



UNIVERSITY  
OF AMSTERDAM

# Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

***Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track***  
***Lecture 6, 05/10/2020***

# Today's lecture

- 📌 Information Theory revisited
- 📌 Generative Models in ML
- 📌 Adversarial Learning and Generative Adversarial Networks
- 📌 Energy-based models
- 📌 Guest lecture by **Dr. Sascha Caron** on applications of ML to high-energy physics

# Information Theory revisited

# Information Theory

the main ideas of **information theory** are ubiquitous in machine learning. In order to discuss generative models and adversarial learning it is useful to review a few basic concepts here

consider a discrete random variable  $x$ . How much information is brought in if we measure an specific value of this variable (its **degree of surprise**)?

*the greater the surprise, the bigger the amount of info  
e.g. ``the sun did not rise this morning'' contains more surprise than ``the sun rose this morning''!*

clearly a measure of ``*how much information a new measurement brings*'' depends on the probability distribution of  $x$ ,  $p(x)$ . Thus we look for an information-measuring function  $h[p(x)]$

*if two events  $x$  and  $y$  are independent, the information we gain from observing both of them is the sum of the information we gain from each of them separately*

$$h(x, y) = h(x) + h(y)$$

*But these two events are statistically independent,  
thus their probabilities obey*

$$p(x, y) = p(x)p(y)$$

# Information Theory

the main ideas of **information theory** are ubiquitous in machine learning. In order to discuss generative models and adversarial learning it is useful to review a few basic concepts

consider a discrete random variable  $x$ . How much information is brought in if we measure an specific value of this variable (its **degree of surprise**)?

*the greater the surprise, the bigger the amount of info  
e.g. ``the sun did not rise this morning'' contains more surprise than ``the sun rose this morning''!*

clearly a measure of ``*how much information a new measurement brings*'' depends on the probability distribution of  $x$ ,  $p(x)$ . Thus we look for an information-measuring function  $h[p(x)]$

*if two events  $x$  and  $y$  are independent, the information we gain from observing both of them is the sum of the information we gain from each of them separately*

$$h(x, y) = h(x) + h(y)$$

*But these two events are statistically independent, thus their probabilities obey*

$$p(x, y) = p(x)p(y)$$

$$h(x) = -\ln p(x)$$

*information is positive!*

$$0 \leq p(x) \leq 1$$

# Information Theory

what is the **total amount of information** contained in a set of instances of  $x$ ?

*It will be the expectation value of  $h(x)$  wrt the probability distribution  $p(x)$*

$$H[x] = \sum_x p(x)h(x) = - \sum_x p(x)\ln p(x)$$

which is known as the **entropy** associated to the random variable  $x$

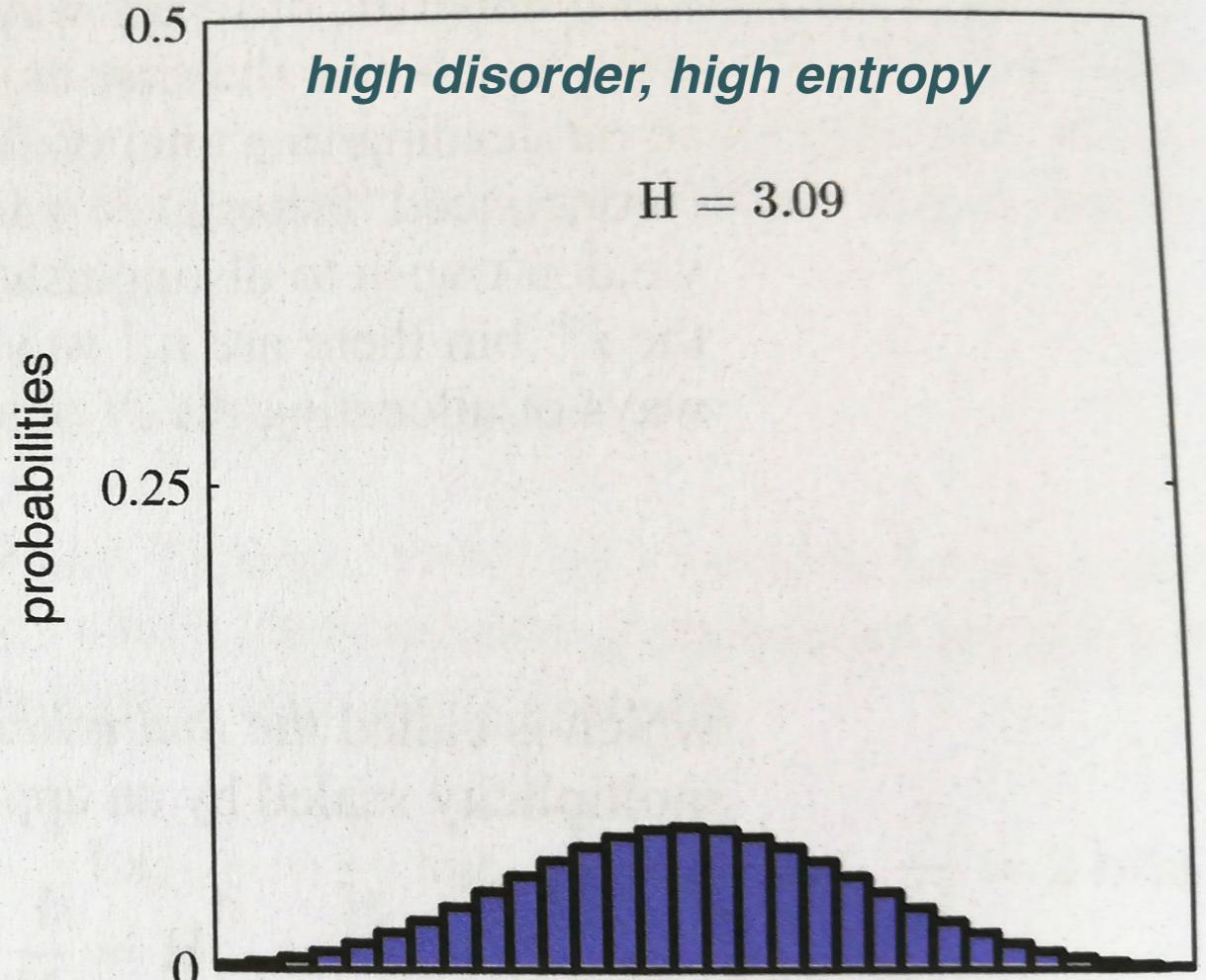
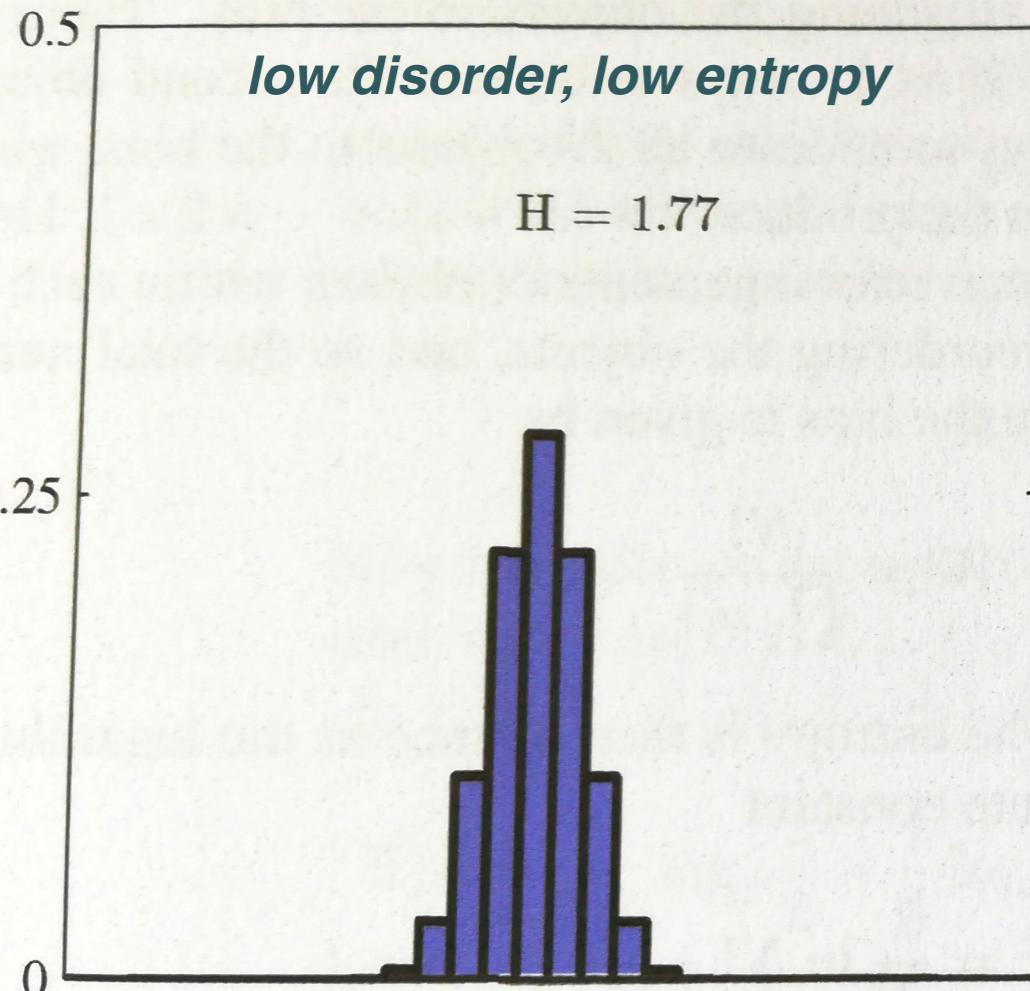
Note some **important properties** about the *surprise function*  $h(x)$  and the entropy  $H[x]$ :

