Deep learning workshop AVIVA

Lancelot Da Costa

10-11th December 2019

Part I: Topics in dimensionality reduction

Why dimensionality reduction? (1)

- Multi-dimensional data visualisation

data frame

	x_1	x_2	x_3	x_4	x_5	...	x_n
1	1.755165	0.841470	15.16638	-3.19198	0.385065	...	-4.67732
2	1.080604	0.909297	4.534220	1.999736	1.015282	...	1.621900
3	0.141474	0.141120	13.29099	-8.99336	1.324239	...	0.306199
4	-0.83229	-0.75680	18.03677	1.355381	0.496793	...	-1.57096
5	-1.60228	-0.95892	21.18148	-6.47613	0.031777	...	-1.55290
6	-1.97998	-0.27941	23.12537	-2.02521	0.475099	...	-3.10106
7	-1.87291	0.656986	10.53740	-2.40270	0.037270	...	3.931002
8	-1.30728	0.989358	2.372823	-0.63531	0.627276	...	1.168063
9	-0.42159	0.412118	12.85550	-4.39144	1.412286	...	-4.55986
10	0.567324	-0.54402	11.38763	1.895743	0.643629	...	4.772541
11	1.417339	-0.99999	6.965379	-9.04229	0.754703	...	0.008559
12	1.920340	-0.53657	18.95395	-7.93374	0.241731	...	1.665364
13	1.953175	0.420167	12.80815	-5.27022	1.493793	...	-4.53836
14	1.507804	0.990607	1.293854	-6.83280	0.253613	...	4.246339
15	...	...	...	...	...	...	...

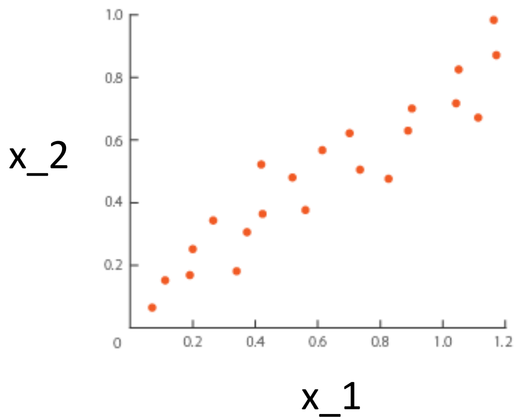
Why dimensionality reduction? (1)

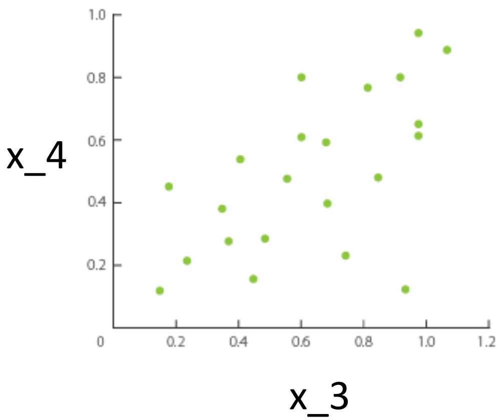
- Multi-dimensional data visualisation

data frame

	x_1	x_2	x_3	x_4	x_5	...	x_n
1	1.755165	0.841470	15.16638	-3.19198	0.385065	...	-4.67732
2	1.080604	0.909297	4.534220	1.999736	1.015282	...	1.621900
3	0.141474	0.141120	13.29099	-8.99336	1.324239	...	0.306199
4	-0.83229	-0.75680	18.03677	1.355381	0.496793	...	-1.57096
5	-1.60228	-0.95892	21.18148	-6.47613	0.031777	...	-1.55290
6	-1.97998	-0.27941	23.12537	-2.02521	0.475099	...	-3.10106
7	-1.87291	0.656986	10.53740	-2.40270	0.037270	...	3.931002
8	-1.30728	0.989358	2.372823	-0.63531	0.627276	...	1.168063
9	-0.42159	0.412118	12.85550	-4.39144	1.412286	...	-4.55986
10	0.567324	-0.54402	11.38763	1.895743	0.643629	...	4.772541
11	1.417339	-0.99999	6.965379	-9.04229	0.754703	...	0.008559
12	1.920340	-0.53657	18.95395	-7.93374	0.241731	...	1.665364
13	1.953175	0.420167	12.80815	-5.27022	1.493793	...	-4.53836
14	1.507804	0.990607	1.293854	-6.83280	0.253613	...	4.246339
15	...	...	...	...	...	...	...

- Want to discover structure in the data
- Can plot variables pairwise:





Why dimensionality reduction? (1)

- Multi-dimensional data visualisation

data frame

	x_1	x_2	x_3	x_4	x_5	...	x_n
1	1.755165	0.841470	15.16638	-3.19198	0.385065	...	-4.67732
2	1.080604	0.909297	4.534220	1.999736	1.015282	...	1.621900
3	0.141474	0.141120	13.29099	-8.99336	1.324239	...	0.306199
4	-0.83229	-0.75680	18.03677	1.355381	0.496793	...	-1.57096
5	-1.60228	-0.95892	21.18148	-6.47613	0.031777	...	-1.55290
6	-1.97998	-0.27941	23.12537	-2.02521	0.475099	...	-3.10106
7	-1.87291	0.656986	10.53740	-2.40270	0.037270	...	3.931002
8	-1.30728	0.989358	2.372823	-0.63531	0.627276	...	1.168063
9	-0.42159	0.412118	12.85550	-4.39144	1.412286	...	-4.55986
10	0.567324	-0.54402	11.38763	1.895743	0.643629	...	4.772541
11	1.417339	-0.99999	6.965379	-9.04229	0.754703	...	0.008559
12	1.920340	-0.53657	18.95395	-7.93374	0.241731	...	1.665364
13	1.953175	0.420167	12.80815	-5.27022	1.493793	...	-4.53836
14	1.507804	0.990607	1.293854	-6.83280	0.253613	...	4.246339
15	...	...	...	...	...	...	...

- Want to discover structure in the data
- Can plot variables pairwise
- This won't uncover higher order structure in complex data
- E.g. $x_n = f(x_1, ..., x_{n-1})$

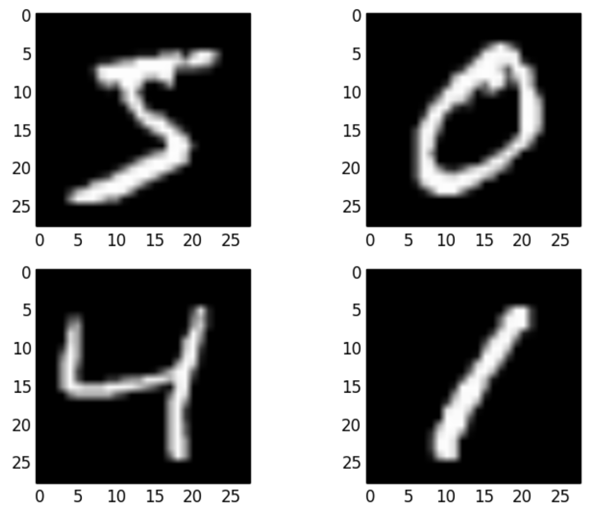
Why dimensionality reduction? (2)

- Data compression
- JPEG:

- MP3

Why dimensionality reduction? (3)

- Noise filtering / informative feature extraction



- Can we get rid of the uninformative components of the data?
- Machine learning on compressed representations:
  - Slimmer models
  - Faster training
  - Less overfitting

Outline

- Principal component analysis
- Autoencoders
- Variational autoencoders and generative modelling
- Persistent homology
- Mapper
- Fourier transform
- The (log) signature method

