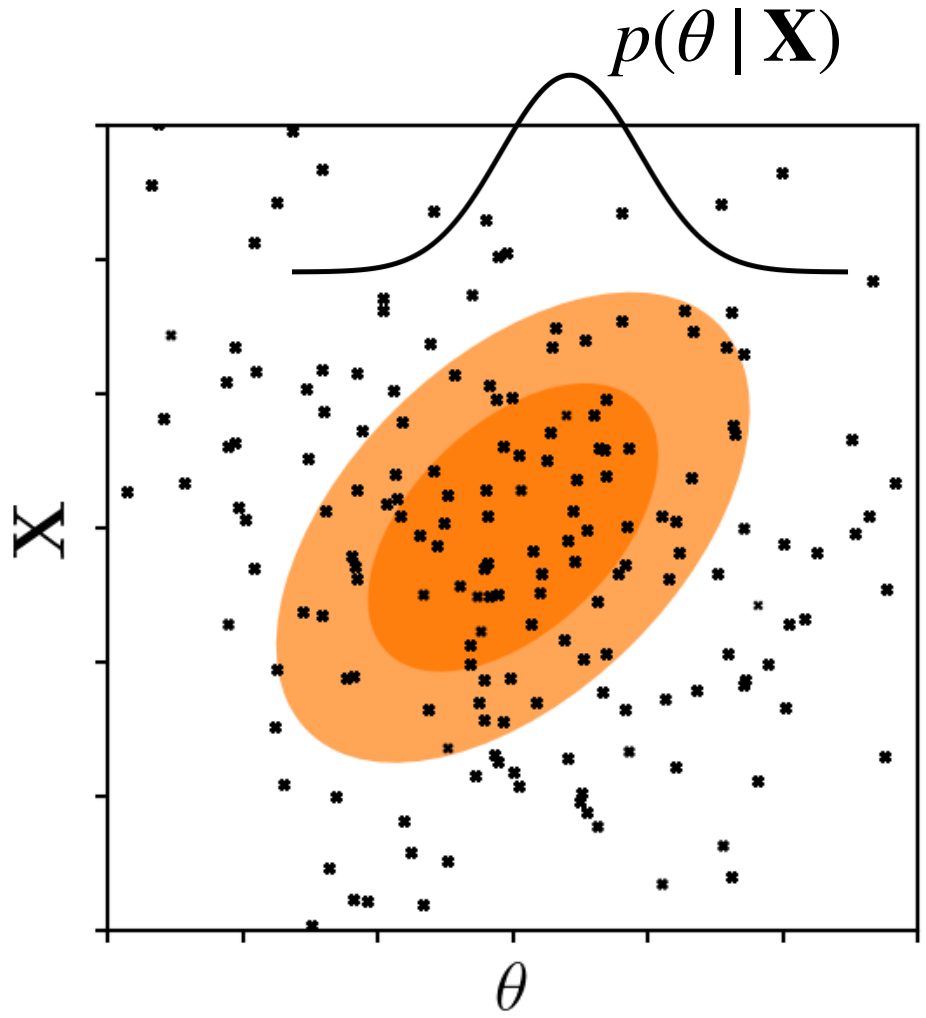
- Assuming we have a perfect simulator, we have (data $\mathbf{x}$, parameter θ) pairs. How do we do inference?

$$\underbrace{P(\boldsymbol{\theta}|\mathbf{x})}_{\text{“Posterior”}} \propto \underbrace{P(\mathbf{x}|\boldsymbol{\theta})}_{\text{“Likelihood”}} \underbrace{P(\boldsymbol{\theta})}_{\text{“Prior”}}$$

Neural **P**osterior **E**stimation

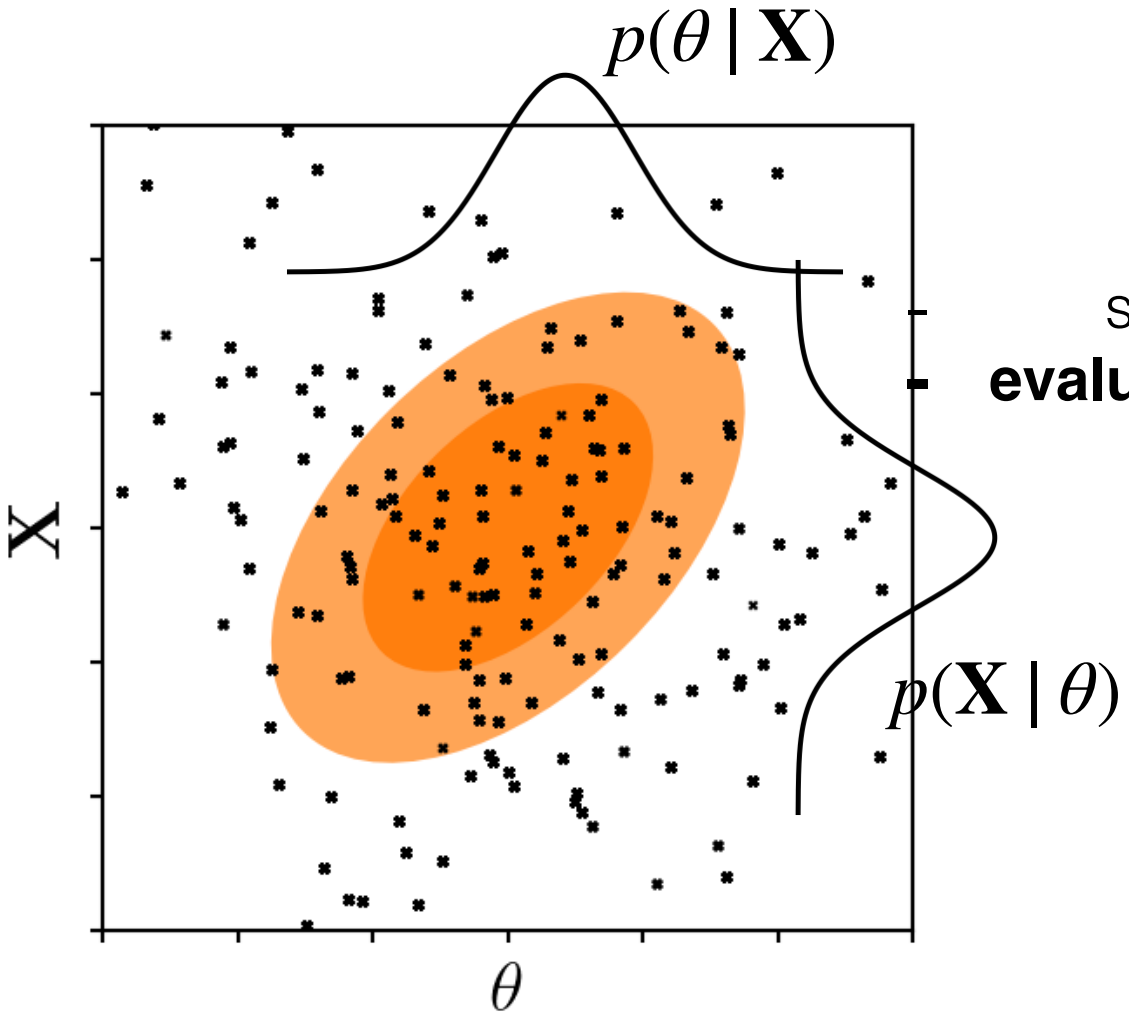
Neural **L**ikelihood **E**stimation

simulation-based inference is a **density estimation problem!**



- **sample from the posterior (amortized)**

simulation-based inference is a **density estimation problem!**



sample from the posterior

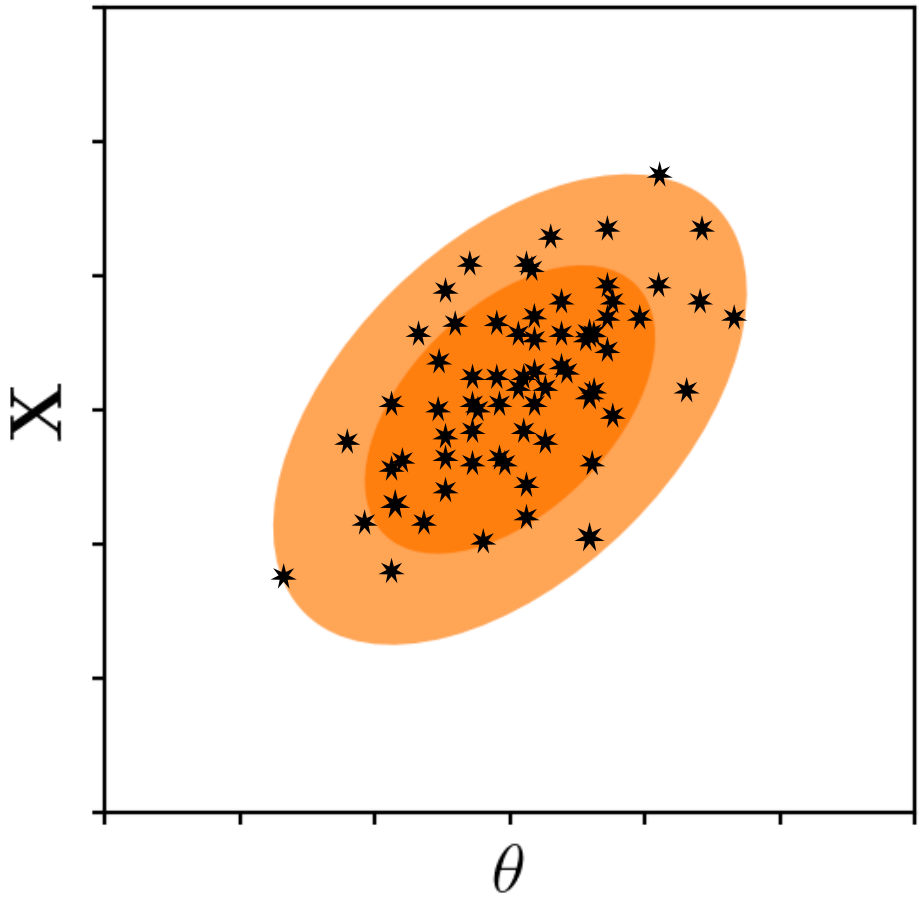
evaluate likelihood (e.g. NF) and then MCMC

- Assuming we have a perfect simulator, we have ($\mathbf{x}$ data, θ parameter) pairs. How do we do inference?

