Machine Learning:

A new toolbox for Theoretical Physics

Juan Rojo

VU Amsterdam & Theory group, Nikhef

D-ITP Advanced Topics in Theoretical Physics

02/12/2019

Unsupervised Learning

Unsupervised Learning

In ML context, **unsupervised learning** is concerned with discovering underlying structures in **unlabelled data**

an important example of unsupervised learning is **clustering:** the aim is to group unlabelled data into clusters using some **distance or similarity measure**

let us illustrate these ideas with ***K*-means clustering**

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

unlabelled dataset: N points with p features each

$$\{\boldsymbol{\mu}_k\}_{k=1}^{K} \qquad \boldsymbol{\mu}_k = (\mu_{k,1}, \mu_{k,2}, \ldots, \mu_{k,p})$$

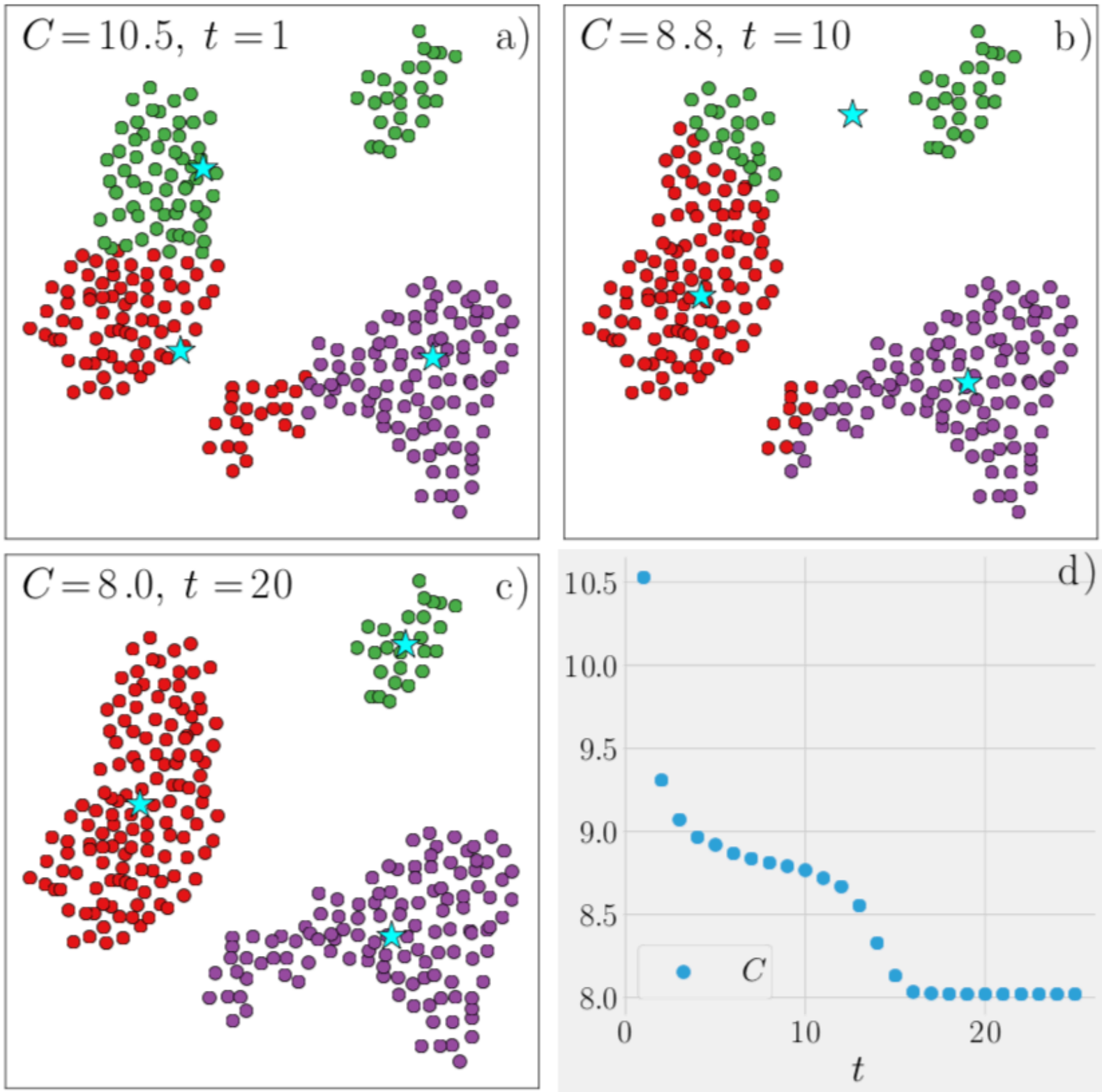
***cluster means**: K clusters with p features each*

the intuitive idea is that the cluster means represent the **main features of each cluster**, to which the data points will be assigned in the clustering procedure

Clustering

2D example of clustering: each colour represents a cluster, with stars indicating their *centers*

how is this clustering achieved in practice?



Clustering

in *K*-means clustering, the **cluster means** and the **data point assignments** are determined from the minimisation of a cost function:

$$C\left(\boldsymbol{x};\boldsymbol{\mu}\right) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\left(\boldsymbol{x}_n - \boldsymbol{\mu}_k\right)^2$$

binary assignment variable

Euclidean distance between n-th data point and k-th cluster centre

$r_{nk} = 1$ ⟶ *the n-th point is assigned to the k-th cluster*

$r_{nk} = 0$ ⟶ *the n-th point is not assigned to the k-th cluster*

furthermore since clustering is exclusive one needs to impose: $\sum_{k=1}^{K} r_{nk} = 1 \quad \forall n$

one sees that K-means clustering aims to **minimise the variance within each cluster**

Clustering

Let's describe an algorithm that implements *K*-means clustering by minimising the cost function

$$C\left(\boldsymbol{x};\boldsymbol{\mu}\right)=\sum_{n=1}^{N}\sum_{k=1}^{K}r_{nk}\left(\boldsymbol{x}_n-\boldsymbol{\mu}_k\right)^2$$

this algorithm alternates between two main steps:

- ***(1) Expectation*:** starting from set of cluster assignments $\{r_{nk}\}$ minimise *C* wrt cluster means

$$\frac{\partial}{\partial\boldsymbol{\mu}_k}C\left(\boldsymbol{x};\boldsymbol{\mu}\right)=0 \quad\rightarrow\quad \boldsymbol{\mu}_k=\frac{1}{N_k}\sum_{n=1}^{N}r_{nk}\boldsymbol{x}_n \qquad N_k=\sum_{n=1}^{N}r_{nk}$$

number of points in k-th cluster

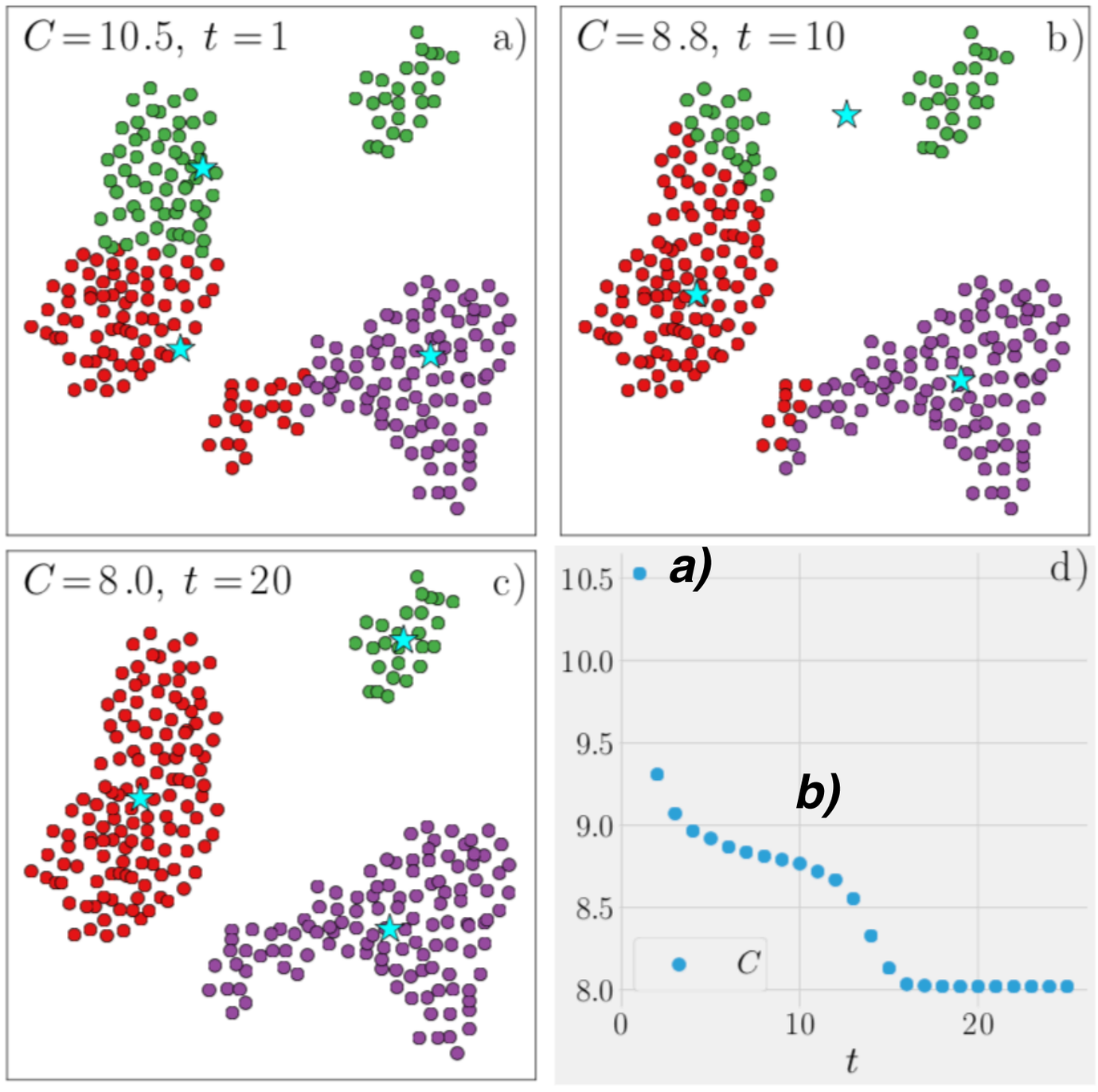
- ***(2) Maximization*:** given the *K* cluster centers, the assignments $\{r_{nk}\}$ should minimise *C*. This can be achieved by assigning each data point to its closest cluster-mean

$$r_{nk}=1 \quad \text{if} \quad k=\arg\min_{k'}(\boldsymbol{x}_n-\boldsymbol{\mu}_{k'})$$

$$r_{nk}=0 \quad \text{if} \quad k\neq\arg\min_{k'}(\boldsymbol{x}_n-\boldsymbol{\mu}_{k'})$$