Principal Component Analysis

- Canonical dimension reduction technique

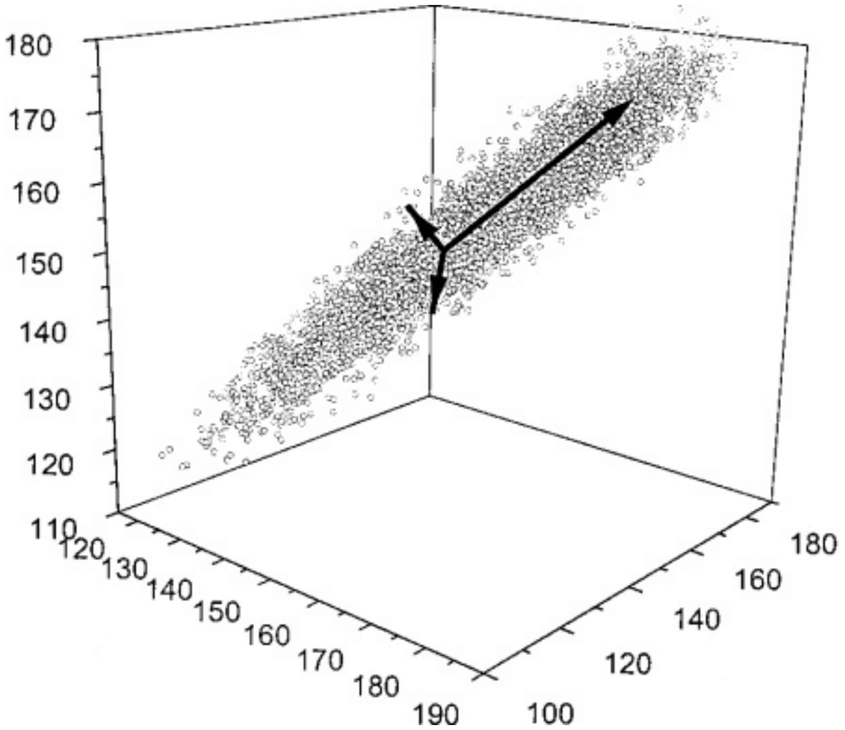
Principal Component Analysis

- Canonical dimension reduction technique

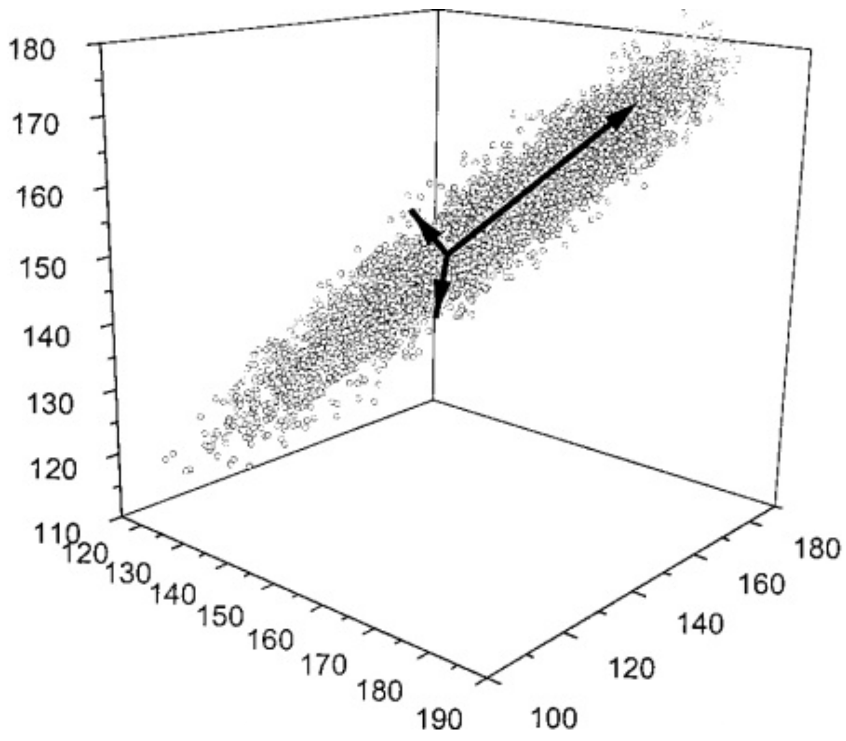
data frame

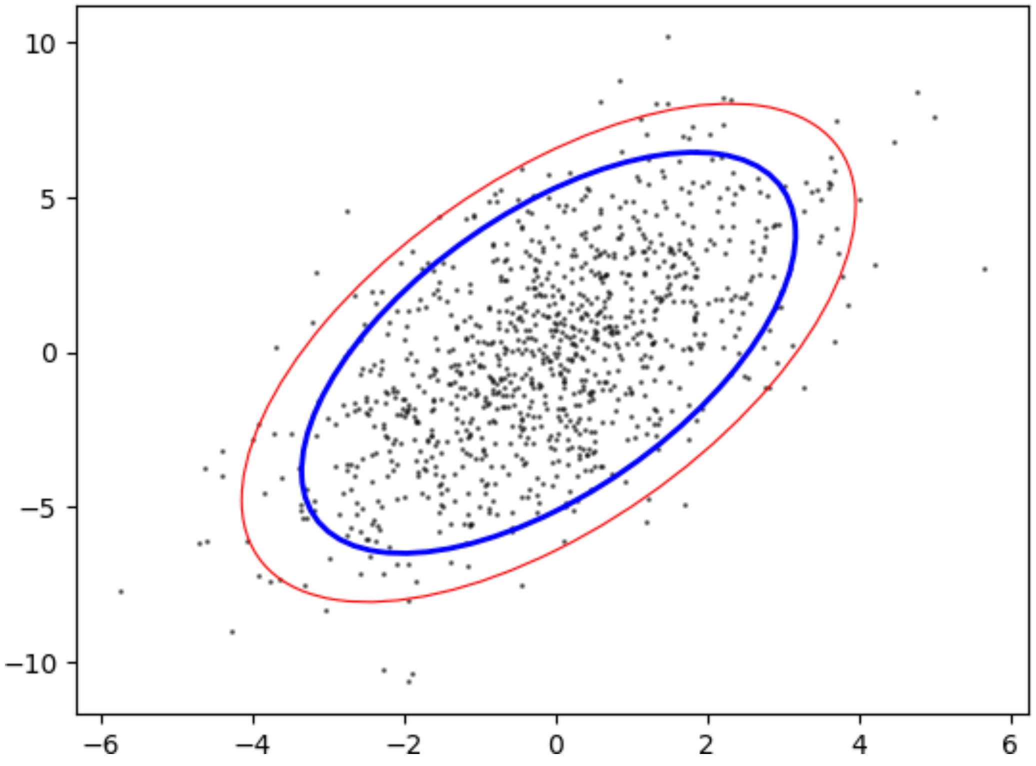
	x_1	x_2	x_3	x_4	x_5	...	x_n
1	1.755165	0.841470	15.16638	-3.19198	0.385065	...	-4.67732
2	1.080604	0.909297	4.534220	1.999736	1.015282	...	1.621900
3	0.141474	0.141120	13.29099	-8.99336	1.324239	...	0.306199
4	-0.83229	-0.75680	18.03677	1.355381	0.496793	...	-1.57096
5	-1.60228	-0.95892	21.18148	-6.47613	0.031777	...	-1.55290
6	-1.97998	-0.27941	23.12537	-2.02521	0.475099	...	-3.10106
7	-1.87291	0.656986	10.53740	-2.40270	0.037270	...	3.931002
8	-1.30728	0.989358	2.372823	-0.63531	0.627276	...	1.168063
9	-0.42159	0.412118	12.85550	-4.39144	1.412286	...	-4.55986
10	0.567324	-0.54402	11.38763	1.895743	0.643629	...	4.772541
11	1.417339	-0.99999	6.965379	-9.04229	0.754703	...	0.008559
12	1.920340	-0.53657	18.95395	-7.93374	0.241731	...	1.665364
13	1.953175	0.420167	12.80815	-5.27022	1.493793	...	-4.53836
14	1.507804	0.990607	1.293854	-6.83280	0.253613	...	4.246339
15	...	...	...	...	...	...	...

$\in \mathbb{R}^n$



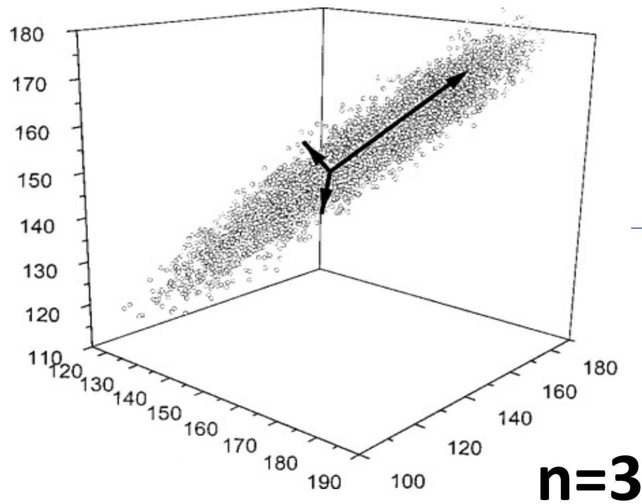
Principal Component Analysis



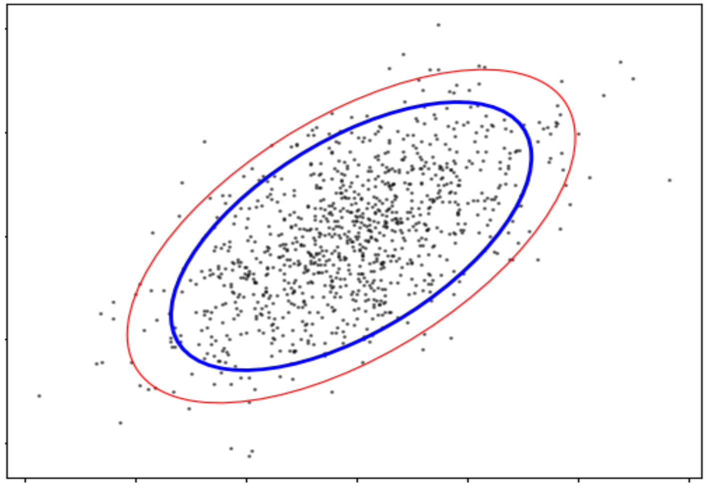


Principal Component Analysis

- **Input**: dimension $d < n$
- Projects the data onto a d dimensional plane (aka d coordinates)
  - Maximise the variance of output data
  - Minimise the distance of original datapoints to the plane



