Machine Learning for Physics and Astronomy

Juan Rojo

VU Amsterdam & Theory group, Nikhef

Natuur- en Sterrenkunde BSc (Joint Degree), Honours Track

Lecture 7, 19/10/2021

Today's lecture

- Information Theory for Machine Learning
- Generative Models
- Adversarial Learning and Generative Adversarial Networks
- Kernel Methods and Support Vector Machines

Information Theory for Machine Learning

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)]**

Q: have you encountered before a function that we can use to measure information

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 (additive)

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

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 (additive)

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

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

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

information is positive!

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

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

Information Theory

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

Ii 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***

Q1: which kind of events dominate the calculation of the total entropy?

Q2: is entropy larger in a ``smooth'' or in a ``spiky'' probability distribution?

Information Theory

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

Ii 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***

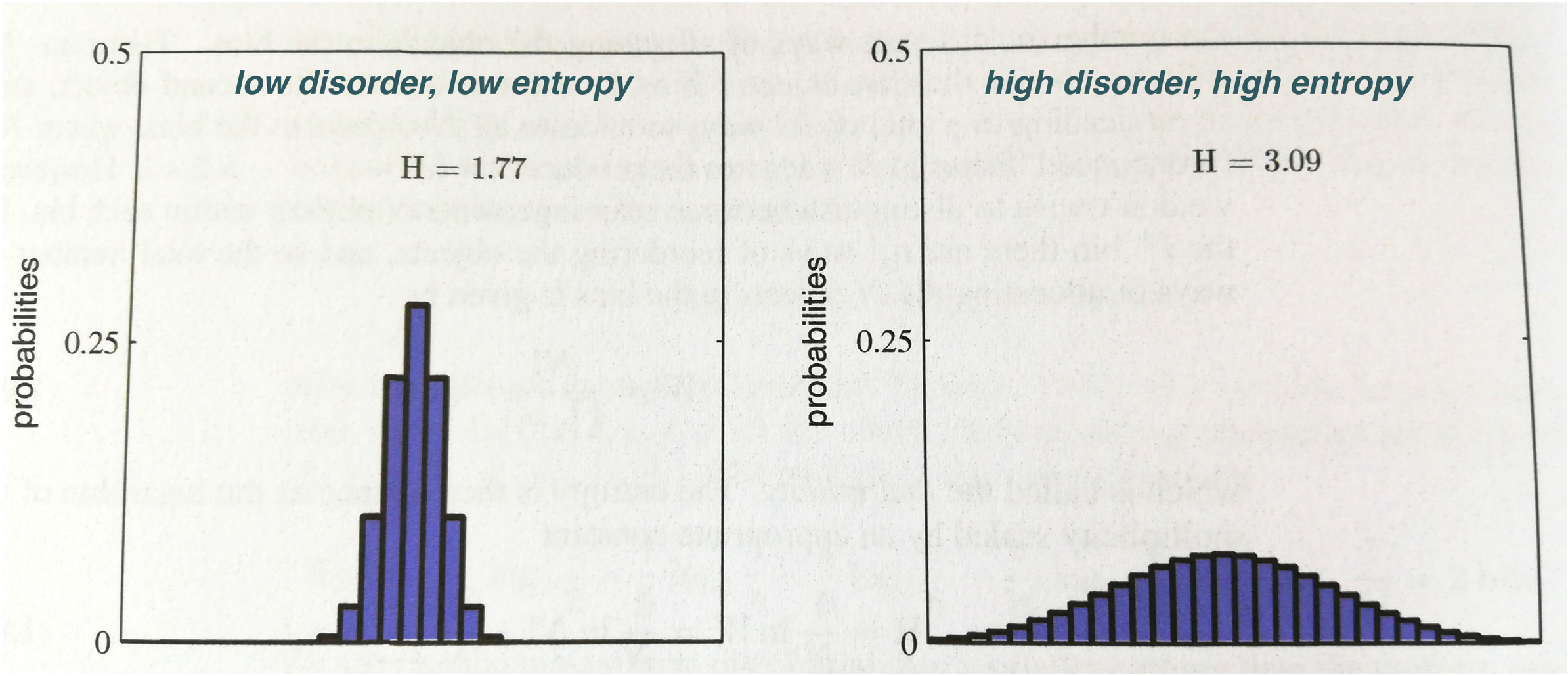
Note some **important properties** about the ***surprise function*** *h(x)* and the **entropy** *H[x]:*

- ☑ 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

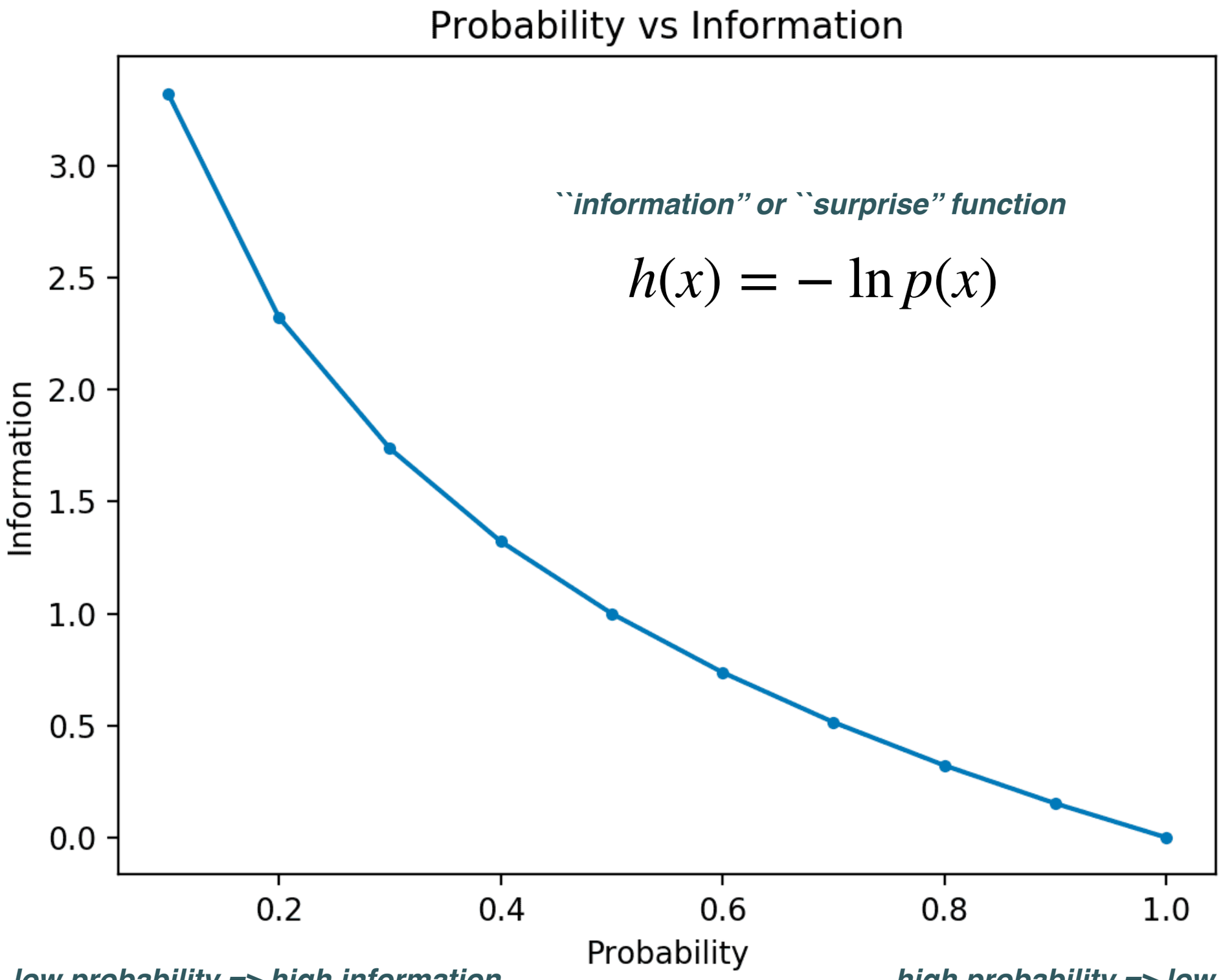
Information Theory

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

Information Theorv



``information" or ``surprise" function

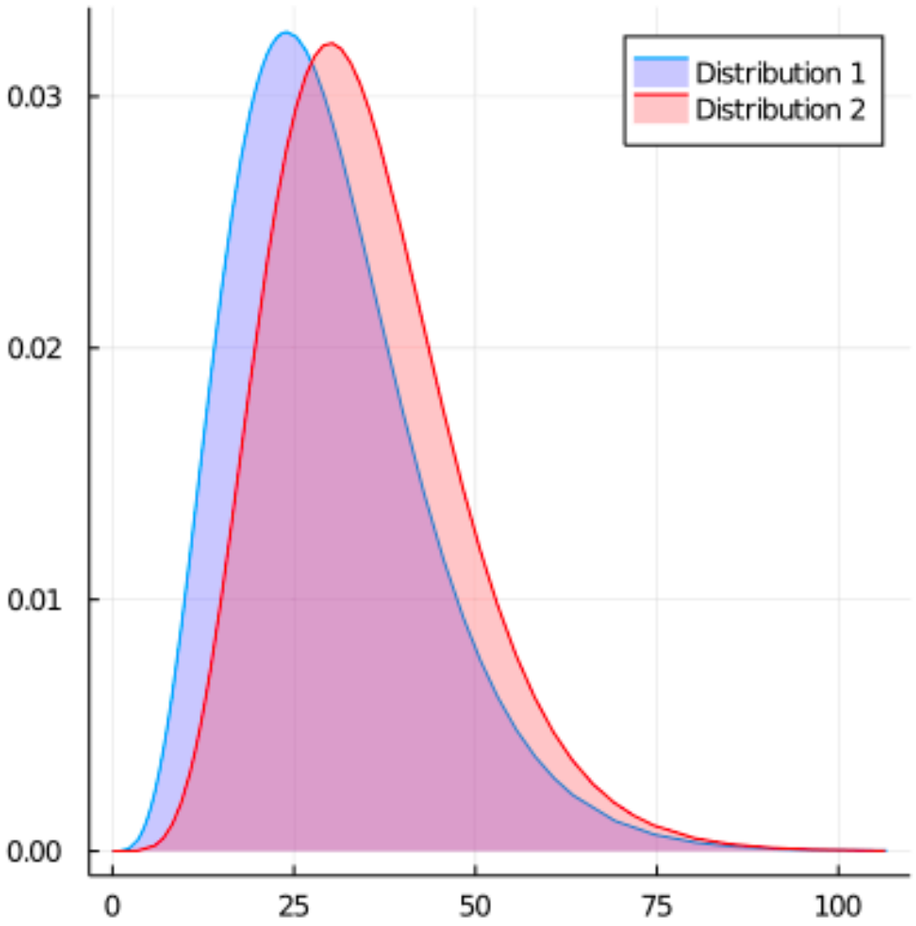
$$h(x) = -\ln p(x)$$

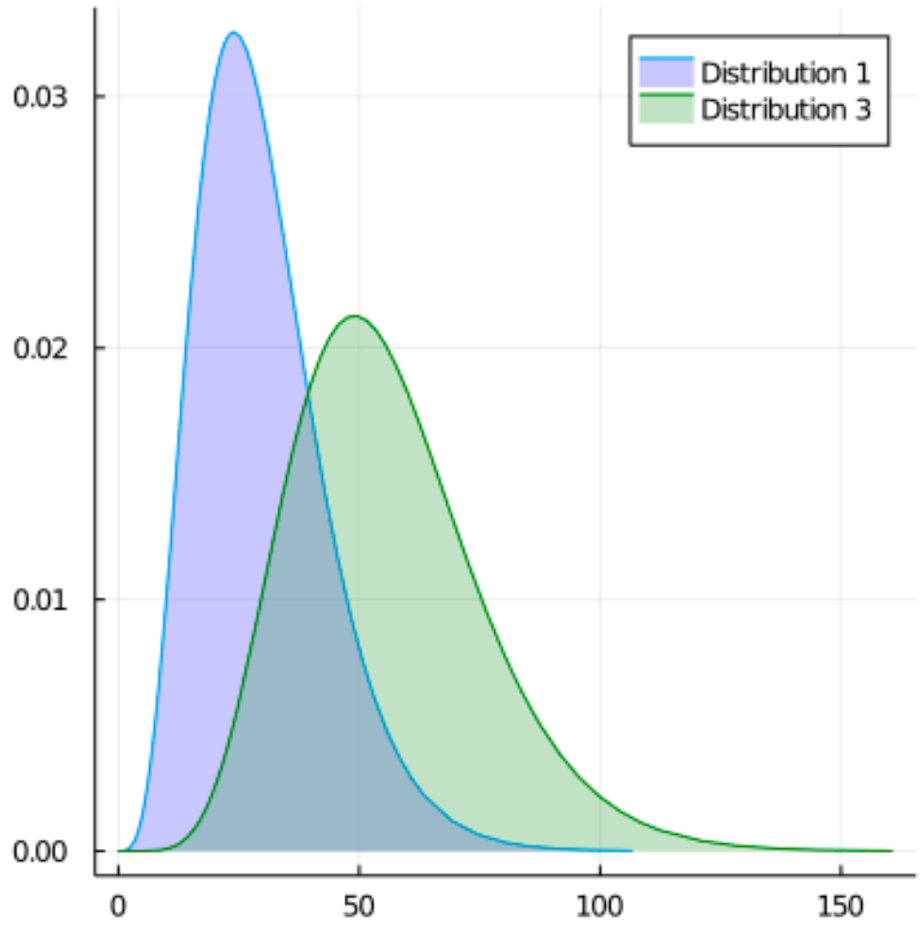
low probability => high information

high probability => low information

Similarity of probability distributions

In many ML applications we need to assess the **degree of similarity** between two probability distributions





Q1: which quantities can we use to determine if Distribution #2 is closer to Distribution #1 or to Distribution #3?