Clustering

these two steps are iterated until some **convergence criterion** is achieved, *e.g.* when the change in the cost function between two iterations is below some threshold



K-means clustering can lead to spurious results since the underlying assumption is that the latent model has uniform variances

Hierarchical clustering

- Another approach to clustering is based on **agglomerative methods**, where one starts from small clusters which are progressively merged into bigger clusters

- This hierarchical structure provides information on the relations between clusters and the **subcomponents of individual clusters**

- As before, we need to specify a **distance**, this time between two clusters *X, Y*

$$d(X, Y) \in \mathscr{R}$$

- At each iteration, the two clusters closer to each other (quantified by *d*) are *merged*

the **agglomerative cluster algorithm** works as follows:

- ***(1)*** Assign **each data point to be its own cluster**

- ***(2)*** Given the resulting set of K clusters, find the closest pair

$$(X_i, X_j) \quad \text{such that} \quad (i, j) = \arg\min_{i'j'} d(X_{i'}, X_{j'})$$

- ***(3)*** Merge the pair into a single cluster. Iterate ***(2)*** and ***(3)*** until a single cluster remains

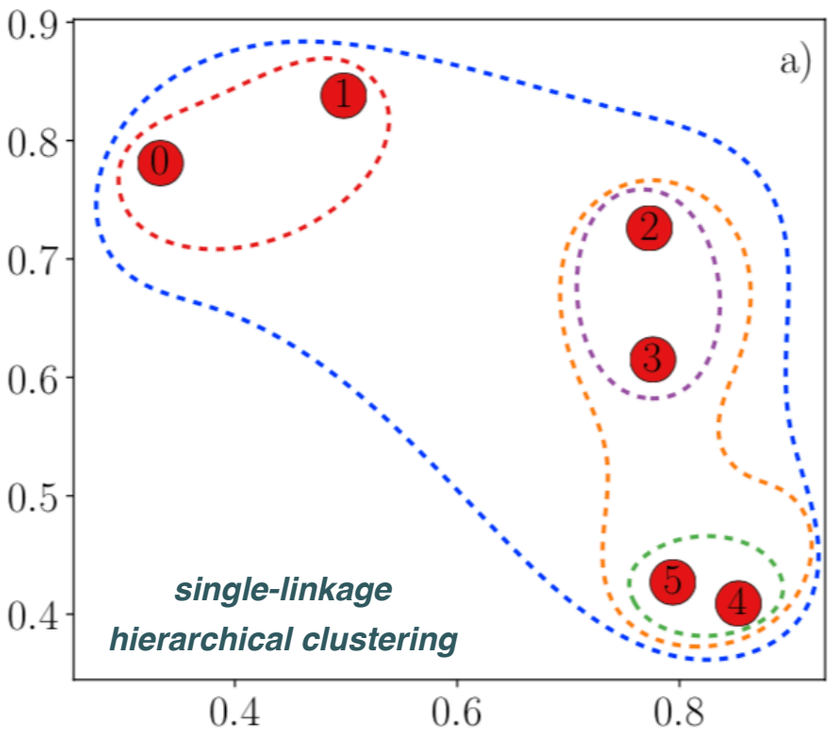
Hierarchical clustering

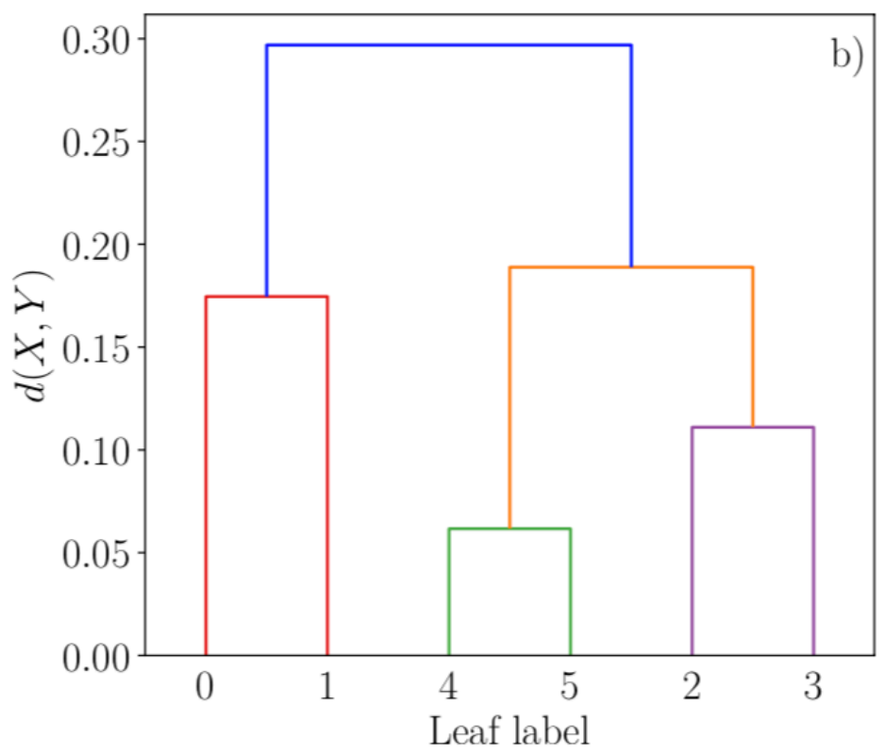
Clearly the results of hierarchical clustering depend on the **choice of distance**

single linkage $\longrightarrow$ $d(X_i, X_j) = \min_{\boldsymbol{x}_i \in X_i, \boldsymbol{x}_j \in X_j} ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2$

complete linkage $\longrightarrow$ $d(X_i, X_j) = \max_{\boldsymbol{x}_i \in X_i, \boldsymbol{x}_j \in X_j} ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2$

Euclidean distance





hierarchical clustering methods do not scale well for large *N*, so they are typically **combined with *K*-means clustering** in the initial steps to define small clusters

Ensemble Methods

Combining models

A powerful strategy in machine learning is that of **ensemble methods** that **combine predictions from multiple statistical models** to improve predictive performance

- Key aspect: assess the **degree of correlation** between the models of the ensemble
- The reason is that **combining correlated models reduces the overall variance less** than in the uncorrelated case
- Also these correlations can **increase the bias of the combined model**, offsetting potential reductions in variance from ensemble averaging

Combining models

A powerful strategy in machine learning is that of **ensemble methods** that **combine predictions from multiple statistical models** to improve predictive performance

- Key aspect: assess the **degree of correlation** between the models of the ensemble
- The reason is that **combining correlated models reduces the overall variance less** than in the uncorrelated case
- Also these correlations can **increase the bias of the combined model**, offsetting potential reductions in variance from ensemble averaging

e. g. Assume that your model predicts *X,* and you have *n* models. If each of these models has variance σ, then the variance of their sum is

$$\mathrm{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \mathrm{Var}(X_i) + 2 \sum_{1 \le i \le j \le n} \mathrm{Cov}(X_i, X_j)$$

$$\mathrm{Var}\left(\overline{X}\right) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n} \mathrm{Var}(X_i)$$

$$\mathrm{Var}\left(\overline{X}\right) = \frac{\sigma^2}{n} + \frac{n-1}{n}\rho\sigma$$

reduction as n increased

correlations increase variance

Bagging

Bootstrap AGGgregation (Bagging) is a popular **ensemble combination method**

we start from a large dataset that is partitioned into *M* smaller datasets:

$$\mathcal{L} \quad \rightarrow \quad \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_M\} \qquad \sum_{m=1}^{M} \mathcal{L}_m = \mathcal{L}$$

So that each of the *M* datasets is large enough to train a predictor. The **aggregate predictor** is then constructed from those trained in each separate dataset