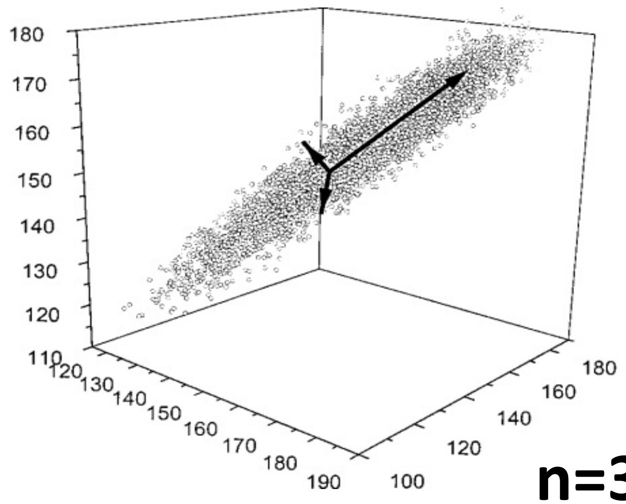
n=3



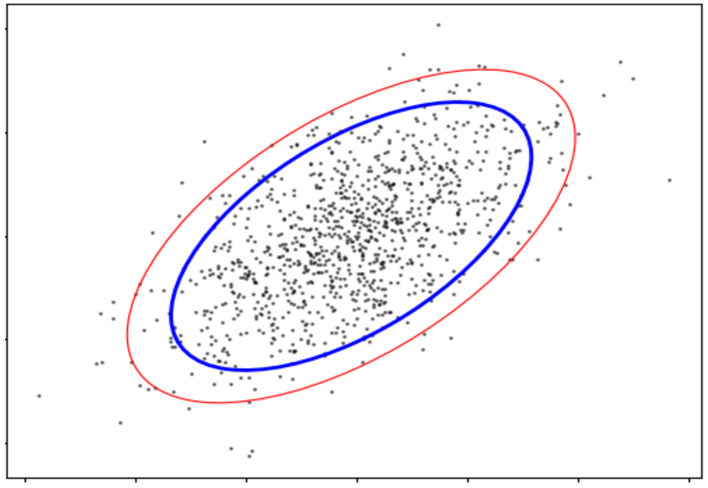
d=2

Principal Component Analysis

- **Input**: dimension $d < n$
- Projects the data onto a d dimensional plane (aka d coordinates)
  - Maximise the variance of output data
  - Minimise the distance of original datapoints to the plane
  - These two characterisations of PCA are equivalent!



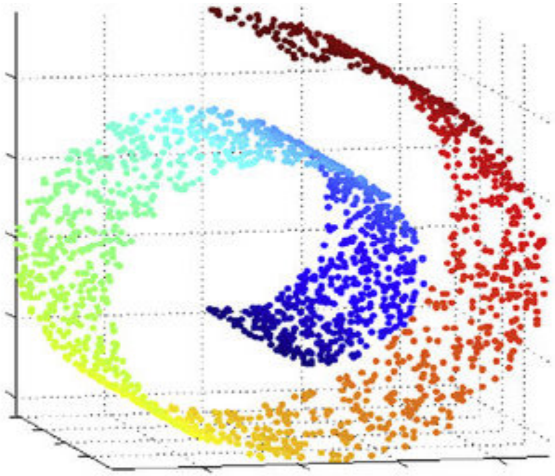
n=3

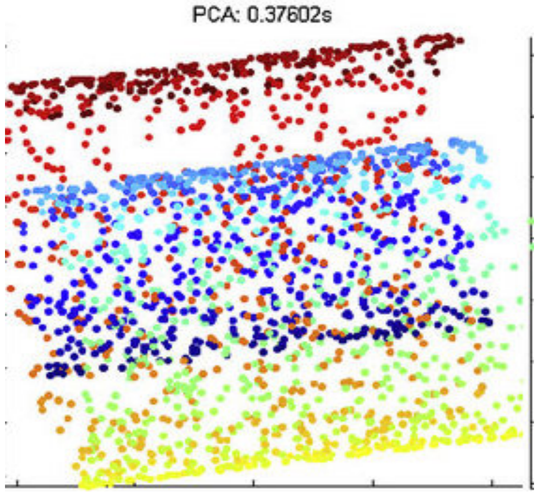


d=2

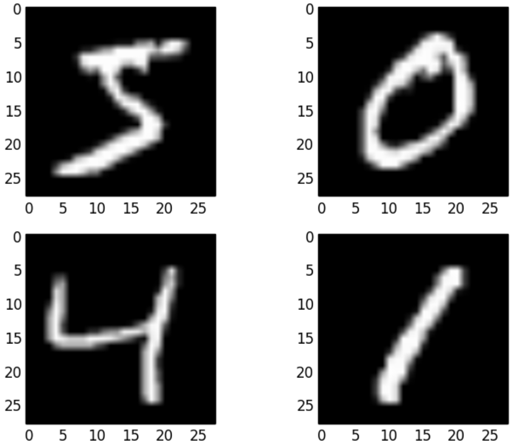
PCA: limitations

- The ‘Swiss’ roll

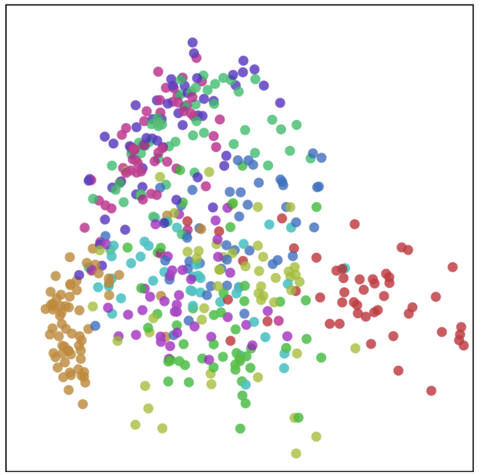




- MNIST digits



...

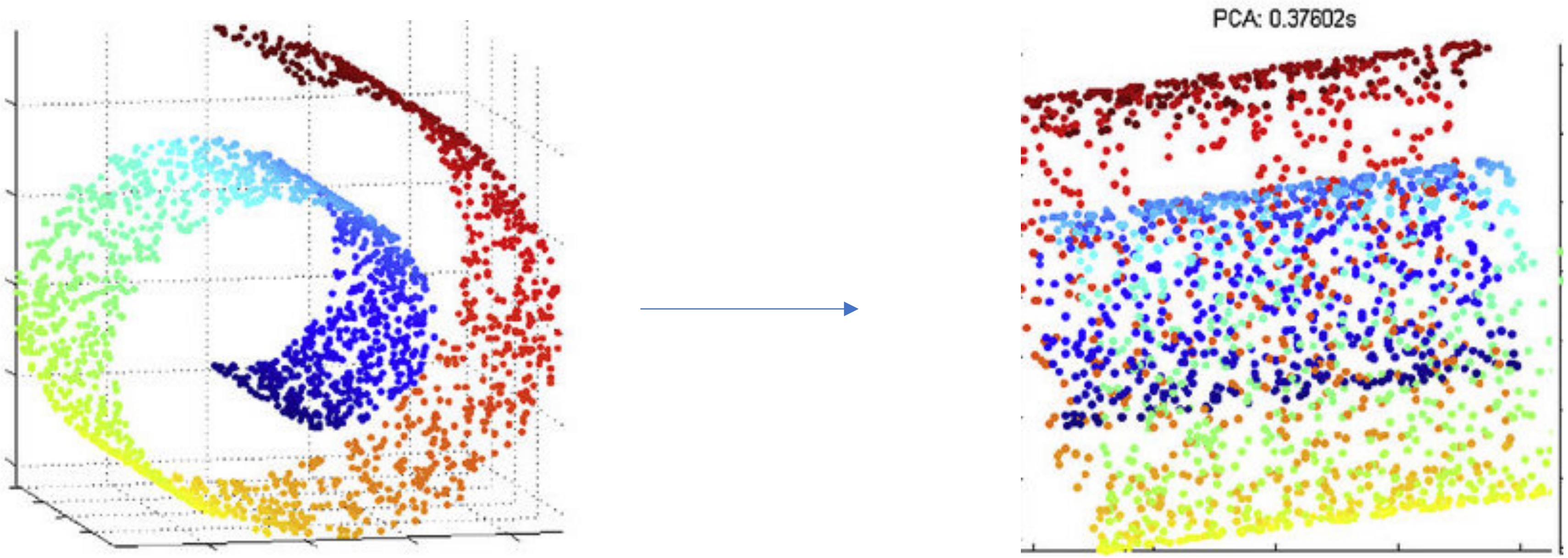


1 colour
= 1 digit

http://colah.github.io/posts/2014-10-Visualizing-MNIST/

PCA: Summary

- Runs fast (problem can be reduced to linear algebra)
- Result is easy to interpret
- Does not preserve the overall ‘shape’ of the data
- Dimension reduction is purely linear