Q2: is similarity an absolute or a relative concept?

Similarity of probability distributions

In many ML applications we need to assess the **degree of similarity** between two probability distributions

We can use the concepts of Information Theory and compute the relative entropy between two distributions

$$H[x] = -\sum_x p(x)\ln p(x) \simeq \int dx\, p(x)\ln p(x)$$

Entropy of stochastic variable x, distributed following p(x)

$$\int dx\, p(x)\ln q(x)$$

Relative entropy of q(x) with respect to p(x)

now compute difference between actual entropy and relative entropy

$$\int dx\, p(x)\ln p(x) - \int dx\, p(x)\ln q(x) = \int dx\, p(x)\ln \frac{p(x)}{q(x)}$$

the smallest this quantity, the more similar the two probability distributions *p(x)* and *q(x)* are

Similarity of probability distributions

This quantity is the **Kullback-Leibler divergence**

$$D_{\mathrm{KL}}(p\,||\,q) = \int dx\, p(x) \ln \frac{p(x)}{q(x)}$$

Let's develop some intuition: what so you think we should find when comparing two Gaussian distributions?

Similarity of probability distributions

This quantity is the **Kullback-Leibler divergence**

$$D_{\mathrm{KL}}(p\,||\,q) = \int dx\, p(x) \ln \frac{p(x)}{q(x)}$$

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_p}} e^{-(x-x_p)^2/2\sigma_p^2} \qquad q(x) = \frac{1}{\sqrt{2\pi\sigma_q}} e^{-(x-x_q)^2/2\sigma_q^2}$$

$$D_{\mathrm{KL}}(p\,||\,q) = \ln \frac{\sigma_q}{\sigma_p} + \frac{1}{(2\pi\sigma_p^2)^{1/2}} \int_{-\infty}^{\infty} dx\, e^{-(x-x_p)^2/2\sigma_p^2} \times \left(-\frac{(x-x_p)^2}{2\sigma_p^2} + \frac{(x-x_q)^2}{2\sigma_q^2} \right)$$

$$D_{\mathrm{KL}}(p\,||\,q) = \ln \frac{\sigma_q}{\sigma_p} + \frac{1}{2\sigma_q^2} \left((x_q - x_p)^2 + (\sigma_p^2 - \sigma_q^2) \right)$$

The closer the means and the standard deviations, the smaller the KL divergence for two gaussian distributions

Similarity of probability distributions

In many ML applications we need to assess the **degree of similarity** between two probability distributions

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

$$D_{KL}\left(p\,||\,q\right) = \int d\boldsymbol{x}\, p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \qquad D_{KL}\left(q\,||\,p\right) = \int d\boldsymbol{x}\, q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}$$

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

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

Jensen-Shannon divergence (symmetric version of KL divergence)

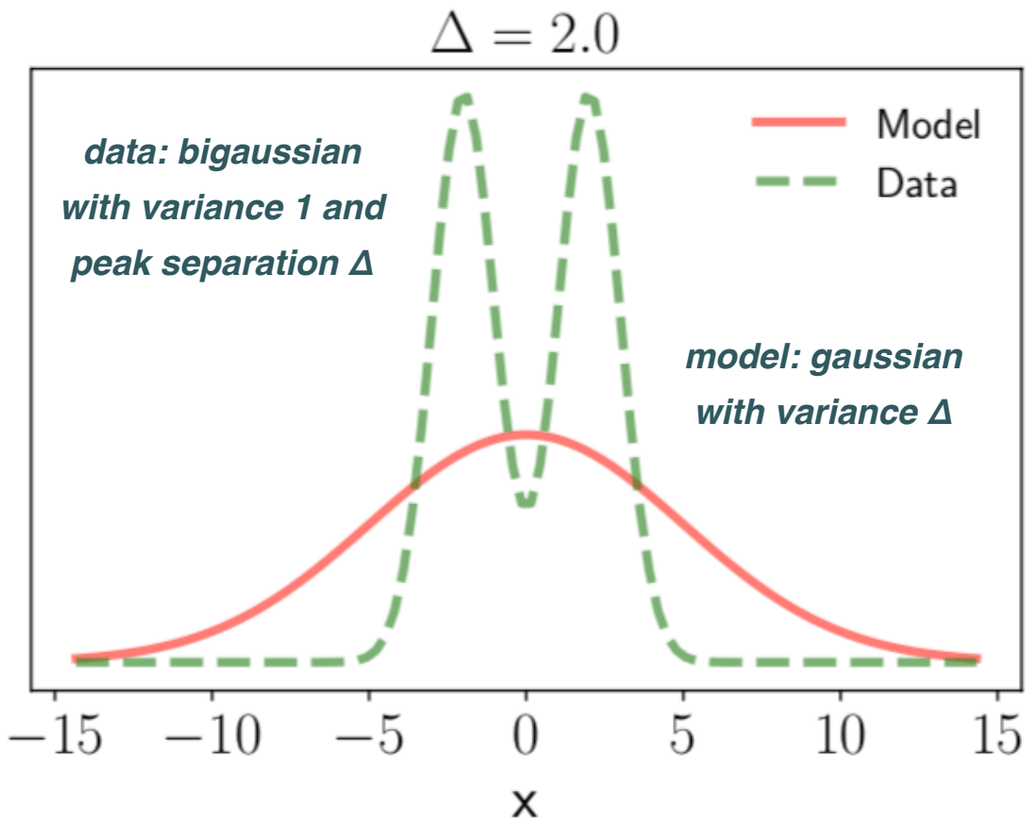
The KL-divergence is **positive-definite**, and only vanishes when *p(**x**)*= q*(**x**)*

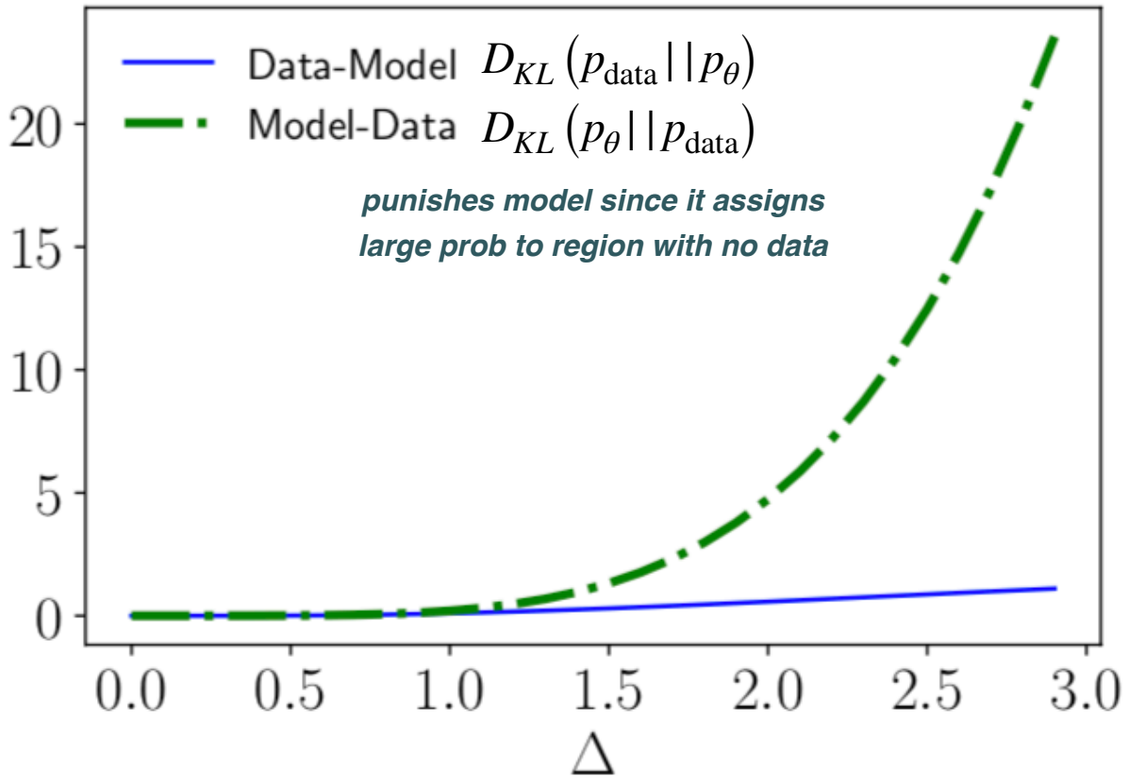
$$D_{KL}\left(p\,||\,q\right) \geq 0$$

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

Similarity of probability distributions

Similarity between probability distributions is a subtle concept!





$D_{KL}\left(p_{\text{data}}||p_\theta\right)$ → misses important information when comparing the data and theory probability distributions

$D_{KL}\left(p_\theta||p_{\text{data}}\right)$ → accounts for non-trivial features beyond the lowest moments of the distribution

identifies when model gives too much wight 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

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

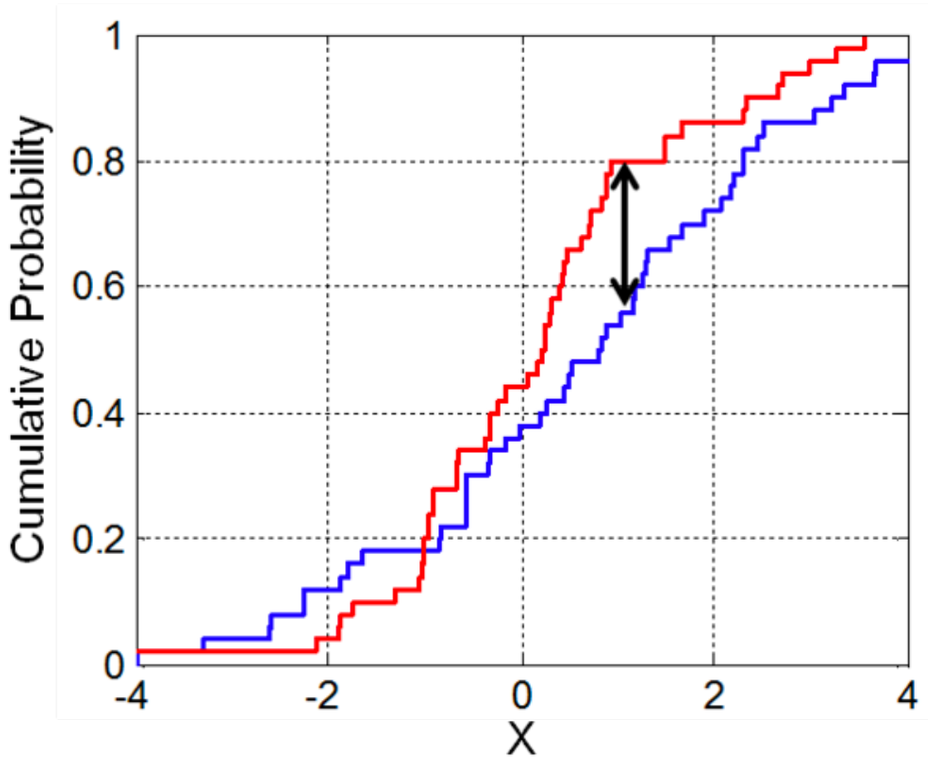
we can then define the **KS-statistic** between two probability distributions as

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

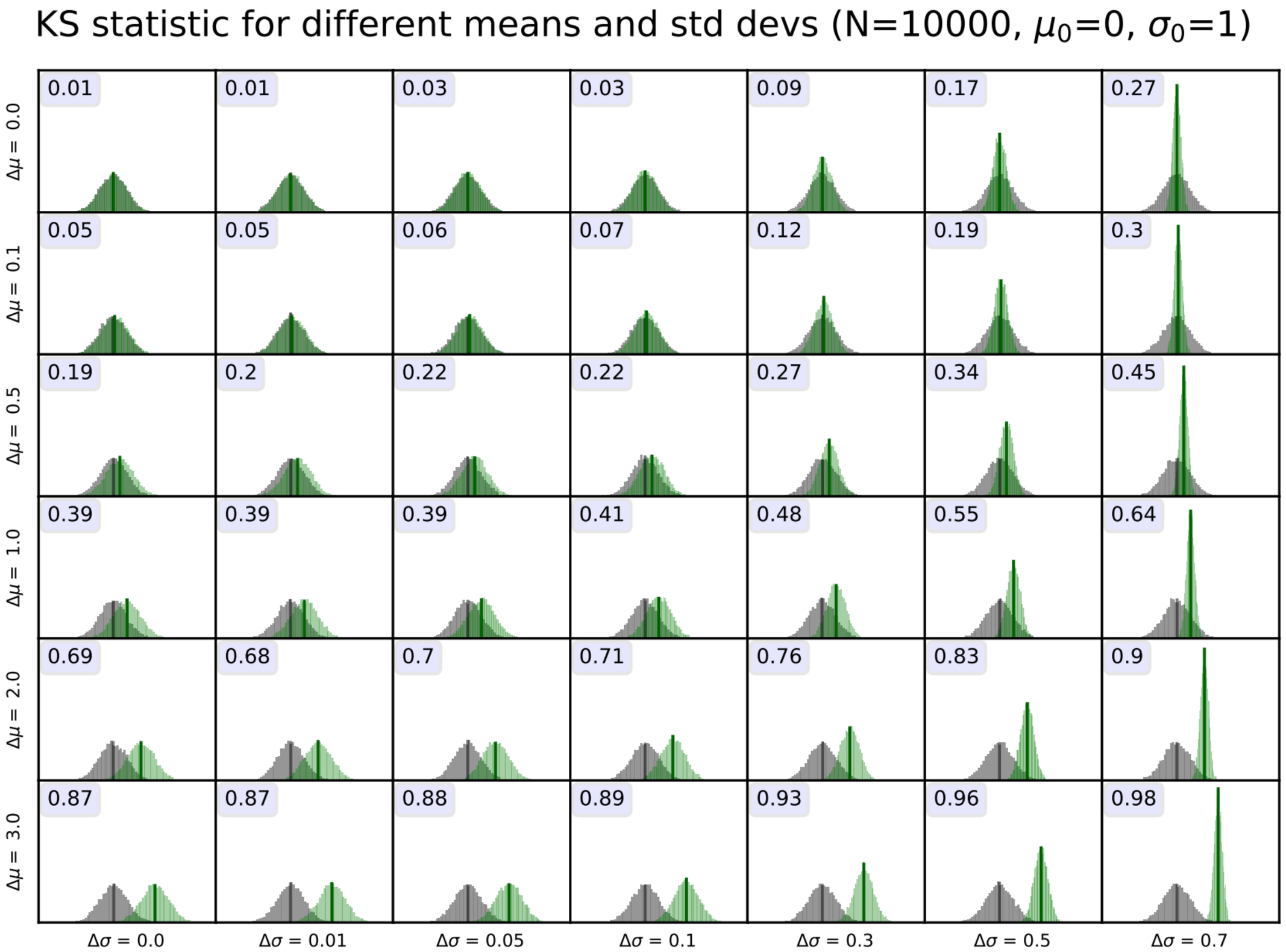
EDF sample 1 (n elements)