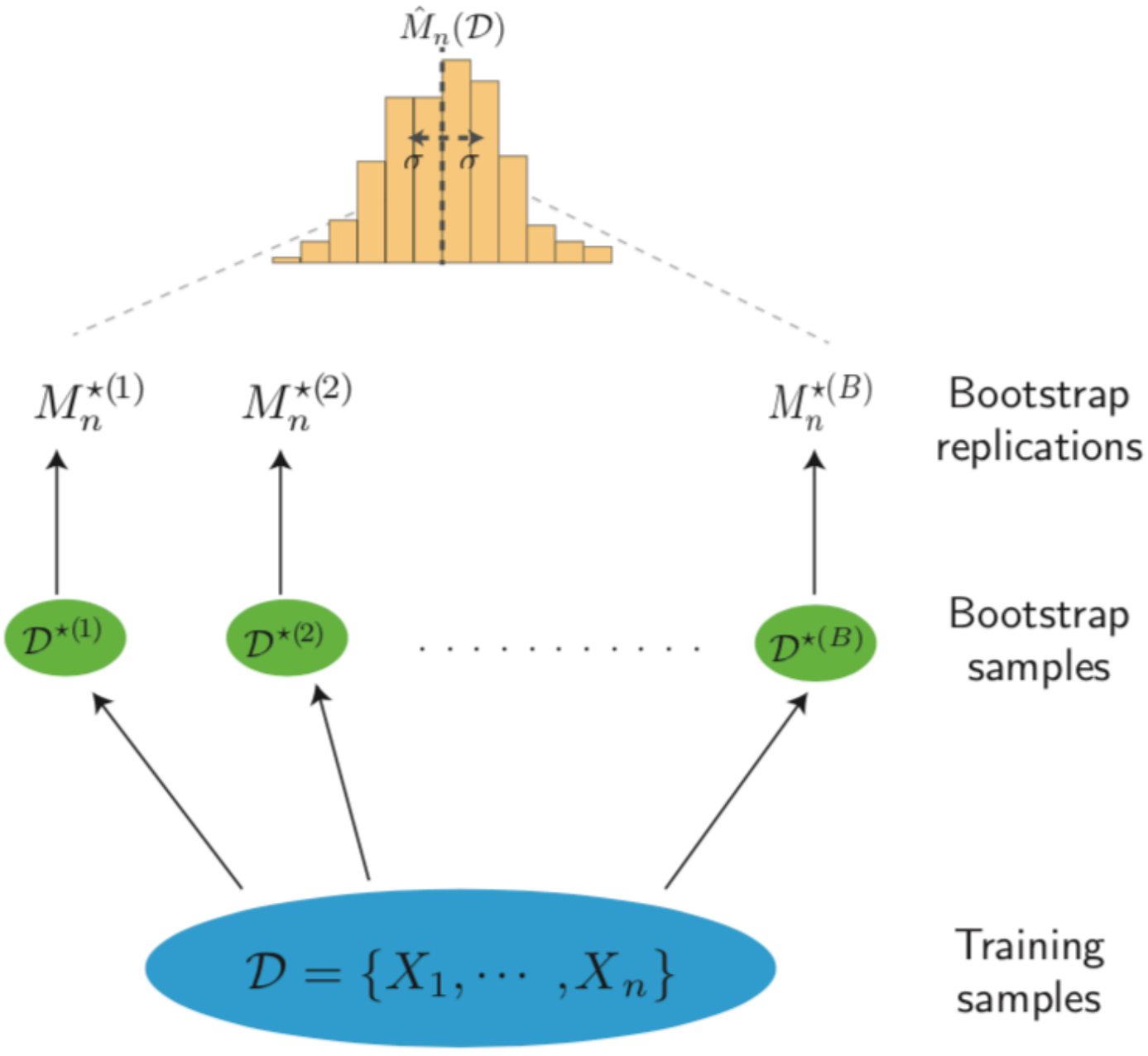
$$\widehat{g}^{A}_{\mathcal{L}}(\boldsymbol{x}) = \frac{1}{M} \sum_{i=1}^{M} g^{A}_{\mathcal{L}_i}(\boldsymbol{x})$$

for continuous datasets, analogous expressions for discrete classifiers

Such aggregation can **reduce the variance without increasing the bias**

But what should we do if we don't have a very large dataset? If each partitioned set one has few data points then the prediction will be poor …

Bootstrapping



In ML applications, bootstrapping is frequently used to **assign measures of accuracy** (defined in terms of bias, variance, correlations etc to sample estimates

It can be shown that in the large *n* limit the **bootstrap distributions** approximate well the **sampling distributions** from which the training dataset was obtained

Bootstrapping

- Assume we are given a training dataset and we want to compute *eg* confidence intervals

$$\mathscr{D} = \{X_1, \ldots, X_n\}$$

- This can be done by **sampling *n* points with replacement** to get *B* new datasets

$$\mathscr{D}^{*(1)} = \{X_1^{*(1)}, \ldots, X_n^{*(1)}\}$$

$$\vdots$$

$$\mathscr{D}^{*(B} = \{X_1^{*(B)}, \ldots, X_n^{*(B)}\}$$

bootstrap samples (with repeated elements)

- With the bootstrap samples we can construct statistical quantities of interest:

$$\widehat{\mathrm{Var}}_B(M_n) = \frac{1}{B-1}\sum_{k=1}^{B}\left(M_n^{*(k)} - \overline{M}_n^*\right)^2 \qquad \overline{M}_n^* = \frac{1}{B}\sum_{k=1}^{B} M_n^{*(k)}$$

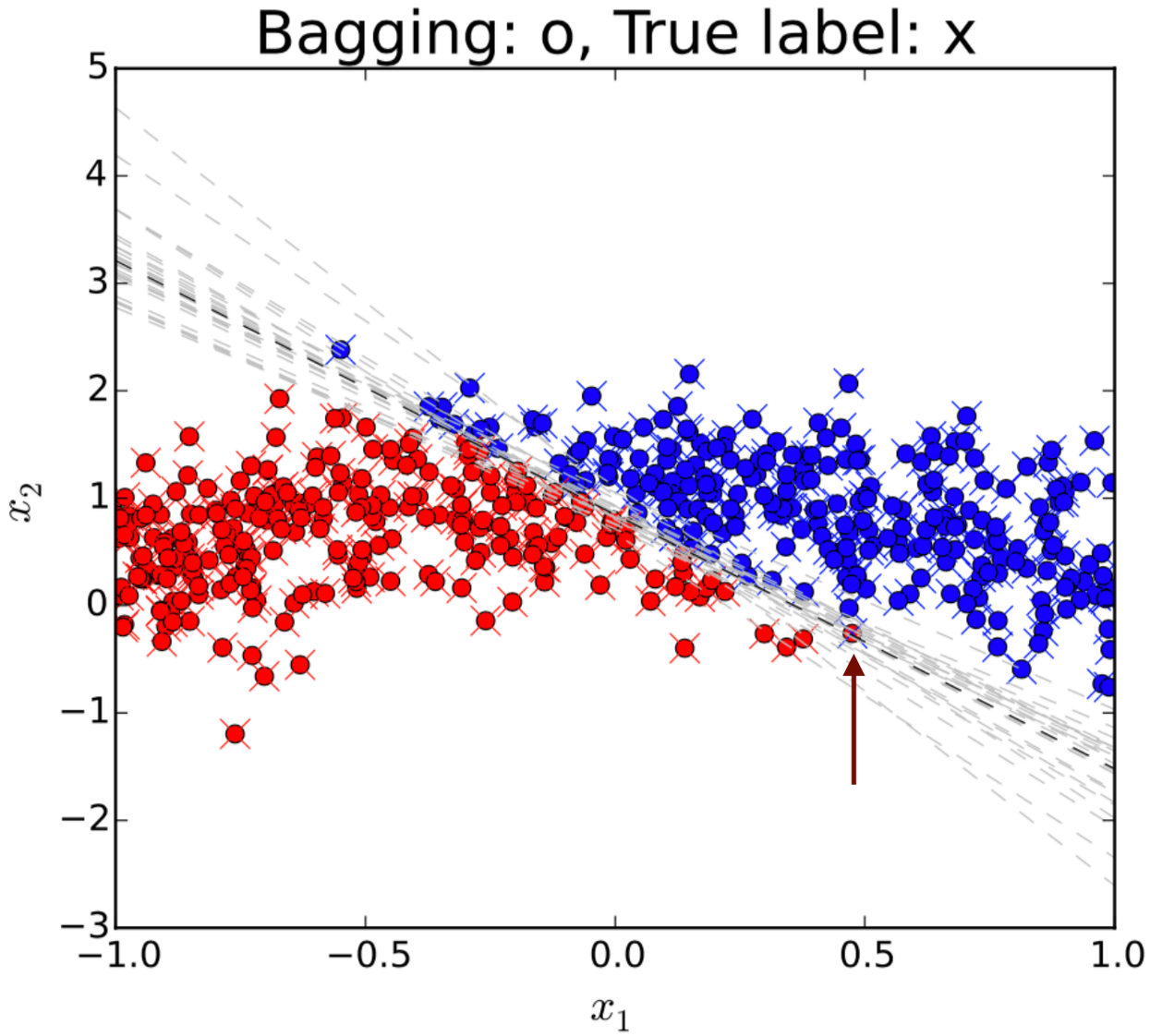
It can be shown that in the large *n* limit the **bootstrap distributions** approximate well the **sampling distributions** from which the training dataset was obtained

