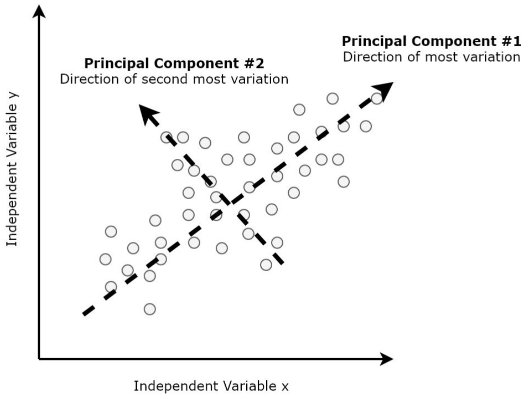
Autoencoders and Generative Networks

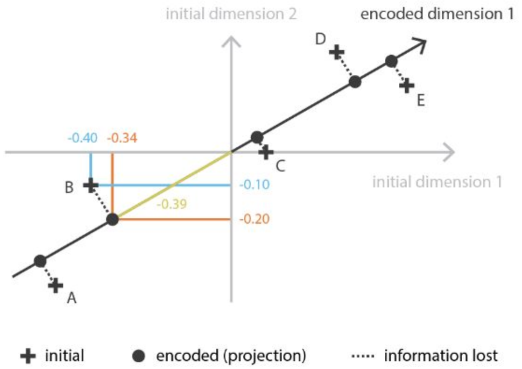
Computing Methods for Experimental Physics and Data Analysis

Andrea.Rizzi@unipi.it

Dimensionality reduction task

- We have as input N numbers, we want to transform them to M numbers, with M < N, that contains as much information as possible of the initial numbers
- PCA is a possible way to do this dimensionality reduction
    - Do PCA, and only save the coordinate along the 1st (or first X) axis

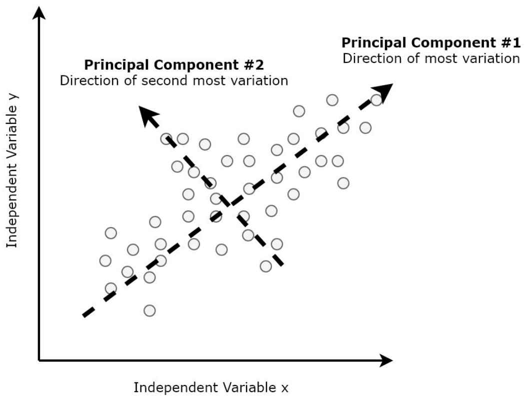


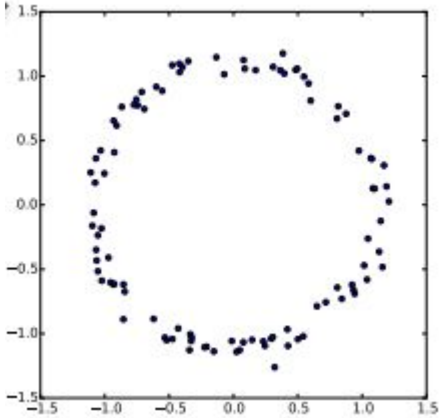


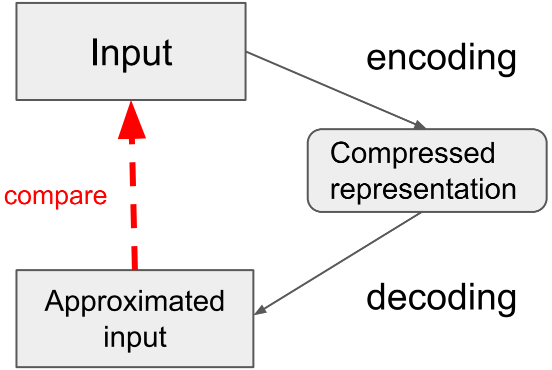
Point	Initial	Encoded	Decoded
A	(-0.50, -0.40)	-0.63	(-0.54, -0.33)
B	(-0.40, -0.10)	-0.39	(-0.34, -0.20)
C	(0.10, 0.00)	0.09	(0.07 0.04)
D	(0.30, 0.30)	0.41	(0.35, 0.21)
E	(0.50, 0.20)	0.53	(0.46, 0.27)

Dimensionality reduction task

- We have as input N numbers, we want to transform them to M numbers, with M < N, that contains as much information as possible of the initial numbers
- PCA is a possible way to do this dimensionality reduction
  - Do PCA, and only save the coordinate along the 1st (or first X) axis
- There are (even simple) data distributions where PCA is not going to help
- **Autoencoders** can help with this task
  - Encode, decode, define a loss based on input vs output difference