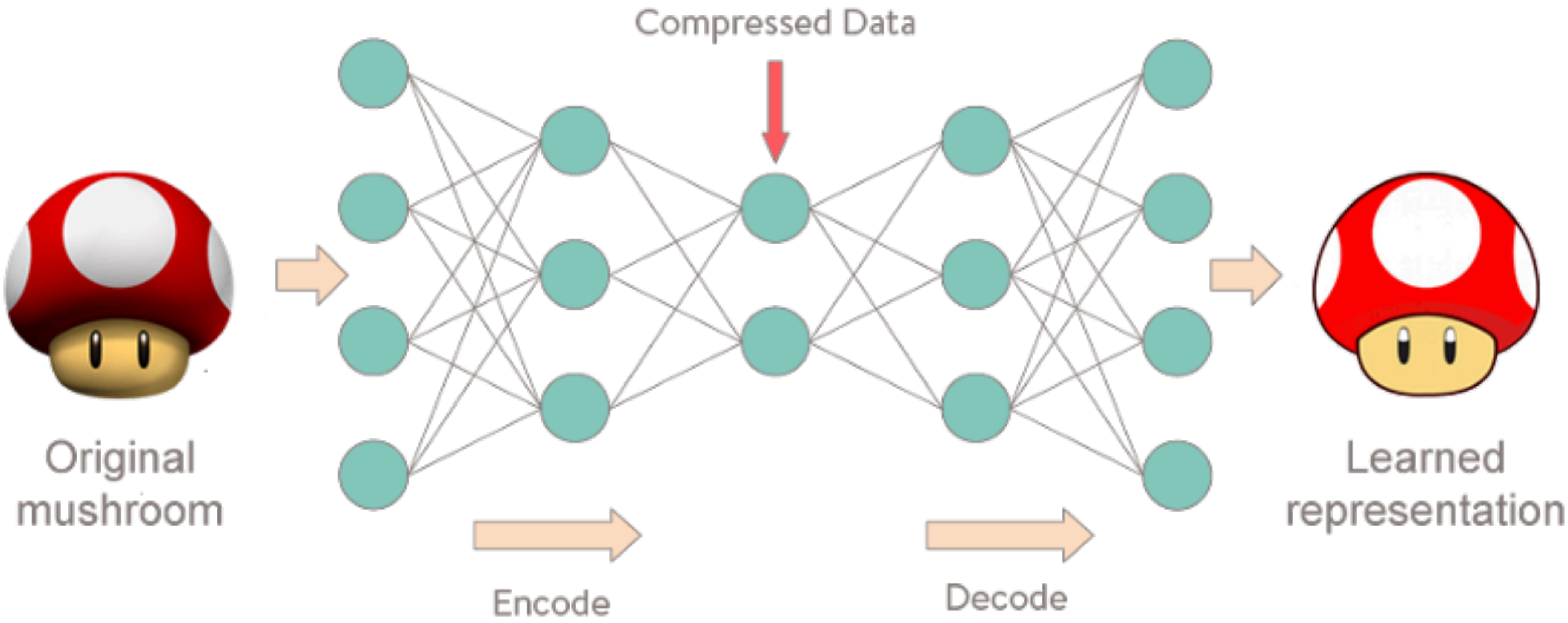


- In Python scikit-learn: see sklearn.decomposition.PCA

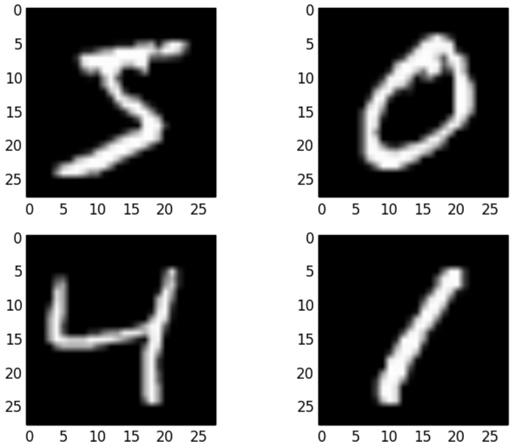
Autoencoders

- Neural network trained to approximate the identity function

$$f(x) = x \qquad dist(f(x), x) \searrow 0$$

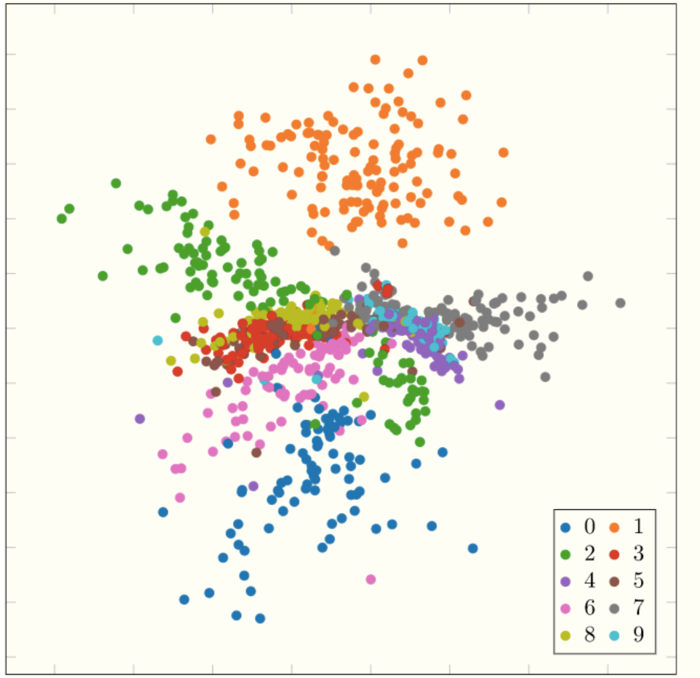


Autoencoder VS PCA: example

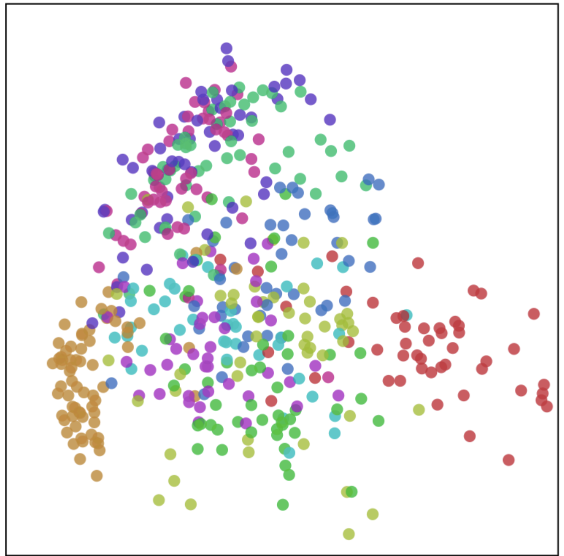


...