Bagging with bootstrap

same as before, but now the datasets have been partitioned with bootstrapping

$$\widehat{g}^{\mathrm{BS}}_{\mathscr{L}}(\boldsymbol{x}) = \frac{1}{M}\sum_{i=1}^{M} g^{A}_{\mathscr{L}^{\mathrm{BS}}_{i}}(\boldsymbol{x})$$

which can lead to a **variance reduction** to the price of an increase in the bias



ex: bagging with bootstrap (2D classification)

n=500, *B*=25 (50 points each)

grey dashed: bootstrap predictions

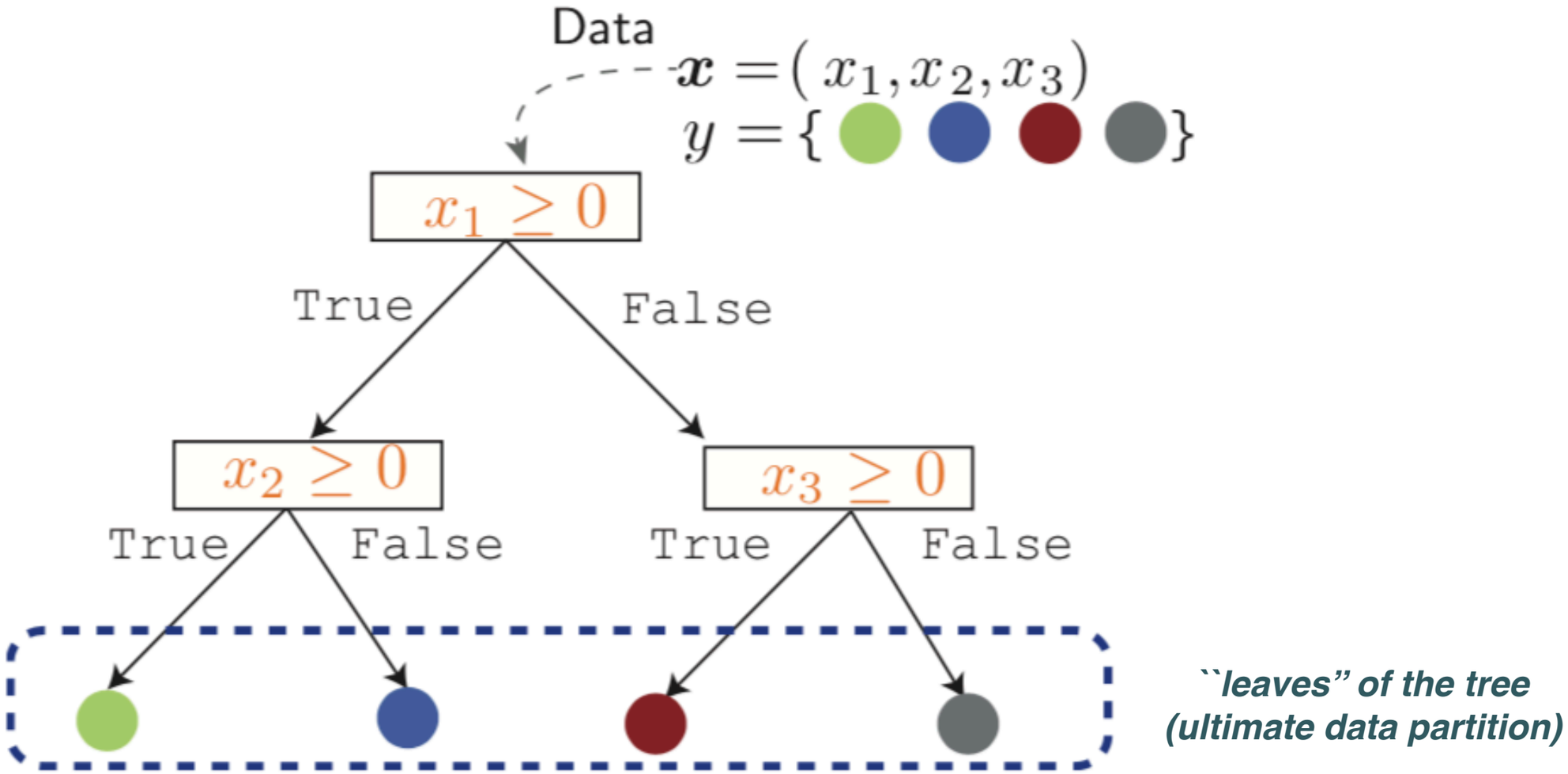
black dahed: bagging average

individual predictors poor, bagging much better!

bagging is specially useful for **unstable learning algorithms** where small changes in the training dataset result in large changes in the prediction

Random Forests

Ensemble method widely used in complex classification tasks constructed from **randomised tree-based classifier decision trees**



In ML, a **decision tree** is an algorithm which uses a series of questions to hierarchically partition the data, where each branch of the tree splits the data into smaller subsets

Random Forests

individual trees have often **high variance** and are weak classifiers:

we can improve by incorporating them in an ensemble method

→ We need an ensemble of **randomised decision trees** (minimised correlations)

- ***(1)*** train each decision tree on a different bootstrapped dataset: **bagged decision tree**
- ***(2)*** use different random subset of features at each split: **random forest**

reduces correlations between trees that arise
when only few features are strongly predictive

Dimensional Reduction & Data Visualisation

Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space,** using **dimensional reduction**

Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space,** using **dimensional reduction**

This is easier said that done: many pitfalls associated to high-dimensionality datasets

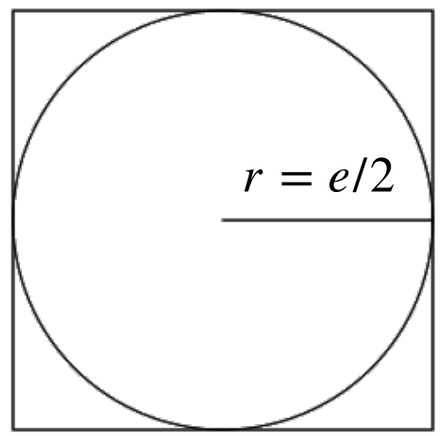
High-dimensional data lives near the edge of the sample space

consider data distributed at random in a *D*-dimensional hypercube $C = [-e/2, e/2]^D$

consider a *D*-dimensional sphere *S* of radius *e/2* entered at origin

probability that random point from *C* is sampled inside the sphere *S* is

$$P\left(\boldsymbol{x}_i \in \mathcal{S}\right) \simeq \frac{\pi^{D/2}(e/2)^D/\Gamma(D/2+1)}{e^D} = \simeq \frac{\pi^{D/2}}{2^D D^D} \to 0$$



so most of the data lies close to the hypercube edge!

Dimensional reduction

Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

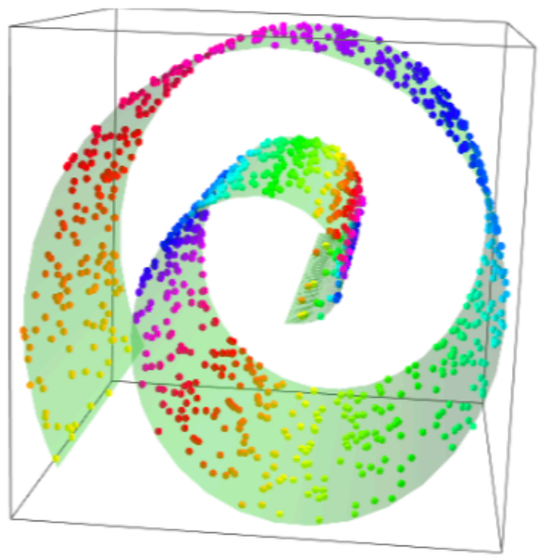
Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space,** using **dimensional reduction**

This is easier said that done: many pitfalls associated to high-dimensionality datasets

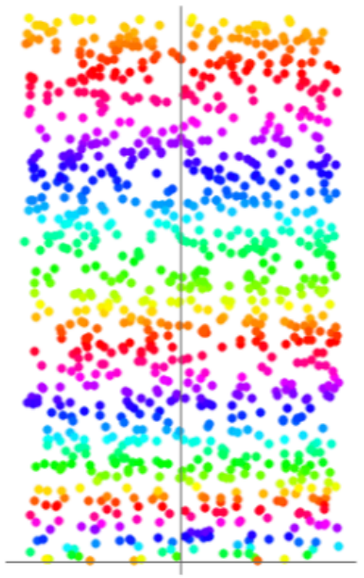
- *High-dimensional data lives near the edge of the sample space*
- *We need to conserve information on original pair-wise distances or similarities when transforming to the latent space*

the minimum number of parameters needed to capture the original patters is the **intrinsic dimensionally of the data**

a) ***original data space: 3D***



b) ***latent space: 2D***



Dimensional reduction

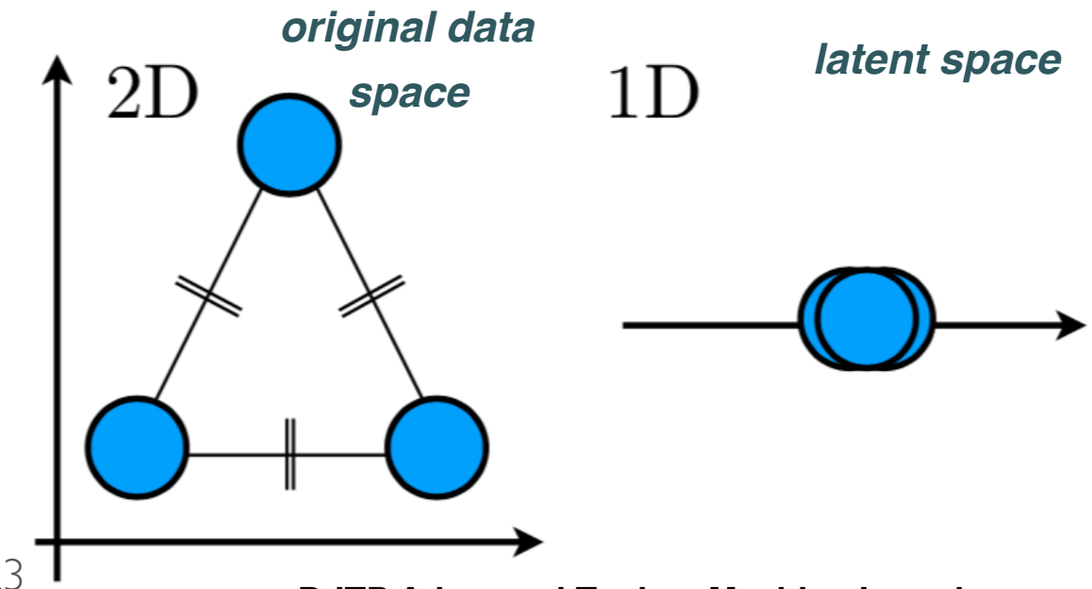
Efficient **data visualisation techniques** are essential to construct better models in ML applications eg by identifying correlated, redundant, or irrelevant features