- Low-probability events have associated a **high information content**
- However low-probability events carry a **small weight** in the total entropy
- One can show that steeply-varying probability distributions lead to smaller entropies than smooth distributions (largest entropy is for uniform distributions)
- Entropy can be understood to as the **average amount of information** needed to specify the state of a random variable

*entropy is of course closely connected  
to disorder: the higher the disorder, the largest the associated entropy*

# Information Theory

*the broader the distribution, the higher the disorder and thus the higher the entropy*



*the highest possible value of the entropy corresponds to a uniform distribution*

# Statistical distances and similarity

In many applications for machine learning we want to assess the **degree of similarity** between two probability distributions: these metrics are known as **statistical distances**

The Kullback-Leibler divergence is a measure of the **similarity** between two probability distributions  $p(\mathbf{x})$  and  $q(\mathbf{x})$  and plays an important role in machine learning models

$$D_{KL}(p \parallel q) = \int d\mathbf{x} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \quad D_{KL}(q \parallel p) = \int d\mathbf{x} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})}$$

*note that KL divergence is not symmetric (as usual with statistical distances)*

$$D_{JS}(p \parallel q) = \frac{1}{2} \left( D_{KL}\left(p \parallel \frac{p+q}{2}\right) + D_{KL}\left(q \parallel \frac{p+q}{2}\right) \right)$$

*Jensen-Shannon divergence (symmetric version of KL divergence)*

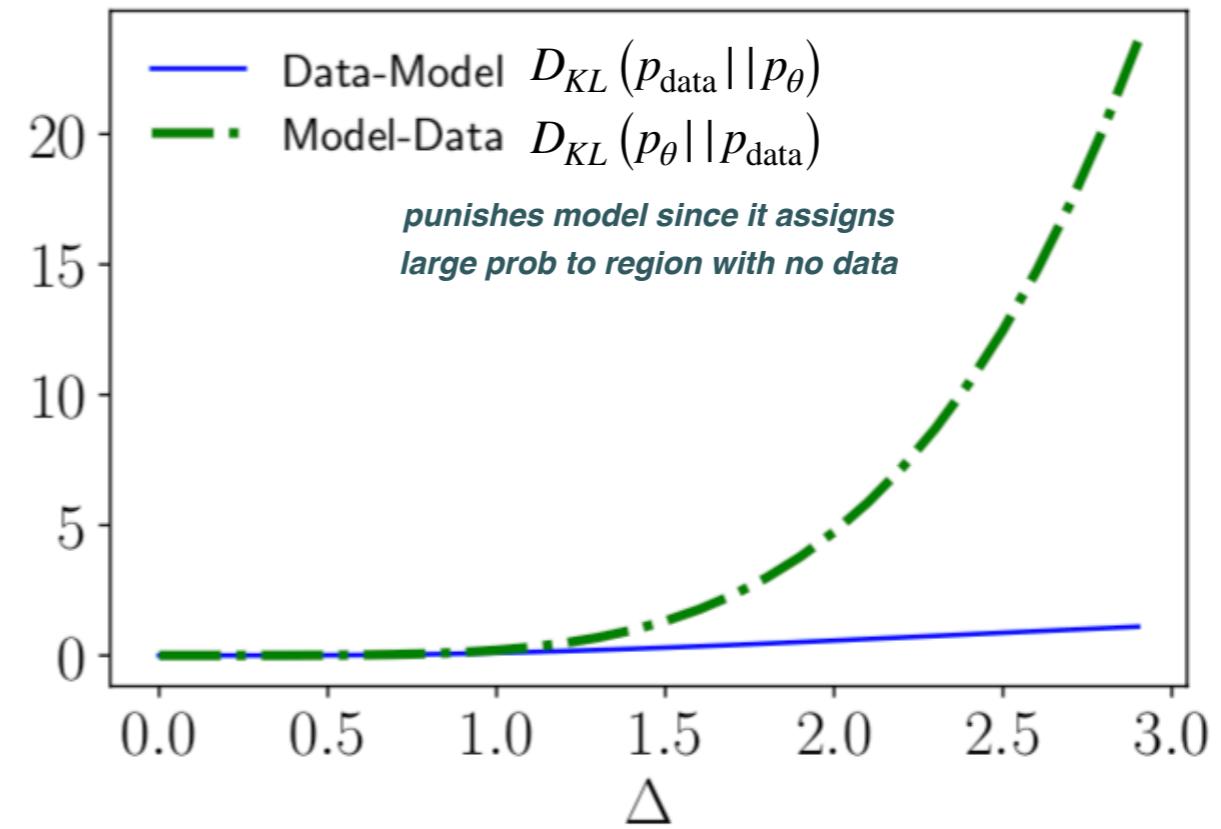
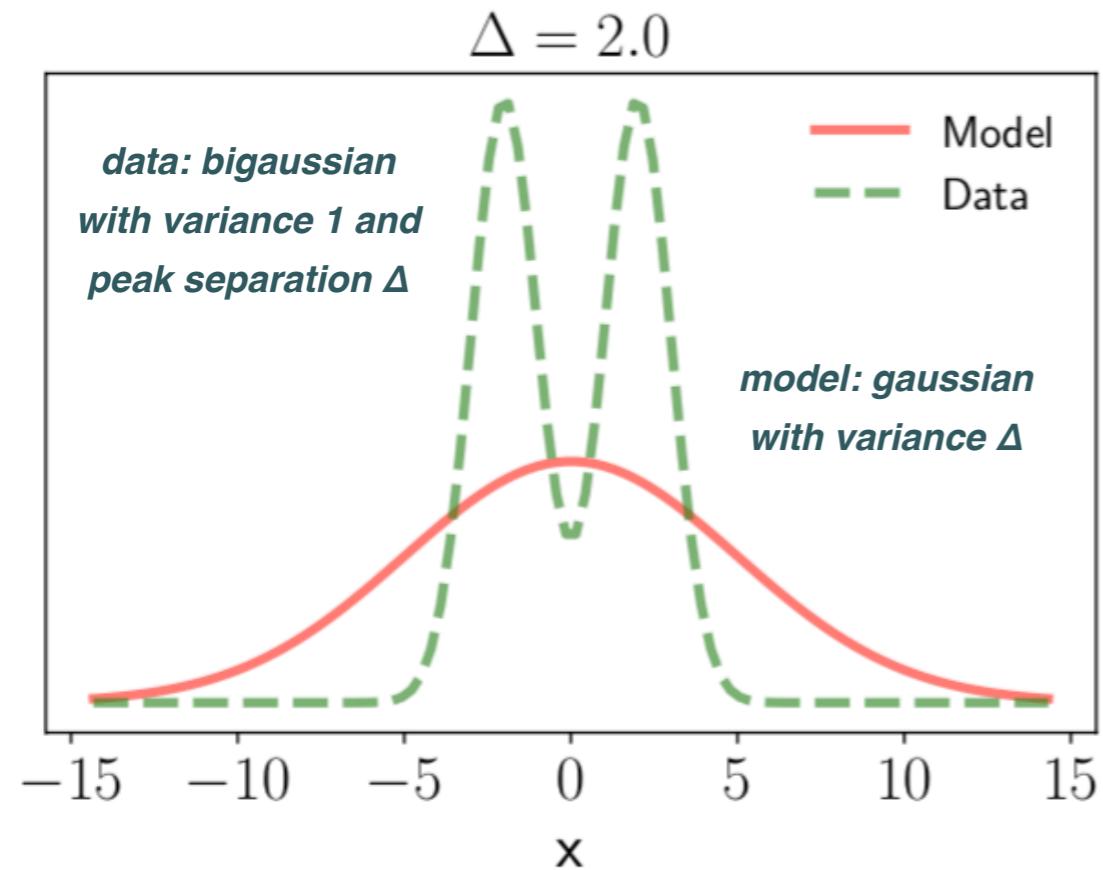
The KL-divergence is **positive-definite**, and only vanishes when  $p(\mathbf{x})=q(\mathbf{x})$

$$D_{KL}(p \parallel q) \geq 0$$

*in general the integral cannot be computed and one needs to sample the two probability distributions by means of a suitable binning*

