Sapienza University of Rome

Master in Artificial Intelligence and Robotics
Master in Engineering in Computer Science

Machine Learning

A.Y. 2020/2021

Prof. L. Iocchi, F. Patrizi

15. Dimensionality reduction

L. Iocchi, F. Patrizi

with contributions from Valsamis Ntouskos

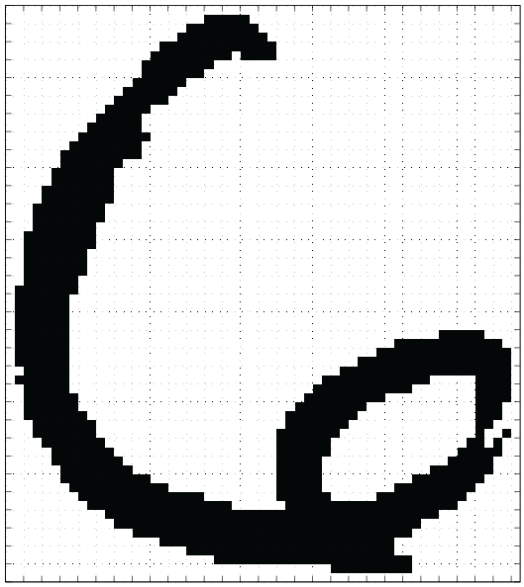
Overview

- Continuous latent variables
- Principal Component Analysis (PCA)
- Probabilistic PCA
- Non-linear latent variable models
- Autoencoders
- Generative Models

Reference
C. Bishop. Pattern Recognition and Machine Learning. Chapter 12.