Autoencoder

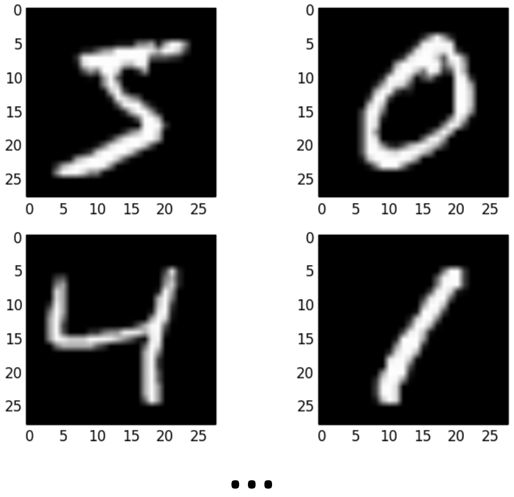


PCA

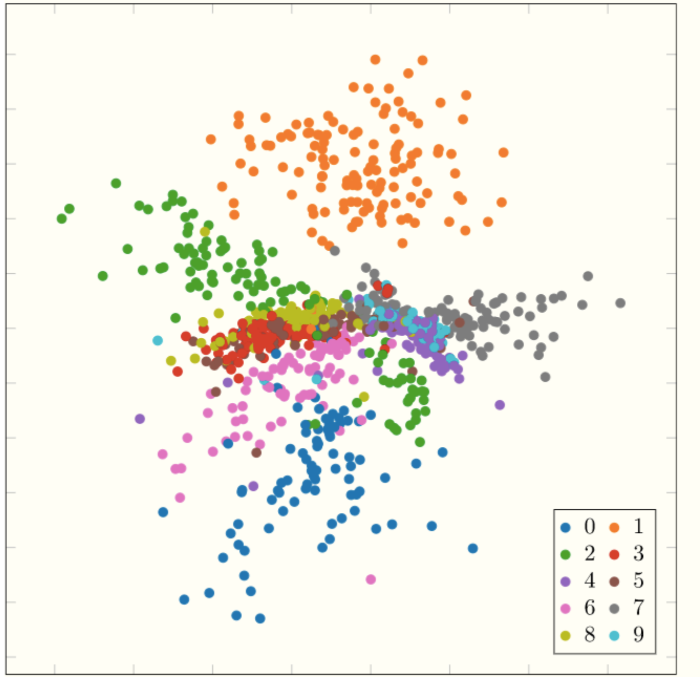


https://gertjanvandenburg.com/blog/autoencoder/

Autoencoder: feature extraction



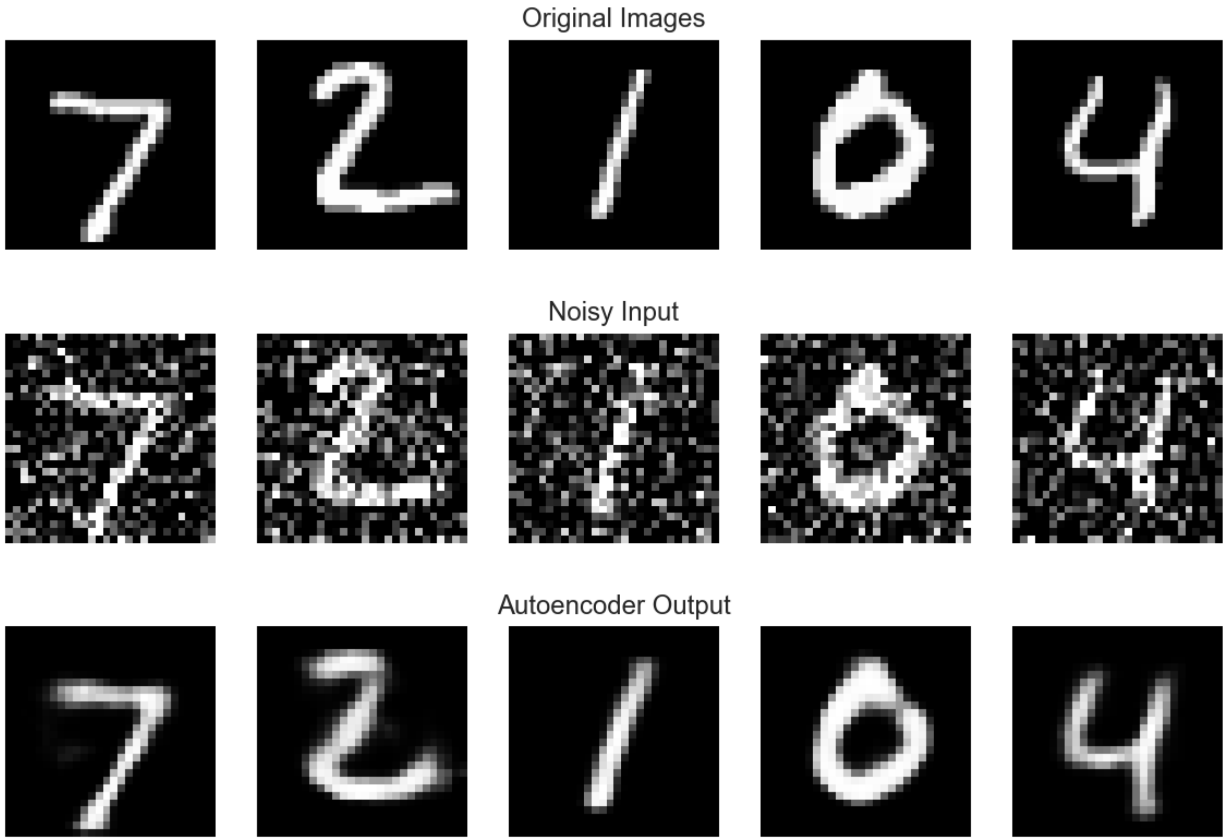
Autoencoder



- Resulting features can be used in machine learning, e.g.,
  - Classification
  - Regression
  - etc

https://gertjanvandenburg.com/blog/autoencoder/

Denoising autoencoder



For state of the art see:

- Generative adversarial denoising
- Variational autoencoder denoising

https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Autencoders

+ Data compression

+ Data denoising

+ Informative feature extraction

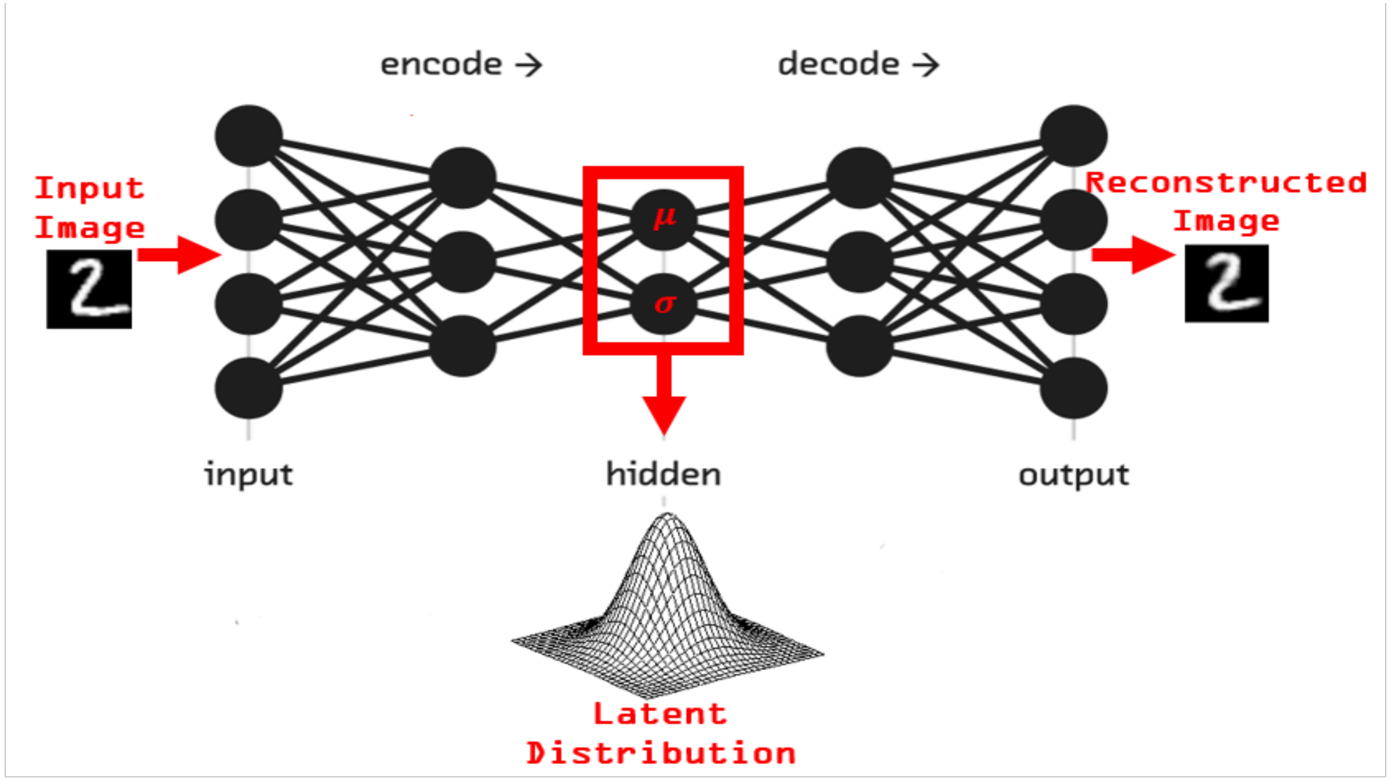
- Usually latent representation is not useful for visualisation

The intrinsic dimension of data

"You only truly understand some data, when you can capture the hidden causes that generated it, and generate the data yourself".

A complementary perspective on VAEs

VAE Motto: “An autoencoder, which does not overfit!”



Compresses each datapoint onto a Gaussian distribution

VAE intuition

- There are ‘few’ important variables in observable data *o*
- Let us call these ‘hidden’ variables. E.g.,
  - Orientation, depth, colour.
- Want to determine what these hidden states *s* are
- More generally, want to build a good model of the data

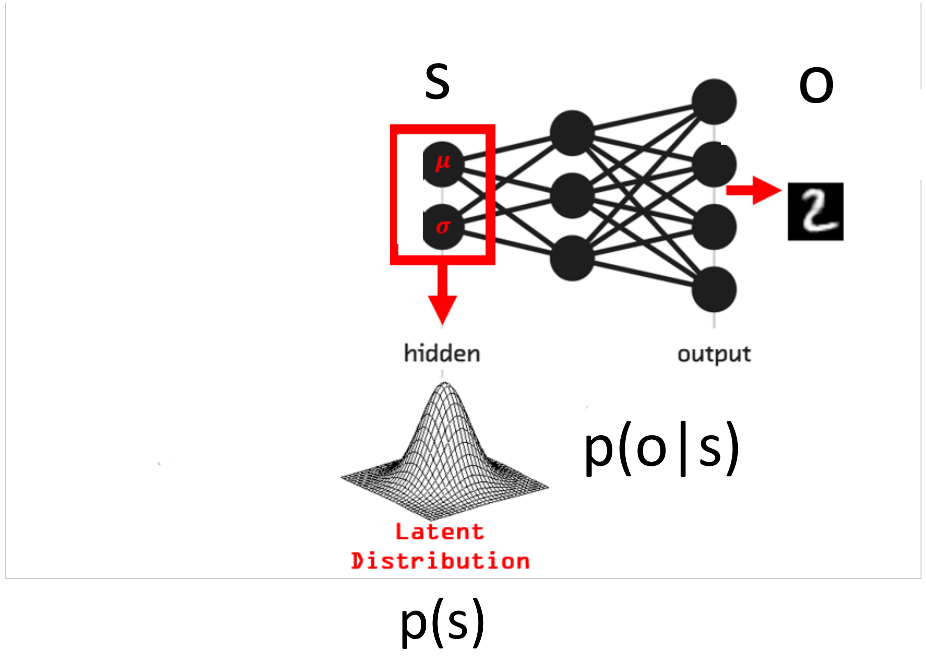
$$p(o|s)p(s)$$

A probability distribution over ‘hidden’ states and how these give rise to outcomes.