# Statistical distances and similarity

Similarity between probability distributions is a subtle concept!



$$D_{KL}(p_{\text{data}} \parallel p_{\theta}) \longrightarrow$$

misses important information when comparing the data and theory probability distributions

$$D_{KL}(p_{\theta} \parallel p_{\text{data}}) \longrightarrow$$

accounts for non-trivial features beyond the lowest moments of the distribution

*identifies when model gives too much weight to regions without data!*

# Statistical distances and similarity

another popular statistical distance is the **Kolmogorov-Smirnov statistic**

Assume we have  $n$  observations of the random variable  $x$ . The **empirical distribution function** is

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n I_{[-\infty, x]}(x_i)$$

*indicator function*

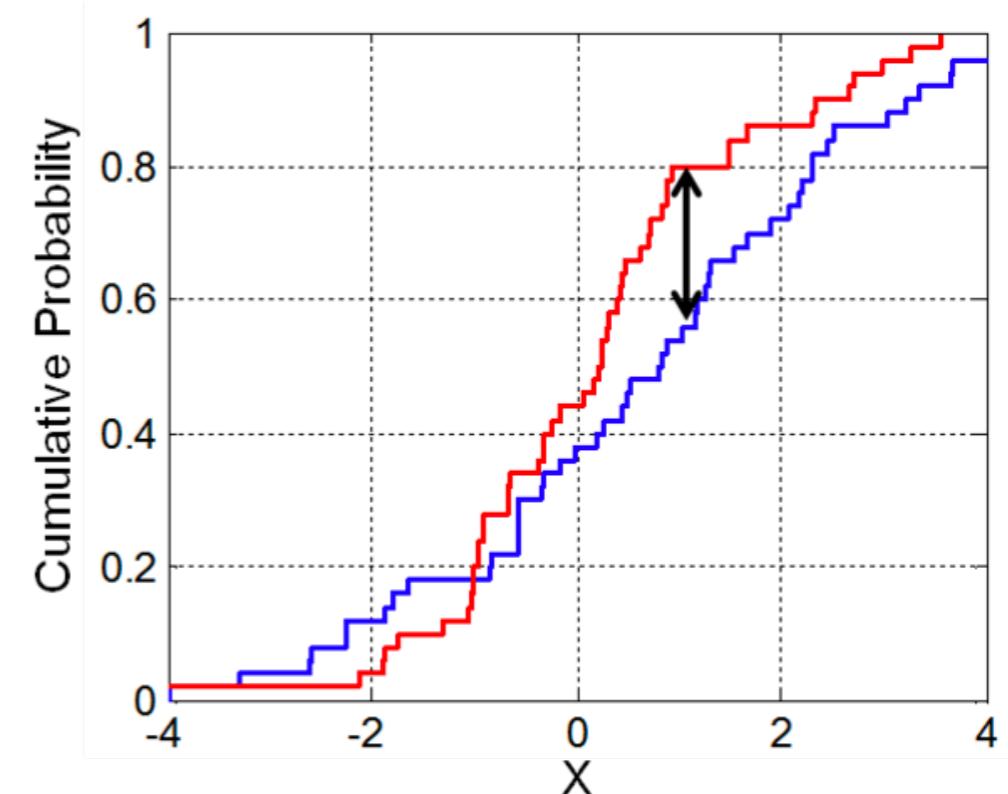
$$I_{[-\infty, x]}(x_i) = 1, \text{ for } x_i \leq x, \quad I_{[-\infty, x]}(x_i) = 0, \text{ for } x_i > 0$$

we can then define the **KS-statistic** between two probability distributions as

$$\text{KS} = \sup_x [F_{1,n}(x) - F_{2,m}(x)]$$

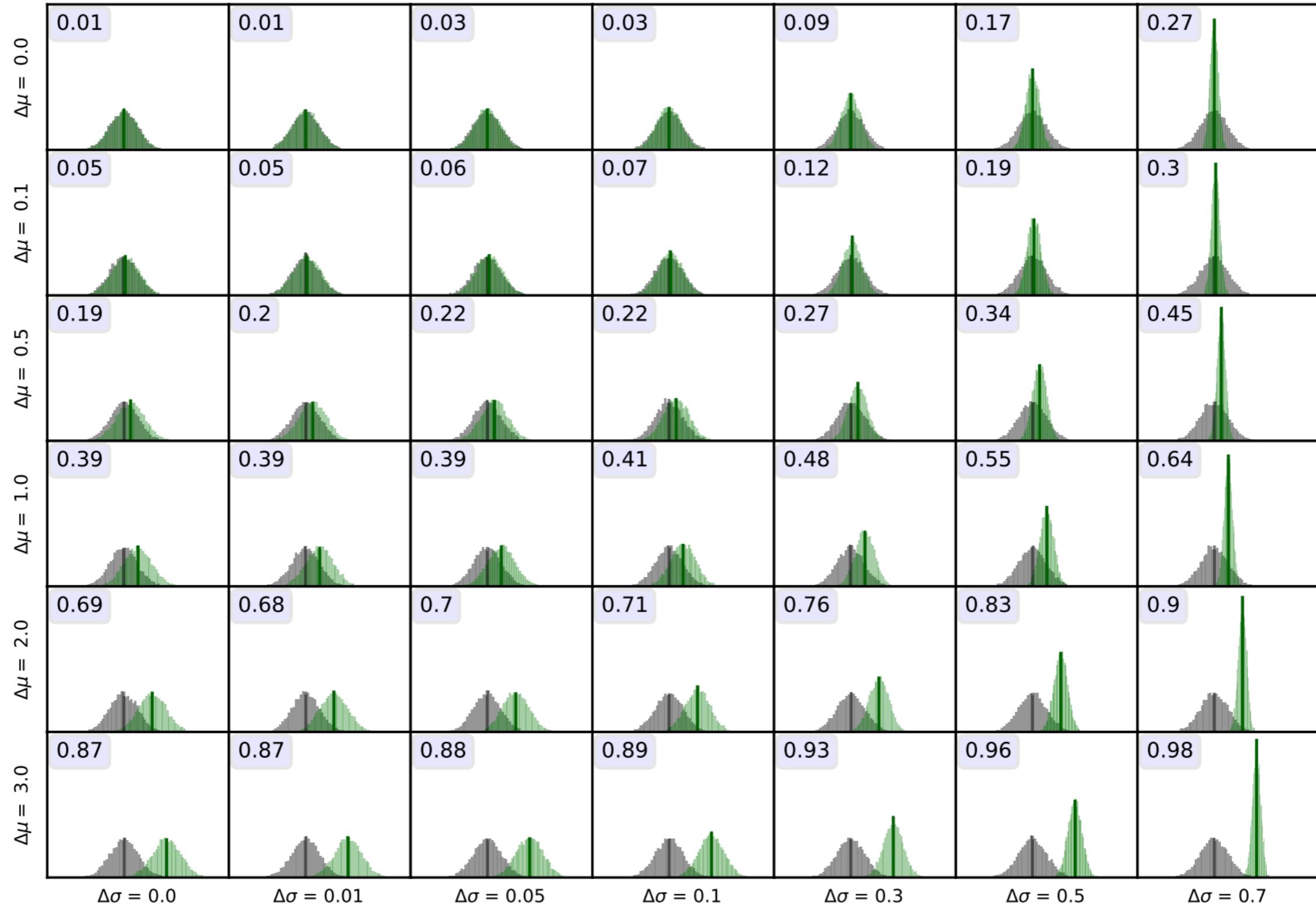
*EDF sample 1  
( $n$  elements)*      *EDF sample 2  
( $m$  elements)*