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



Generative Models

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

discriminative models: tell part muffins from chihuahuas

generative models: determine the probability distribution associated to "muffin" …

Generative Models

Q: what is the key property of **generative models** which makes them distinct from **discriminative models?**

discriminative models

$$p\left(\mathscr{C}_k \mid \boldsymbol{x}\right)$$

probability that instance x belongs to class C_k

generative models

$$p\left(\boldsymbol{x} \mid \mathscr{C}_k\right)$$

probability distribution of x associated to the class C_k

What we can do with generative models that is not possible with discriminative 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 $p(\mathscr{C}_k|\boldsymbol{x})$
- 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

Here discuss **generative models** which aim to model the **distribution in the space of inputs *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(**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 ***{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

$$p\left(\boldsymbol{x} \mid \mathscr{C}_k\right)$$ → ***probability that given a class C_k the instance x is found in it***

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

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

$$p\left(\mathscr{C}_k \mid \boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x} \mid \mathscr{C}_k\right) p(\mathscr{C}_k)}{p(\boldsymbol{x})}$$

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

$$p(\boldsymbol{x}) = \sum_k p\left(\boldsymbol{x} \mid \mathscr{C}_k\right) p(\mathscr{C}_k)$$ → ***probability to generate the instance x***

Generative Models

$$p\left(\mathscr{C}_k \mid \boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x} \mid \mathscr{C}_k\right) p(\mathscr{C}_k)}{p(\boldsymbol{x})}$$

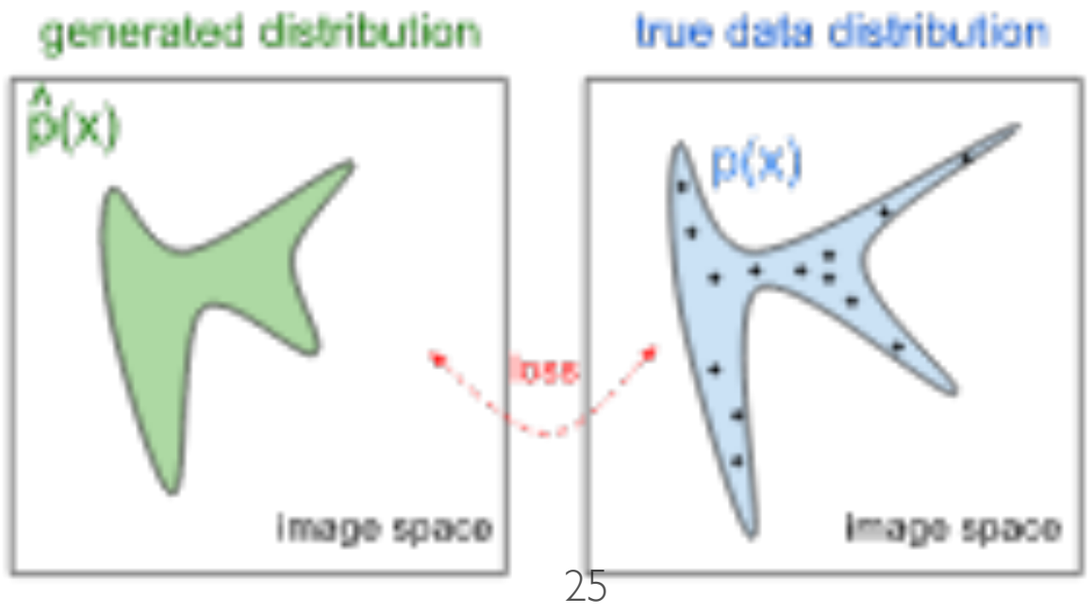
probability that x belongs to class C_k

$$p(\boldsymbol{x}) = \sum_k p\left(\boldsymbol{x} \mid \mathscr{C}_k\right) p(\mathscr{C}_k)$$

probability to generate the instance x

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

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



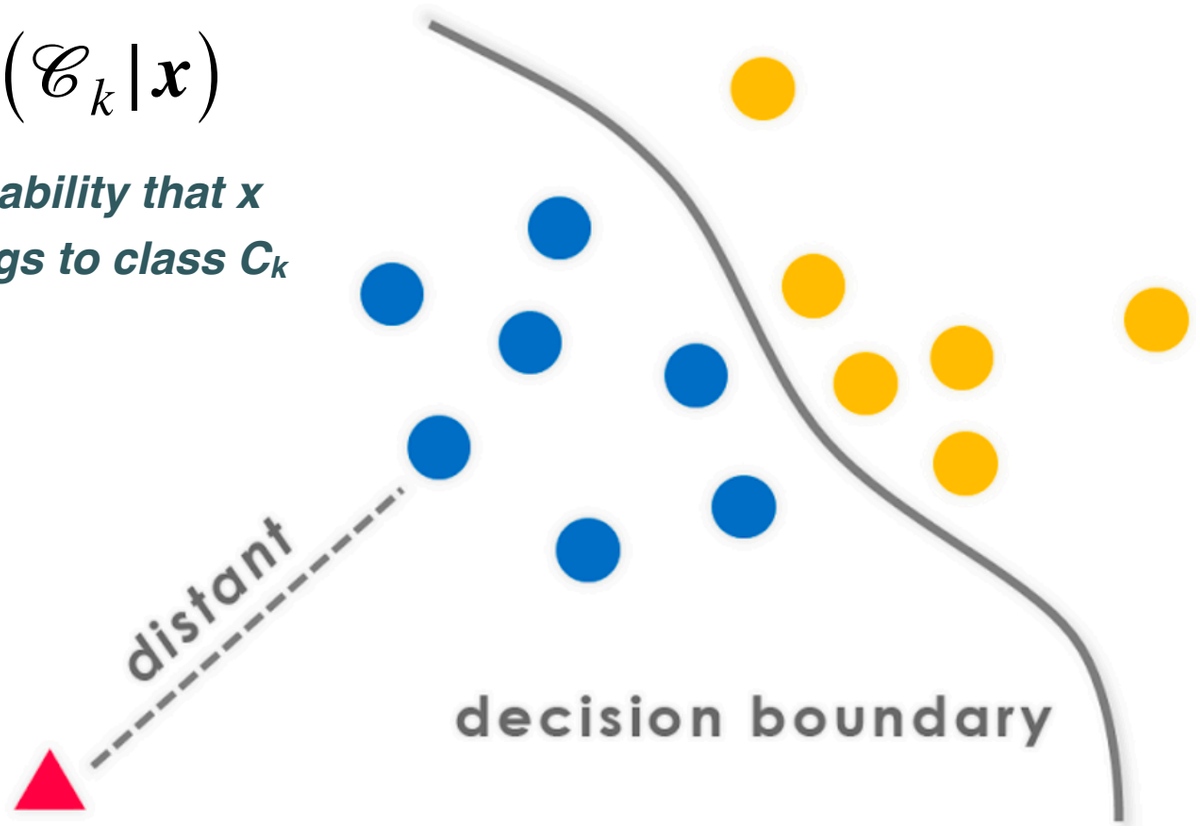
Generative Models

Compare graphically Discriminative and Generative Models for classification

Discriminative

$$p\left(\mathscr{C}_k|\boldsymbol{x}\right)$$

probability that x belongs to class C_k

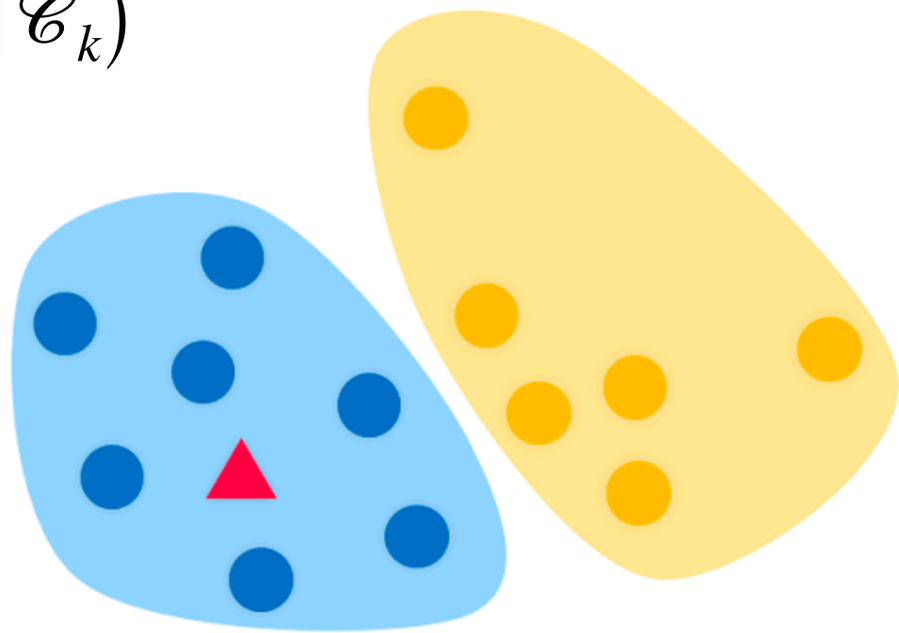


discriminative models works best when new instances are far from the decision boundary

Generative

probability of x in class C_k

$$p\left(x|\mathscr{C}_k\right)$$



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

$$p\left(\mathscr{C}_k|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}|\mathscr{C}_k\right)p(\mathscr{C}_k)}{p(\boldsymbol{x})}$$

$$p\left(\mathscr{C}_1|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}|\mathscr{C}_1\right)p(\mathscr{C}_1)}{p\left(\boldsymbol{x}|\mathscr{C}_1\right)p(\mathscr{C}_1) + p\left(\boldsymbol{x}|\mathscr{C}_2\right)p(\mathscr{C}_2)}$$

which can be simplified as

$$p\left(\mathscr{C}_1|\boldsymbol{x}\right) = \frac{1}{1+\exp(-a)} = \sigma(a) \qquad a = \ln\frac{p\left(\boldsymbol{x}|\mathscr{C}_1\right)p(\mathscr{C}_1)}{p\left(\boldsymbol{x}|\mathscr{C}_2\right)p(\mathscr{C}_2)}$$

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**

$$p\left(\mathscr{C}_k|\boldsymbol{x}\right) + \text{decision theory} \rightarrow \text{class boundaries}$$

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

$$p\left(\boldsymbol{x}\right) \rightarrow \text{generate new instances of } \boldsymbol{x}$$

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

$$p\left(\mathscr{C}_k|\boldsymbol{x}\right) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \qquad a_k = \ln\left(p(\boldsymbol{x}|\mathscr{C}_k)p(\mathscr{C}_k)\right)$$

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

$$p_\theta(\boldsymbol{x}) \qquad p_{\text{data}}(\boldsymbol{x})$$

from model — *from training data*

we aim to construct p_θ to model p_{data}

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

$$D_{KL}\left(p_{\text{data}}||p_\theta\right) = \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}$$

$$= \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_{\text{data}}(\boldsymbol{x}) - \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_\theta(\boldsymbol{x})$$

$$= S_p[p_{\text{data}}] - \left\langle \log p_\theta \right\rangle_{\text{data}}$$

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

$$\left\langle \log p_\theta(\boldsymbol{x}) \right\rangle_{\text{data}} = S_p[p_{\text{data}}] - D_{KL}\left(p_{\text{data}} \,||\, p_\theta\right)$$

Log-likelihood of data under model

entropy of data: independent of model parameters

KL-divergence

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

Adversarial Learning

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

(1) $D_{KL}\left(p_{\text{data}} || p_\theta\right)$ → ***(draw values of θ accordingly to p_θ and compare with data)*** → ***Calculable using sampling***

(2) $D_{KL}\left(p_\theta || p_{\text{data}}\right)$ → ***Large when model over-weights low-density regions near real peaks***

→ ***but not calculable since p_{data} unknown ….***

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

Adversarial Learning

Q: do you think that Adversarial Learning belongs to the family of **Supervised** or of **Unsupervised** machine learning algorithms?

Adversarial Learning

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

(1) $D_{KL}\left(p_{\text{data}}||p_\theta\right)$ → ***(draw values of θ accordingly to p_θ and compare with data)*** → ***Calculable using sampling***

(2) $D_{KL}\left(p_\theta||p_{\text{data}}\right)$ → ***Large when model over-weights low-density regions near real peaks***

→ ***but not calculable since p_{data} unknown ….***

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

Adversarial Learning is a type of **Unsupervised Machine Learning**: there is **no cost function** and the examples are **unlabelled**: the task here is to learn how to **generate new examples** using the existing ones as input!