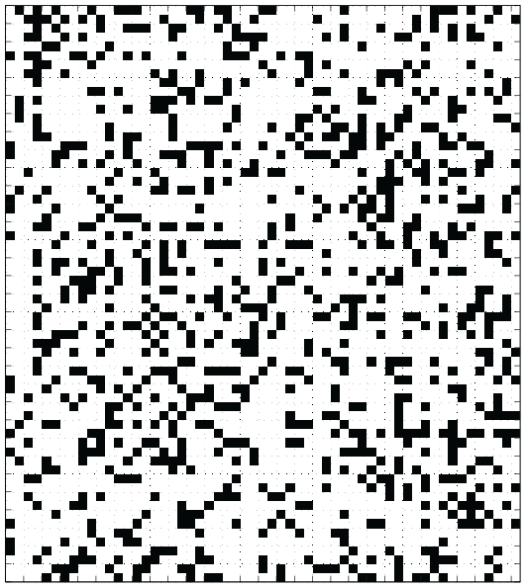
Latent Variables

Example
USPS dataset: 64 rows by 57 columns



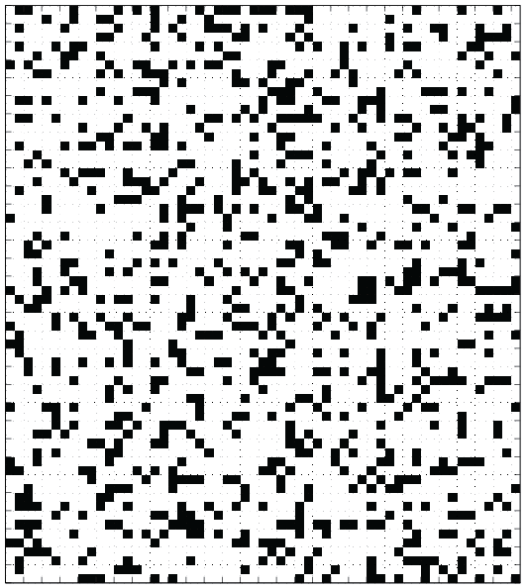
Latent Variables

Data space contains more than just digits



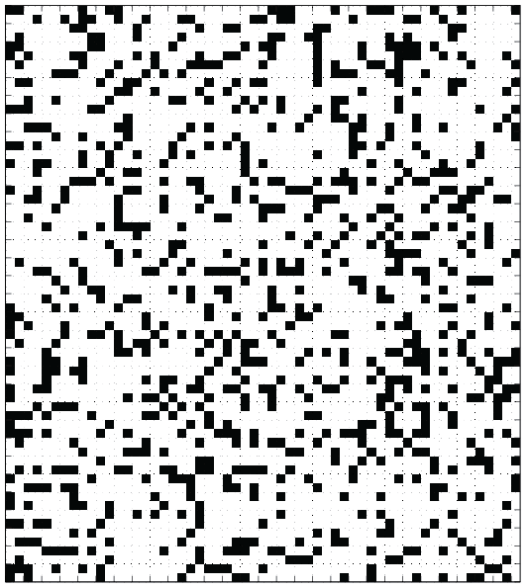
Latent Variables

Data space contains more than just digits



Latent Variables

Data space contains more than just digits

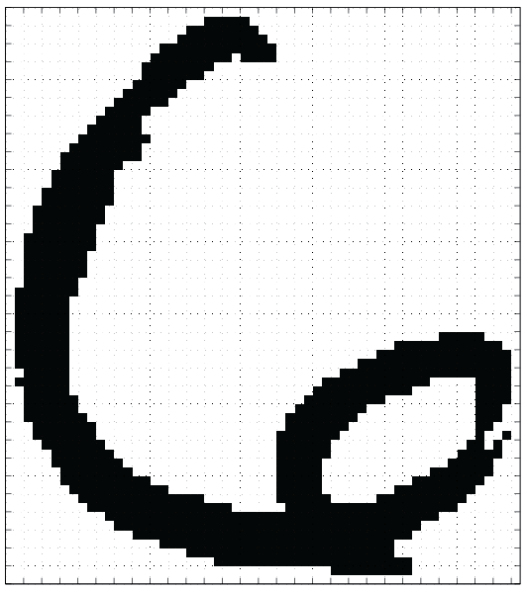


Latent Variables

Prototype rotation (1 dof transformation)

640x480x3

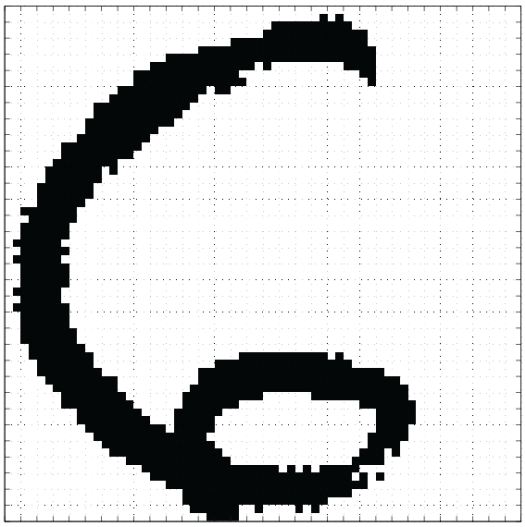
$X \subseteq \mathbb{R}^{W \times H \times C}$



we can represent the image in a smaller dimensional space

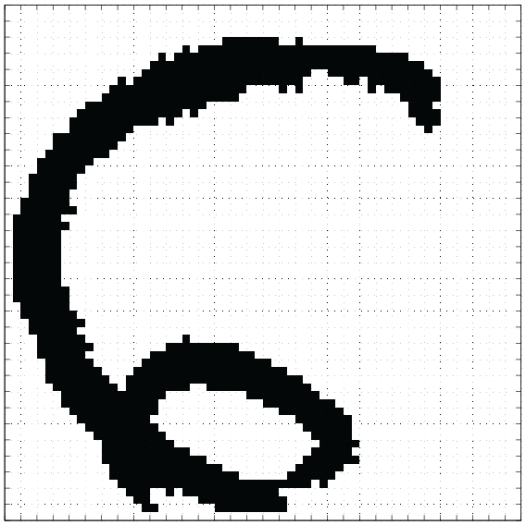
Latent Variables

Prototype rotation (1 dof transformation)