$$\underbrace{P(\boldsymbol{\theta}|\mathbf{x})}_{\text{“Posterior”}} \propto \underbrace{P(\mathbf{x}|\boldsymbol{\theta})}_{\text{“Likelihood”}} \underbrace{P(\boldsymbol{\theta})}_{\text{“Prior”}}$$

Neural **P**osterior **E**stimation	**N**eural **L**ikelihood **E**stimation
Amortized	Requires sampling (e.g. MCMC)
(Need to sample many posteriors for many X’s)	**(Need to build model into hierarchical sampling scheme)**

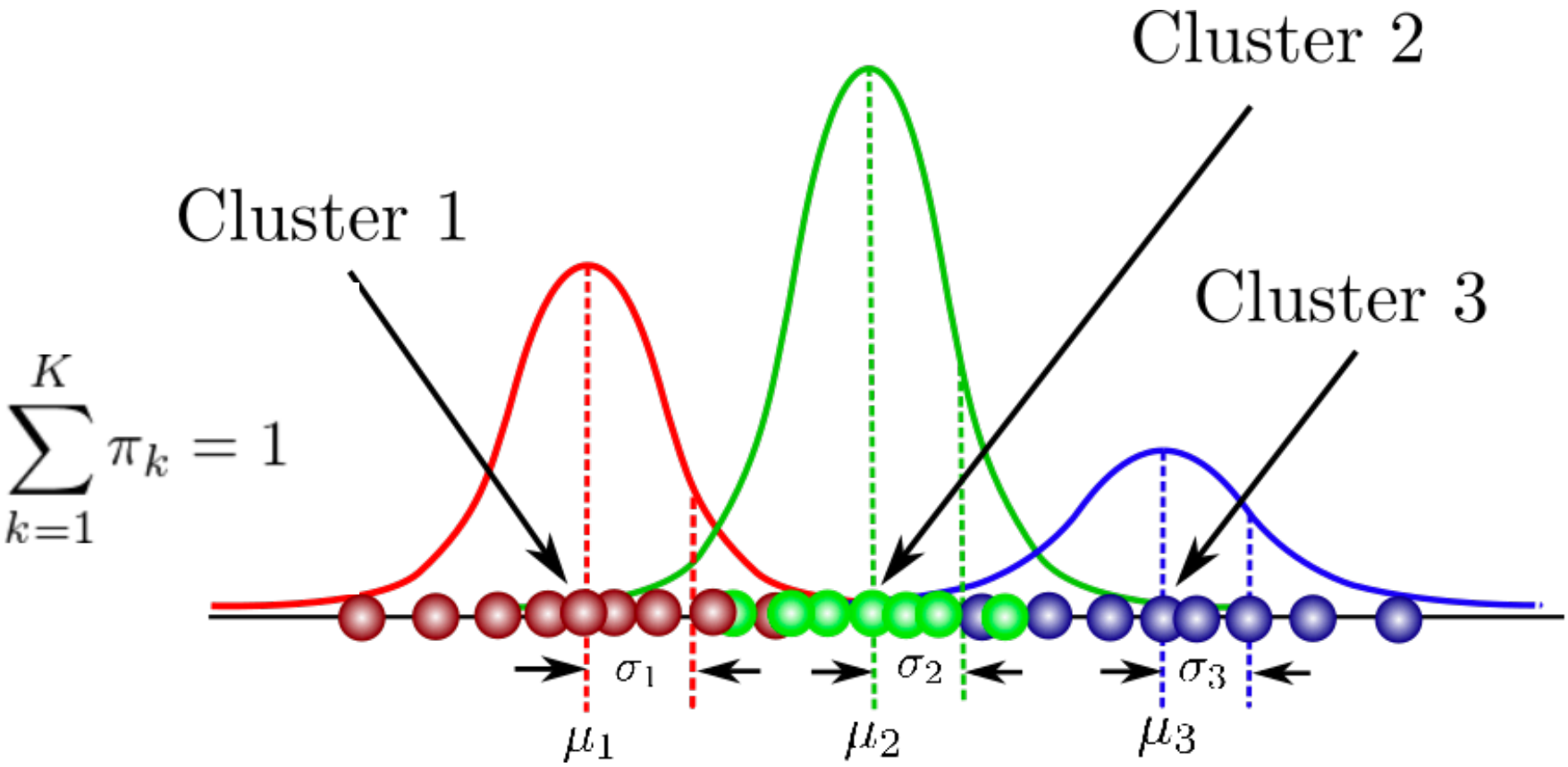
Density estimation



$$p(x, \theta)$$

Given this dataset, the goal of density estimation is to fit a model to the data distribution such that we can synthesize new data points at will by sampling from the distribution.

Standard kernel density can be used for that (GMMs)



means, sigmas and scale factors of each gaussian are free parameters

https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

GMMs

- + Both sampling and evaluating are straightforward with GMMs
- — Scales poorly with increasing dimensionality
- — Depends on initial choices
- — Scales poorly for heavily multimodal distributions

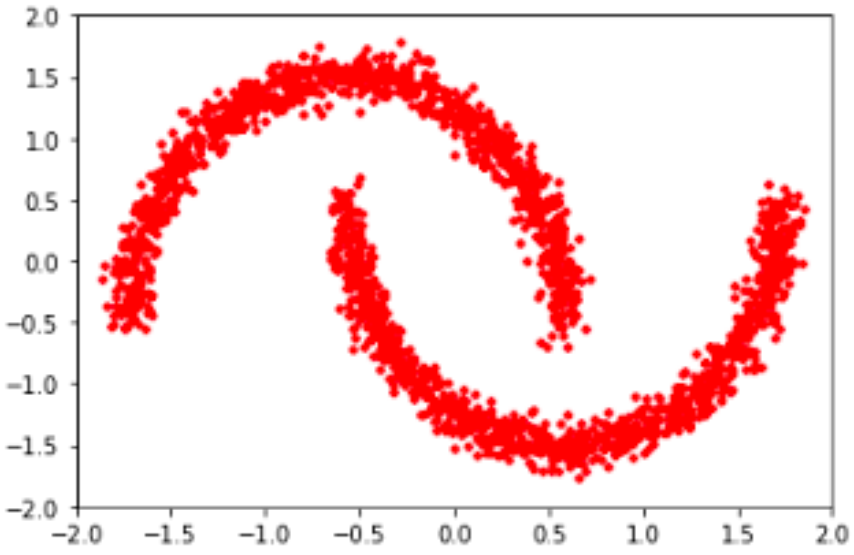
The deep learning revolution has enabled fast density estimation at very high dimension

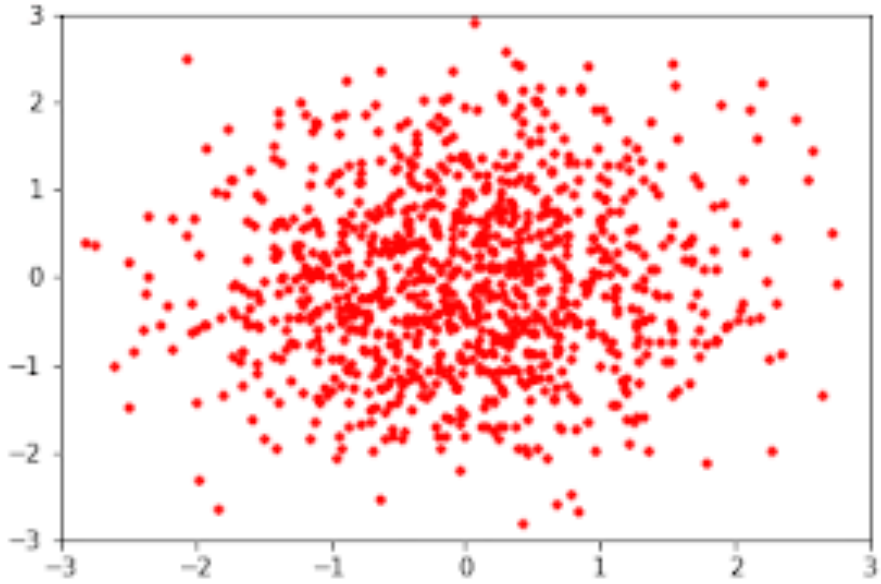
Pixel space!