*the KS statistics is sensitive to all possible differences between the two distributions (which might be excessive in some cases)*



# Statistical distances and similarity

KS statistic for different means and std devs ( $N=10000, \mu_0=0, \sigma_0=1$ )



# **Generative Models**

# Generative Models

let us go back to our discussion of **Decision Theory**. There we saw that a general classification problem can be separated into two distinct steps:

*posterior class probabilities*

- The **inference stage**, where a set of input examples is used to train a model for  $p(\mathcal{C}_k | \mathbf{x})$
- The **decision stage**, where the information on these posterior probabilities is used to make **optimal class assignments**

So far we focused on **discriminative models**, where some criterion (e.g. minimise misclassification) is used together with the posterior probabilities to assign each new instance to a class

In this lecture we will discuss **generative models** which aim to model the **distribution in the space of inputs  $\mathbf{x}$** . The name stems because using this distribution one can **generate synthetic data points** in the input space

one benefit of this approach is that we access the marginal density in the space of input data,  $p(\mathbf{x})$ , which is specially useful to detect new data points that have low probability in the model: **outlier, anomaly, or novelty detection**

# Generative Models

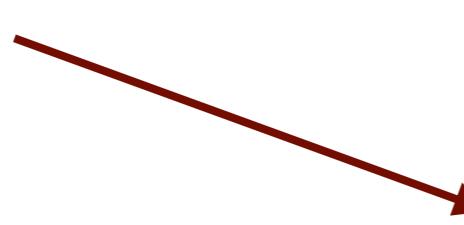
**generative models** can be better understood in the context of **classification**: assume we have a set of  $n$  instances  $\{\mathbf{x}\}$  with  $p$  features each that we want to classify into categories  $C_k$

- First of all, one must **solve the inference problem** and determine the **class-conditional probabilities** for each class individually

$$p(x | \mathcal{C}_k) \longrightarrow \text{probability that given a class } C_k \text{ the instance } x \text{ is found in it}$$

- Then separately infer the **prior class probabilities**  $p(\mathcal{C}_k)$

- Determine the **posterior class probabilities** by means of Bayes' Theorem

$$p(\mathcal{C}_k | x) = \frac{p(x | \mathcal{C}_k) p(\mathcal{C}_k)}{p(x)}$$


*probability that  $x$  belongs to class  $C_k$*

$$p(x) = \sum_k p(x | \mathcal{C}_k) p(\mathcal{C}_k) \longrightarrow \text{probability to generate the instance } x$$

# Generative Models

$$p(\mathcal{C}_k | \mathbf{x}) = \frac{p(\mathbf{x} | \mathcal{C}_k) p(\mathcal{C}_k)}{p(\mathbf{x})}$$

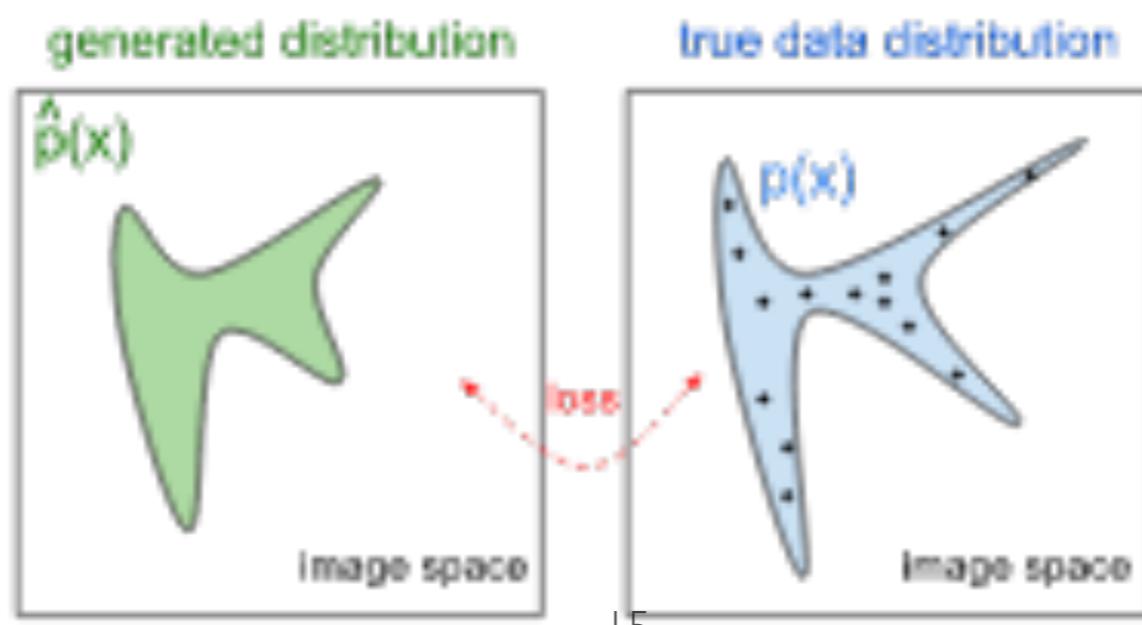
*probability that  $\mathbf{x}$  belongs to class  $C_k$*

$$p(\mathbf{x}) = \sum_k p(\mathbf{x} | \mathcal{C}_k) p(\mathcal{C}_k)$$

*probability to generate the instance  $\mathbf{x}$*

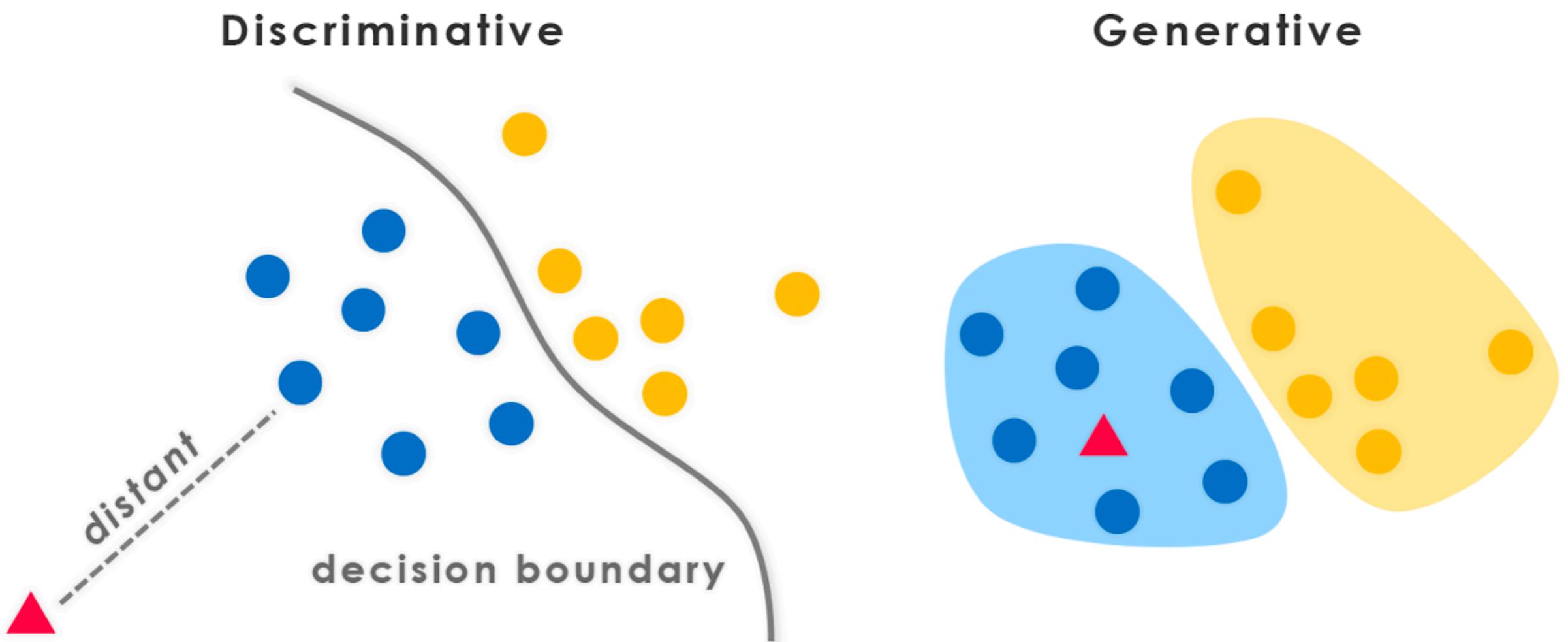
given that we now have the posterior probabilities, we can use **Decision Theory** to determine the class membership for each new instance  $\mathbf{x}$

the key aspect of generative models is that we have access to  $p(\mathbf{x})$ , the probability distribution in the input data space: sampling from  $p(\mathbf{x})$  one can then **generate new synthetic data points** in input space