VAE intuition

- Build probability distribution over 'hidden' states and how these give rise to outcomes:

$$p(o|s)p(s)$$



- This allows to:
    - Create new data by sampling from *p(s)*
    - Compressing data via Bayes rule:

$$p(s|o) = \frac{p(o|s)p(s)}{p(o)}$$

Model evidence

The best model maximises model evidence *p(o)*

$$p(o) = \int p(o \mid s) p(s)\, ds$$

Problems:

- Model evidence is intractable to compute
- This means that we can't compress our data via

$$p(s|o) = \frac{p(o|s)p(s)}{p(o)}$$

*very important in advanced Bayesian machine learning

Workaround: Variational Bayes

The KL divergence measures the discrepancy between probability distributions.

$$0 \leq \mathrm{KL}(q(s)\,||\,p(s\,|\,o)) := \int_S q(s)\,\ln \frac{q(s)}{p(s\,|\,o)}\,ds$$

q= arbitrary distribution

$$= \int_S q(s)\,\ln \frac{q(s)p(o)}{p(o,s)} ds = \underbrace{\int_S q(s)\,\ln \frac{q(s)}{p(o,s)} ds}_{\text{-ELBO(q)}} \quad + \cancel{\int_S q(s)\,\ln p(o)\,ds} = \ln p(o)$$

Compression ⇔ Minimising KL ⇔ maximising ELBO

Optimal *q* = compressed representation

Maximising ELBO => good model

ELBO(q) $\leq$ ln *p(o)* (Jensen's inequality)

Moral of the story:

By maximising ELBO(q) = $\int_S q(s) \ln \frac{q(s)}{p(o,s)} ds$

- one obtains a good model of the data that captures the hidden dimensions
- efficient data compression.

Since <u>Model evidence = Model Accuracy – Complexity</u> => no-overfitting!

VAE Caveats

One fixes:

p(s) standard gaussian

p(o|s), q(s) gaussians parameterised by a neural network

This means that we optimise ELBO wrt the network parameters!

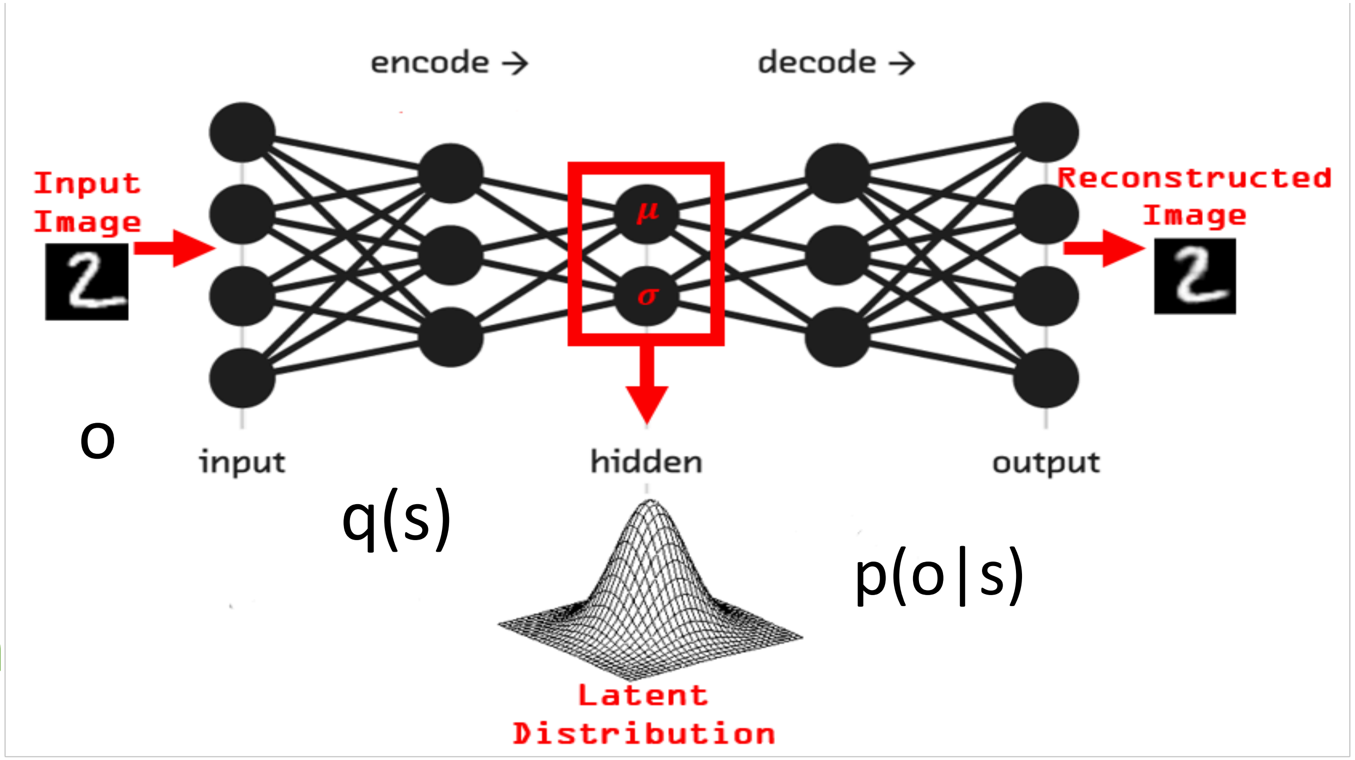
ELBO(q) = $$\int_S q(s)\ \ln \frac{q(s)}{p(o,s)} ds$$

Caveat: with these choices, ELBO is also intractable to compute.

One uses the ‘reparameterization trick’ to approximate it

… et voilà!

Variational autoencoders



+ Data compression
+ Data denoising
+ Informative feature extraction
+ Does not overfit
+ Data generation
- Usually latent representation is not useful for visualisation
- Technically more involved

The topology of data

Idea: real world data is often noisy, therefore one must look at its global structure.

Topology enables to analyse the structure of data, by using methods that are resistant to noise perturbations.

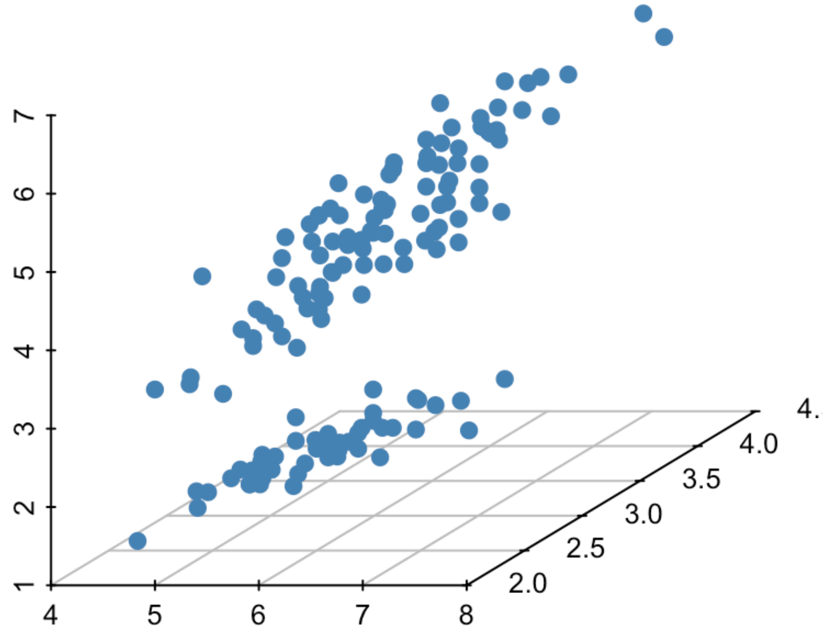
Setup

- Complex multidimensional data

data frame

	x_1	x_2	x_3	x_4	x_5	...	x_n
1	1.755165	0.841470	15.16638	-3.19198	0.385065	...	-4.67732
2	1.080604	0.909297	4.534220	1.999736	1.015282	...	1.621900
3	0.141474	0.141120	13.29099	-8.99336	1.324239	...	0.306199
4	-0.83229	-0.75680	18.03677	1.355381	0.496793	...	-1.57096
5	-1.60228	-0.95892	21.18148	-6.47613	0.031777	...	-1.55290
6	-1.97998	-0.27941	23.12537	-2.02521	0.475099	...	-3.10106
7	-1.87291	0.656986	10.53740	-2.40270	0.037270	...	3.931002
8	-1.30728	0.989358	2.372823	-0.63531	0.627276	...	1.168063
9	-0.42159	0.412118	12.85550	-4.39144	1.412286	...	-4.55986
10	0.567324	-0.54402	11.38763	1.895743	0.643629	...	4.772541
11	1.417339	-0.99999	6.965379	-9.04229	0.754703	...	0.008559
12	1.920340	-0.53657	18.95395	-7.93374	0.241731	...	1.665364
13	1.953175	0.420167	12.80815	-5.27022	1.493793	...	-4.53836
14	1.507804	0.990607	1.293854	-6.83280	0.253613	...	4.246339
15	...	...	...	...	...	...	...