Pixel space

$$p(\theta|x)$$

Not Gaussian

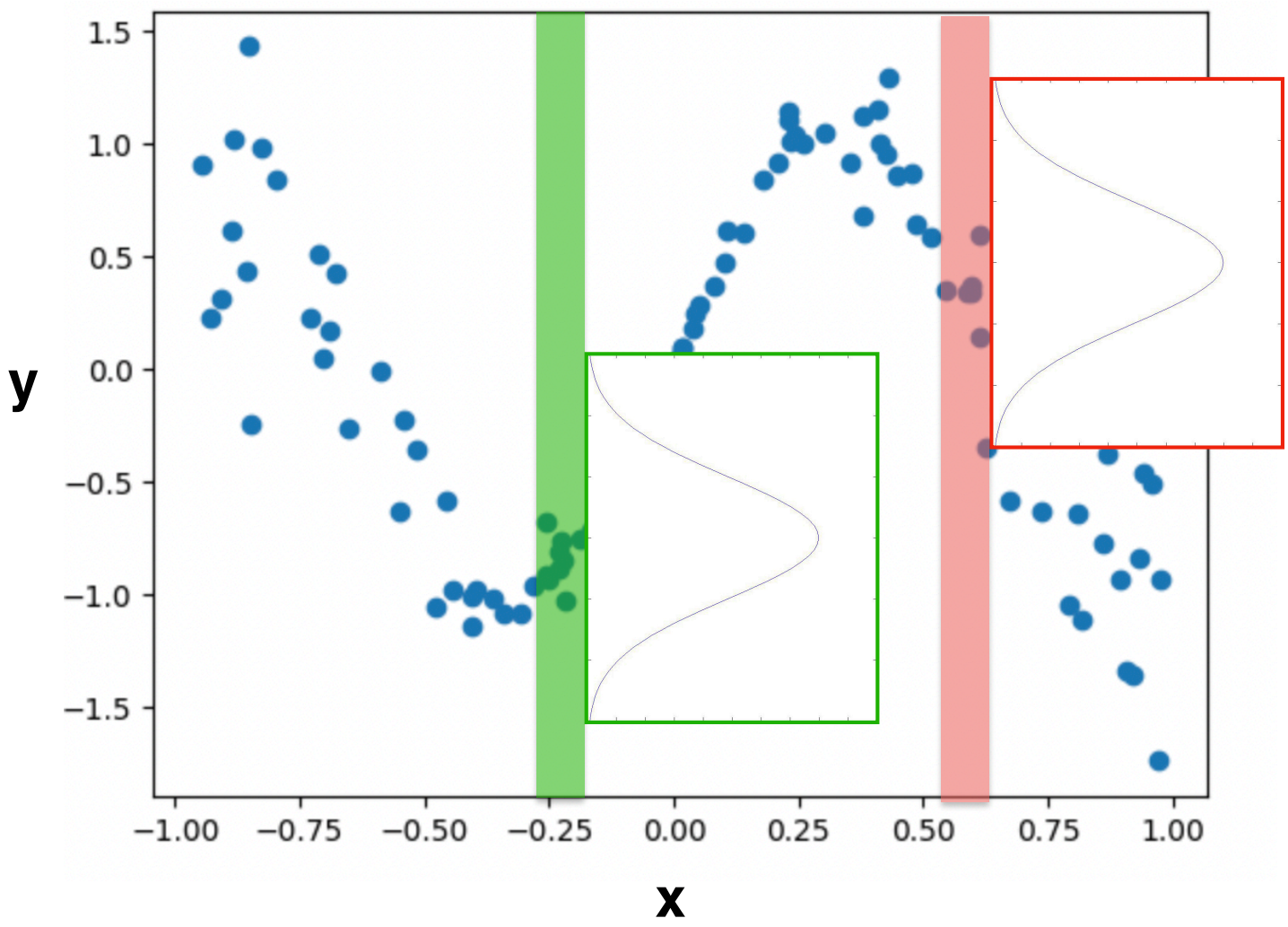




Neural Networks as statistical models

$$f_W(\vec{x}) = \vec{y} \quad \Leftrightarrow \quad p(y|x;W)$$

$$p(y|x) = \mathcal{N}(\mu, 1)$$



Assume that the posterior distribution can be approximated by a Gaussian function with fixed standard deviation

we have a set of independent realizations:

$$(x_i, y_i); i = 1, ..., N$$

and want to find the underlying probability distribution of y given x - p(y|x):

If we assume that p(y|x) is a Gaussian distribution:

$$p_w(y|x) = L(w, \sigma^2) = \prod_{i=1}^{N} (2\pi\sigma^2)^{-N/2} exp(-\frac{1}{\sigma^2}(y_i - f_w(x_i))^2)$$

Network output

weights (free parameters)

So the goal is to find the values of w (weights) that estimate the mean of a Gaussian distribution for y given a set of N independent observations

$$p_w(y|x) = L(w, \sigma^2) = \prod_{i=1}^{N} (2\pi\sigma^2)^{-N/2} exp(-\frac{1}{\sigma^2}(y_i - f_w(x_i))^2)$$

weights (model parameter)

neural network

And then take the log:

$$ln\ L = cst - \sum_{i=1}^{N} \frac{(x_i - f_w(x_i))^2}{2\sigma^2}$$

And now assume sigma = 1:

$$ln\ L = -\sum_{i=1}^{N} (x_i - f_w(x_i))^2 = MSE$$

Regressions

WHEN USING THE MEAN SQUARED ERROR, WE ARE ASSUMING THAT THE POSTERIOR P(y|x) IS A NORMAL DISTRIBUTION WITH STANDARD DEVIATION EQUAL TO 1.

OUR PDF IS THEREFORE FULLY CHARACTERIZED BY ONE UNIQUE PARAMETER: THE MEAN OF THE NORMAL PDF.

THE MEAN IS ESTIMATED BY MAXIMIZING THE LIKELIHOOD OF THE OBSERVATIONS.

What about classification?

$$H = -\frac{1}{N}\sum_{i=1}^{N} y_i.log(f_w(x_i)) + (1 - y_i).log(1 - f_w(x_i))$$

GROUND TRUTH

GROUND TRUTH

NETWORK OUTPUT

NETWORK OUTPUT

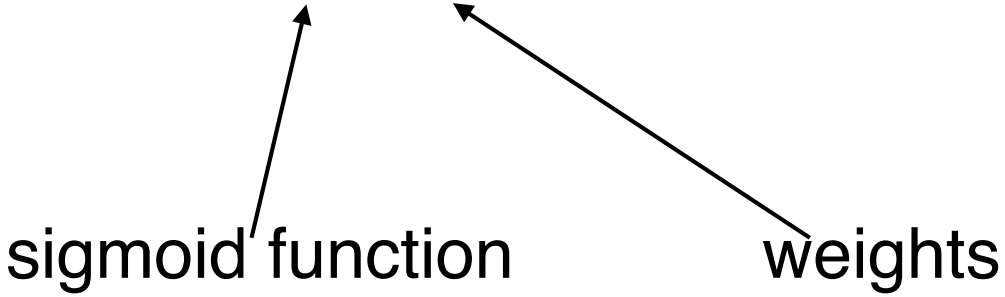
we have a set of independent realizations:

$$(x_i, y_i); i = 1, ..., N$$

and want to find the underlying probability distribution of y given x:

Since y can only take 0 / 1 values, we assume it can be parametrized with a **Bernouilli distribution**:

$$P(y_i = 0|x_i) = 1 - sigm(f_w(x_i)) \qquad P(y_i = 1|x_i) = sigm(f_w(x_i))$$



So the goal is to find the values of w (weights) that generate a Bernouilli distribution for y given a set of N independent observations

This can be achieved via Maximum Likelihood estimation:

The likelihood of a given observation under the Bernouilli assumption can be written as:

$$L(w; x_i, y_i) = [sigm(f_w(x_i))]^{y_i} [1 - sigm(f_w(x_i))]^{1-y_i}$$

which is equal to $[sigm(f_w(x_i))]$ if y=1 and $[1 - sigm(f_w(x_i))]$ if y=0

And the likelihood of the entire sample is the product of the likelihoods:

$$L(w; x_i, y_i) = \prod_{i=1}^{N} [sigm(f_w(x_i))]^{y_i} [1 - sigm(f_w(x_i))]^{1-y_i}$$

So, the log-likelihood:

$$l(w; x_i, y_i) = \sum_{i=1}^{N} y_i \times \log[sigm(f_w(x_i))] + (1 - y_i) \times \log[1 - sigm(f_w(x_i))]$$

THEREFORE EQUIVALENT TO THE CROSS-ENTROPY LOSS:

$$l(w; x_i, y_i) = \sum_{i=1}^{N} y_i \times \log[sigm(f_w(x_i))] + (1 - y_i) \times \log[1 - sigm(f_w(x_i))]$$

$$H = -\frac{1}{N} \sum_{i=1}^{N} y_i.log(f_w(x_i)) + (1 - y_i).log(1 - f_w(x_i))$$

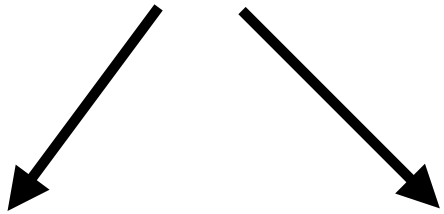
WHEN YOU DO A BINARY CLASSIFICATION WITH NEURAL NETWORKS YOU ARE SIMPLY FINDING THE WEIGHTS OF YOUR MODEL THAT MAXIMIZE THE LIKELIHOOD OF A BERNOUILLI DISTRIBUTION

ASSUMES THAT THE POSTERIOR p(ylx) CAN BE MODELLED BY A BERNOUILLI DISTRIBUTION

A standard NN performs Maximum Likelihood Estimation of the weights W

$$D = (x^{(i)}, y^{(i)})$$

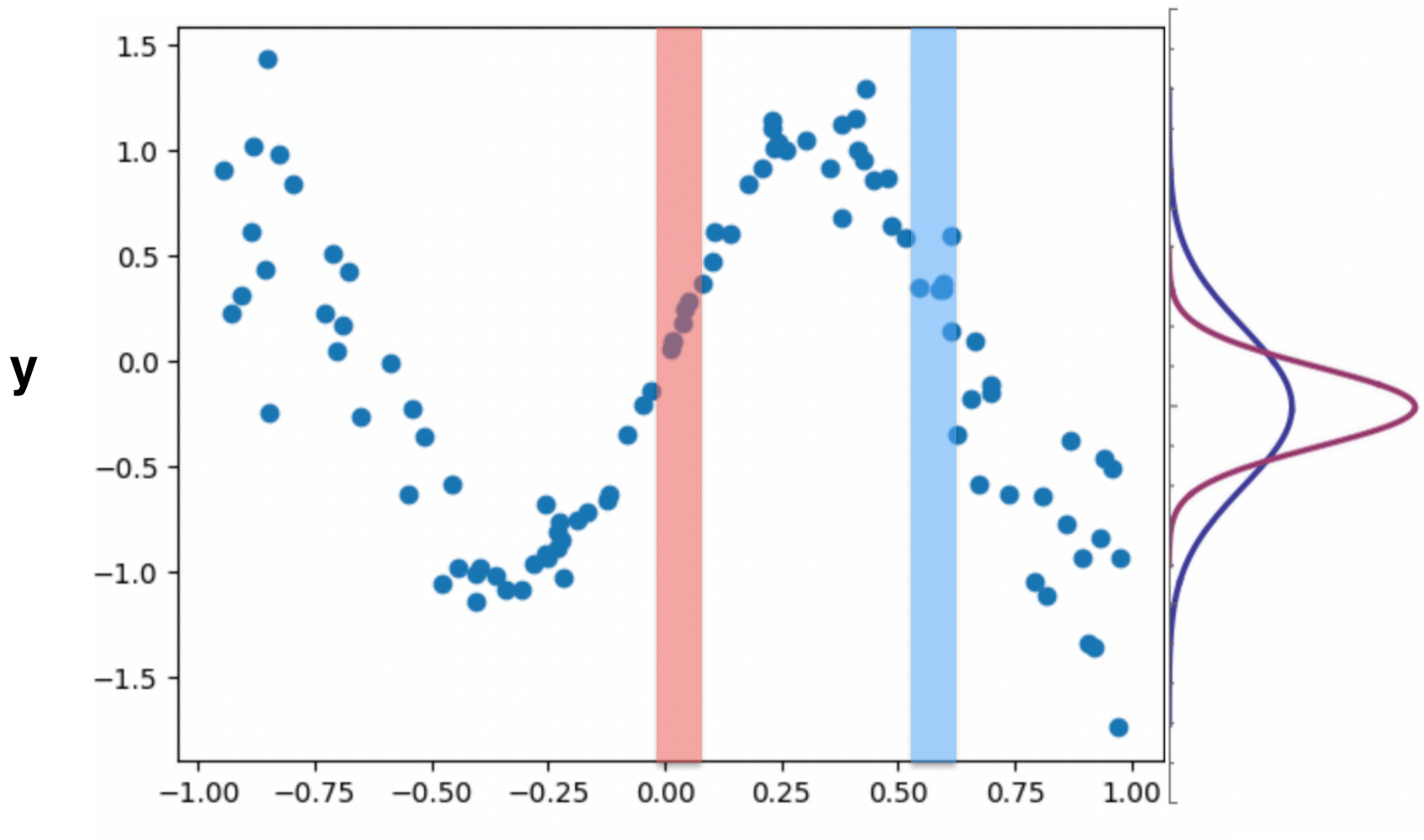
$$p(D|w) = \prod_i p(y^{(i)}|x^{(i)}, W) \sim p_w(y|x)$$