# Generative Models

*Compare graphically Discriminative and Generative Models for classification*



*discriminative models works best when new instances are far from the decision boundary*

*in generative models new instances are degenerated in the bulk of the input space probability distribution*

# Probabilistic generative models

based on these ideas let us construct an explicit **probabilistic generative model** in the context of classification problems with **two categories**. In this case Bayes' Theorem reads

*posterior class  
probabilities*

$$p(\mathcal{C}_k | \mathbf{x}) = \frac{p(\mathbf{x} | \mathcal{C}_k) p(\mathcal{C}_k)}{p(\mathbf{x})}$$

$$p(\mathcal{C}_1 | \mathbf{x}) = \frac{p(\mathbf{x} | \mathcal{C}_1) p(\mathcal{C}_1)}{p(\mathbf{x} | \mathcal{C}_1) p(\mathcal{C}_1) + p(\mathbf{x} | \mathcal{C}_2) p(\mathcal{C}_2)}$$

which can be simplified as

$$p(\mathcal{C}_1 | \mathbf{x}) = \frac{1}{1 + \exp(-a)} = \sigma(a) \quad a = \ln \frac{p(\mathbf{x} | \mathcal{C}_1) p(\mathcal{C}_1)}{p(\mathbf{x} | \mathcal{C}_2) p(\mathcal{C}_2)}$$

in terms of the **logistic sigmoid function** which we discussed in the context of neural nets

# Probabilistic generative models

Note that once we have trained such a model we can do two things:

- Use **posterior probabilities** with **decision theory** to determine **decision boundaries**

$$p(\mathcal{C}_k | \mathbf{x}) + \text{decision theory} \rightarrow \text{class boundaries}$$

- Use the **probability distribution in the space of input data** to generate synthetic samples

$$p(\mathbf{x}) \rightarrow \text{generate new instances of } \mathbf{x}$$

For  $K > 2$  classes the posterior probabilities are given by

$$p(\mathcal{C}_k | \mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad a_k = \ln(p(\mathbf{x} | \mathcal{C}_k)p(\mathcal{C}_k))$$



*normalised exponential of softmax function*

*(smoothed version of the maximum function)*

# Generative Models

we can summarise the main ideas **underlying generative models** as follows

Most ML models discussed here (Supervised NNs, logistic regression, ensemble models) are **discriminative**: designed to identify **differences between groups of data**

*e.g. cats vs dogs discrimination*

these models cannot carry some tasks such as **drawing new examples** from an unknown probability distribution: for this we need **generative models**

*e.g. learn how to draw new examples  
of cat and dog images*

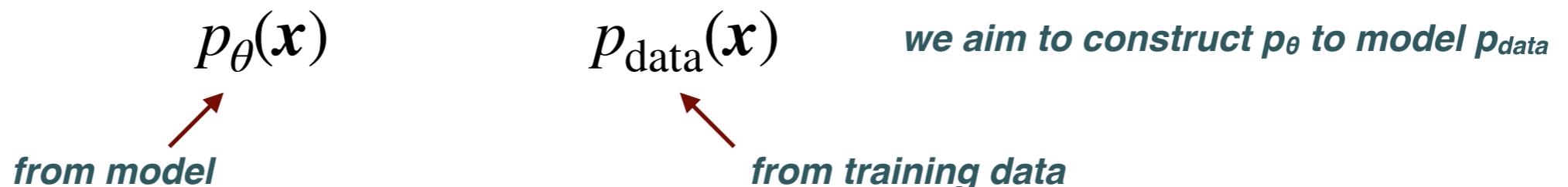
*e.g. generate new samples of a given phase  
of the Ising model*

generative models are Machine Learning techniques that allows to learn **how to generate new examples** similar to those found in a training dataset

# **Adversarial Learning and Generative Adversarial Networks**

# Maximising similarity

In **generative models** one deals with two probability distributions (data and model), which we would like to have as similar as possible



however subtleties about how we define **similarity** have large implications for the model training

maximising the **log-likelihood of the data under the model** is the same as **minimising the KL divergence** between the data distribution and the model distribution

Kullback-Leibler divergence

$$D_{KL} (p_{\text{data}} || p_\theta) = \int d\mathbf{x} p_{\text{data}}(\mathbf{x}) \log \frac{p_{\text{data}}(\mathbf{x})}{p_\theta(\mathbf{x})}$$
$$= \int d\mathbf{x} p_{\text{data}}(\mathbf{x}) \log p_{\text{data}}(\mathbf{x}) - \int d\mathbf{x} p_{\text{data}}(\mathbf{x}) \log p_\theta(\mathbf{x})$$
$$= S_p[p_{\text{data}}] - \langle \log p_\theta \rangle_{\text{data}}$$

Entropy

*Expected value of model probability given the data*

# Maximising similarity

maximising the **log-likelihood** of the data under the model is the same as **minimising the KL divergence** between the data distribution and the model distribution

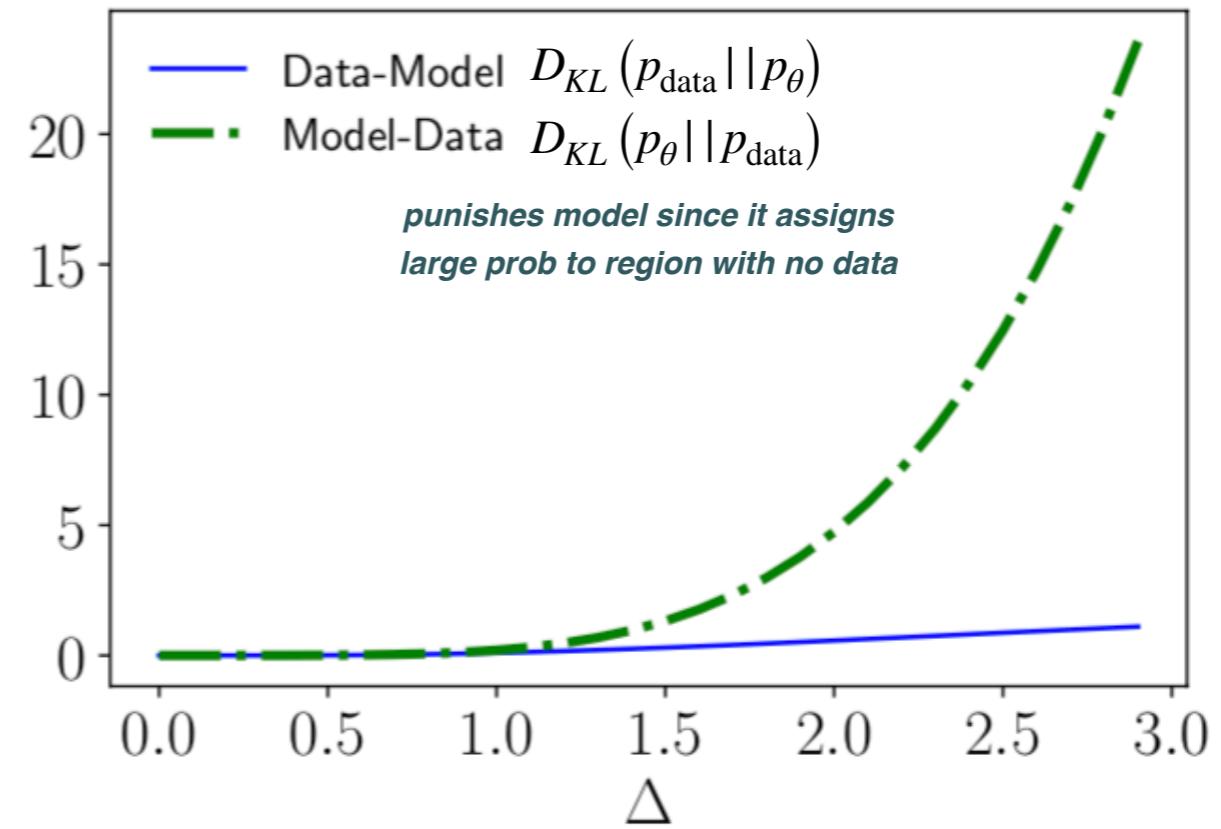
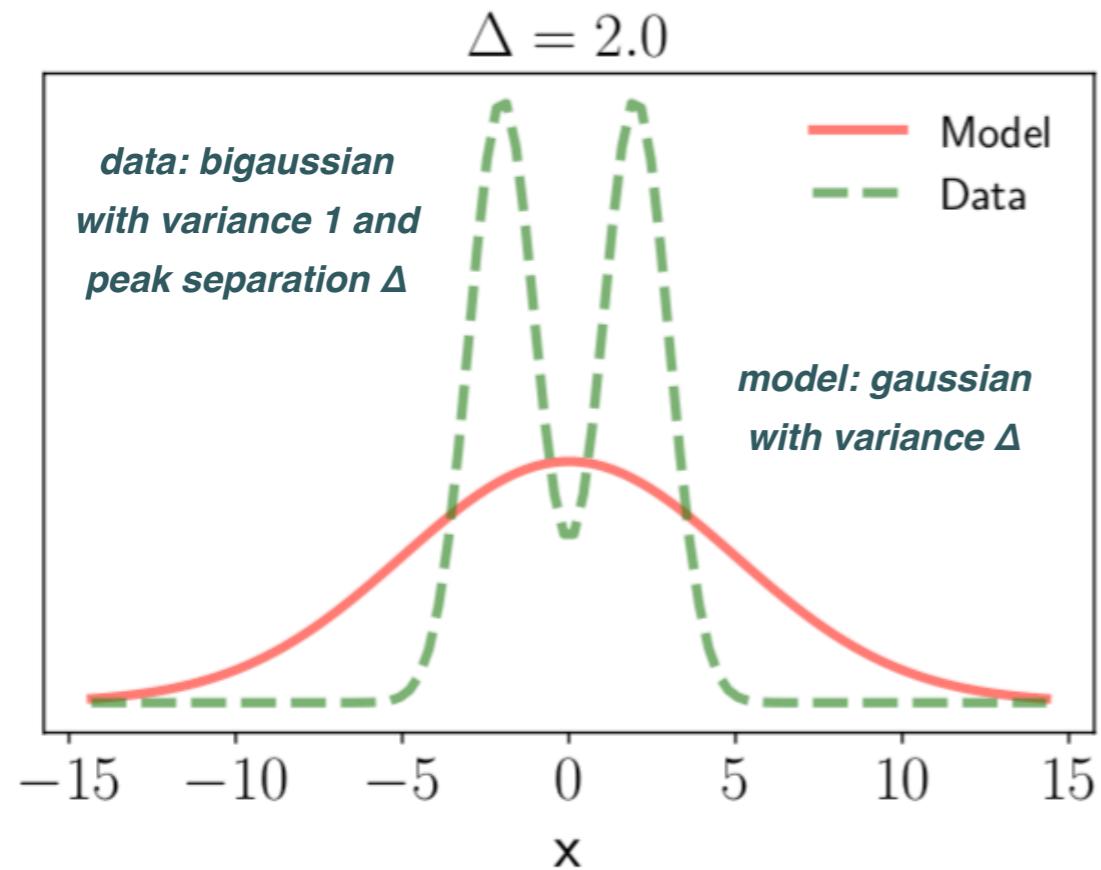
$$\langle \log p_\theta(x) \rangle_{\text{data}} = S_p[p_{\text{data}}] - D_{KL}(p_{\text{data}} \parallel p_\theta)$$