Generative adversarial networks

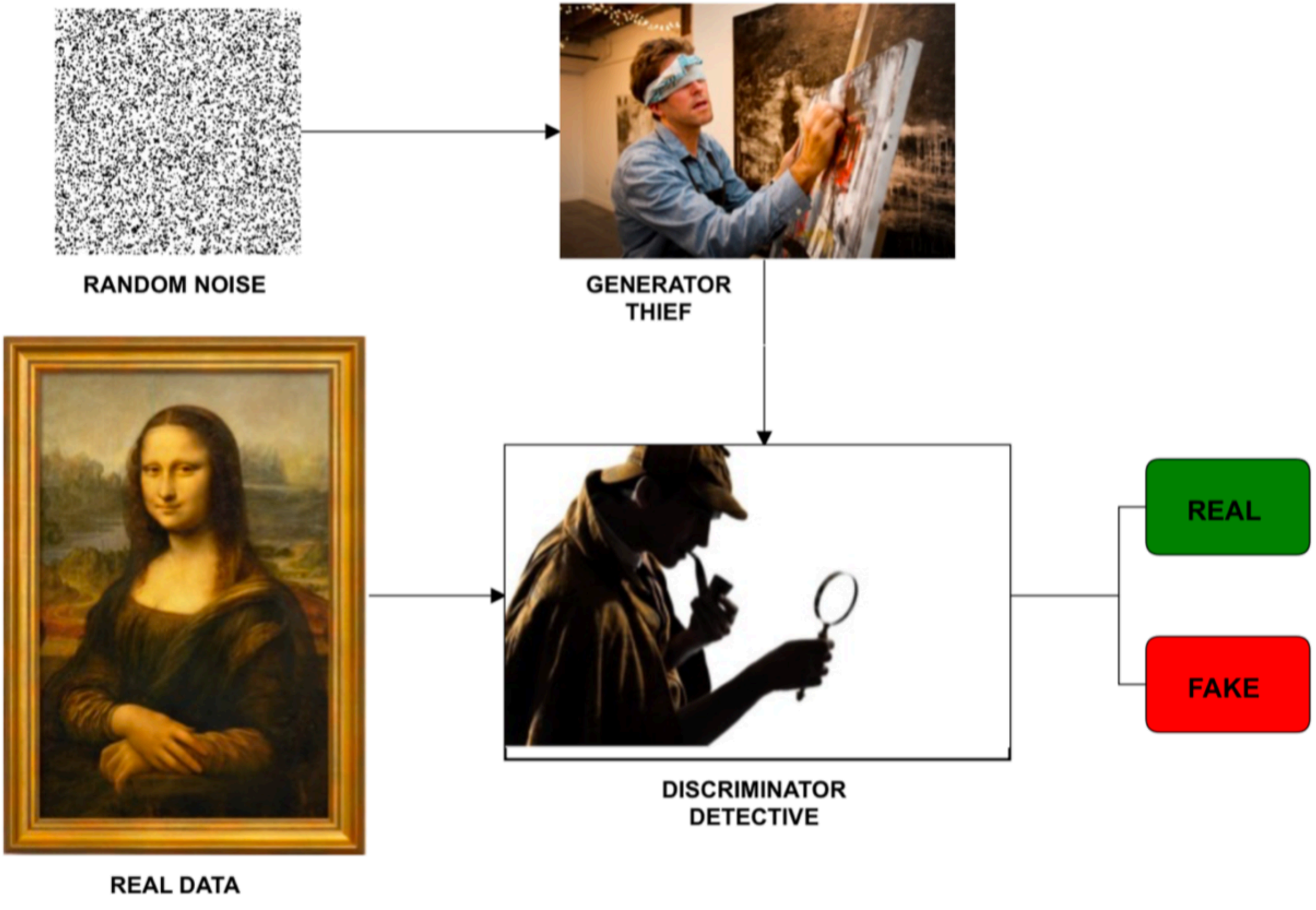
Generative Adversarial Networks (GANs) are deep neural network architectures, composed by two independent NNs which **compete against each other**

- ***(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**

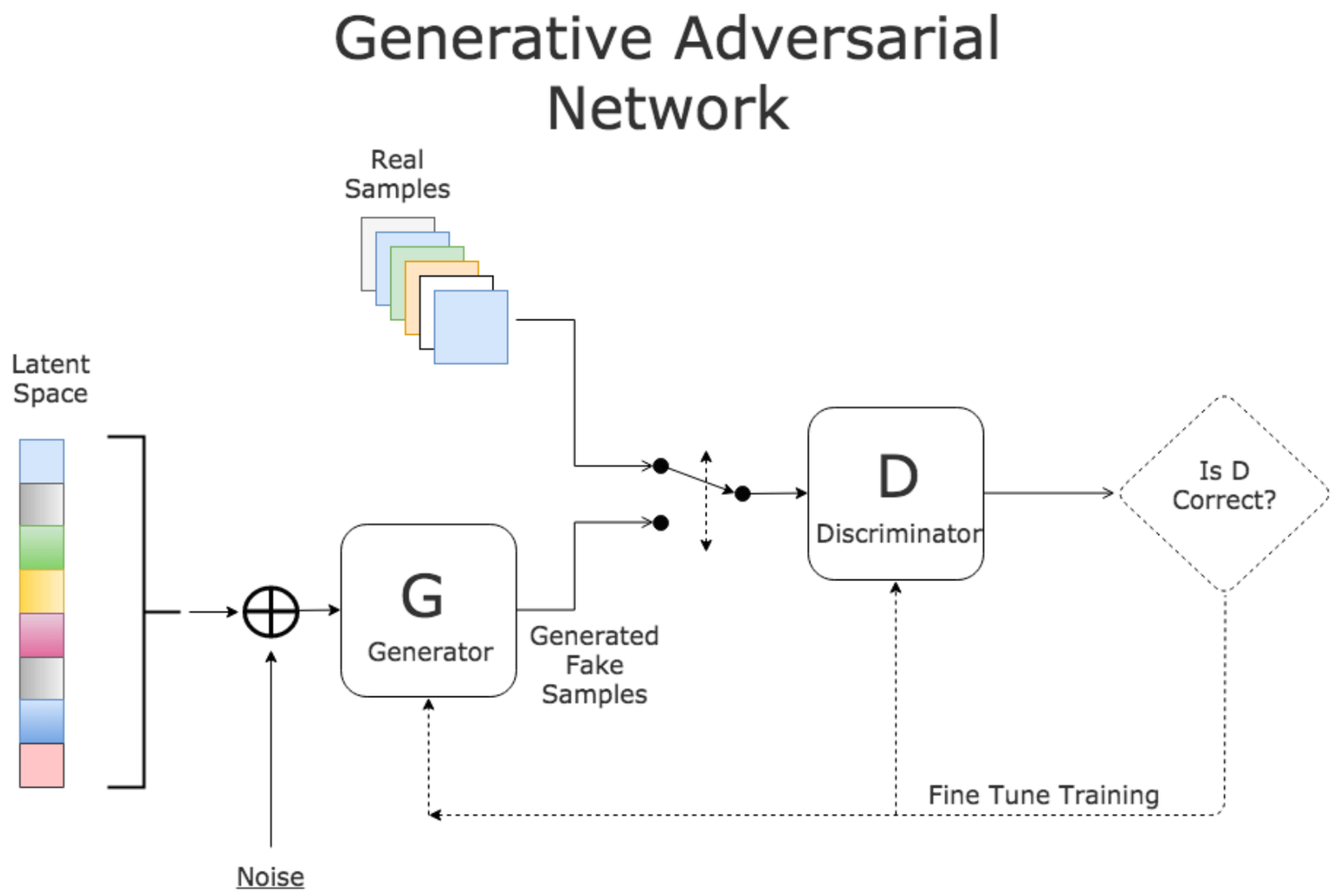
the generator network **G** should be trained to maximise the probability that the discriminator network **D** makes a mistake: that is, **G** should generate pseudo-data samples that are virtually **indistinguishable** from the actual data

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.



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

$$\{\boldsymbol{x}_n\}_{n=1}^{N} \qquad \boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \ldots, x_{n,p})$$

p = number of features per sample

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

$$\{\boldsymbol{z}_n\}_{n=1}^{N} \qquad \boldsymbol{z}_n = (z_{n,1}, z_{n,2}, \ldots, z_{n,p})$$

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

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

NN params of D ***NN params of G*** ***output of D when input a real data sample*** ***output of D when input 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**

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

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

$$\mathbf{v}_t = \eta_t \nabla_{\theta_D} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_{G,t}), \qquad \boldsymbol{\theta}_{D,t+1} = \boldsymbol{\theta}_{D,t} - \mathbf{v}_t$$

- 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** $\{\boldsymbol{z}_n\}_{n=1}^{N}$

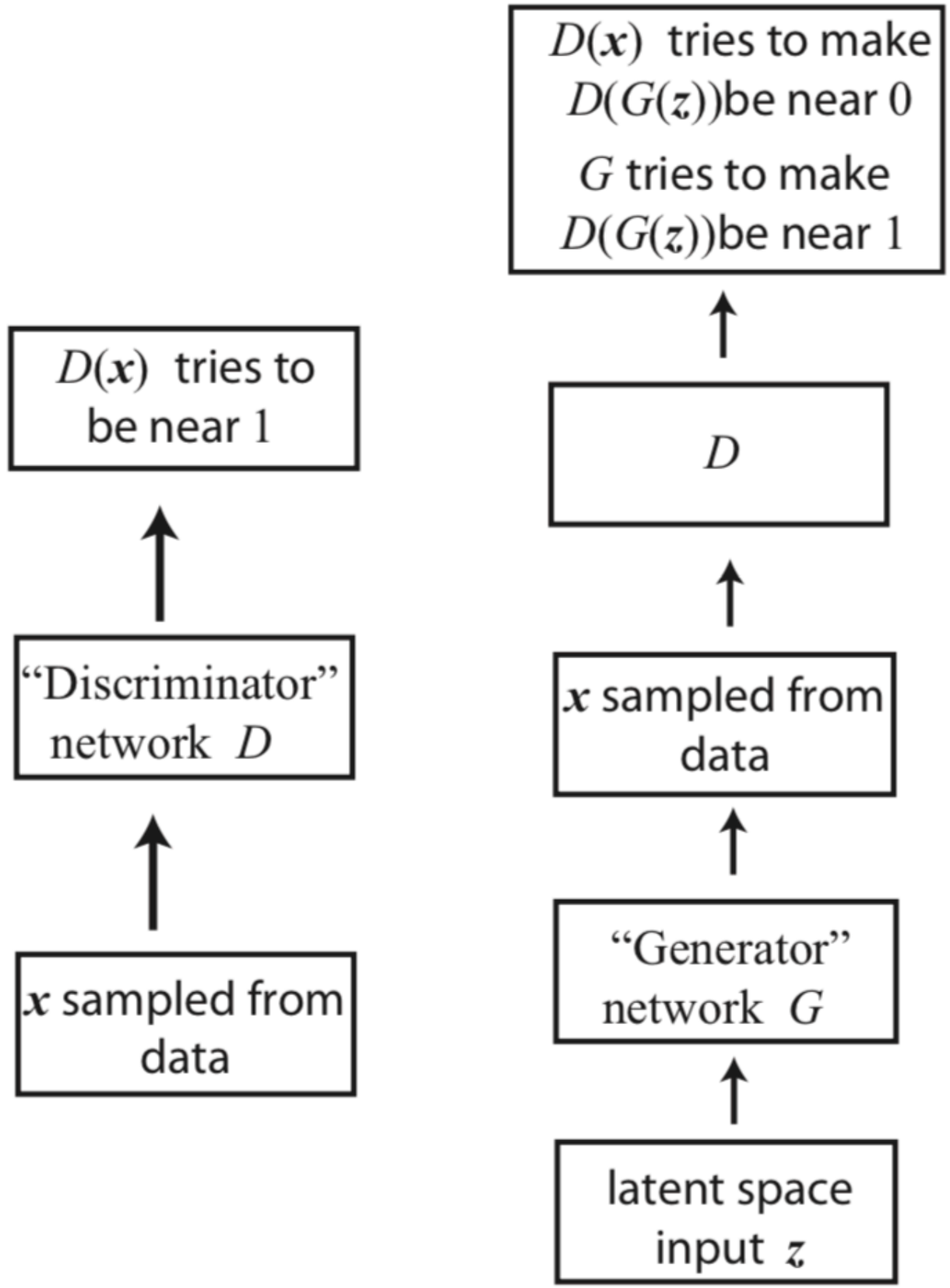
$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\log(1 - D(G(\boldsymbol{z}_i)))$$

output of D (now with its parameters fixed)

$$\mathbf{v}_t = \eta_t \nabla_{\theta_G} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_G), \qquad \boldsymbol{\theta}_{G,t+1} = \boldsymbol{\theta}_{G,t} - \mathbf{v}_t$$

GAN training

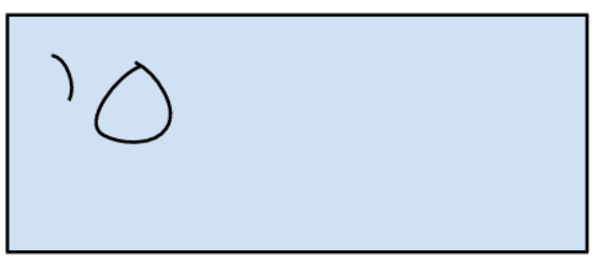
the generator and discriminator are sequentially trained and iterated until convergence is achieved, at this point **D** cannot tell apart the pseudo-data from **G** from the real data



GAN training

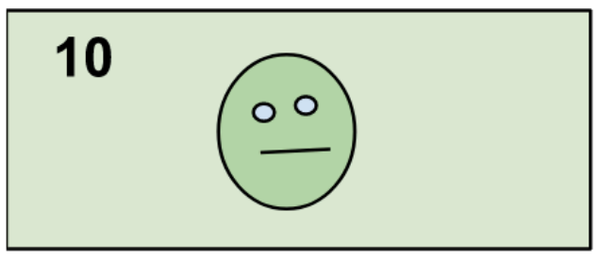
initial training

Generated Data | **Discriminator** | **Real Data**



FAKE REAL

intermediate training



FAKE REAL

long training

REAL REAL

convergence!!