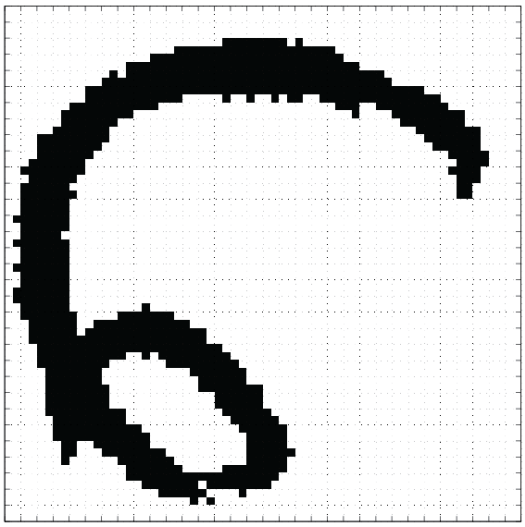
Latent Variables

Prototype rotation (1 dof transformation)



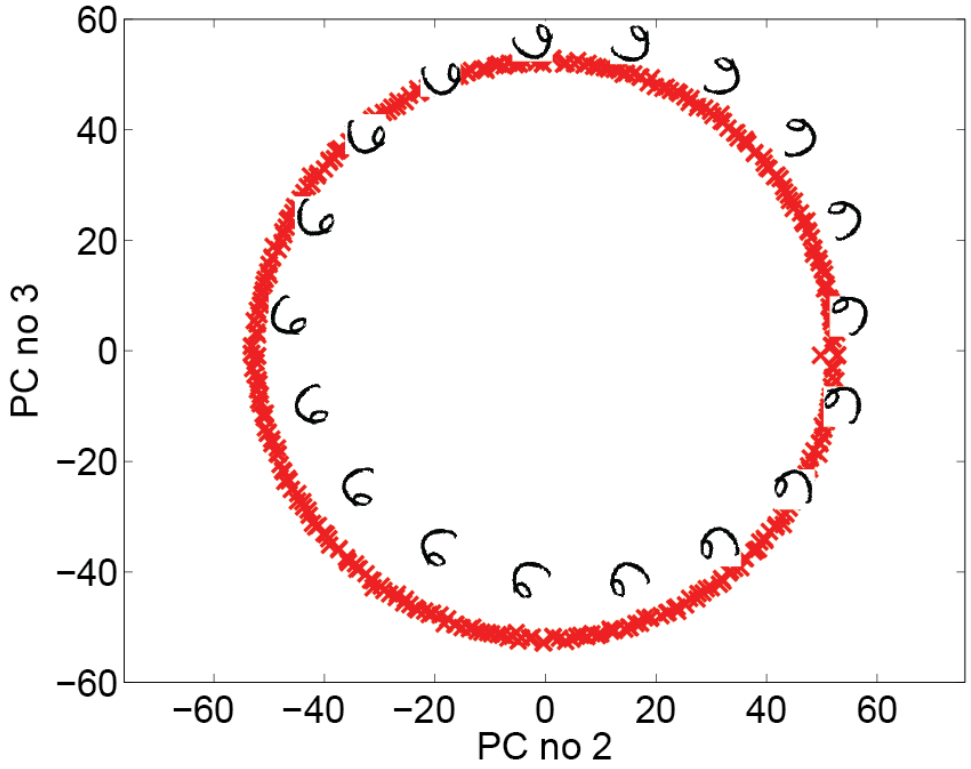
Latent Variables

Prototype rotation (1 dof transformation)