Traditional data visualisation methods are not practical when the datasets involve a large number of features (such as images) and we need to project the data onto a lower-dimensional space, called the **latent space,** using **dimensional reduction**

This is easier said that done: many pitfalls associated to high-dimensionality datasets

- *High-dimensional data lives near the edge of the sample space*
- *We need to conserve information on original pair-wise distances or similarities when transforming to the latent space*
- *Dimensional reduction cannot be such to destroy info on original patterns in the data*

``the crowding problem"



Principal Component Analysis

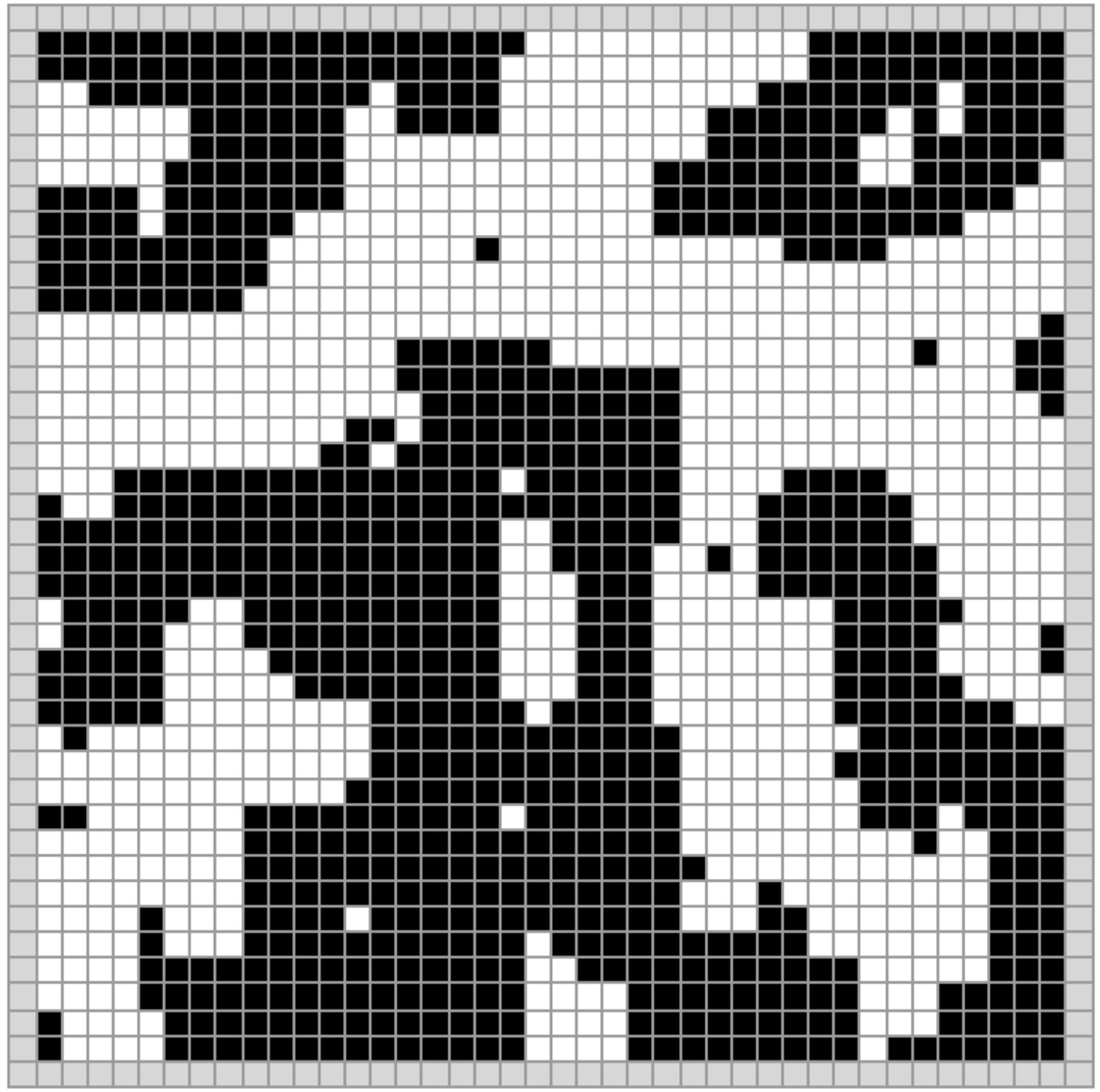
PCA projections often capture the **large-scale structure** of high-dimensional datasets

consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space

can we **measure ``order''** with few parameters?

disordered (random) phase

ordered phase



https://demonstrations.wolfram.com/The2DIsingModelMonteCarloSimulationUsingTheMetropolisAlgorit/

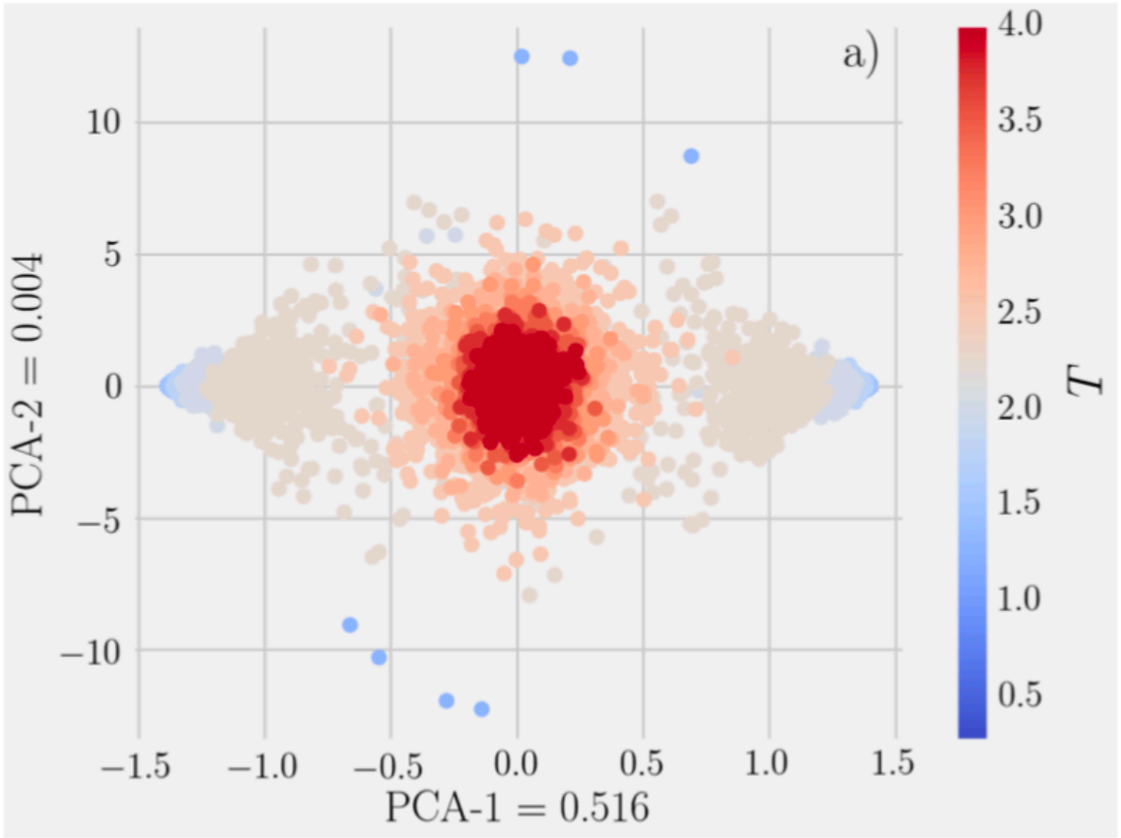
Principal Component Analysis

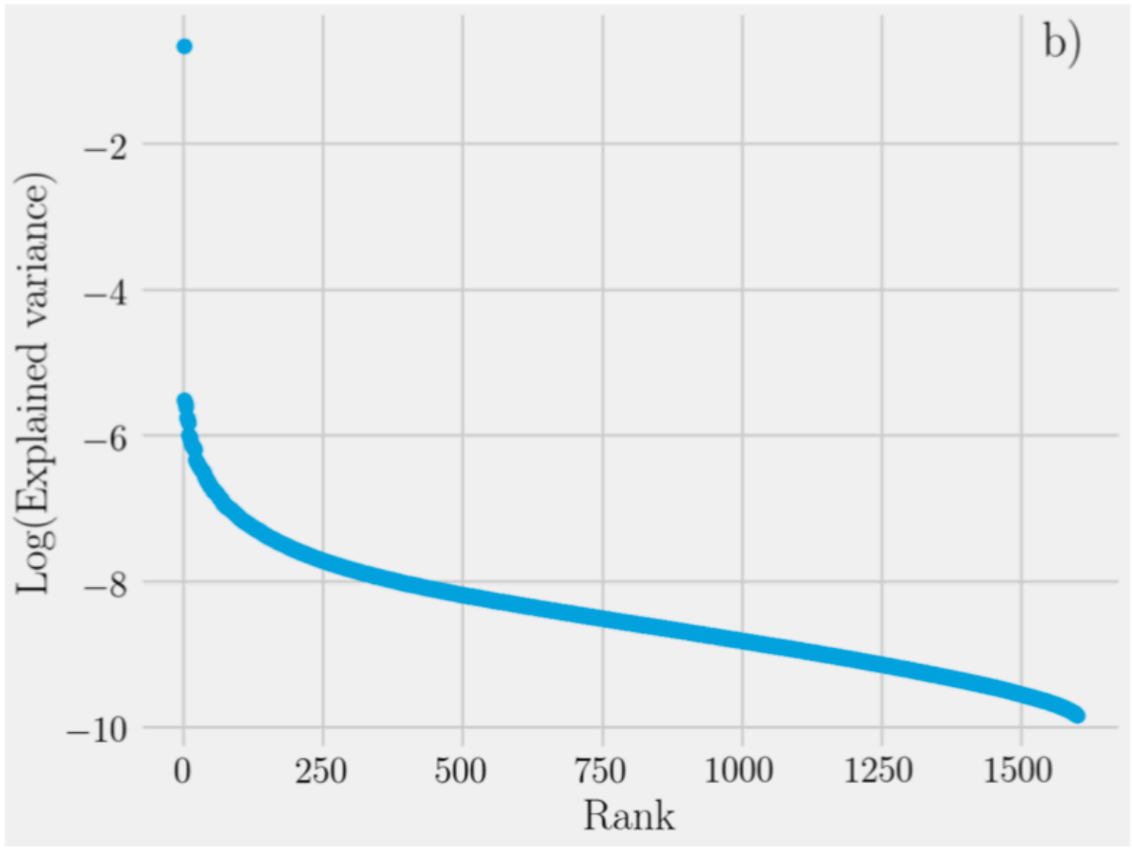
PCA projections often capture the **large-scale structure** of high-dimensional datasets

consider for example the **Ising Model in 2D** with 40 spins: 1600-dimensional space

can we **measure ``order''** with few parameters?