Image generation with GANs

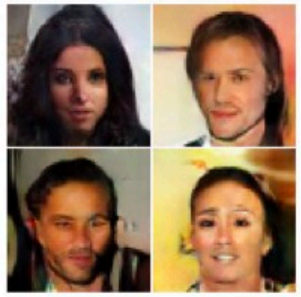
https://thispersondoesnotexist.com/

Image generation with GANs

https://thispersondoesnotexist.com/

Image generation with GANs

DCGAN
11/2015



EBGAN-PT
9/2016

BEGAN
3/2017
128 × 128

Progressive GAN
10/2017
1024 x 1024

horse → zebra

Image generation with GANs

https://deepdreamgenerator.com

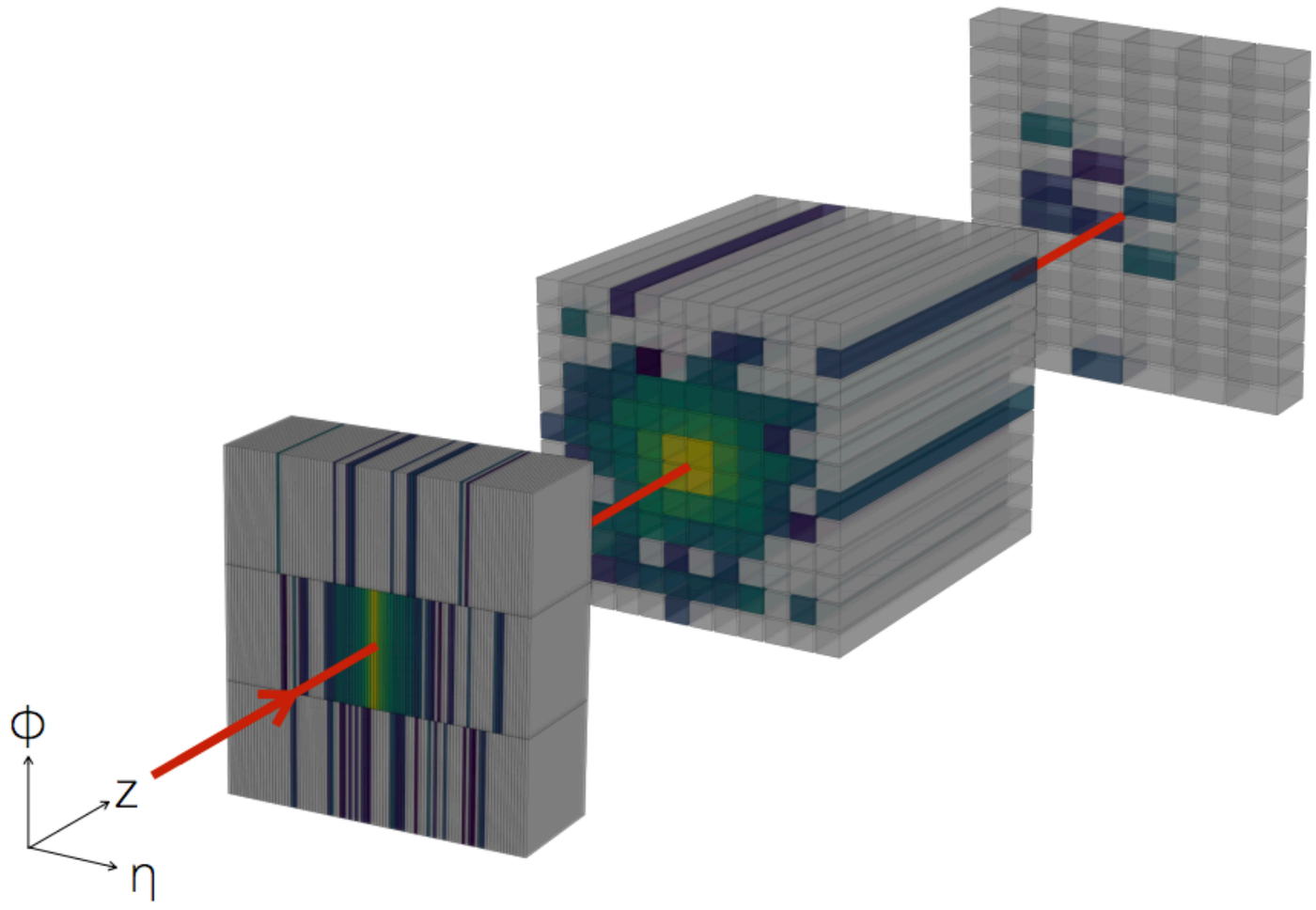
Generative Adversarial Networks

GANs for detector simulation in HEP

- Modelling accurately the response of detectors with the **propagation of high energy particles** is an essential task for present and future HEP experiment

- **Detector simulation at the LHC** is a very CPU-intensive task, dominated by modelling of particle showers inside calorimeters

- **Generative Adversarial Networks** can speed up detector simulation by orders of magnitude



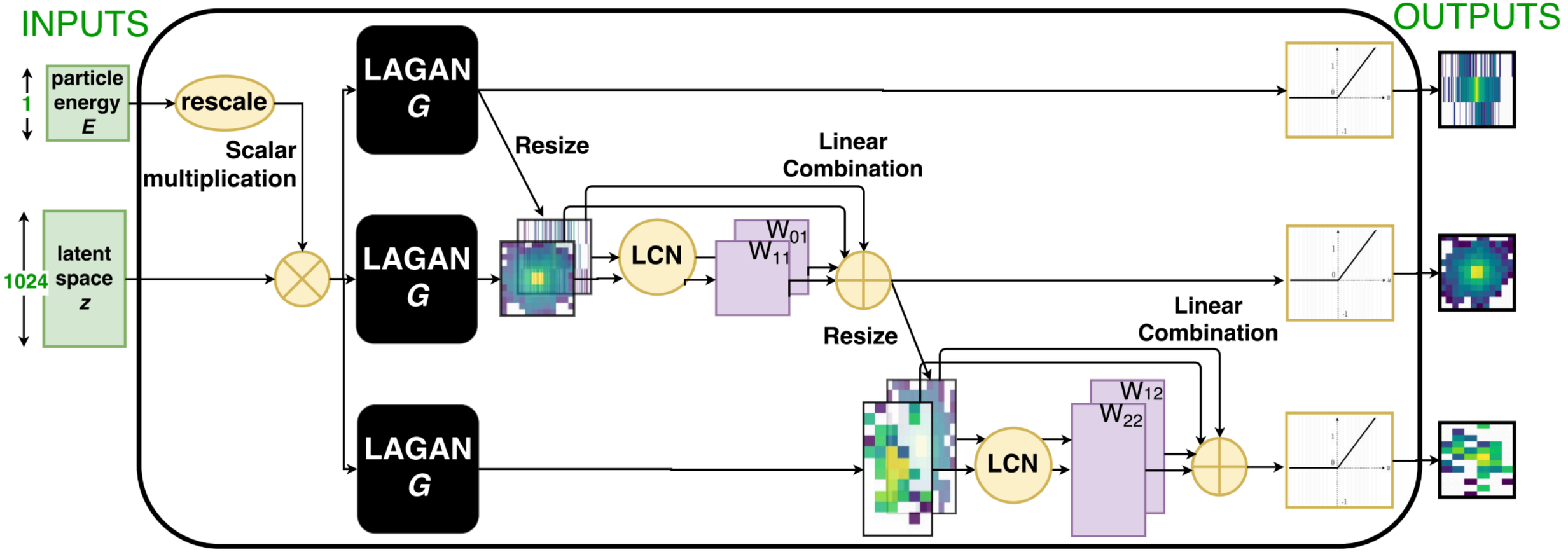
*Task: to efficiently model the **propagation of high energy particles** (and their interaction) within the layers of electromagnetic and hadronic calorimeters*

GANs for detector simulation in HEP

- Use GANs as a tool to **speed up full simulation of particle showers** in a HEP calorimeter
- The generator G learns a map from **a latent space** to space of **generated samples** for training
- Carefully understanding the **underlying physics of particle propagation** in a detector is crucial to optimise the training strategy, *e.g.* relationships between neighbouring detector layers

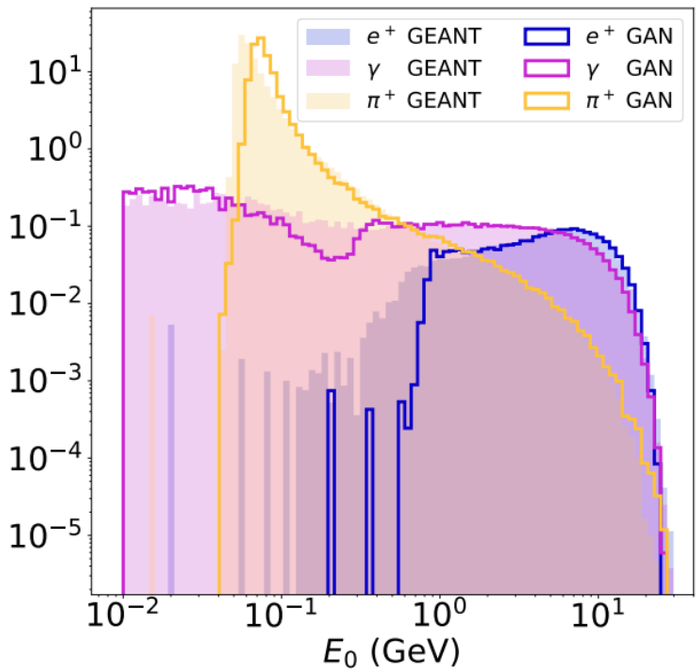
Generator Network

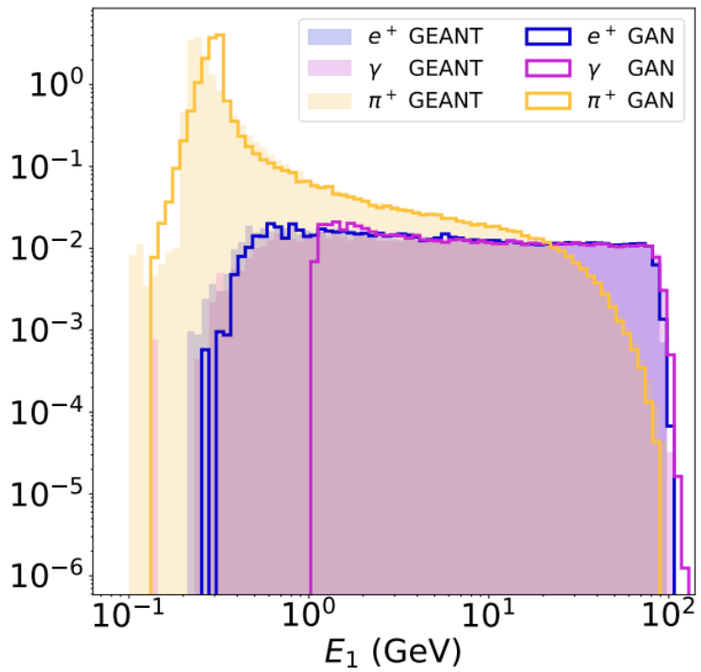
Output: ***generated samples*** *to be used for training D*