Crossentropy
(~ Bernouilli)

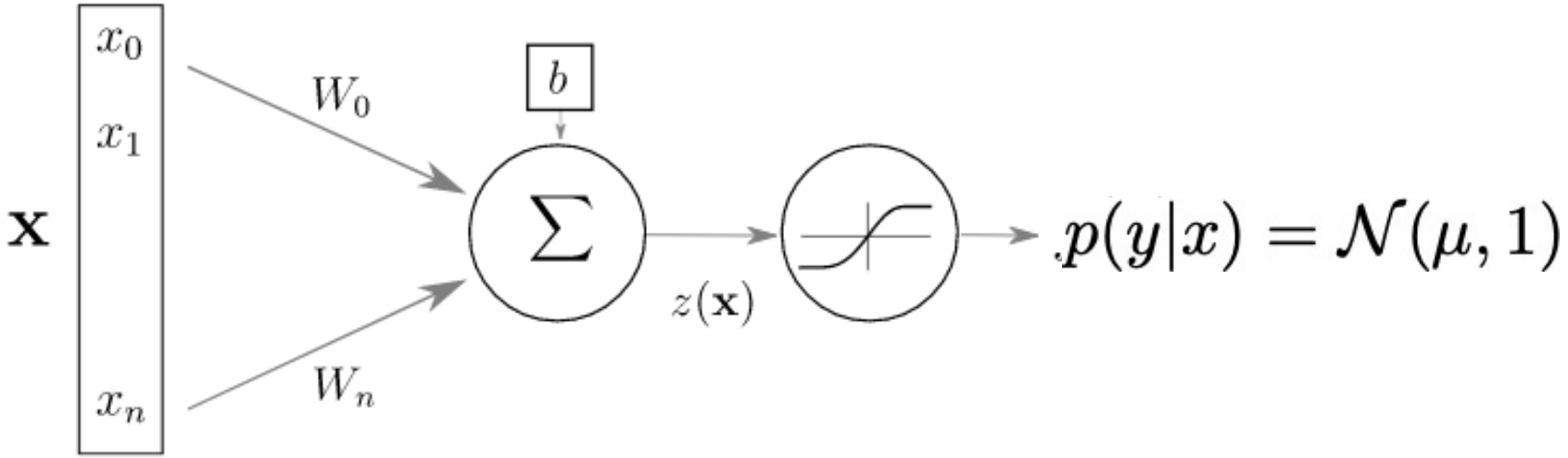
MSE
(~Normal(mu,1))

$$p(y|x) = \mathcal{N}(\mu, 1) \; ?$$

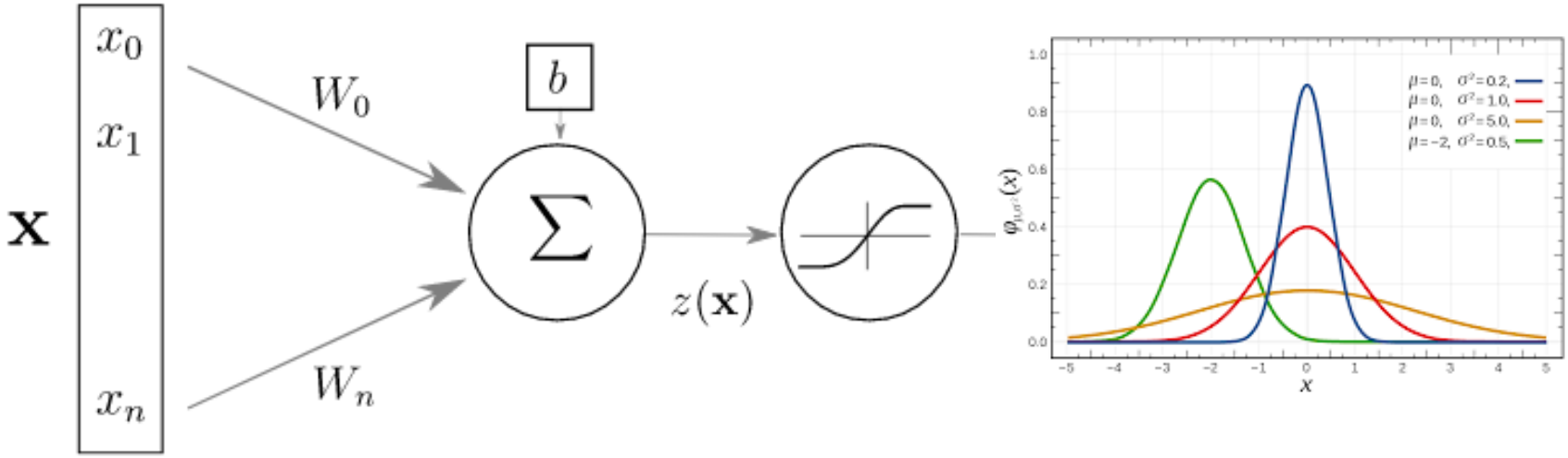


is one parameter enough to characterize the posterior?

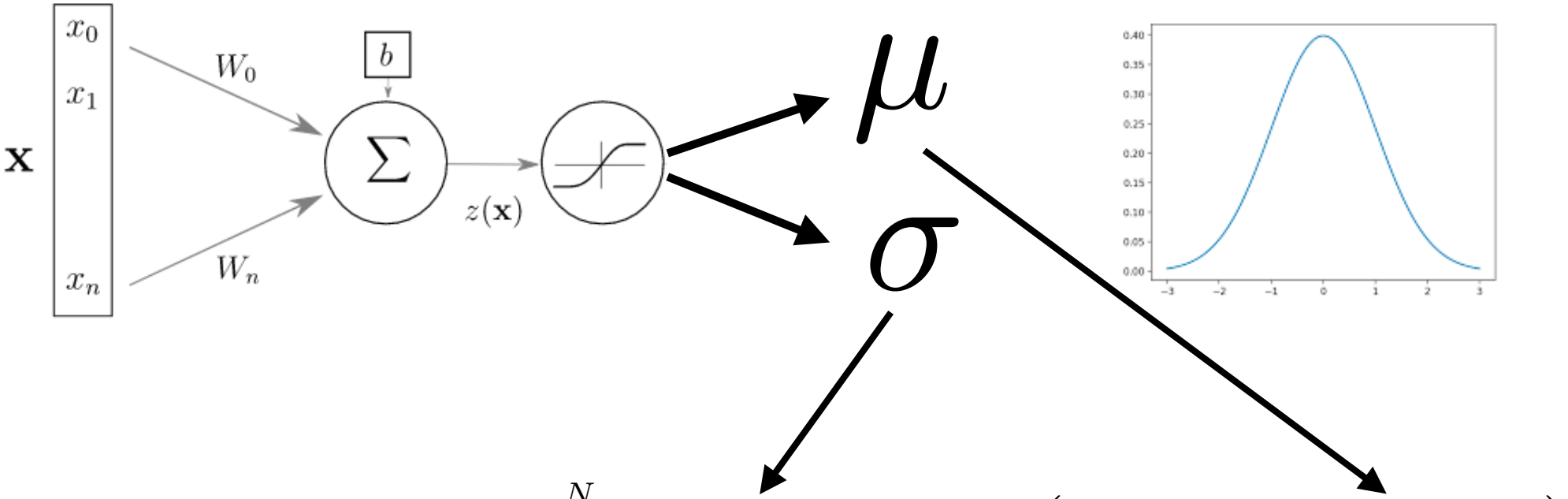
Mixture Density Network



$$p(y|x) = \mathcal{N}(\mu, \sigma)$$



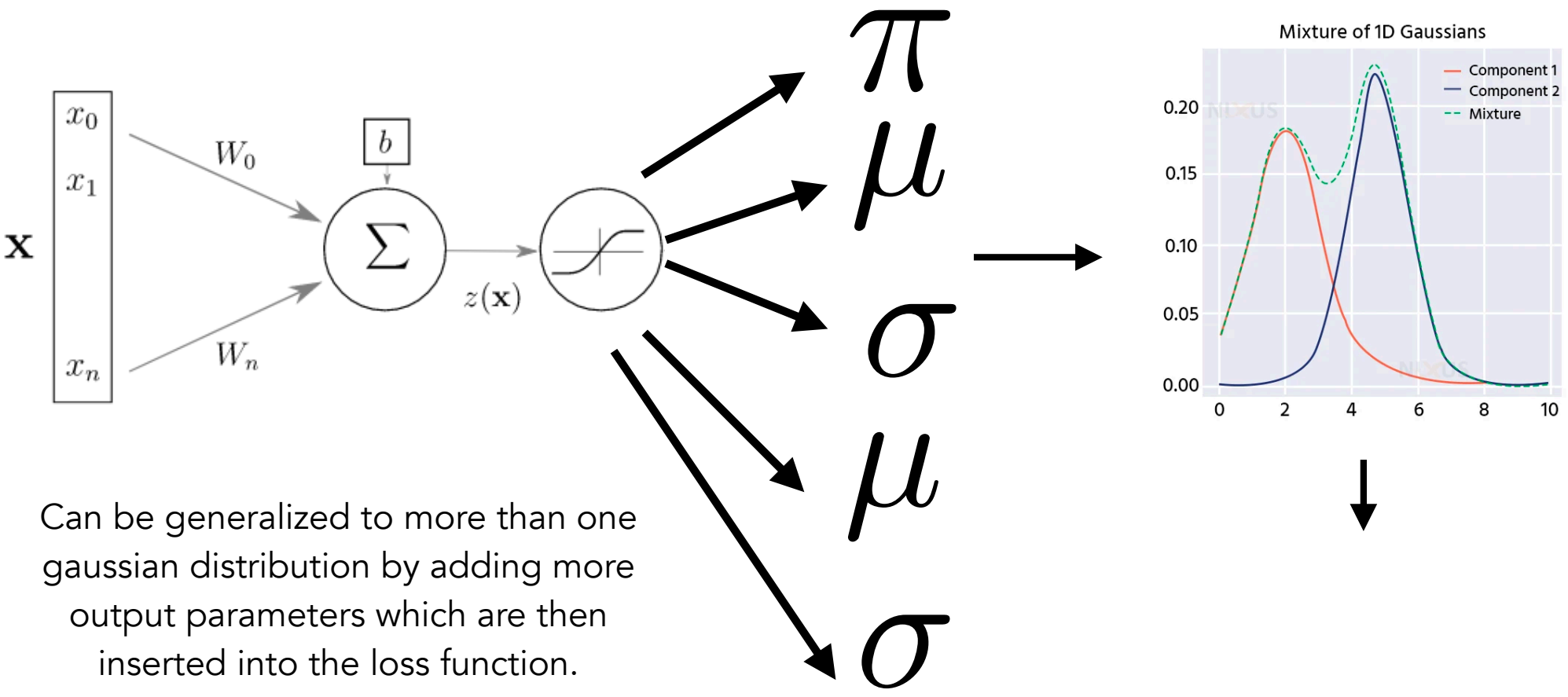
Mixture Density Network (MDN)