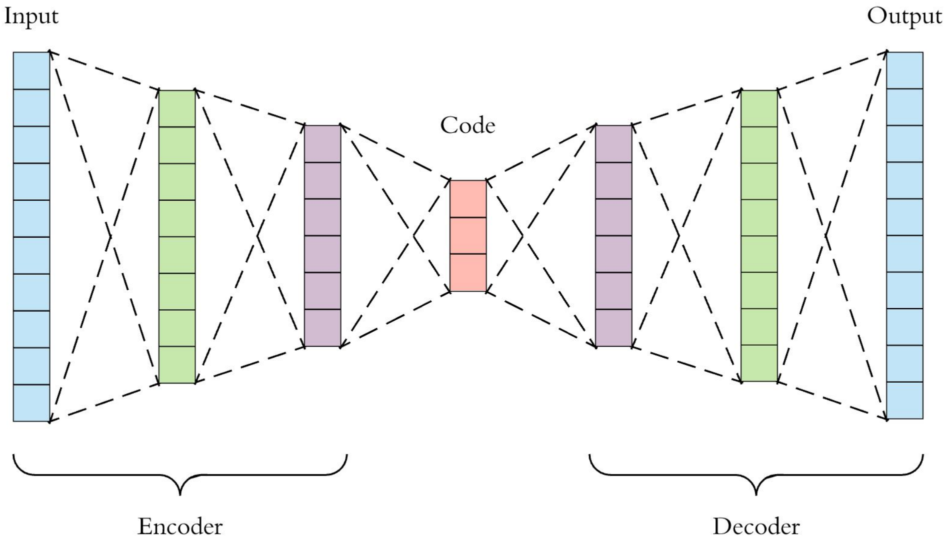






Autoencoder example

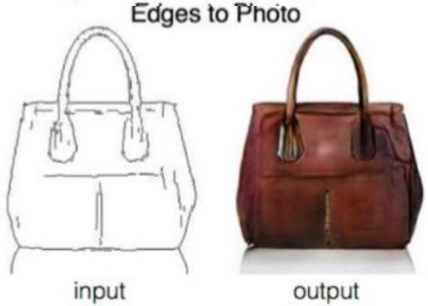
- Create a “bottleneck” to reduce the information
  - A layer with fewer nodes than the input and output
- Define a loss by comparing Output to Input
  - This is an unsupervised algorithm!
  - No need to have labels
- The content of the bottleneck layer is the “compressed representation” or “code”

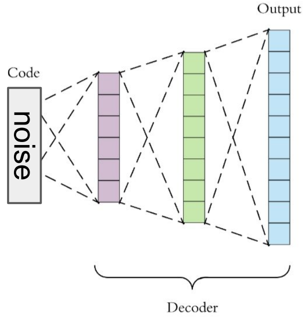


How can we do it in Keras?
model.fit(X,X) <= the target is the input

Generative models

- We may want to generate new samples from a distribution we learned
  - Generating fake images of animals, actors, dresses, etc..
  - E.g. for creating simulations of LHC events
- In many case we want to “conditionally” generate new samples
  - Generate a full picture of a product from a hand made sketch
  - Create color image from B&W
  - Generate realistic “reconstructed LHC event” from generated quarks and leptons
- Two powerful methods
  - With Autoencoders:
    - Train an autoencoder on the data you want to mimic
    - Take the trained “decoder” and start decoding a vector of random noise
    - This works best with so called “Variational Autoencoders”
      - Latent space representing “mean and variance” of the learned features (tutorial)
  - With Generative Adversarial Networks





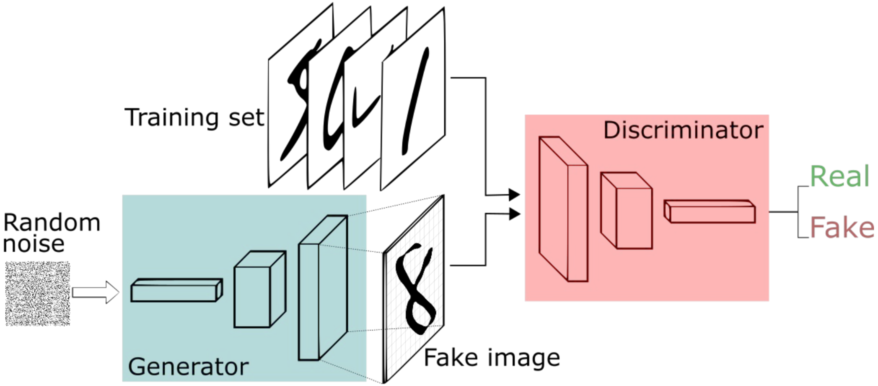
Generative Adversarial Networks

GAN works with two independent networks:

- A generator
- A discriminator

The two networks “compete” against each other

- The **discriminator** tries to distinguish samples of the original training dataset from samples generated by the **generator**
- The generator tries to create samples starting from random noise



- For the **discriminator** training we use a mixture of real and generated samples
  - No labels are needed in the original sample as we can label “0” vs “1” the samples coming from generator vs original
- **Generator** loss is controlled by the **discriminator** being able to recognize the fake

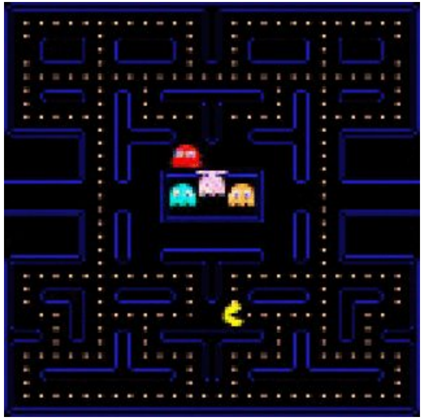
GAN progress

2014: “dogs with three heads”

2018: coherent generation of faces

See also https://thispersondoesnotexist.com/

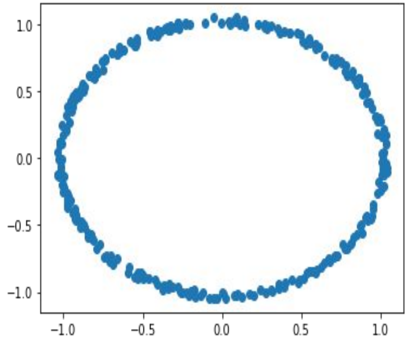
2019: re-create a **playable** video game just by looking at videos of an existing one (so far PacMan)

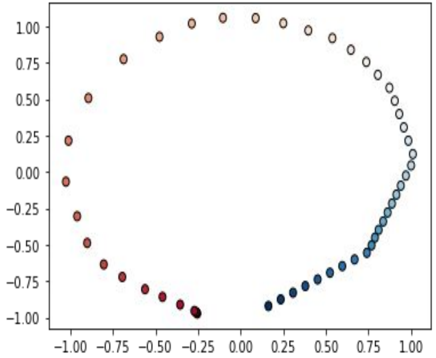


2021: GANTheftAuto

Assignment 5

- Create an autoencoder to compress a ring like distribution
  - As the input is 2 dimensional, can only be 1 number
- Steps for the exercise
  - Generate 1000 events in a ring with 0.95<R<1.05
  - Create an autoencoder with
    - An input with dimension 2
    - 2 encoding hidden layers with ~50 nodes per layer
    - A latent layer with a single node (sigmoid output) <= give it a name to later reuse
    - 2 decoding hidden layers with ~50 nodes per layer
    - An output with 2 nodes
  - Reuse the latent layer to create two models
    - Encoder (i.e. Input -> latent)
    - Decoder (i.e. latent -> output)
  - Make few tests like:
    - How are (0,1) and (1,0) mapped to “the code”
    - If we scan the code from 0 to 1, how does it map to (x,y)



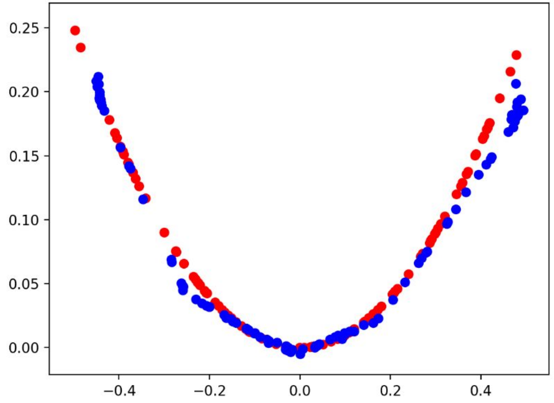


Assignment 6

Follow the tutorial at

https://machinelearningmastery.com/how-to-develop-a-generative-adversarial-network-for-a-1-dimensional-function-from-scratch-in-keras/

Nicely following a similar approach to what we had in this lectures: start from something simple and under your complete control instead of loading the usual ML datasets (MNIST, Iris, etc..)



- Generate points in a x1,x2 plane following a known function
- Ask the GAN to produce “samples” that look like our dataset (i.e. follow the same distribution)