1000 samples for each T





The first principal component accounts for > 50% of total variability!

This first PCA component corresponds to the **magnetisation order parameter**, which we have thus identified without any prior physical knowledge of the system

Principal Component Analysis

consider *n* data points that live in a *p*-dimensional feature space

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

assume for simplicity that the mean of these points vanishes

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = 0$$

Now denote the **design matrix** as

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^T$$

In a design matrix, each row represents an individual data point and each each columns one of the data features

where rows are the data points and columns are features

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n,1} & x_{n,2}\,. & \cdots & x_{n,p} \end{pmatrix}$$

Principal Component Analysis

the associated (symmetric) *p*-dimensional covariance matrix is then

$$\boldsymbol{\Sigma}(X) = \frac{1}{n-1} X^T X \quad \rightarrow \quad \boldsymbol{\Sigma}_{lm} = \frac{1}{n-1} \sum_{i=1}^{n} X_{li} X_{im}$$

from where we see that $\boldsymbol{\Sigma}_{lm}$ measures the correlation between features *l* and *m*

The goal of PCA is to rotate this matrix to a **new feature-basis** (other than the one present in original data) that emphasises high-variability directions. This can be done by a linear transformation that **reduces the covariance between features**

recall the **eigenvalue decomposition** for a square matrix

$$A = Q \Lambda Q^T$$

columns are eigenvectors of A

diagonal matrix of eigenvalues

columns are eigenvectors of A

Principal Component Analysis

the associated (symmetric) *p*-dimensional covariance matrix is then

$$\boldsymbol{\Sigma}(X) = \frac{1}{n-1} X^T X \quad \rightarrow \quad \boldsymbol{\Sigma}_{lm} = \frac{1}{n-1} \sum_{i=1}^{n} X_{li} X_{im}$$

from where we see that $\boldsymbol{\Sigma}_{lm}$ measures the correlation between features *l* and *m*

The goal of PCA is to rotate this matrix to a **new feature-basis** (other than the one present in original data) that emphasises high-variability directions. This can be done by a linear transformation that **reduces the covariance between features**

for this we will use **Singular Value Decomposition** (SVD), a factorisation of a real or complex matrix that generalizates the eigendecomposition of a positive semidefinite matrix

$$X = USV^T$$

left (right) singular vectors of X: orthonormal eigenvectors of XX^* (X^*X)

columns are left singular vectors of X

diagonal matrix of singular values

columns are right singular vectors of X

Principal Component Analysis

Using SVD we can express the data covariance matrix as

$$\boldsymbol{\Sigma}(\boldsymbol{X}) = \frac{1}{n-1}\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{V}\left(\frac{S^2}{n-1}\right)\boldsymbol{V}^T \equiv \boldsymbol{V}\Lambda\boldsymbol{V}^T$$

U,V are unitary matrices

diagonal matrix with eigenvalues in decreasing order along diagonal

columns of $\boldsymbol{V}$ ⟶ principal directions of $\boldsymbol{\Sigma}$

at this point we are ready to use PCA to reduce the dimensionality of data **from *p* to *p'* < *p***

$$\widetilde{\boldsymbol{Y}} = \boldsymbol{X}\widetilde{\boldsymbol{V}}_{p'}$$

reduced dataset: n points with p' < p features each

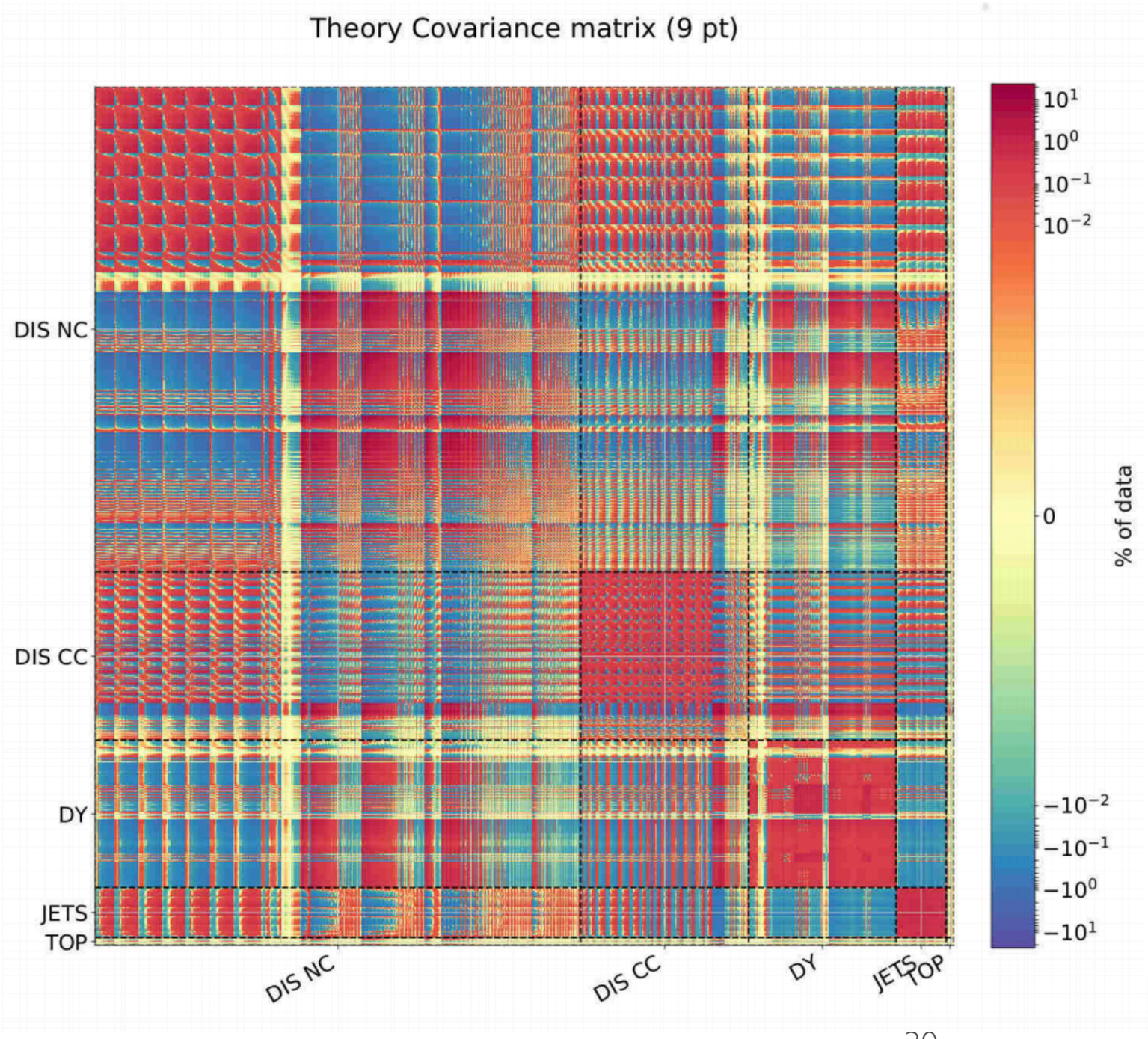
original dataset: n points with p features each

projection matrix: select the p' largest eigenvectors

with PCA only the **p' directions with higher variability** remain

PCA for propagation of theory errors

assume that some gives you the covariance matrix of theory errors of a global PDF fit (3000-dimensional space!). What are the **relevant components**?