$$p_w(y \mid x) = L(w, \sigma^2) = \prod_{i=1}^{N} (2\pi\sigma_w^0(x_i)^2)^{-N/2} \exp\left(-\frac{1}{\sigma_w^0(x_i)^2}(y_i - f_w^0(x_i))^2\right)$$

The neural network outputs the parameters of a distribution, on which one can back propagate (2 in the example above)

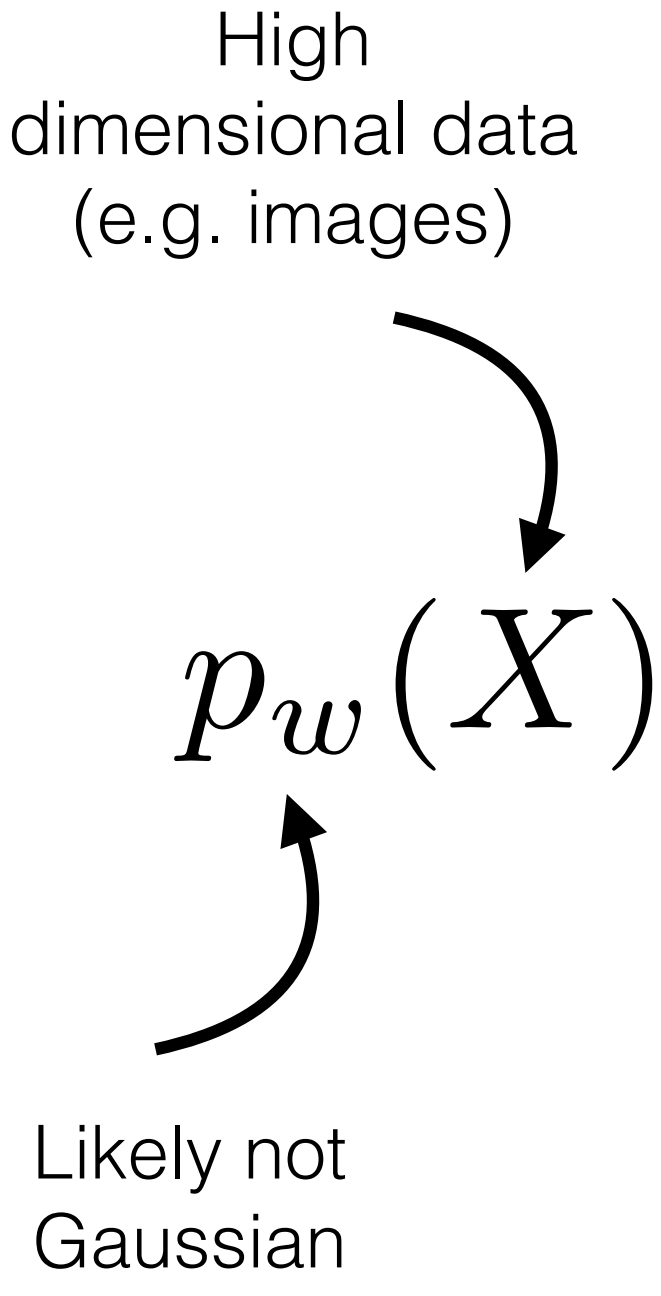
What if the posterior is not gaussian?

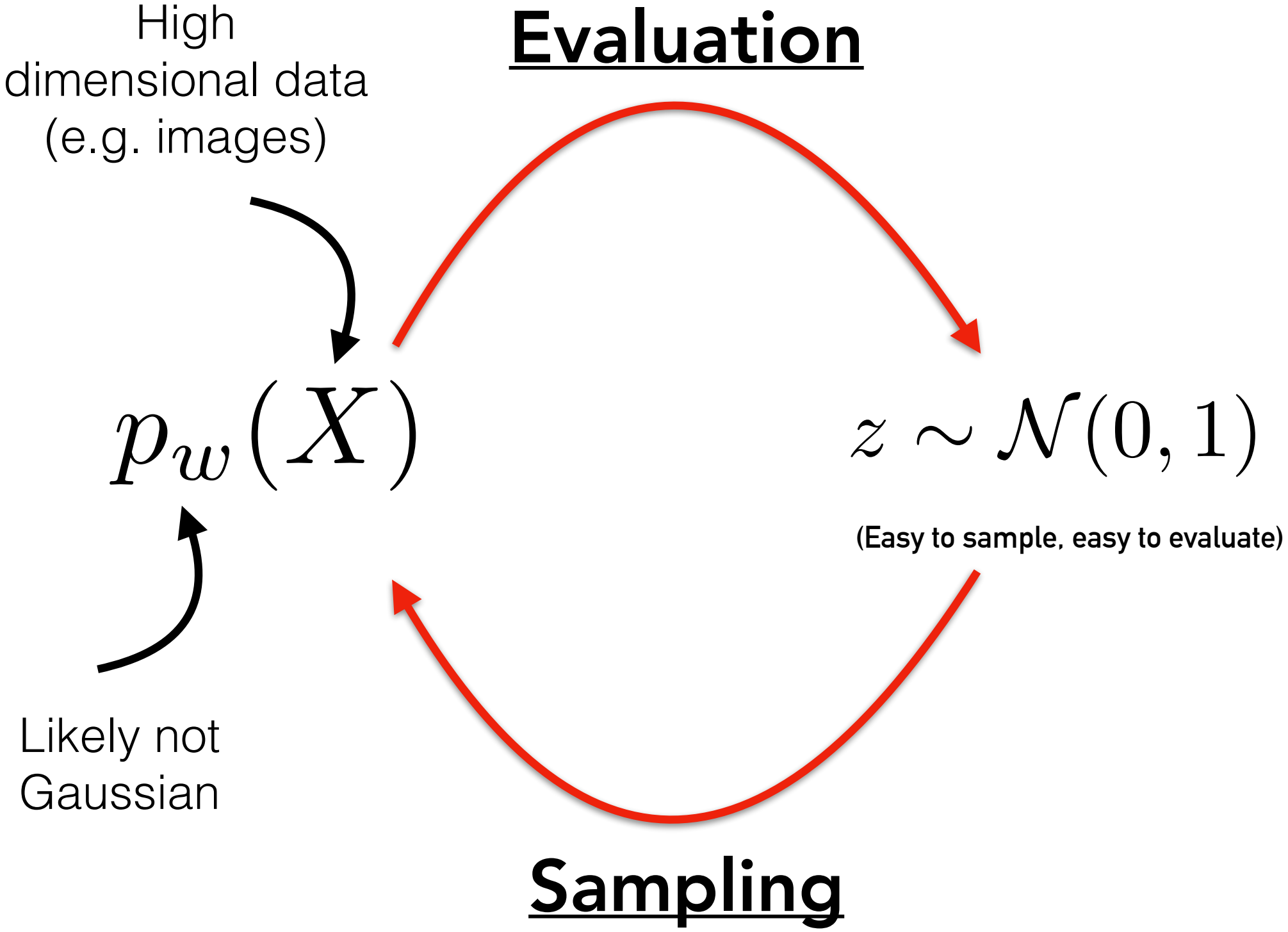


Can be generalized to more than one gaussian distribution by adding more output parameters which are then inserted into the loss function.

$$p_w(y \mid x) = L(w) = \prod_{i=1}^{N} \sum_{k=1}^{2} \pi_w^{(k)}(x_i) \, \frac{1}{\sqrt{2\pi}\,\sigma_w^{(k)}(x_i)} \exp\left(-\frac{\left(y_i - f_w^{(k)}(x_i)\right)^2}{2\left[\sigma_w^{(k)}(x_i)\right]^2} \right)$$

The loss function remains the max of the (log) likelihood


High
dimensional data
(e.g. images)
$p_w(X)$
Likely not
Gaussian

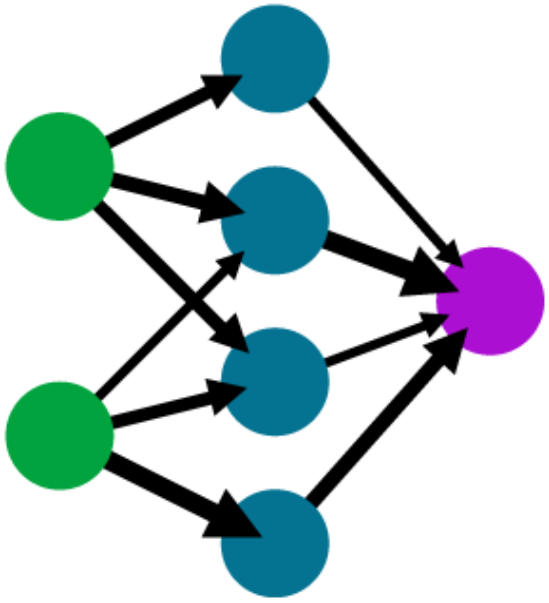

High dimensional data (e.g. images)
Evaluation
$p_w(X)$
$z \sim \mathcal{N}(0, 1)$
(Easy to sample, easy to evaluate)
Likely not Gaussian
Sampling