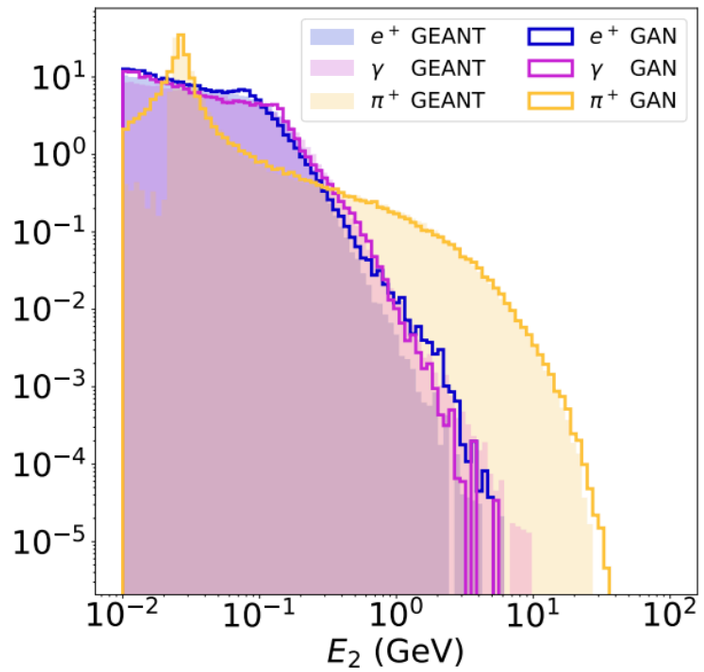


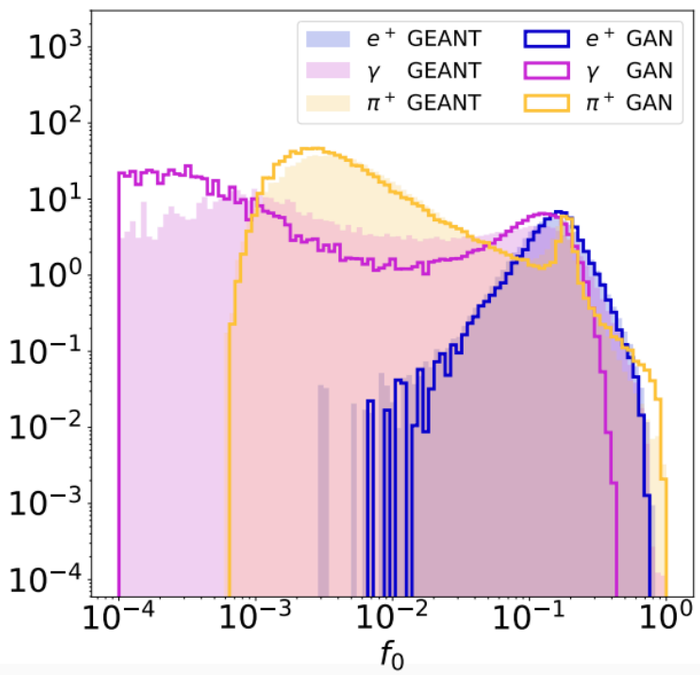
Paganini et al. 17

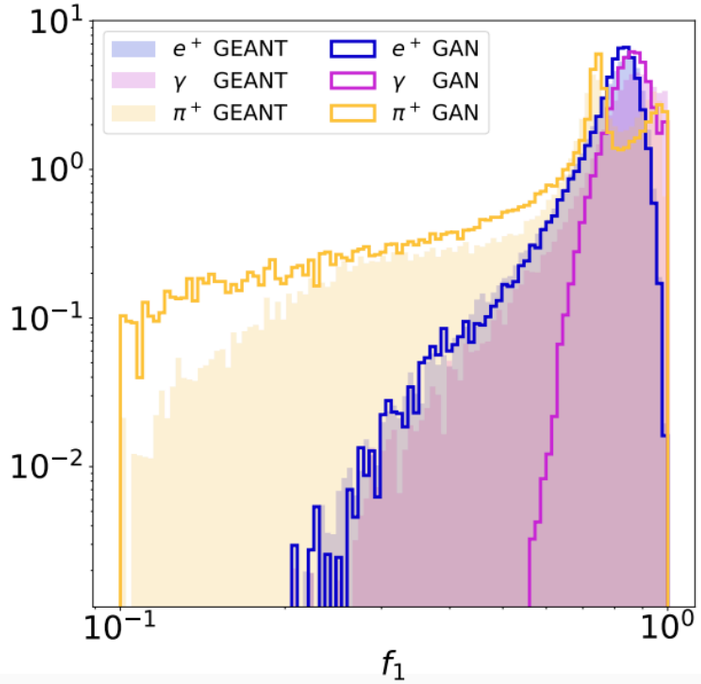
GANs for detector simulation in HEP

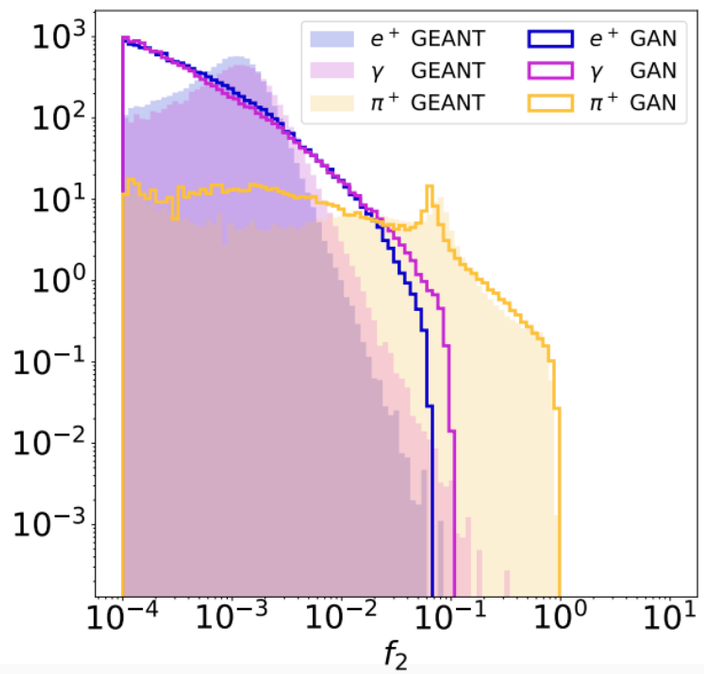










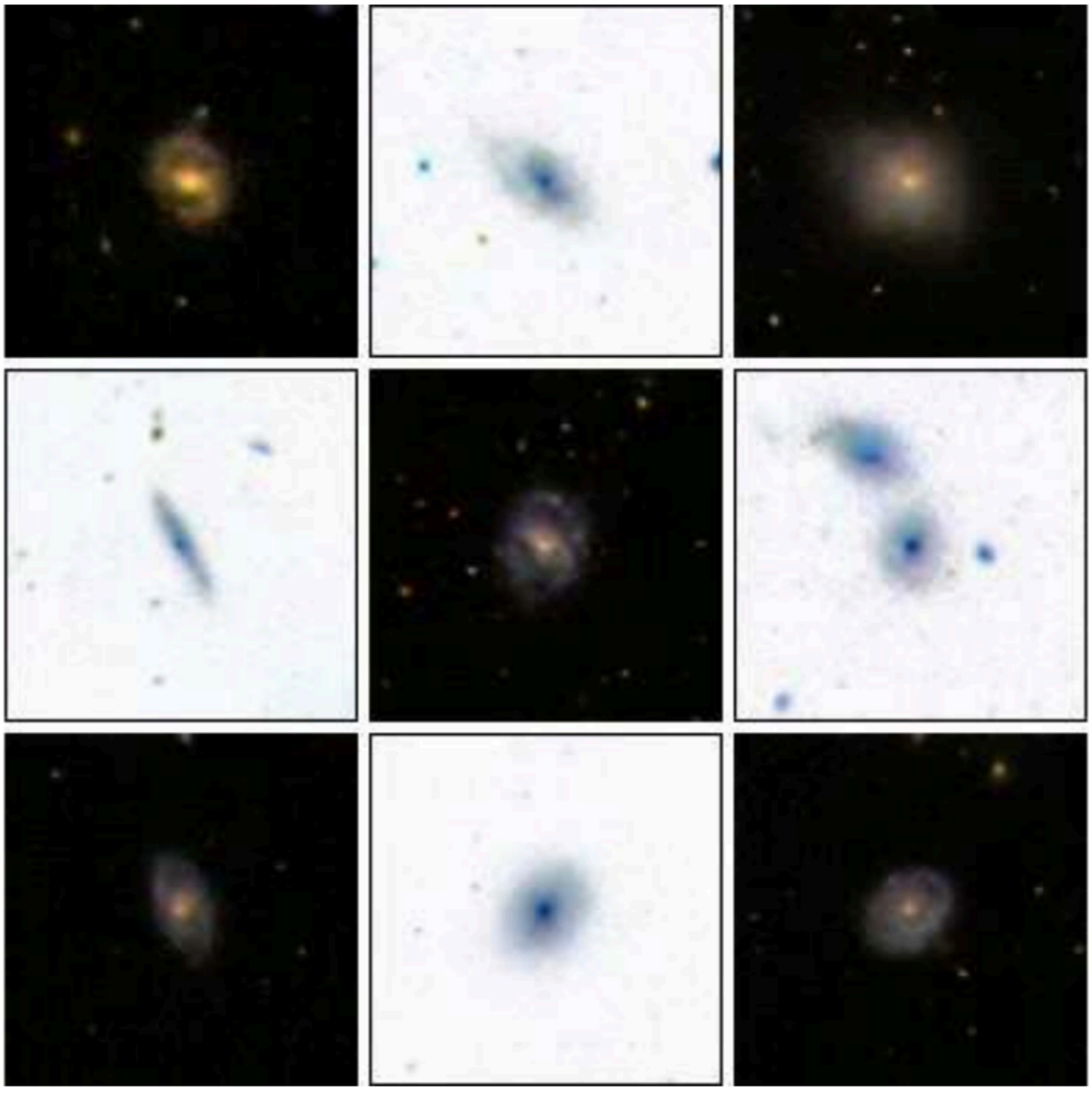


GANs for detector simulation in HEP

Simulator	Hardware	Batch Size	ms/shower
GEANT4	CPU	N/A	1772
CALOGAN	CPU	1	13.1
		10	5.11
		128	2.19
		1024	2.03
	GPU	1	14.5
		4	3.68
		128	0.021
		512	0.014
		1024	0.012

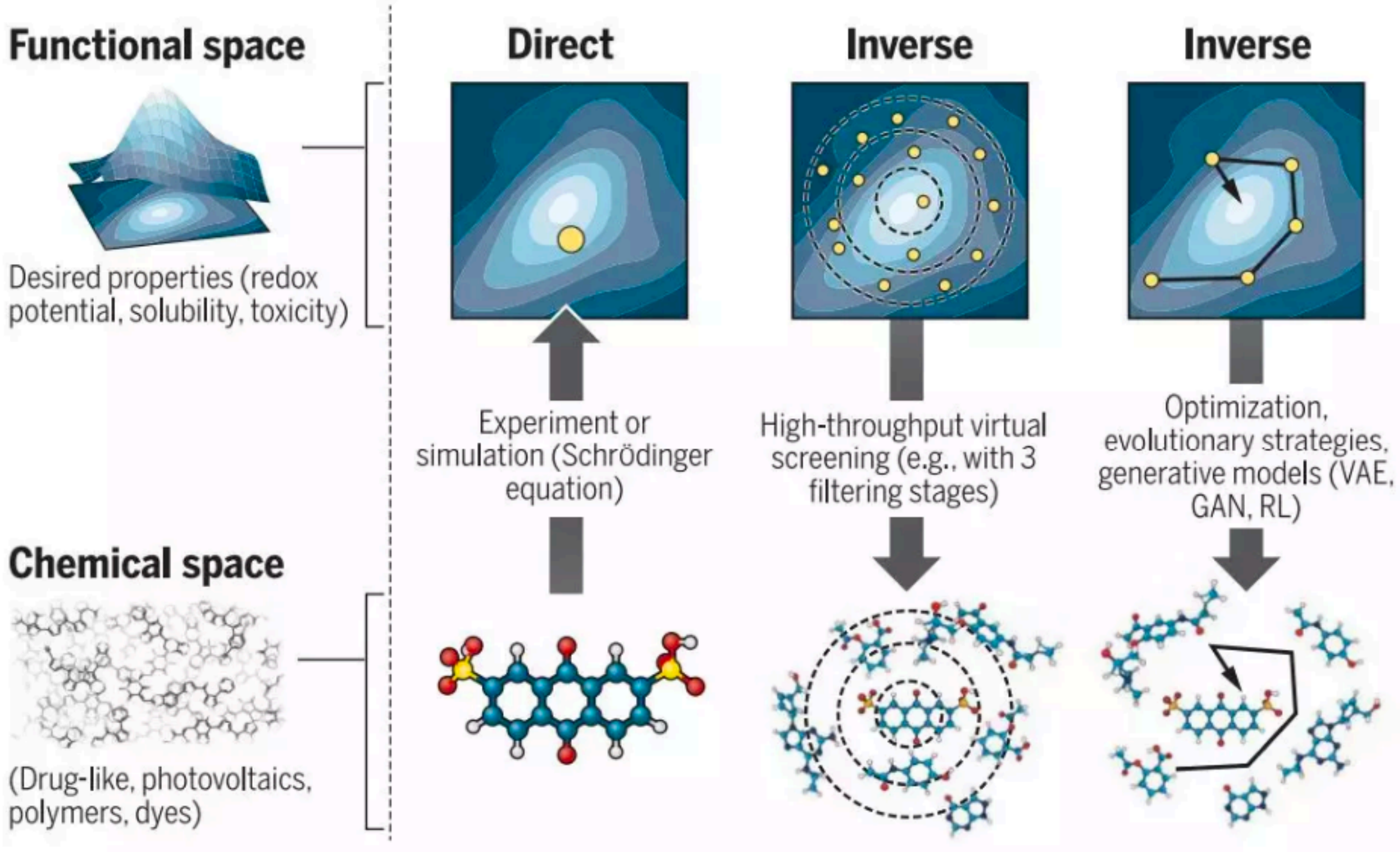
Speed-up by several orders of magnitude, specially when running in GPUs

Synthetic Galaxies from GANs



Useful for applications where a large number of synthetic images of galaxies speed up analysis

GANs in chemistry



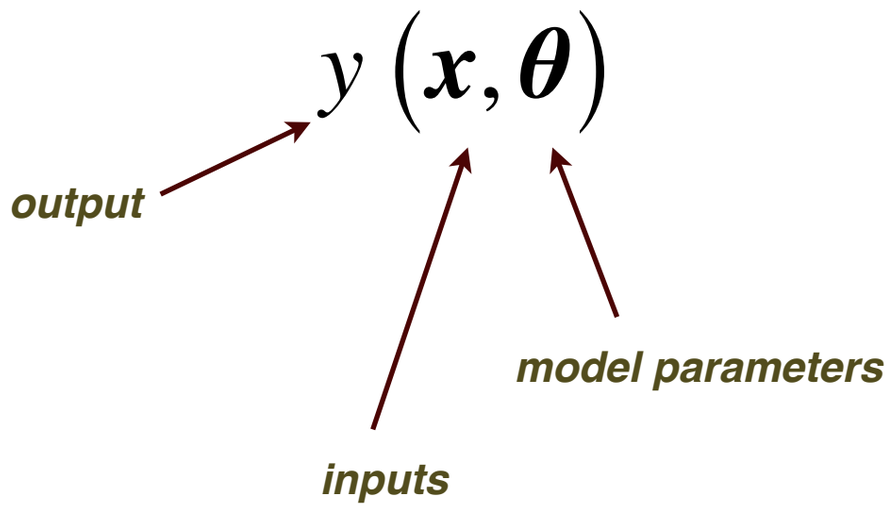
Start from molecule, evaluate space of properties