*↑*                                   *↑*                                   *↑*  
*Log-likelihood of data under model*      *entropy of data:*      *KL-divergence*  
*independent of model parameters*

by minimising the KL divergence between two distributions, we can build a model that reproduces the **input data distribution**

# Statistical distances and similarity

Similarity between probability distributions is a subtle concept!



$$D_{KL}(p_{\text{data}} \parallel p_{\theta}) \longrightarrow$$

misses important information when comparing the data and theory probability distributions

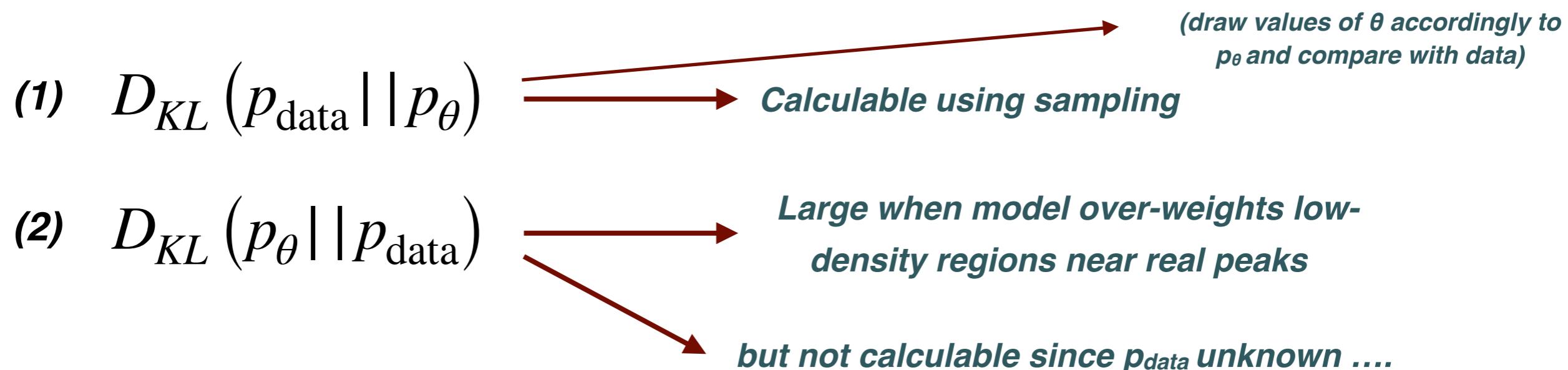
$$D_{KL}(p_{\theta} \parallel p_{\text{data}}) \longrightarrow$$

accounts for non-trivial features beyond the lowest moments of the distribution

*identifies when model gives too much weight to regions without data!*

# Adversarial Learning

A subtle point concerns which of the two versions of the KL-divergence to minimise:



In **Adversarial Learning** we achieve a similar goal as that of minimising (2) by training a **discriminator** to distinguish between real data points and samples from the model

By punishing the model for generating points that can be easily discriminated from the data, Adversarial Learning decreases the **weight of regions in the model space that are far away from data points**, regions that inevitably arise when maximising the likelihood