High dimensional data (e.g. images)

Evaluation

NNs == universal approximators

$p_w(X)$



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

(Easy to sample, easy to evaluate)

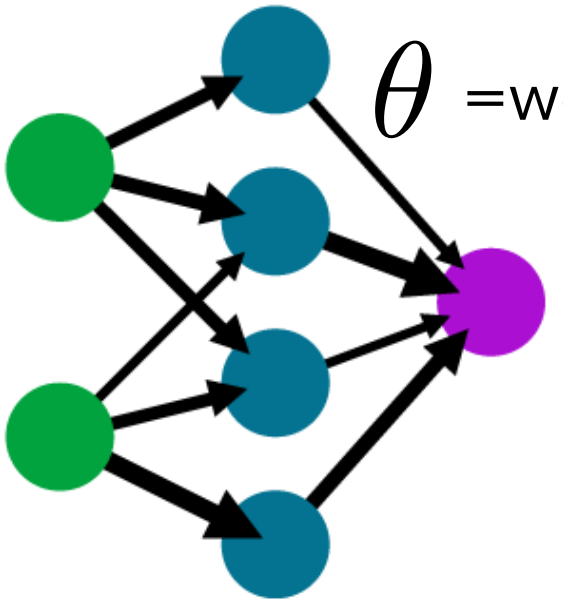
(Optimization problem)

Likely not Gaussian

Sampling

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

$$p(z)$$

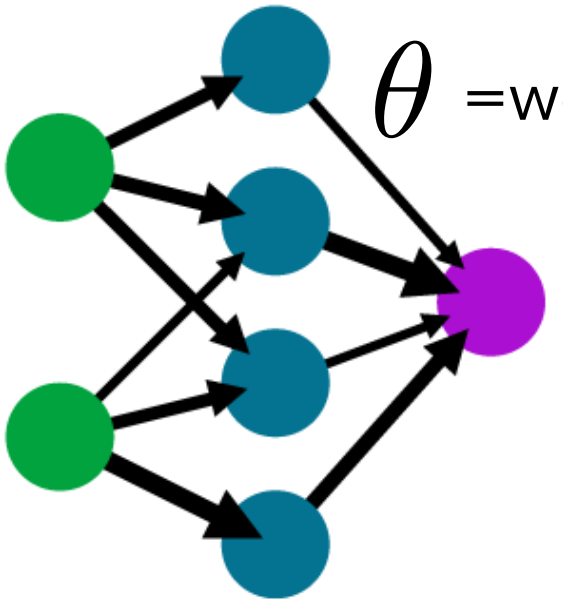


θ =weights

$$p_\theta(x)$$

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

$p(z)$



θ =weights

$p_\theta(x)$

Types of generative models			
Likelihood based models	directly learn the distribution's probability density function via (approximate) maximum likelihood	VAEs, Normalizing Flows	$\max_\theta \sum_{i=1}^{N} \log p_\theta(x_i)$
Implicit generative models	the probability distribution is implicitly represented by a model of its sampling process	GANs	adversarial
Score based models	model the gradient of the log probability density function	Score based diffusion	$\nabla_x \log p_\theta(x)$

Types of generative models		
Likelihood based models	directly learn the distribution's probability density function via (approximate) maximum likelihood	VAEs, Normalizing Flows
Implicit generative models	the probability distribution is implicitly represented by a model of its sampling process	GANs
Score based models	model the gradient of the log probability density function	Score based diffusion

Normalizing Flows

Based on change of variables:

$$p_X(x) = p_Z\big(f(x)\big) \left|\det J_f(x)\right|, \qquad J_f(x) = \frac{\partial f(x)}{\partial x^\top}$$

unknown pdf

tractable pdf
(typically Gaussian)

Invertible differentiable
function

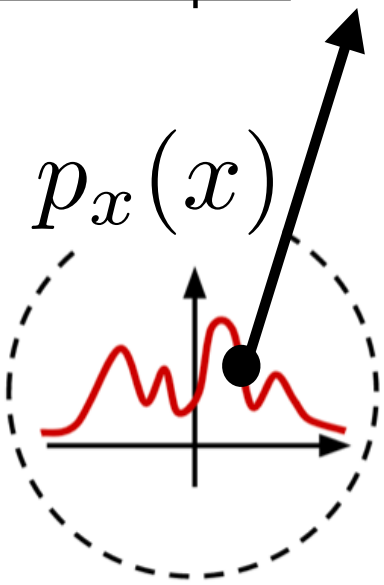
Determinant of the Jacobian
of f(x)

the neural network parametrizes the function f: $f_\theta(x)$

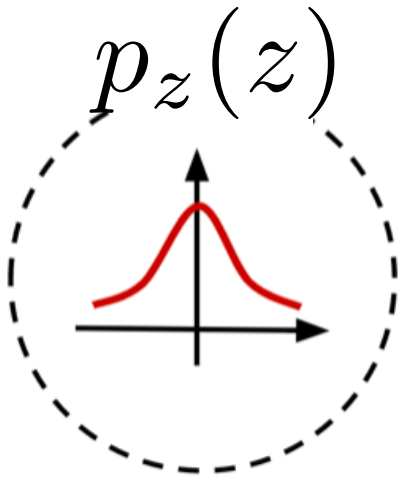
and optimize so that: $p_Z\big(f_\theta(x)\big) \;\; \left|\det J_f(x)\right|$ is maximum

training procedure

a sample x_i



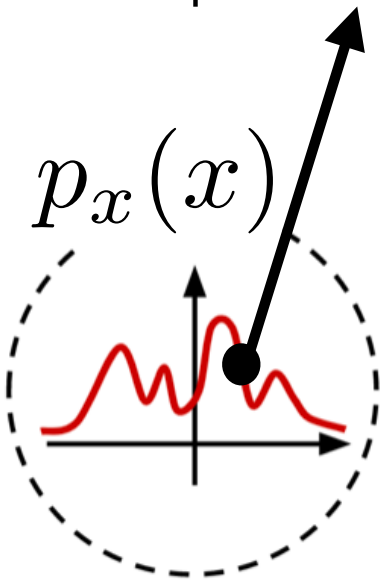
(unknown)



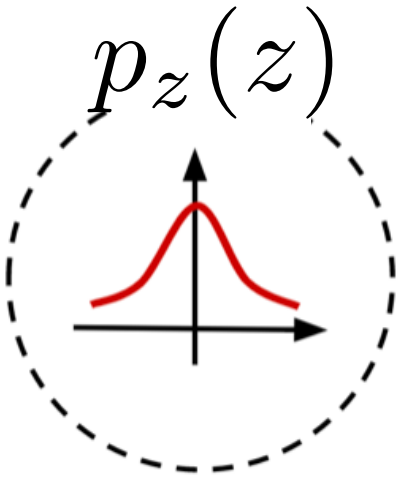
(known and easy)

training procedure

a sample x_i $\longrightarrow$ $f_\theta(x_i)$

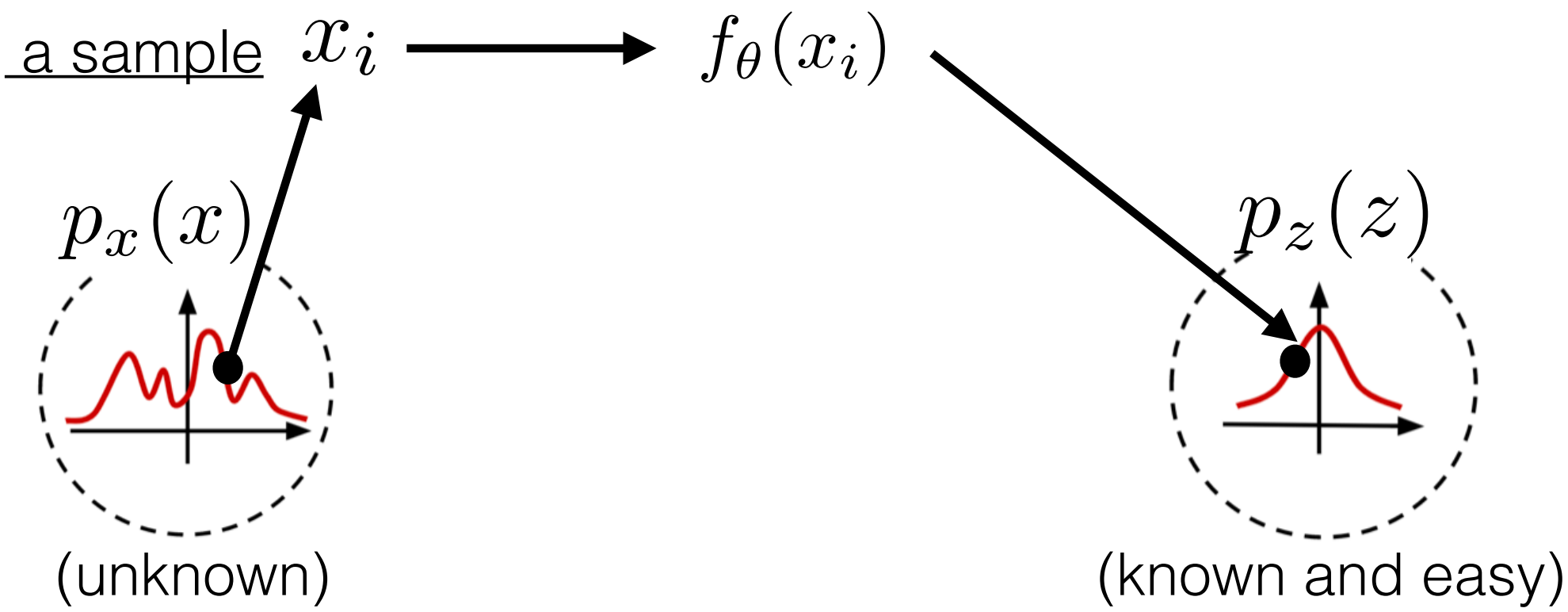


(unknown)