Start from space of properties, design new molecules

Kernel Methods

Kernel Methods

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs



after the training, the input examples are **discarded** and not relevant for subsequent **predictions**: the information contained in the original training examples is now **entirely contained** on the model parameters

Kernel Methods

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs

$$y\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$$

info on training samples entirely contained in model params

were the **training examples are discarded** after the model parameters (or their posterior distribution) have been determined

here we introduce models where the training dataset is also used in the **prediction phase**

Feature space mapping

A key concept of **kernel methods** is that of a **feature space mapping**

This is a transformation, in general non-linear, between the **input data space *X*** and a **feature space *F*** of the same or different dimensions:

$$F = \{\phi(x) : x \in X\}$$

Example of linear feature-space transformation

x	$\phi(x)$
Height	*Height+Weight*
Weight	*Weight-Height*
Age	*Age + 2*Weight*

Example of non-linear feature-space transformation

x	$\phi(x)$
Height	*Height/Weight*
Weight	*Weight+Height*
Age	*Age * Weight*

such mapping can be chosen **to increase the efficiency** of the ML model training

note that the dimensions of feature space can be much larger than of data space

Feature space mapping

most of the ML models we have considered for regression and classification are based on the **parametric mapping** (linear or non-linear) between inputs and outputs

$$y\left(\boldsymbol{x}, \boldsymbol{\theta}\right)$$

info on training samples entirely contained in model params

were the **training examples are discarded** after the model parameters (or their posterior distribution) have been determined

here we introduce models where the training dataset is also used in the **prediction phase**

the dimensions of the **feature space mapping** are in general different from those of input space

$$\boldsymbol{x} \longrightarrow \boldsymbol{\phi}\left(\boldsymbol{x}\right)$$

Height

Weight

Age

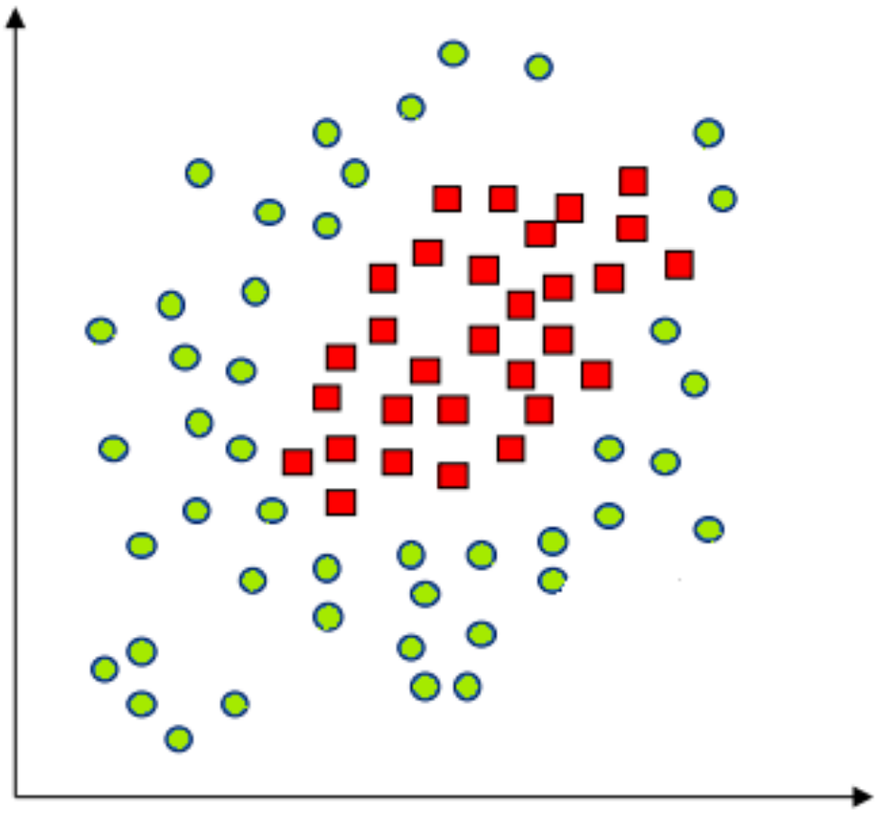
Height/Weight

Weight + Height

Age * Weight

Height * Weight+Age

Feature space mapping

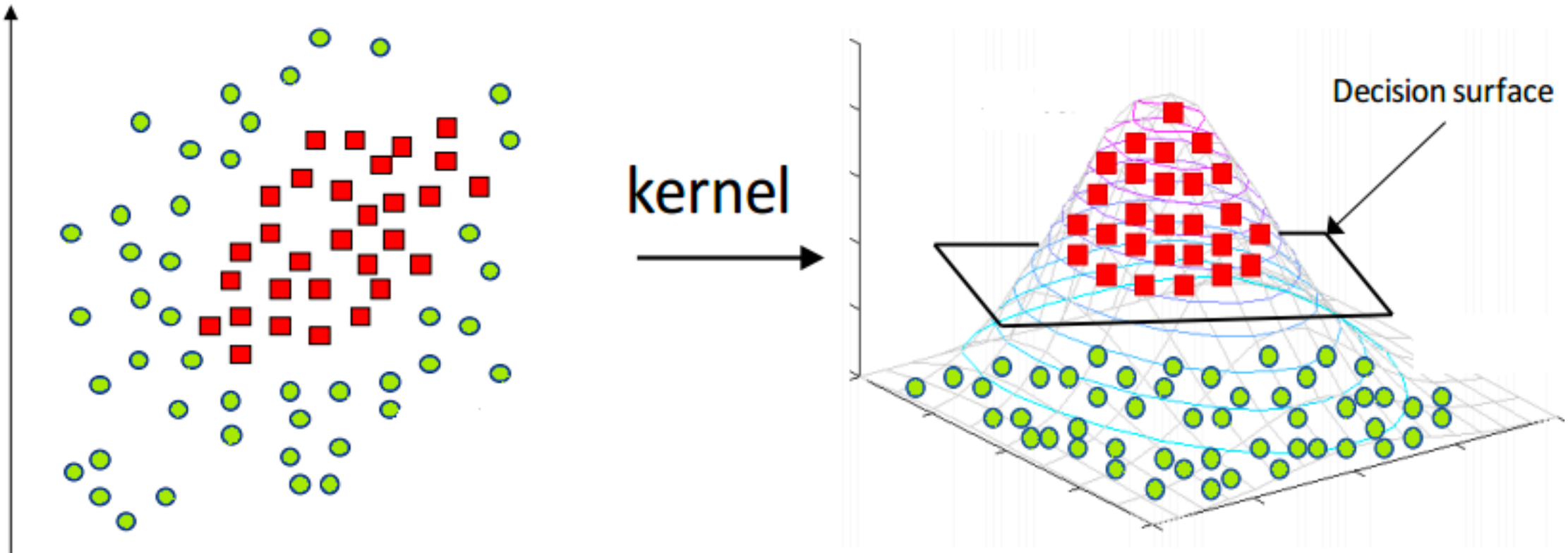


Data Space (p=2):

linear classification impossible

Q: can we go to a higher-dimensional ``feature space'' where linear classification is possible?

Feature space mapping



Data Space (p=2):

linear classification impossible

Feature space (p=3):

linear classification possible!

Kernel Methods

the key ingredient of **kernel methods** is the **kernel function** evaluated at the training points

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

inner product in feature space

kernel function (scalar)

feature space mapping

the simplest kernel is the **linear kernel** (identity mapping in feature space)

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}' = \sum_{k=1}^{p} x_i x_i'$$

sum over p,

dimension of input data space

Kernel Methods

in kernel methods, the training dataset is also used in the **prediction phase**

The advantages of kernel methods arise when choice special, non-linear kernels

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}')$$

for example, assume a two-dimensional problems (***p=2***) with the following feature mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1 x_2, x_1 - x_2\right)$$

then the corresponding kernel will be given by

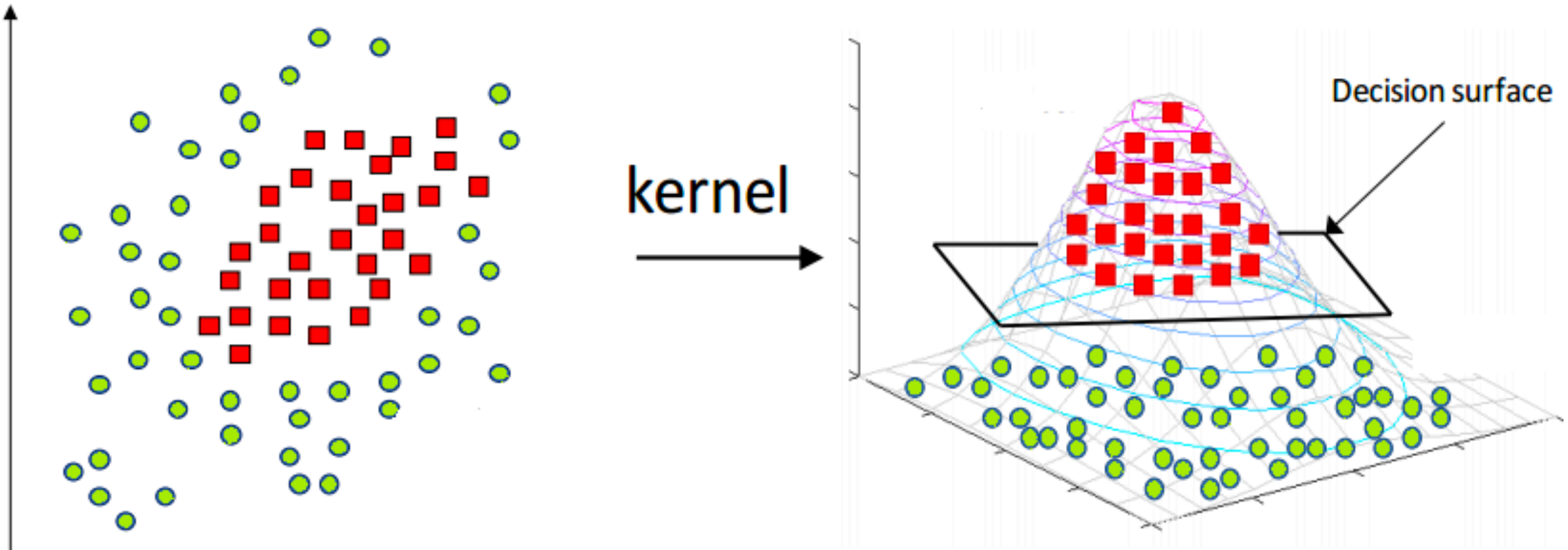
$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}') = x_1 x_2 x_1' x_2 + (x_1 - x_2)(x_1' - x_2')$$

as we will see, kernel methods exploit the **kernel trick,** whereby they can work directly at the level of kernels **without the need to specify the feature space mapping**

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p,** but classification requires a feature space of **dimension p' > p**



Data Space (p=2):

linear classification impossible

Feature space (p=3):

linear classification possible!

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p=3,** but classification requires a feature space of **dimension p=9**

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2\right)^T$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y}) = \sum_{i,j=1}^{3} x_ix_jy_iy_j$$

scales as $O(n^2)$

Kernel Methods

Why working with kernels is advantageous?

Assume we work in a data space of **dimension p=3,** but classification requires a feature space of **dimension p=9**

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, x_1x_2, x_1x_3, x_2x_1, x_2^2, x_2x_3, x_3x_1, x_3x_2, x_3^2\right)^T$$

$$\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y}) = \sum_{i,j=1}^{3} x_ix_jy_iy_j$$

scales as O(n²)

The same result can be obtained working with an appropriate **kernel**

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y})^2 = \left(x_1y_1 + x_2y_2 + x_3y_3\right)^2 = \boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{y})$$

scales as O(n)

efficient and **computationally advantageous** method to transform data to higher dimensions!