Latent Variables

Manifold

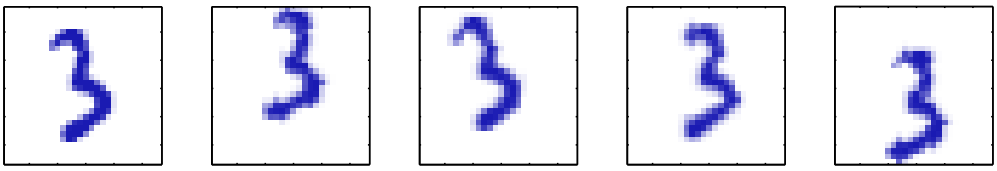


Latent Variables

Each of the resulting images is represented by a point in the 100 x 100 = 10 000-dimensional data space. However, across a data set of such images, there are only three degrees of freedom of variability, corresponding to the vertical and horizontal translations and the rotations. The data points will therefore live on a subspace of the data space whose intrinsic dimensionality is three.

Another example

the manifold will be nonlinear because. for instance. if we translate the digit past (oltre) a particular pixel, that pixel value will go from zero (white) 10 one (black) and back to zero again. which is clearly a nonlinear function of the digit position



the translation and rotation parameters are latent variables because we observe only the image vectors and are not told which values of the translation or rotation variables were used to create them.

3 degrees of freedom transformation (2D translation + rotation)

it means that the actual variability of the dataset is just 3 and not WxHxD=640x480x3

For real digit image data, there will be a further degree of freedom arising from scaling.

Latent Variables

For data with 'structure'*

- We expect fewer distortions than dimensions
- data live on a lower dimensional manifold

Conclusion: deal with high dimensional data by looking for lower dimensional embedding

*from Raquel Urtasun's slides

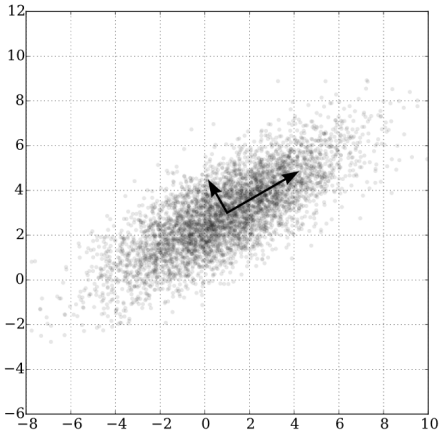
Principal Component Analysis

Principal Component Analysis (PCA) is a widely used technique for various tasks as

that maximize the variance with respect to the distribution

- dimensionality reduction
- data compression (lossy)
- data visualization
- feature extraction

there are 2 commonly used definitions of PCA that give rise to the same algorithm. PCA can be defined as the orthogonal projetion of the data onto a lower dimensional linear space, known as the principal subspace, such that the variance of the projected data is maximized. it can be defined as the linear projection that minimizes the average projection cost, defined as the mean squareed distance between the data points and their projection



PCA - Variance Maximization

Given data $\{\mathbf{x}_n\} \in \mathbb{R}^D$

dim of the original input space before the reduction

M: dim of the reduced space

our goal is to project the data onto a space having dimensionality M<D while maximizing the variance of the projected data

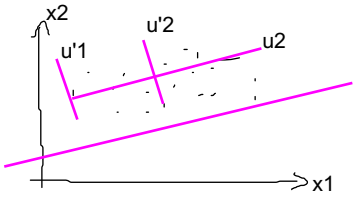
Goal: Maximize data variance after projection to some direction $\mathbf{u}_1$

we assume that the value of M is given

Projected points:

We can define the direction of this space using a D-dimensional vector U1

$\mathbf{u}_1^T \mathbf{x}_n$



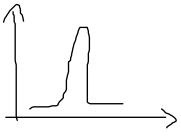
Each data point xn is then projected onto a scalar value u1^(T)xn

Note: $\mathbf{u}_1^T \mathbf{u}_1 = 1$

we want to pass from R^2 to R^1

we project the points on the line u1. However u'1 (ortogonal to u2)is not a good choice because the projection are near to zero.

projection of u2



projection on u'2

PCA - Variance Maximization

Mean value of data points:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

Data-centered matrix $\mathbf{X}$ $(N \times D)$:

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}_1 - \bar{\mathbf{x}})^T \\ \cdots \\ (\mathbf{x}_n - \bar{\mathbf{x}})^T \\ \cdots \\ (\mathbf{x}_N - \bar{\mathbf{x}})^T \end{bmatrix}$$

PCA - Variance Maximization

Mean of projected points:

$$\mathbf{u}_1^T\bar{\mathbf{x}}$$

Variance of projected points:

$$\frac{1}{N}\sum_{n=1}^{N}[\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\bar{\mathbf{x}}]^2 = \mathbf{u}_1^T S\mathbf{u}_1$$

with S $(D \times D)$ convariance matrix of the dataset

$$S = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T = \frac{1}{N}\mathbf{X}^T\mathbf{X}$$

$(\mathbf{x}_n - \bar{\mathbf{x}})$: difference between each point in the dataset and the corresponding mean

PCA - Variance Maximization

Problem definition

Maximize the projected variance

$$\max_{\mathbf{u}_1} \ \mathbf{u}_1^T S \mathbf{u}_1$$

subject to constraint $\mathbf{u}_1^T \mathbf{u}_1 = 1$

Equivalent to unconstrained maximization with a Lagrange multiplier

$$\max_{\mathbf{u}_1} \ \mathbf{u}_1^T S \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

PCA - Variance Maximization

the solution will be given in terms of eigenvalues and eigenvectors of the matrix S that contains the variance of the data in the dataset

Solution

Setting derivative w.r.t. $\mathbf{u}_1$ to zero we have

$$S\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

highest eigenvalue

eigenvector related to highest eigenvalue

$\mathbf{u}_1$ must be an eigenvector of S

Left-multiplying by $\mathbf{u}_1^T$ and using $\mathbf{u}_1^T \mathbf{u}_1 = 1$, we have

$$\mathbf{u}_1^T S \mathbf{u}_1 = \lambda_1$$

which is the variance after the projection.

PCA - Variance Maximization

we can extend the problem to the second best eigenvalue

Solution

$$\mathbf{u}_1^T S \mathbf{u}_1 = \lambda_1$$

Variance is maximal when $\mathbf{u}_1$ is the eigenvector corresponding to the largest eigenvalue λ_1.

This is called the first **principal component**.

PCA - Variance Maximization

Repeat to find other directions which

- maximize variance of projected data
- are orthogonal to the previous directions

Summary:
To perform PCA in a M-dimensional projection space, with $M < D$

- compute $\bar{\mathbf{x}}$: mean of the data
- compute S: covariance matrix of the dataset
- find M eigenvectors of S corresponding to the M largest eigenvalues

let's consider another formulation of the problem, based on projection error minimization

PCA - Error minimization

given a dataset in D dimension and given a orthonormal basis so a set of vectort u1...uD that correspond to an orthonormal base in D dimension, this means that we can consder the u vectors as a new reference space in which we want to translate the original system

Consider a complete orthonormal D-dimensional basis such that

$$\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$$

with $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$

Each data point can be written as

$$\mathbf{x}_n = \sum_{i=1}^{D} \alpha_{ni} \mathbf{u}_i$$

Using the orthonormality property we have $\alpha_{nj} = \mathbf{x}_n^T \mathbf{u}_j$, hence

$$\mathbf{x}_n = \sum_{i=1}^{D} (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i$$

PCA - Error minimization

Goal: Approximate $\mathbf{x}_n$ using a lower-dimensional representation.