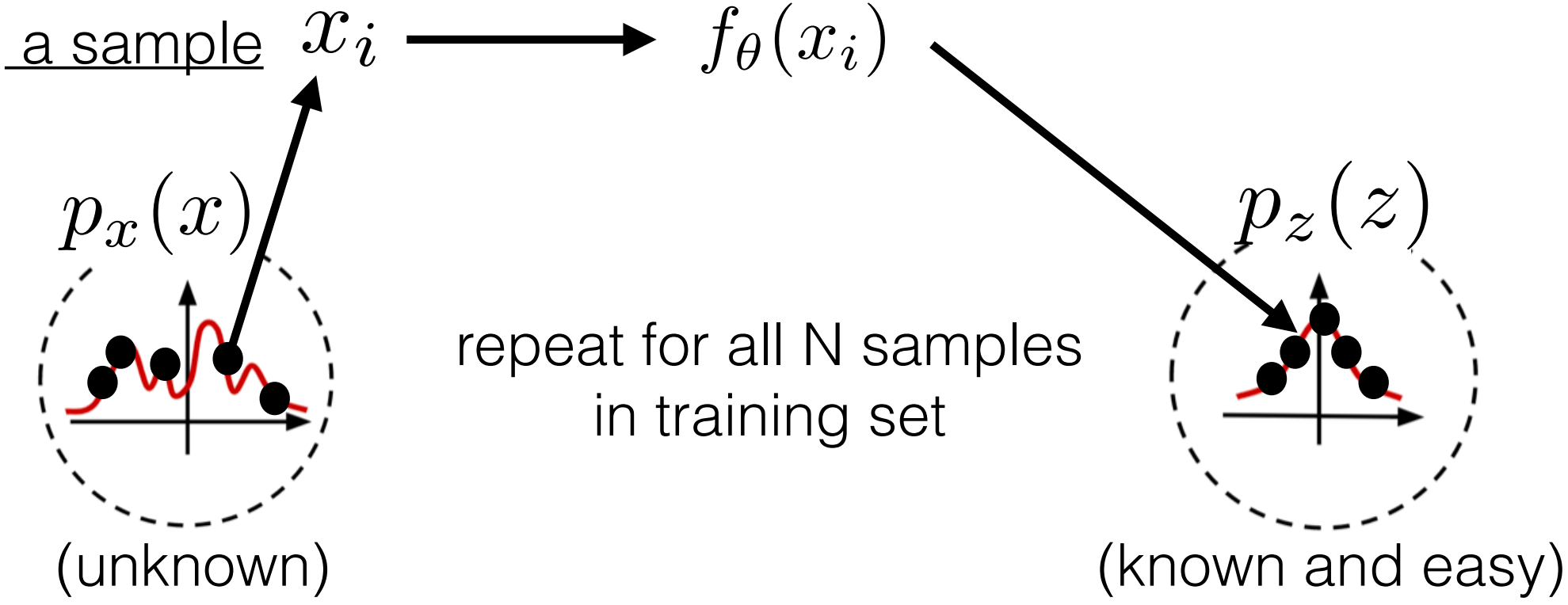


(known and easy)