Kernel Methods

Kernel methods are used in a **wide variety of Machine Learning** applications:

- Pattern analysis
- Clustering
- Dimensional reduction
- Classification
- Anomaly detection

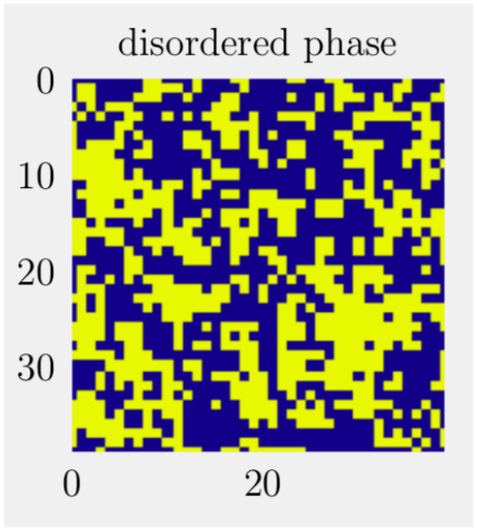
A kernel is **a similarity function:** it measures how similar two inputs are

working with **kernels** offers many advantages over working with **feature vectors,** which in general have infinite dimensionality

Kernel Methods

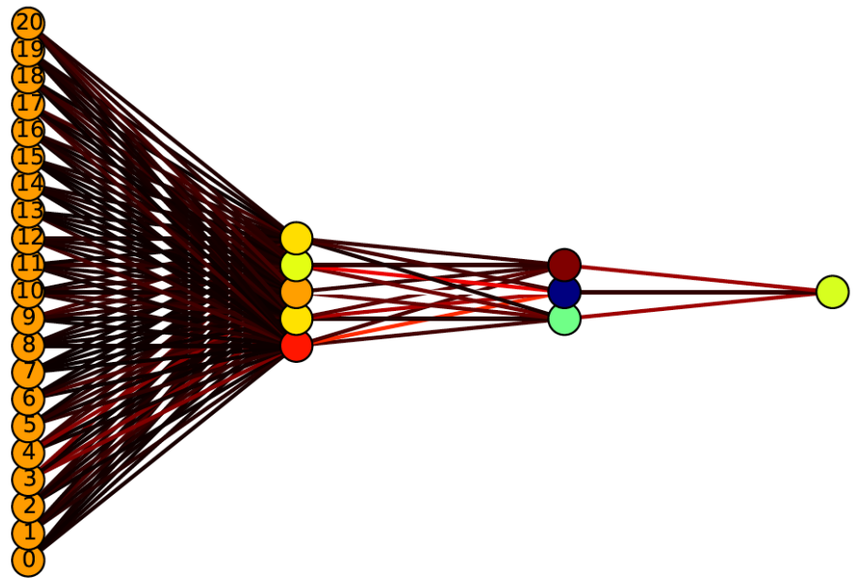
Classifier working on **feature space**

labelled training examples => Feature map => input to classifier



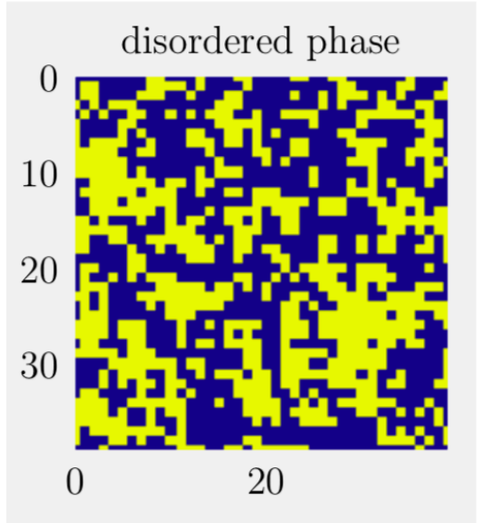
array with spin values

+ label indicating phase



Classifier working with **kernel method**

labelled training examples + kernel (similarity measure) => Classifier



$$k\left(\boldsymbol{x}, \boldsymbol{x}', \alpha, \beta, \ldots\right)$$

optimise hyper-params of the kernel

to achieve good classification

how to select a suitable kernel?

Support Vector Machines

Support Vector Machines

SVM are **kernel methods** popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**

The advantages of support vector machines include:

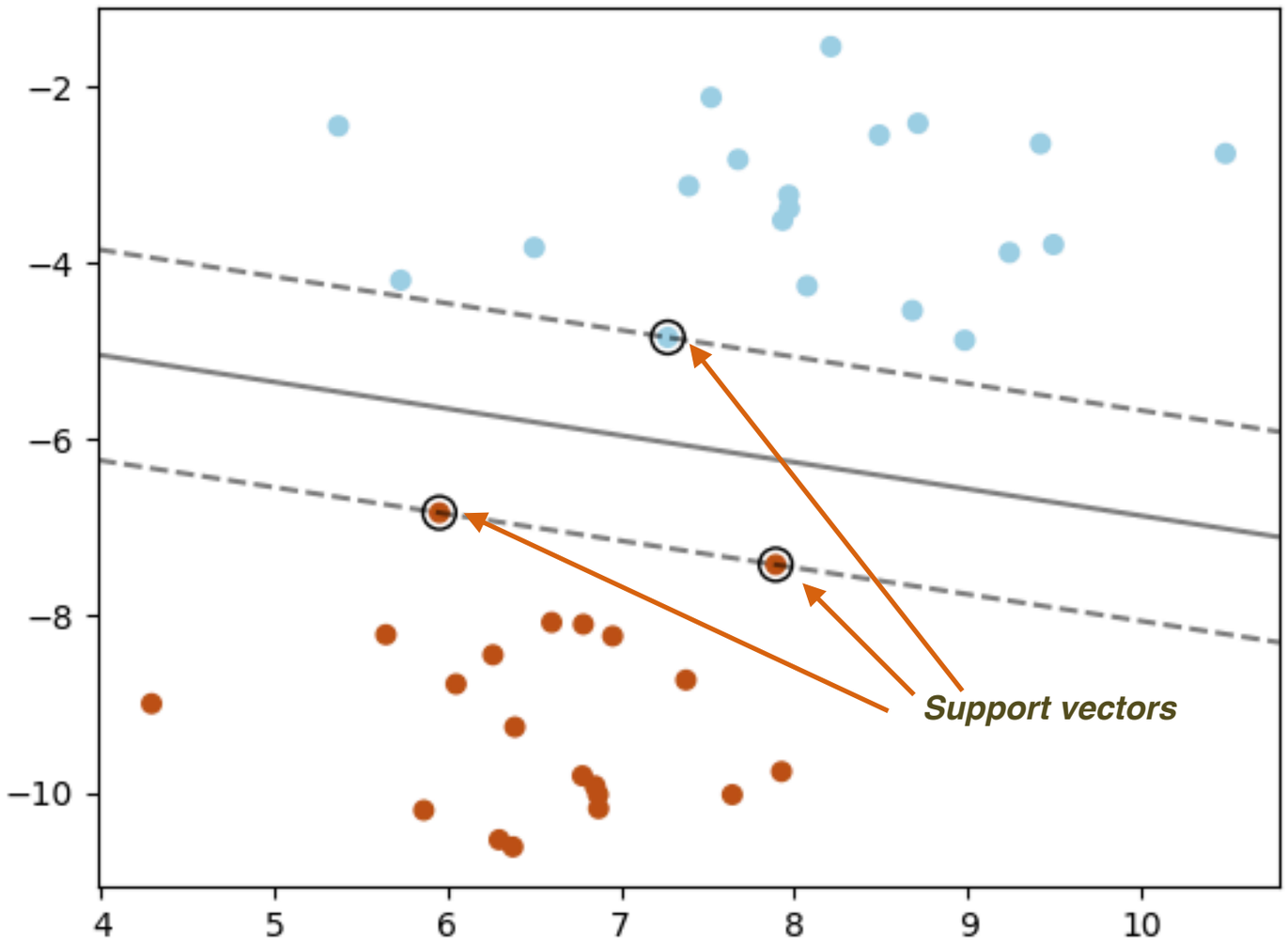
- Effective in **high dimensions.**
- Effective when number of dimensions is greater than number of samples.
- Includes on a **subset of training points** in the decision function (**support vectors**)
- Different **kernels functions** can be used for the figure of merit

kernel methods operate in a high-dimensional,
implicit feature space without computing the coordinates in data space

Support Vector Machines

SVM are ML algorithms popular for classification, regression, and novelty detection

A peculiar feature of SVM is that the model parameters are found by a solution of a convex optimisation problem: any local solution is also a **global optimum**



In SVM classification, the support vectors are samples that lie in the margin boundaries (at least for linearly separable problems)

Support Vector Machines

As other **classification algorithms**, the problem that SVM try to solve is the following. Starting from a training dataset

$$\{\boldsymbol{x}_i^T, t_i\}, \quad i = 1,\ldots,N, \quad t_i \in (-1,1)$$

binary classification

our goal is to find the **model parameters** such that the prediction from

$$\text{sign}(\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}) + b)$$

linear in model parameters, but non-linear in feature space

is correct for most of the samples. Here $\boldsymbol{\phi}$ is the **feature-space transformation**, where the *feature space* is where our data lives

Example of a feature-space transformation

$\boldsymbol{x}$	$\boldsymbol{\phi}(\boldsymbol{x})$
Height	***Height/Weight***
Weight	***Weight+Height***
Age	***Age * Weight***

Support Vector Machines

the problem then that SVM are aiming to solve is

inverse of margin between the two categories

$$\min_{\theta,b,\xi}\left(\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}+C\sum_{i=1}^{n}\xi_i\right)$$

subject to the **additional constraint** that

Class label of sample i → $t_i \times \left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_i)+b\right) \geq 1-\xi_i$ ← *tolerance (regulator)*

which essentially aims to **maximise the margin between the two categories** while including a penalty when when a sample is misclassified or within the margin boundary, where the team proportional to *C* plays the role of a **regulator**

> SVM are an example of a **constrained optimisation problem,** where one tries to minimise a quadratic function subject to **linear constraints.** They are most efficiently solved by means of the Lagrange multiplier method

Support Vector Machines

In which respect SVD is a **kernel method**? So far it seems to work only in **feature space …**

we can reformulate the problem such that the feature space disappears completely!

many ML models for regression and classification can be reformulated in a **dual representation** where the kernel functions arise naturally

in doing so, we will also show in which respect kernel methods are special in that they **explicitly include the training examples** in the final decision model

Dual representations

many ML models for regression and classification can be reformulated in a **dual representation** where the kernel functions arise naturally

consider a linear regression model with a regularised least-squares error function

$$E_{\mathrm{tr}}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right)^2 + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

regulator

inner product in feature space

sum over training samples

output from training examples

same structure as SVD!

the model parameters are determined **analytically** by requiring the vanishing of the gradient

$$\boldsymbol{\theta} = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x_n}), \quad a_n = -\frac{1}{\lambda}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\right)$$

recall that inner products take place in feature space

one can formulate a dual representation of the problem in terms of the **parameter vector**

$$\boldsymbol{a} = \left(a_1, a_2, \ldots, a_N\right)^T \qquad \boldsymbol{t} = \left(t_1, t_2, \ldots, t_N\right)^T$$

Dual representations

with some algebra one can show that the original figure of merit of the model

$$E_{\text{tr}}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}\left(\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right)^2 + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$$

admits a **dual representation** in terms of **kernel function**

model params nor feature space maps do not appear in this formulation

$$E_{\text{tr}}(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

where ***K*** is known as the **Gram matrix**, an *N x N* symmetric matrix with entries

$$K_{nm} = k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m)$$

matrix in space of input data

given by the **kernel function** evaluated over two of the **input training examples**

the crucial property of this dual representation is that now the predictions for new inputs will explicitly **depend on the training examples**

SVD can be expressed entirely in terms of kernels

Dual representations

the crucial property of this dual representation is that now the predictions for new inputs will explicitly **depend on the training examples**

$$y(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^T\left(\boldsymbol{K} + \lambda \boldsymbol{I}_N\right)^{-1}\boldsymbol{t}$$

output of trained model

new input

depends on the N original inputs

$$\boldsymbol{k}(\boldsymbol{x}) = \left(k(\boldsymbol{x}_1, \boldsymbol{x}), k(\boldsymbol{x}_2, \boldsymbol{x}), \ldots\right)$$

in the dual formulation we invert a *N x N* matrix (data space) rather than a *M x M* one (feature space), so this does not appear to be advantageous …

the main benefit is being able to **work directly with kernel functions** and bypass a choice of feature map

this allows one to work in feature spaces of **very high (even infinite) dimensionality**

Support Vector Machines

https://www.youtube.com/watch?v=5zRmhOUjjGY

The kernel trick

recall that the kernel function is constructed from a **feature space mapping**

$$k\left(\boldsymbol{x}_n, \boldsymbol{x}_m\right) = \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m) = \sum_{i=1}^{M} \phi_i(\boldsymbol{x}_n)\phi_i(\boldsymbol{x}_m)$$

one can also construct kernel functions directly, provided they can be expressed as above

e.g., for a 2D input space

$$k\left(\boldsymbol{x}, \boldsymbol{z}\right) = \left(\boldsymbol{x}^T \boldsymbol{z}\right)^2 = \left(x_1 z_1 + x_2 z_2\right)^2 = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{z})$$

$$\boldsymbol{\phi}(\boldsymbol{x}) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right)$$

note that feature space dimension (M=3) is bigger than input space dimensionality (d=2)

there exist more general methods to construct acceptable kernels e.g. using **basis functions**

a popular choice is Gaussian kernel, built from an **infinite-dimensional feature map**

$$k\left(\boldsymbol{x}, \boldsymbol{z}\right) = \exp\left(-||\boldsymbol{x} - \boldsymbol{z}||^2 / 2\sigma^2\right)$$

The kernel trick

a popular choice is Gaussian kernel, built from an **infinite-dimensional feature map**

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-||\boldsymbol{x} - \boldsymbol{z}||^2 / 2\sigma^2\right)$$

we can demonstrate that this is a **valid kernel** as follows: given two valid kernels

$$k_1(\boldsymbol{x}, \boldsymbol{z}) \ , \quad k_2(\boldsymbol{x}, \boldsymbol{z})$$

we can construct **new valid kernels** from them using the following rules

$$k(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x}) k_1(\boldsymbol{x}, \boldsymbol{z}) f(\boldsymbol{z}) \qquad k(\boldsymbol{x}, \boldsymbol{z}) = \exp(k(\boldsymbol{x}, \boldsymbol{z}))$$

so now we can check that the Gaussian kernel is indeed a *bona-fide* kernel by expanding

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\boldsymbol{x}^T \boldsymbol{x} / 2\sigma^2\right) \exp\left(-\boldsymbol{x}^T \boldsymbol{z} / 2\sigma^2\right) \exp\left(-\boldsymbol{z}^T \boldsymbol{z} / 2\sigma^2\right)$$

linear kernel

the feature map associated to this kernel has infinite dimensionality!

Summary

- Many problems in ML require assessing the **similarly between probability distributions**

- **Information theory** provides several quantities (e.g. KL divergence) to quantify this

- Generative models are design to evaluate **new instances of the training data,** with features as close as possible to the real ones

- Generative Adversarial Networks are a particular implementation of a generative model where **two networks compete againts each other** to evaluate the generative probability