Process Type	Dataset	Reference	$N_{\rm dat}$	$N_{\rm dat}$ (total)
DIS NC	NMC	[28,29]	134	1593
	SLAC	[30]	12	
	BCDMS	[31,32]	530	
	HERA σ^p_{NC}	[36]	886	
	HERA σ^c_{NC}	[37]	31	
DIS CC	NuTeV dimuon	[33,34]	41	552
	CHORUS	[35]	430	
	HERA σ^p_{CC}	[36]	81	
DY	ATLAS W,Z, 7 TeV 2010	[42]	30	484
	ATLAS W,Z, 7 TeV 2011	[43]	34	
	ATLAS low-mass DY 2011	[44]	4	
	ATLAS high-mass DY 2011	[45]	5	
	ATLAS Z p_T 8 TeV (p_T^{ll}, M_{ll})	[46]	44	
	ATLAS Z p_T 8 TeV (p_T^{ll}, y_Z)	[46]	48	
	CMS Drell-Yan 2D 2011	[51]	88	
	CMS W asy 840 pb	[52]	11	
	CMS W asy 4.7 pb	[53]	11	
	CMS W rap 8 TeV	[54]	22	
	CMS Z p_T 8 TeV (p_T^{ll}, M_{ll})	[55]	28	
	LHCb Z 940 pb	[60]	9	
	LHCb $Z \to ee$ 2 fb	[61]	17	
	LHCb $W,Z \to \mu$ 7 TeV	[62]	29	
	LHCb $W,Z \to \mu$ 8 TeV	[63]	30	
	CDF Z rap	[38]	29	
	D0 Z rap	[39]	28	
	D0 $W \to e\nu$ asy	[40]	8	
	D0 $W \to \mu\nu$ asy	[41]	9	
JET	ATLAS jets 2011 7 TeV	[47]	31	164
	CMS jets 7 TeV 2011	[56]	133	
TOP	ATLAS $\sigma^{\rm top}_{tt}$	[48,49]	3	26
	ATLAS $t\bar{t}$ rap	[50]	10	
	CMS $\sigma^{\rm top}_{tt}$	[57,58]	3	
	CMS $t\bar{t}$ rap	[59]	10	
Total			2819	2819

Generative Models & Adversarial Learning

The Kullback-Leibler divergence

The KL divergence, a measure of the similarity between two probability distributions *p(**x**)* and *q(**x**)* plays an important role in generative models

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

which can be symmetrised to construct a **squared metric** (distance)

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

Jensen-Shannon divergence

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

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

The Kullback-Leibler divergence

In **generative models** such as GANs one deals with two probability distributions, 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*

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

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

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

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

The Kullback-Leibler divergence

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

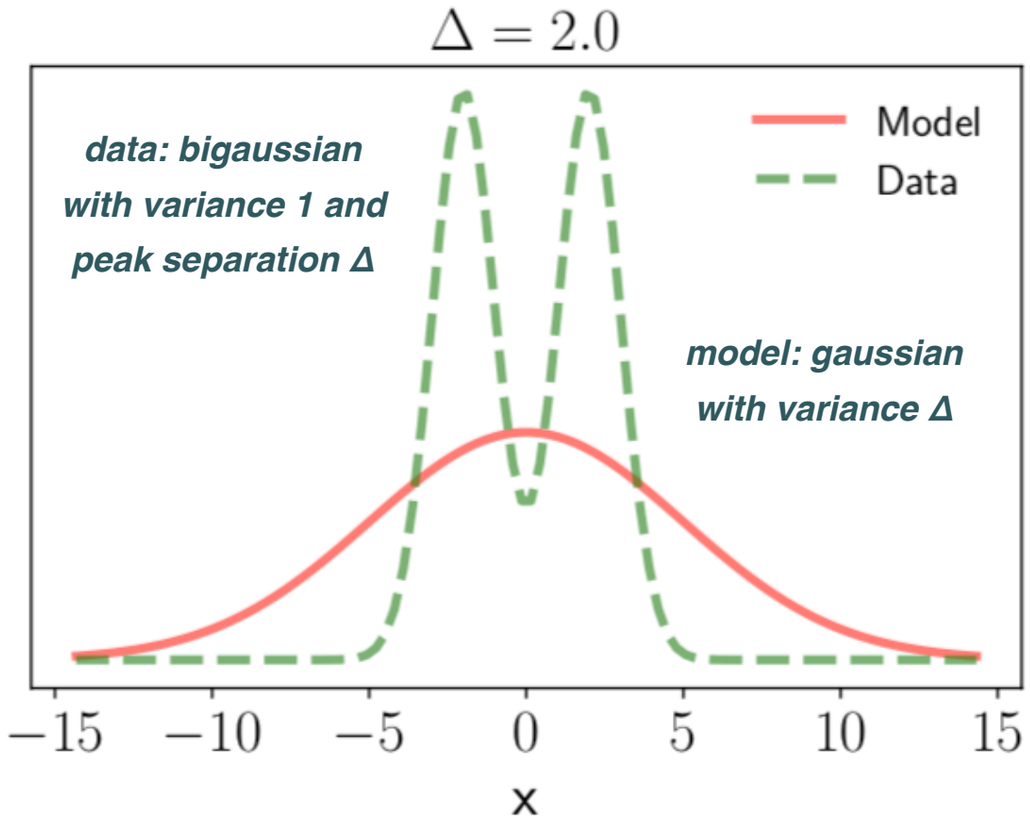
$$\left\langle \log p_\theta(\boldsymbol{x}) \right\rangle_{\text{data}} = S[p_{\text{data}}] - D_{KL}\left(p_{\text{data}} \,||\, p_\theta\right)$$

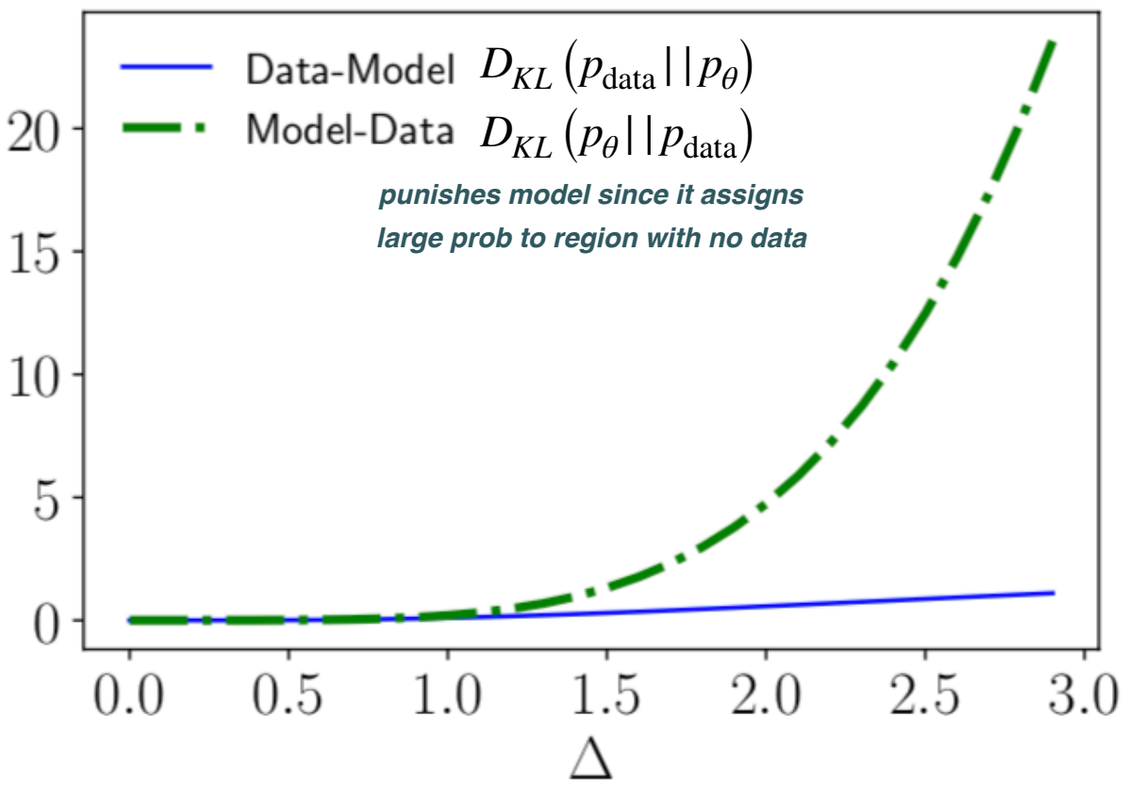
Log-likelihood of data under model

entropy of data: independent of model parameters

KL-divergence

similarity is a subtle concept ….





Adversarial Learning

(1) $D_{KL}\left(p_{\text{data}}||p_\theta\right)$ → ***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 maximizing the likelihood

Generative adversarial networks

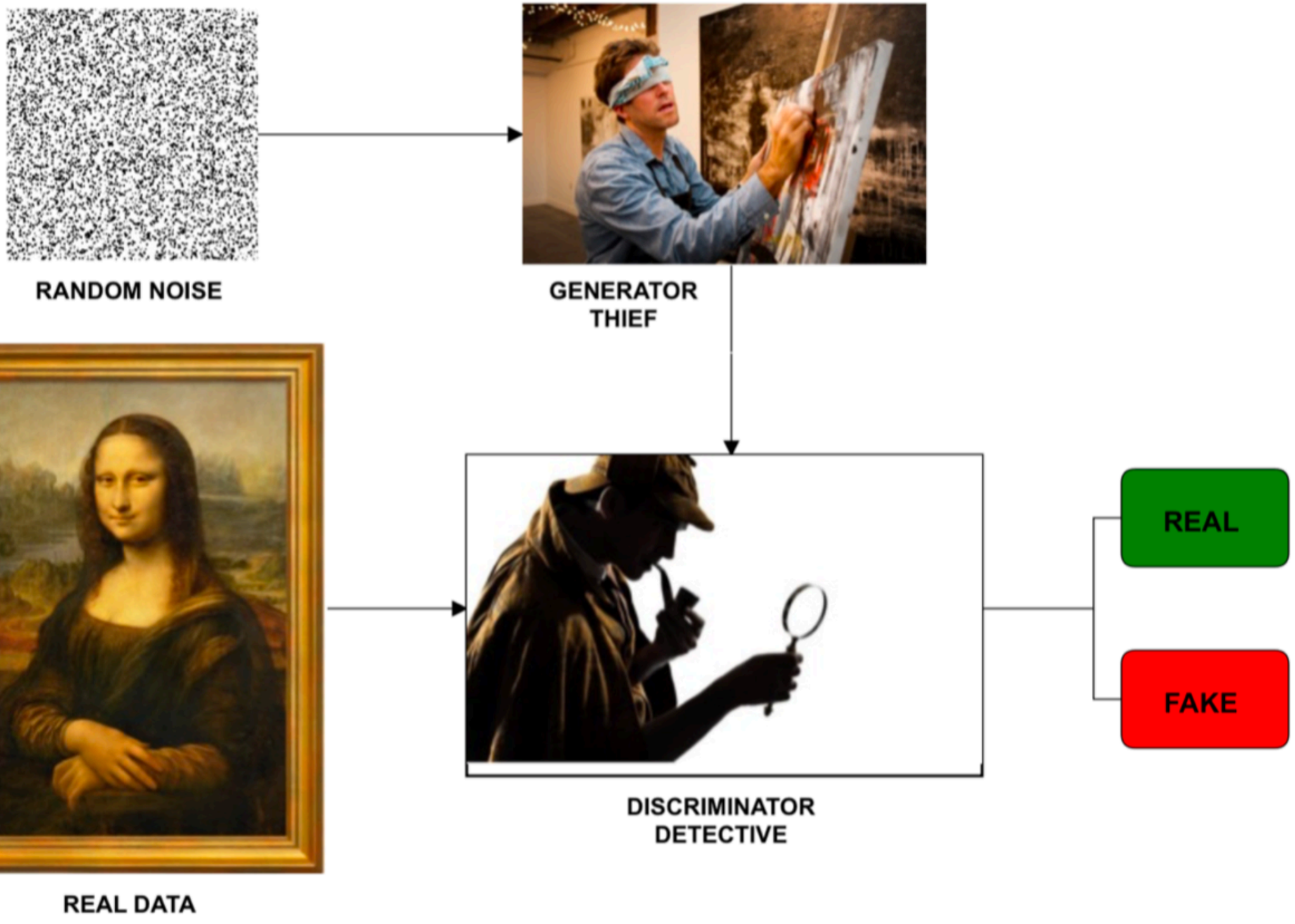
Generative Adversarial Networks (GANs) are deep neural networks architectures, composed by two independent NNs which **compete against each other**