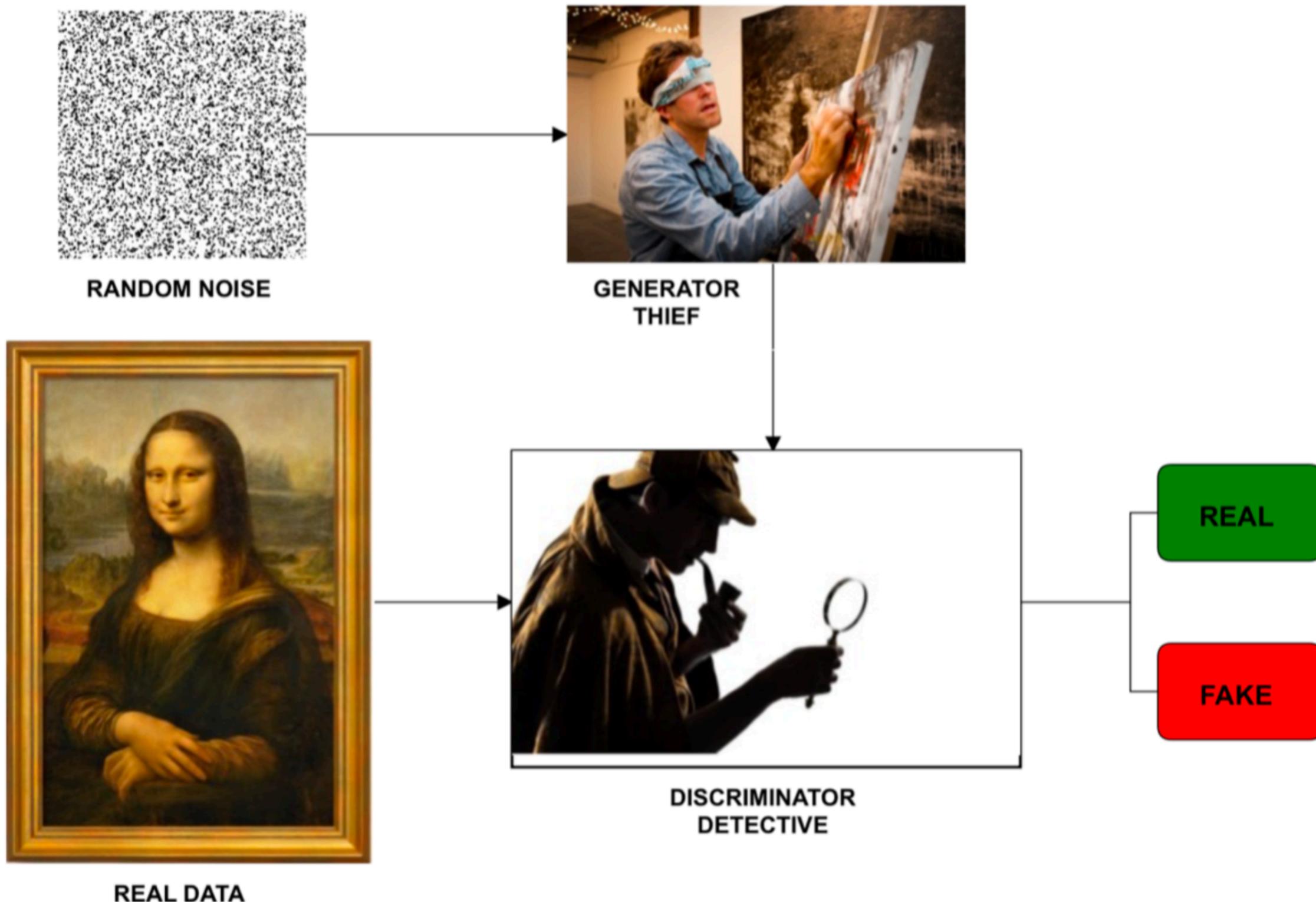
# Generative adversarial networks

Generative Adversarial Networks (GANs) are deep neural network architectures, composed by two independent NNs which **compete against each other**

- (1) A **generator G** NN that creates (samples) pseudo-data by inferring the probability distribution associated to the training dataset
- (2) A **discriminator D** NN which determines the probability of a given sample arises from the actual training data rather than having been produced by **G**

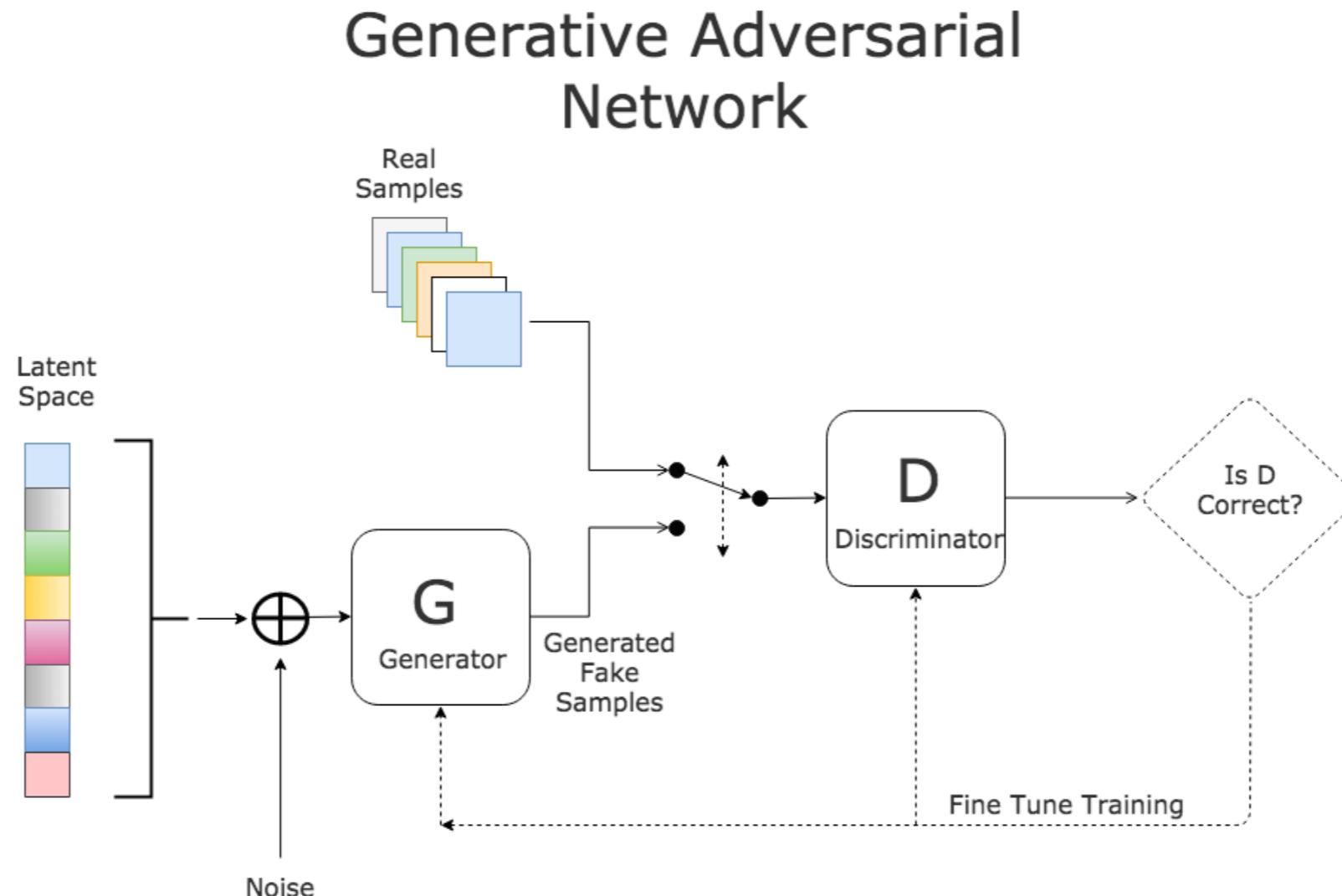
the generator network **G** should be trained to maximise the probability that the discriminator network **D** makes a mistake: that is, **G** should generate pseudo-data samples that are virtually **indistinguishable** from the actual data

# Generative adversarial networks



# Generative Adversarial Networks

- Architecture for an **unsupervised neural network training** (unlabelled samples)
- Based on two **independent nets** that work separately and act as **adversaries**:
  - the **Discriminator (D)** undergoes training and plays the role of classifier
  - the **Generator (G)** and is tasked to generate random samples that **resemble real samples** with a twist rendering them as fake samples.



# Generative Adversarial Networks

# GAN training

As with other NN architectures one uses GD to train GANs, but now one has to **update sequentially** the model parameters of both **G** and **D**

- Take a sample of  $N$  data points from the training set

$$\{\mathbf{x}_n\}_{n=1}^N \quad \mathbf{x}_n = (x_{n,1}, x_{n,2}, \dots, x_{n,p}) \quad p = \text{number of features per sample}$$

- Produce a sample of  $N$  pseudo-data points from generator **G** (at  $\text{ite}_0$  this is random noise)

$$\{\mathbf{z}_n\}_{n=1}^N \quad \mathbf{z}_n = (z_{n,1}, z_{n,2}, \dots, z_{n,p})$$

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C(\theta_D, \theta_G) = \frac{1}{N} \sum_{n=1}^N (\log D(\mathbf{x}_i) + \log(1 - D(G(\mathbf{z}_i))))$$

NN params of **D**      output of **D** when input  
NN params of **G**      a real data sample      a "fake" data sample produced by **G**

- Train **D** using GD to maximise its discrimination capability

# GAN training

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C(\theta_D, \theta_G) = \frac{1}{N} \sum_{n=1}^N (\log D(x_i) + \log(1 - D(G(z_i))))$$

- Train **D** using GD to maximise its discrimination capability

$$\mathbf{v}_t = \eta_t \nabla_{\theta_D} C(\theta_{D,t}, \theta_{G,t}), \quad \theta_{D,t+1} = \theta_{D,t} - \mathbf{v}_t$$

- At this point **D** can tell apart data from pseudo-data pretty well, so we need to train **G** to generate better (closer to the training set) pseudo-data samples

- Produce a sample of  $N$  pseudo-data points from the generator **G**  $\{z_n\}_{n=1}^N$