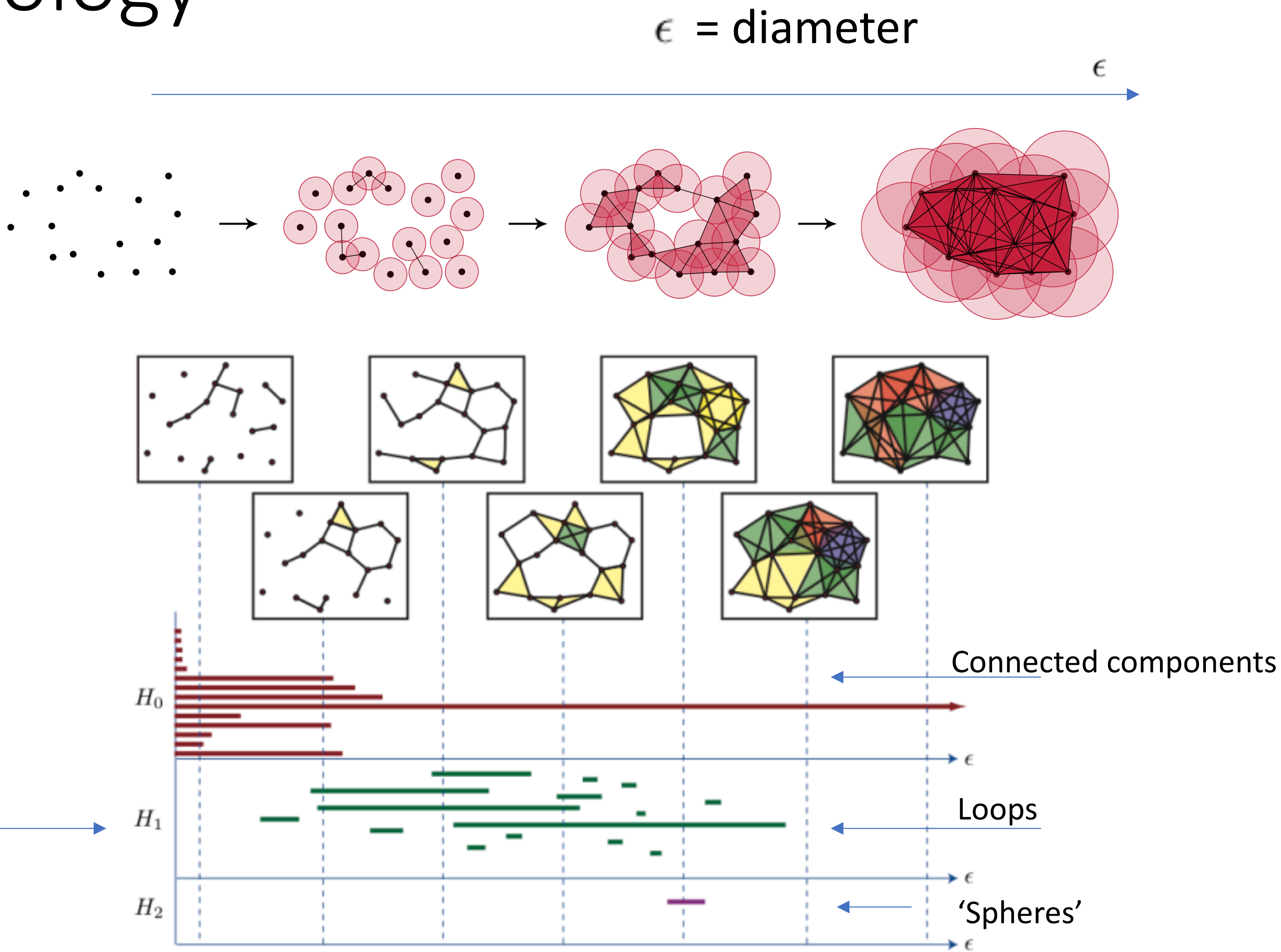
$\in \mathbb{R}^n$



Persistent homology

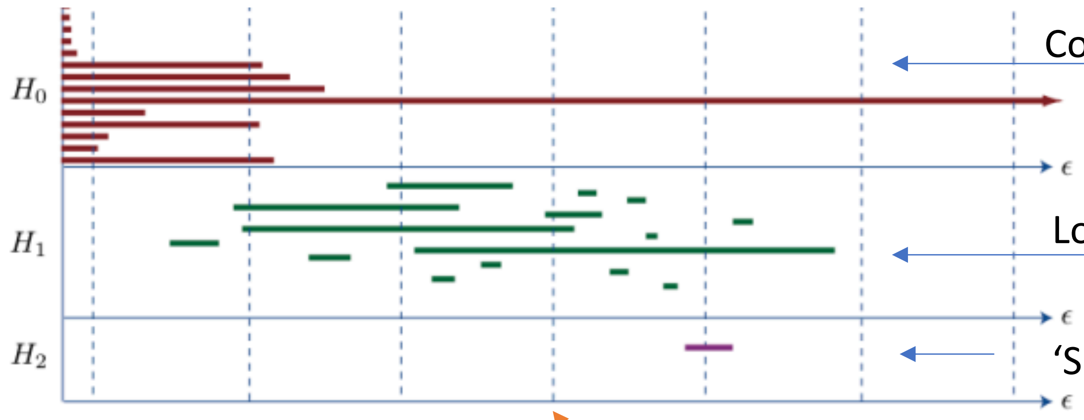
Illustrations:

Persistence barcodes



Persistence diagram

Persistence barcodes



Connected components

Loops

'Spheres'

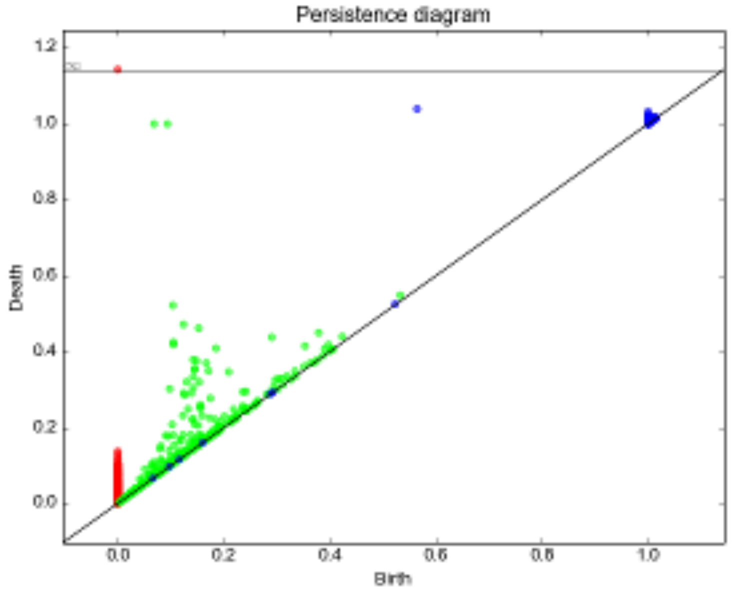
Persistence diagram:

Each dot is (birth, death)

Of a:

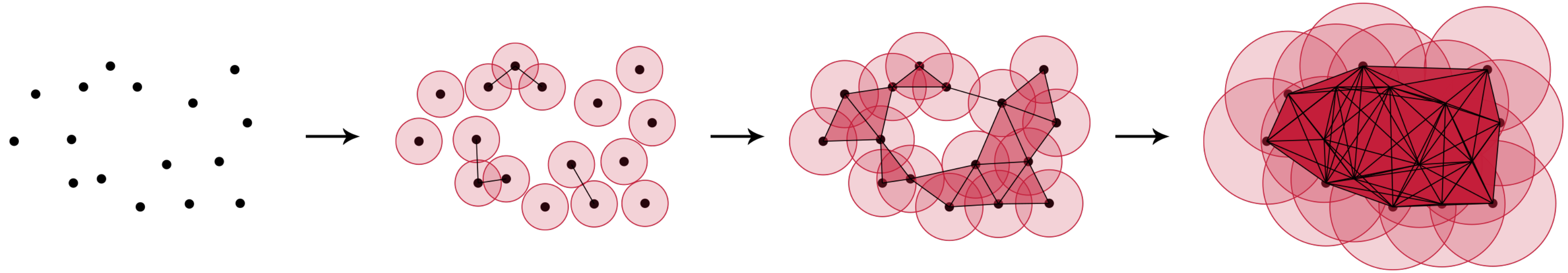
- Connected component
- Loop
- Sphere

Depending on colour



Beware these are not from the same dataset!