- ***(1)*** A **generator G** NN that creates 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**

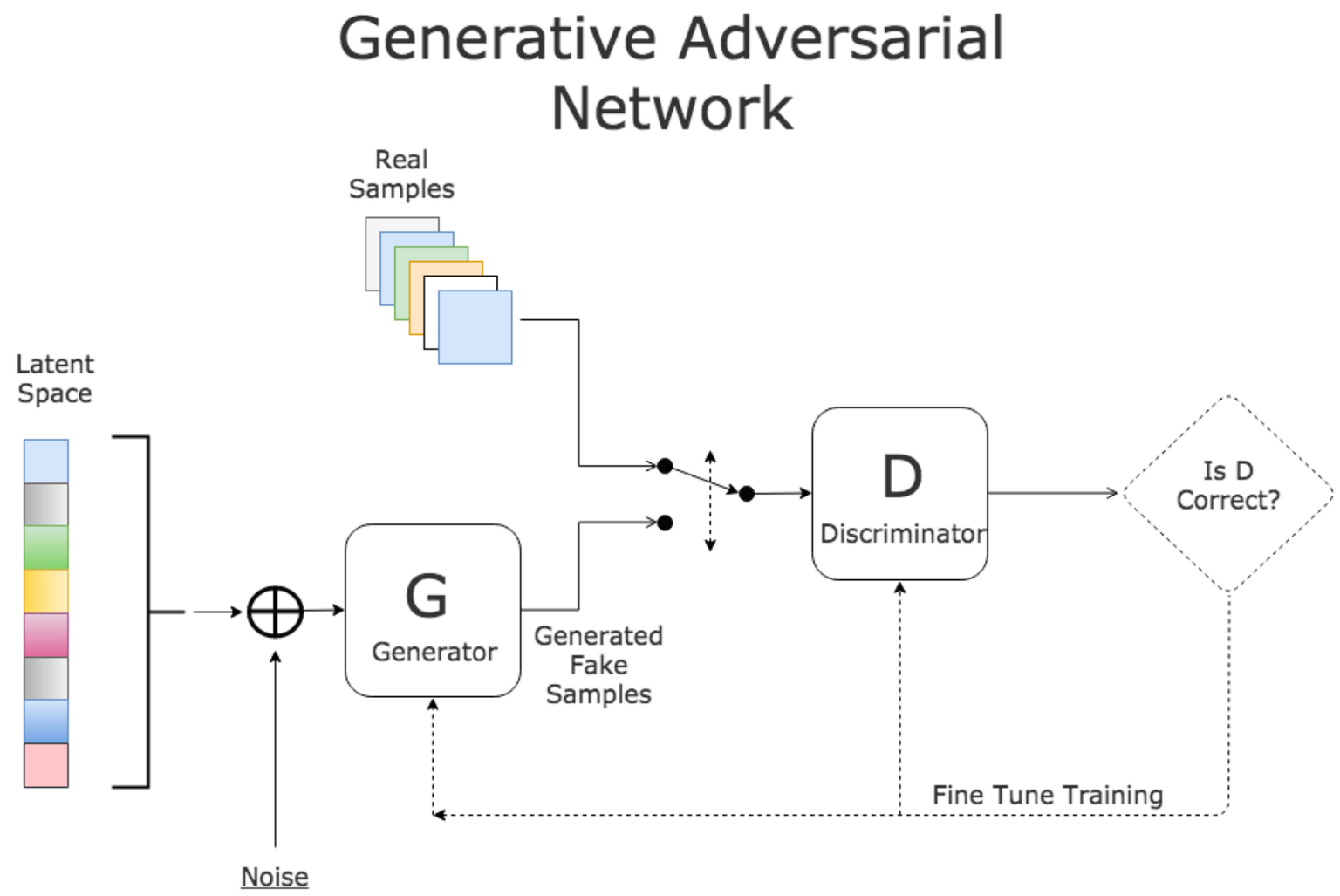
The generator network **G** should be trained to maximise the probability that the discriminator network **D** makes a mistake: that is, **G** should generate pseudo-data samples that are virtually indistinguishable from the actual data

Generative adversarial networks



Generative Adversarial Networks

- New 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 ite0 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

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

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

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

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

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

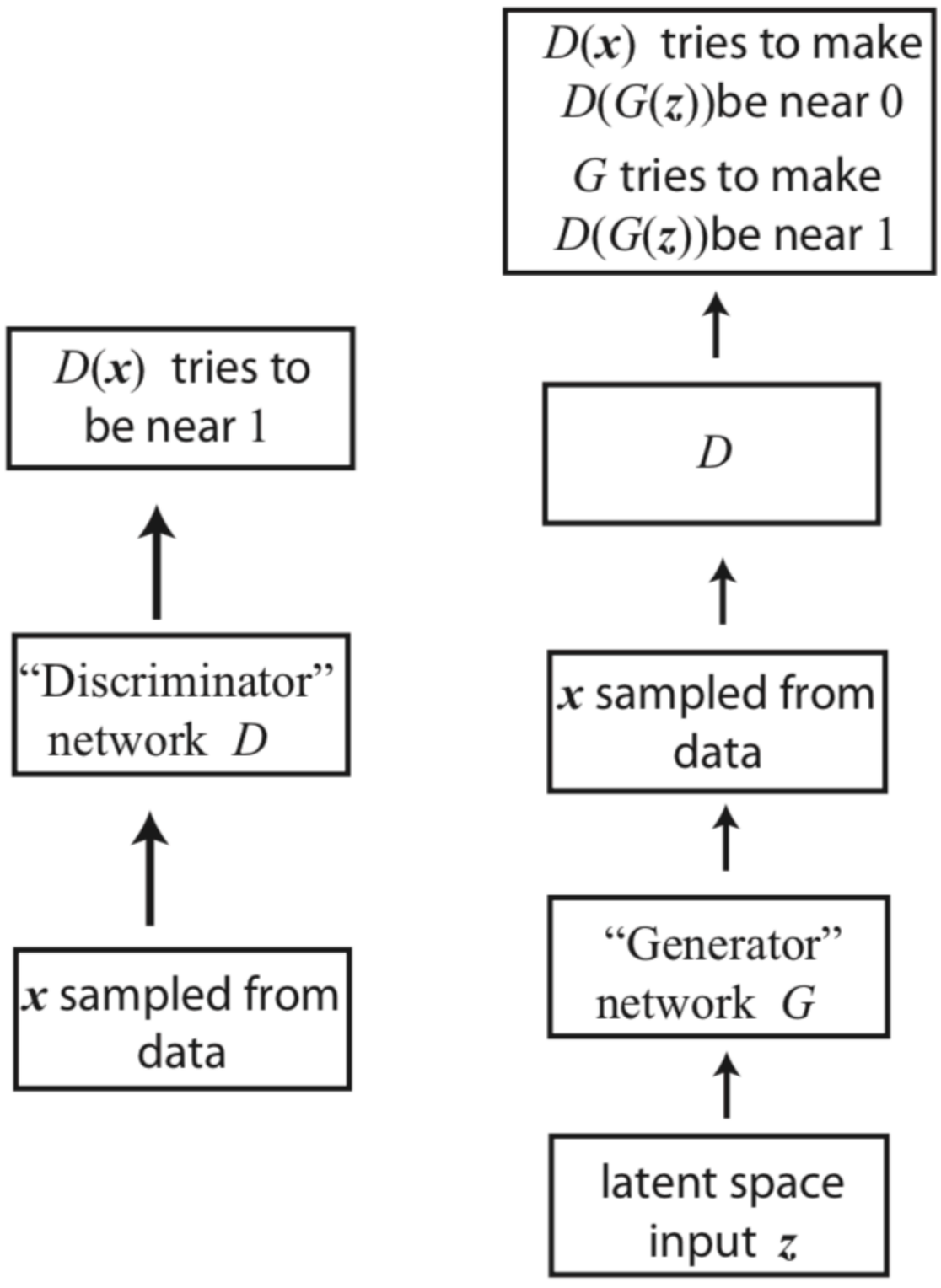


Image generation with GANs

https://thispersondoesnotexist.com/