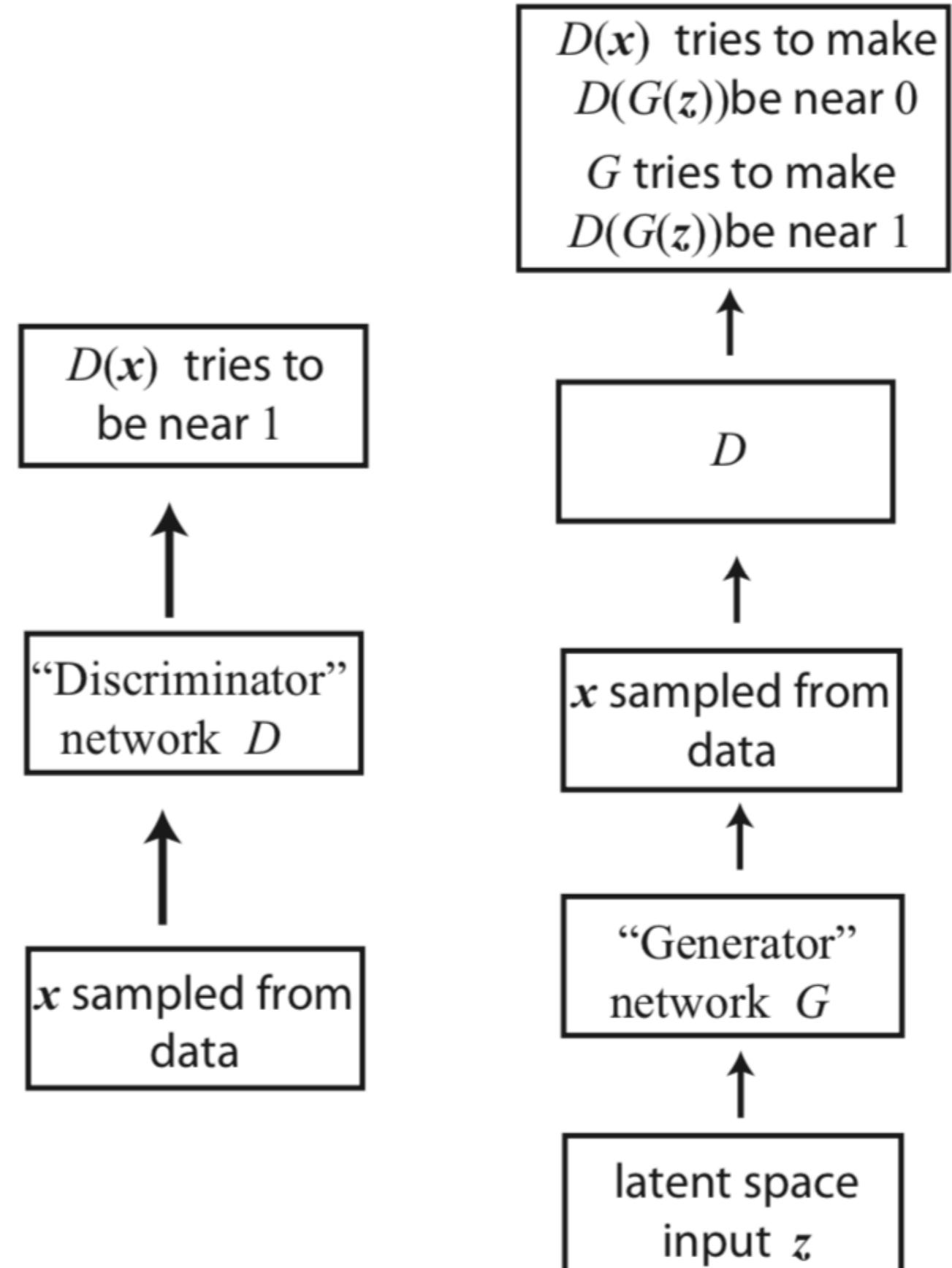
$$C(\theta_D, \theta_G) = \frac{1}{N} \sum_{n=1}^N \log(1 - D(G(z_i)))$$

*output of D (now with its parameters fixed)*

$$\mathbf{v}_t = \eta_t \nabla_{\theta_G} C(\theta_{D,t}, \theta_G), \quad \theta_{G,t+1} = \theta_{G,t} - \mathbf{v}_t$$

# GAN training

the generator and discriminator are sequentially trained and iterated until convergence is achieved, at this point **D** cannot tell apart the pseudo-data from **G** from the real data



# Image generation with GANs



*<https://thispersondoesnotexist.com/>*

# Image generation with GANs

**<https://thispersondoesnotexist.com/>**

# Generative Adversarial Networks

# **Energy-based Models**

# Maximum Entropy Generative Models

basic concept is the **Shannon information-theoretic entropy**, which quantifies the statistical uncertainty one has about a random variable drawn from a probability distribution

$$S_p = \text{Tr}_x p(x) \log p(x)$$

*↑ sum/integral over all  
possible values of variable*

*variables whose prob dist  
I would like to estimate*

assume we have a **set of models**, functions of  $x$ , whose average should coincide with some observed values. What should be their **underlying prob dist?**

$$\begin{array}{ccc} \{f_i(x)\} & \langle f_i \rangle_{\text{obs}} & i = 1, \dots, n_{\text{dat}} \\ \text{model for sample } i \text{ as function of inputs } x & \text{models} & \text{observations} \end{array}$$

**Principle of Maximum Entropy:** choose the probability distribution with the largest uncertainty (Shannon entropy) subject to the observational constraints

$$\langle f_i \rangle_{\text{model}} = \int d\mathbf{x} f_i(\mathbf{x}) p(\mathbf{x}) = \langle f_i \rangle_{\text{obs}}$$

*the selected distribution is the one that makes admits the most ignorance beyond the stated data*

# Maximum Entropy Generative Models

this condition can be expressed as a **Lagrange Multiplier problem** by minimising:

$$\mathcal{L}[p] = -S_p + \sum_i \lambda_i \left( \langle f_i \rangle_{\text{obs}} - \int d\mathbf{x} f_i(\mathbf{x}) p(\mathbf{x}) \right) + \gamma \left( 1 - \int d\mathbf{x} p(\mathbf{x}) \right)$$

*Shannon entropy*                    *observational constraints*                    *normalisation*

whose solution gives us the **Maximum Entropy distribution**

$$p(\mathbf{x}) = \frac{\exp \left( \sum_i \lambda_i f_i(\mathbf{x}) \right)}{\int d\mathbf{x} \exp \left( \sum_i \lambda_i f_i(\mathbf{x}) \right)} = \frac{\exp \left( \sum_i \lambda_i f_i(\mathbf{x}) \right)}{Z}$$

*partition function*                     $E/k_B T = - \sum_i \lambda_i f_i(\mathbf{x})$

which is nothing but the **Boltzmann distribution in statistical mechanics**, and where the parameters of the distribution are fixed by the observations

$$\partial_{\lambda_i} \log Z = \langle f_i \rangle_{\text{data}}$$

# Energy-based Generative Models

these MaxEnt models can be used to **infer the underlying probability distributions** from a finite set of observations, which subsequently can be used to **generate new instances**

training an energy-based generative model: using the data to infer the model parameters

$$E(\mathbf{x}; \boldsymbol{\theta}) = - \sum_i \theta_i f_i(\mathbf{x}) \quad \xrightarrow{\text{model parameters}}$$

as in Supervised Learning, we need to specify a **cost function**, which however in the case of generative models is much subtler: what defines a good model?

the most useful method is to **maximise the log-likelihood** of the training set

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \mathcal{L}(\{\boldsymbol{\theta}\})$$

$$\mathcal{L}(\{\boldsymbol{\theta}\}) = \langle \log p_{\boldsymbol{\theta}}(\mathbf{x}) \rangle_{\text{data}} = - \langle E(\mathbf{x}; \boldsymbol{\theta}) \rangle_{\text{data}} - \log Z(\{\boldsymbol{\theta}\})$$

where we have used that the **generative probability distribution** is of the Boltzmann form and that the partition function does not depend on the data

# Boltzmann machines

The training of **energy-based generative models** proceeds usually via SGD

MaxEnt generative models are defined by the choice of the **energy**

$$p_{\theta}(\mathbf{x}) = \frac{\exp \left( \sum_i \theta_i f_i(\mathbf{x}) \right)}{\int d\mathbf{x} \exp \left( \sum_i \theta_i f_i(\mathbf{x}) \right)} = \frac{\exp \left( \sum_i \theta_i f_i(\mathbf{x}) \right)}{Z} = \frac{\exp (-E(\mathbf{x}; \boldsymbol{\theta}))}{Z}$$

*model parameters* 

one can construct various **other generative models** with different choices of the energy

e.g. *Restricted Boltzmann machines*

$$E(\mathbf{x}; \boldsymbol{\theta}) = - \sum_i a_i(v_i) - \sum_{\mu} b_{\mu}(h_{\mu}) - \sum_{i\mu} W_{i\mu} v_i h_{\mu}$$