training procedure



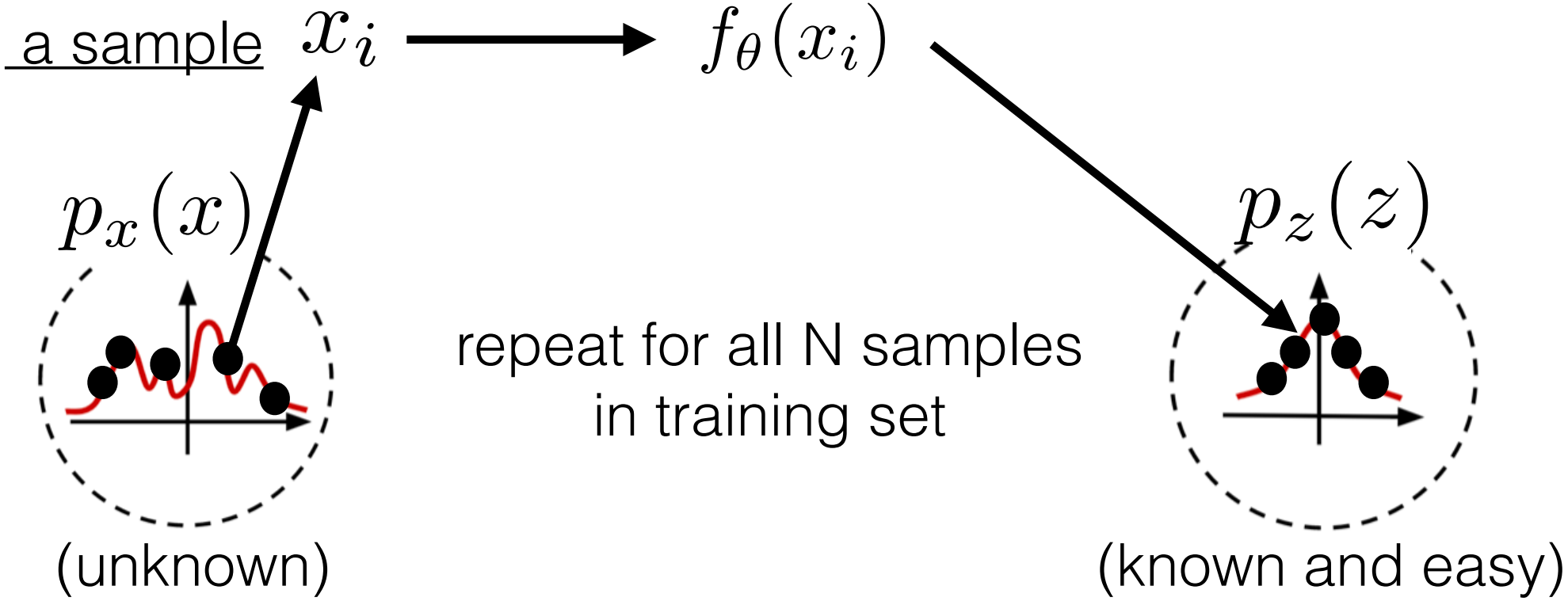
training procedure



And Maximize this

$$\sum_{i=1}^{N} \log p_z(x_i) + \log \det J_f(x_i)$$

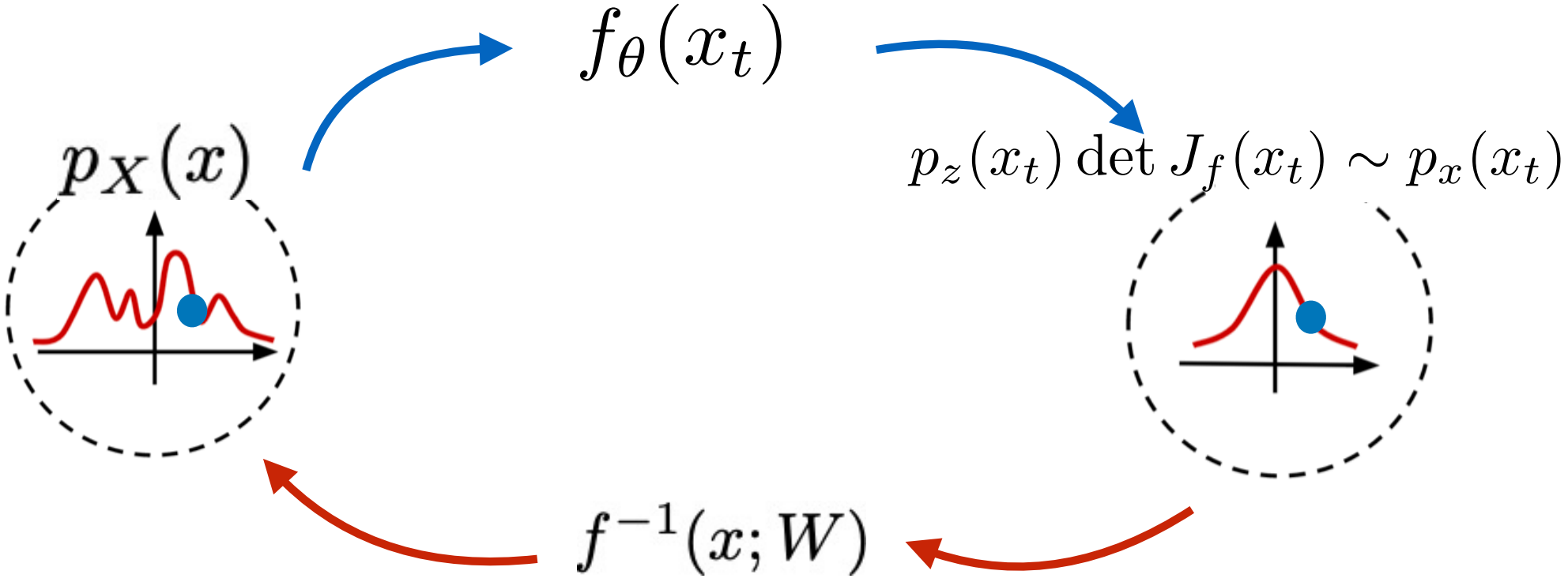
training procedure



achieved by maximizing this loss function

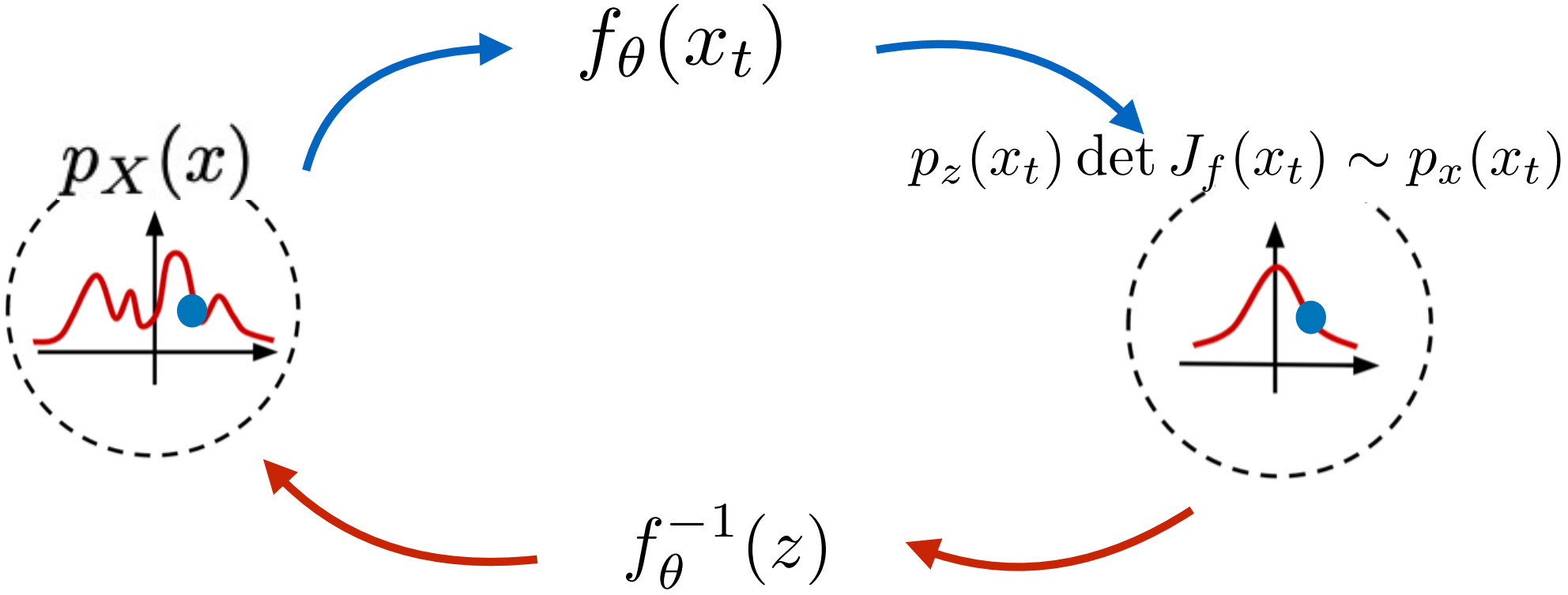
$$\sum_{i=1}^{N} \log p_z(x_i) + \log \det J_f(x_i)$$

Likelihood evaluation



Sampling

Likelihood evaluation



Sampling?

The price to pay is that we need to put some restrictions on the NN architecture

- 1. Easily invertible - to enable sampling
- 2. Jacobian tractable - to optimize the network

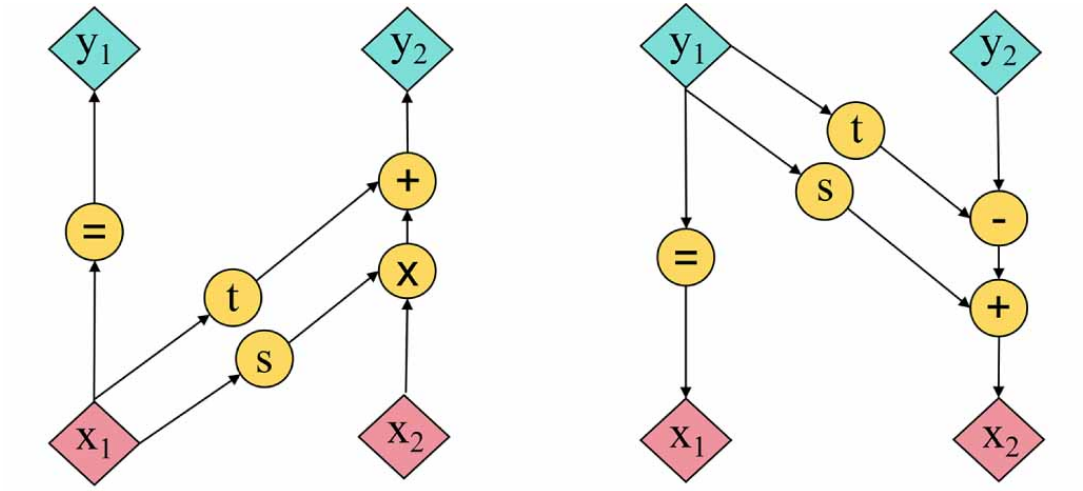
What functions satisfy these conditions?

An example are: RealNVPs (Real-valued Non-Volume Preserving)

$$\mathbf{y}_{1:d} = \mathbf{x}_{1:d}$$

$$\mathbf{y}_{d+1:D} = \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d})$$

NN



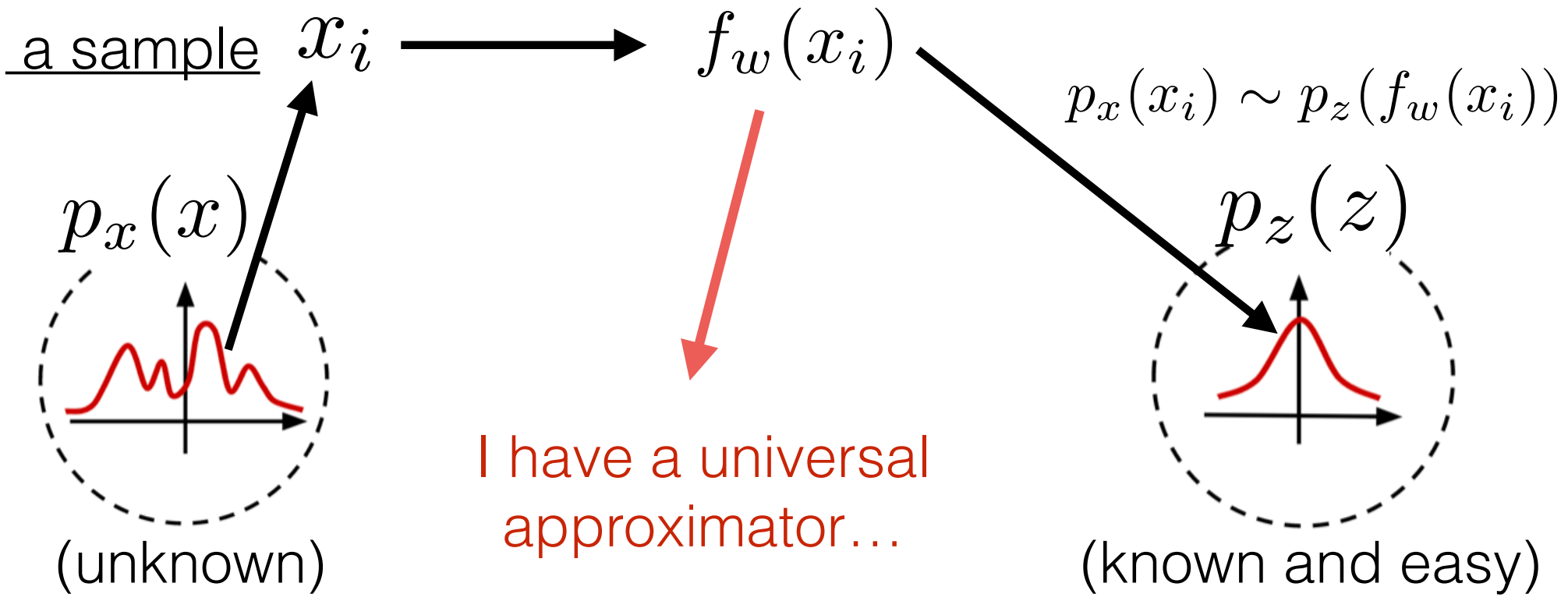
(a) Forward transformation (b) Inverse transformation

- Easily invertible:

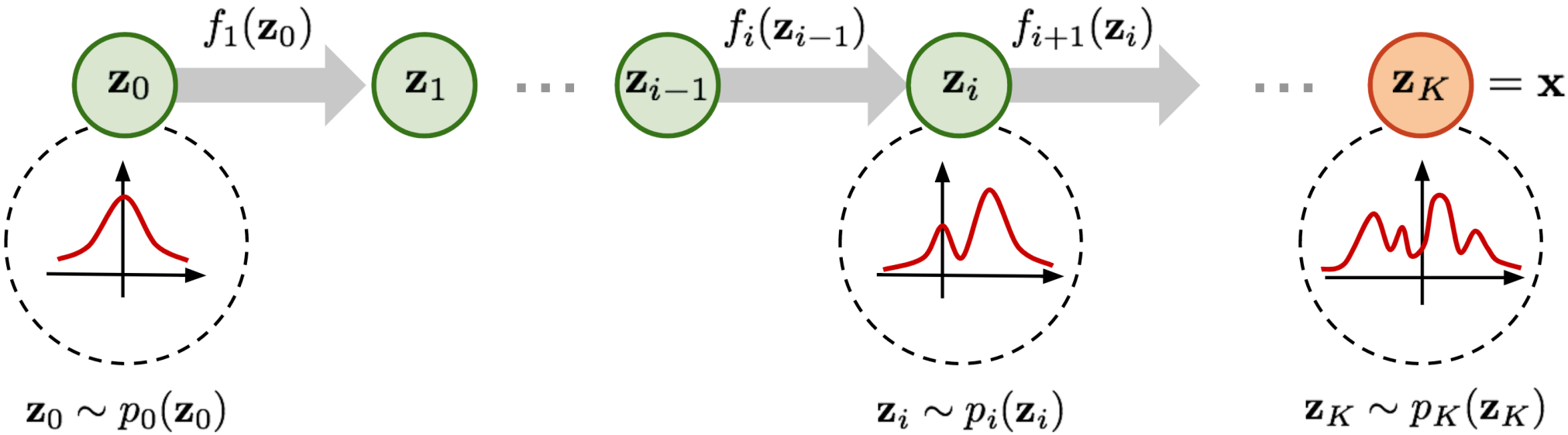
$$\begin{cases} \mathbf{y}_{1:d} & = \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} & = \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \end{cases} \Leftrightarrow \begin{cases} \mathbf{x}_{1:d} & = \mathbf{y}_{1:d} \\ \mathbf{x}_{d+1:D} & = (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d})) \end{cases}$$

- Jacobian:

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d\times(D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \operatorname{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix} \qquad \det(\mathbf{J}) = \prod_{j=1}^{D-d} \exp(s(\mathbf{x}_{1:d}))_j = \exp(\sum_{j=1}^{D-d} s(\mathbf{x}_{1:d})_j)$$


a sample x_i
$f_w(x_i)$
$p_x(x_i) \sim p_z(f_w(x_i))$
$p_x(x)$
$p_z(z)$
I have a universal approximator…
(unknown)
(known and easy)

Normalizing flow: in practice we use a concatenation of neural networks (called bijectors)



$z_i = f_i(z_{i-1})$ are **invertible** and **differentiable** transformations

$f = f_1 \circ f_2 \ldots \circ f_{k-1} \circ f_k$ is also invertible and differentiable