Persistence homology: algorithm



From ϵ = 0 to D

Draw spheres around each datapoint of diameter ϵ

Link datapoints when spheres intersect

Let a k-clique be a fully connected set of k+1 points that is not within a k+2 fully connected set of points

For each k from 0 to N: record the number of k-cliques

Output: birth and death of each k-clique as a function of epsilon

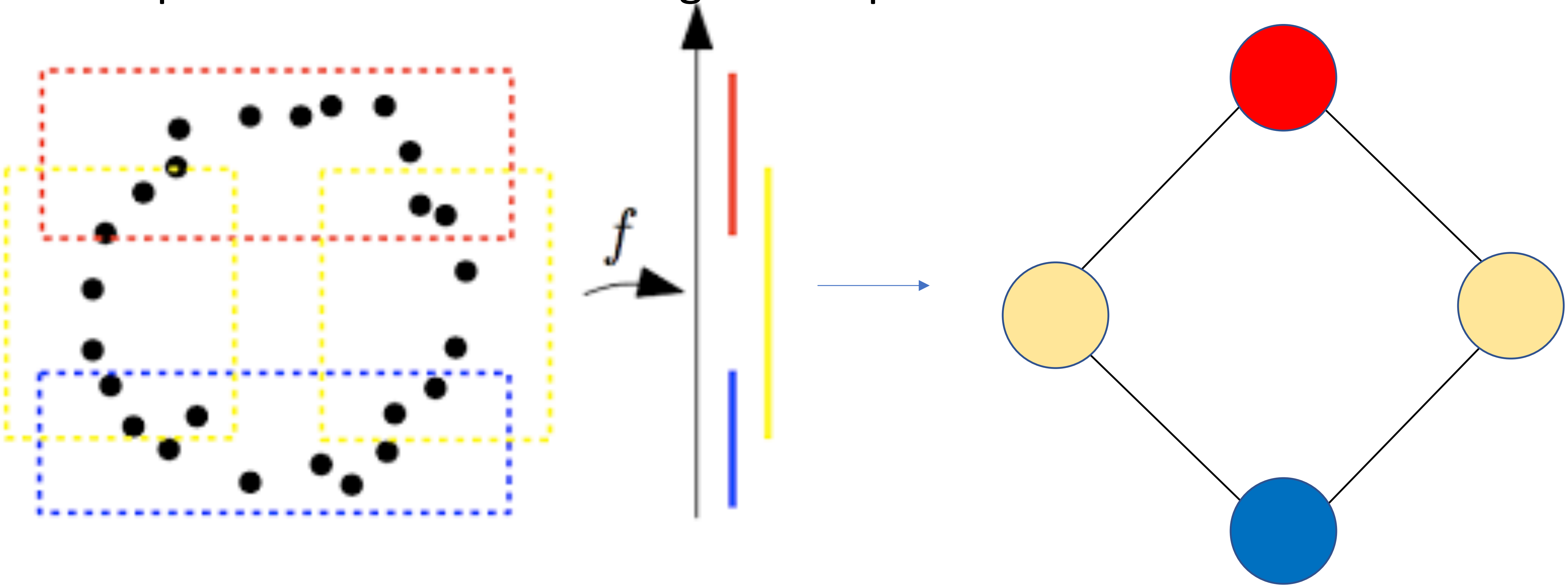
Persistence homology: summary

+ Informative feature extraction
+ Easy to interpret
+ Useful exploratory data analysis technique
+ Resistant to noise in data
- Computationally intensive

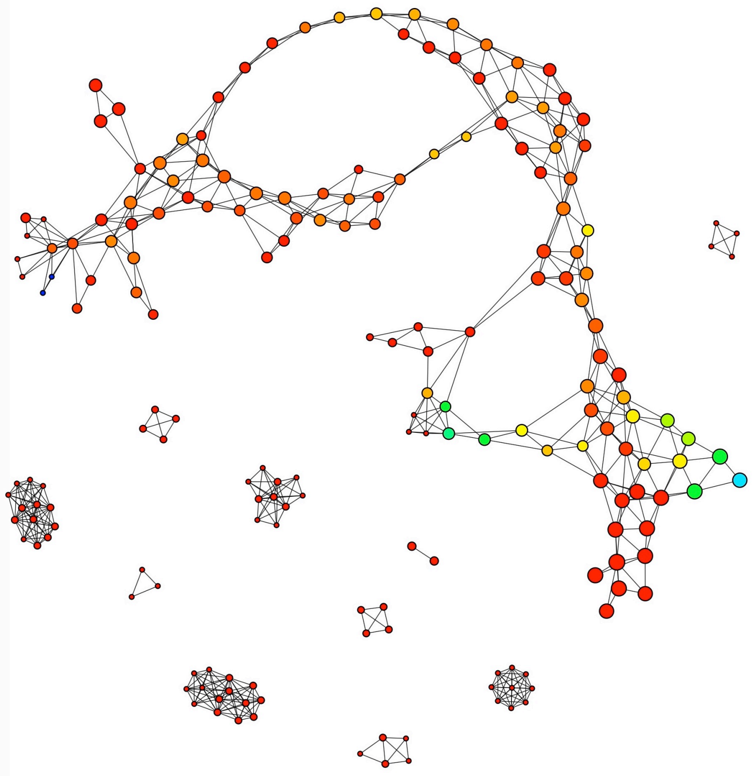
Implementations: see Python module Giotto

Mapper

- Visualising the 'shape' of data
- Output: network summarising the shape of the data



Mapper: algorithm

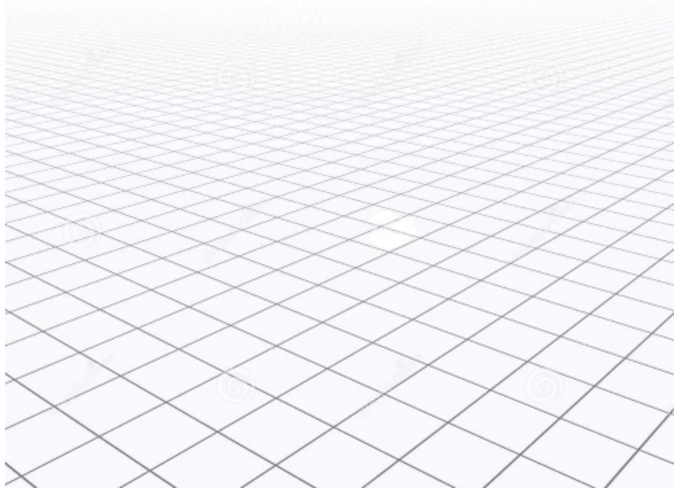


- Input:
  - Function of data *f (lens)*
  - Bin size
  - Bin percentage % overlap
- For each bin
  - Take the datapoints with get mapped into the bin by *f*
  - Perform clustering algorithm on such data
  - Create a node for each cluster
  - Link node to other existing nodes if they share at least a datapoint
- Output: network summarising the shape
- Geometric structure of visual output doesn't matter – connections only

Mapper: lens functions of data

- Note: this algorithm works for non-scalar functions *f*!
  - Need to bin target space and perform algorithm
  - Curse of dimensionality

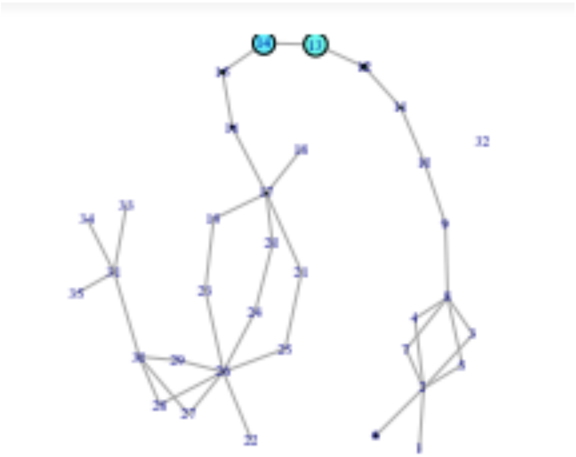
- There are many possible (and natural choices) for *f*
  - Projection onto an important variable
  - Distance to the center of mass
  - PCA
  - Autoencoder
  - …
- Intuitively, *f* must disentangle the data well



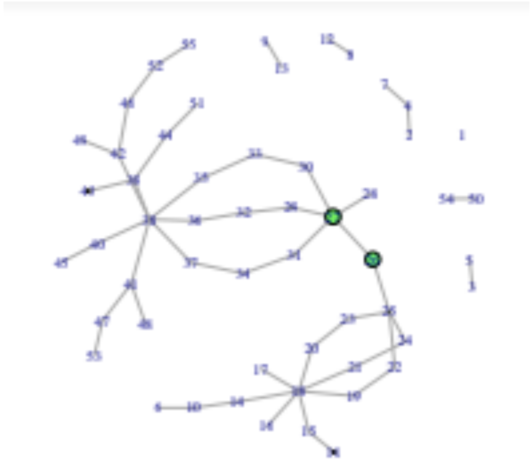
If target space is the plane

Mapper: example

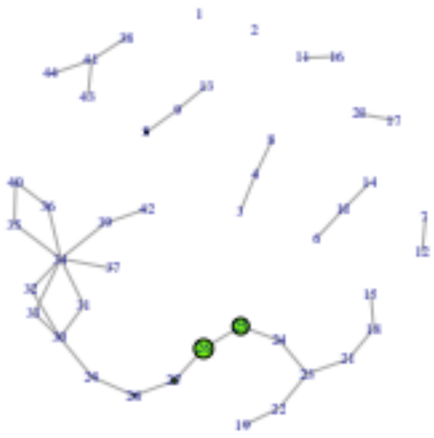
- Patterns in tree growth and meteorological data



Lens 1



Lens 2



Lens 3

- Conclusion: there is no network that rules them all!

Mapper: summary

+ Visualise data through different facets
+ Useful exploratory data analysis technique
+ Highly resistant to noise in data
- Computationally intensive
- Output may vary greatly with respect to input variables
    - One has to run in many different ways and interpret the globality of outputs

Python: see module Giotto

R: see package TDAmapper

Thank you for your attention