We can write

M components

remaining D components

approximator value

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} z_{ni} \mathbf{u}_i + \sum_{i=M+1}^{D} b_i \mathbf{u}_i$$

Evaluate approximation error as

$$J = \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2$$

Minimizing w.r.t. z_{nj} we get

$$z_{nj} = \mathbf{x}_n^T \mathbf{u}_j, \; j = 1, \ldots, M$$

Minimizing w.r.t. b_j we get

$$b_j = \bar{\mathbf{x}}^T \mathbf{u}_j, \; j = M+1, \ldots, D$$

PCA - Error minimization

Using these expression we get

$$\mathbf{x}_n - \tilde{\mathbf{x}}_n = \sum_{i=M+1}^{D} [(\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i]\mathbf{u}_i$$

The overall approximation error becomes

$$J = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} (\mathbf{x}_n^T \mathbf{u}_i - \bar{\mathbf{x}}^T \mathbf{u}_i)^2 = \sum_{i=M+1}^{D} \mathbf{u}_i^T S \mathbf{u}_i$$

PCA - Error minimization

Minimize the approximation error subject to constraint $\mathbf{u}_i^T \mathbf{u}_i = 1$:

$$\tilde{J} = \sum_{i=M+1}^{D} \mathbf{u}_i^T S \mathbf{u}_i + \lambda_i (1 - \mathbf{u}_i^T \mathbf{u}_i)$$

Setting derivative of a $\mathbf{u}_i$ to zero we have:

$$S\mathbf{u}_i = \lambda_i \mathbf{u}_i$$

here we want to minimize the error

Hence $\mathbf{u}_i$ is an eigenvector of S with eigenvalue λ_i.

PCA - Error minimization

The approximation error is then given by

$$J = \sum_{i=M+1}^{D} \lambda_i$$

This is minimized by selecting $\mathbf{u}_i$ as the eigenvectors corresponding to the $D - M$ smallest eigenvalues.

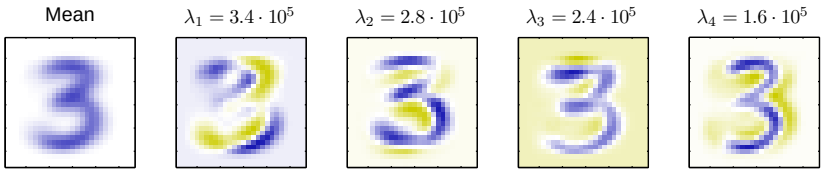
Note: Choosing $D - M$ smallest eigenvalues of S corresponds to finding M highest eigenvalues of S as in the maximum variance formulation.

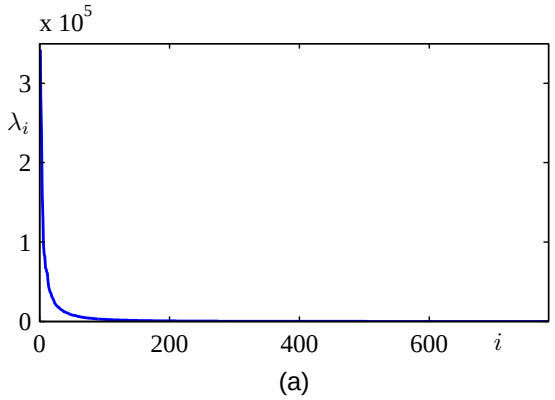
PCA - Algorithms

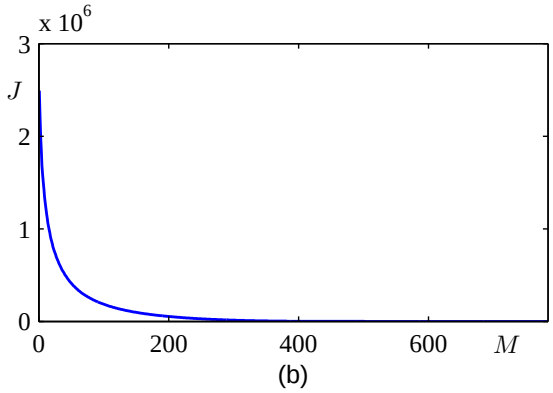
method in which, given an unsupervised dataset we compute the covariance matrix S, the highest eigenvalues and the corresponding eigenvectors and just transform and project all the samples in the dataset in this dimension

1. Full eigenvalue decomposition of S (slow)
2. Efficient eigenvalue decomposition - only M eigenvectors
3. Singular value decomposition of centered data matrix $\mathbf{X}$

PCA - Example





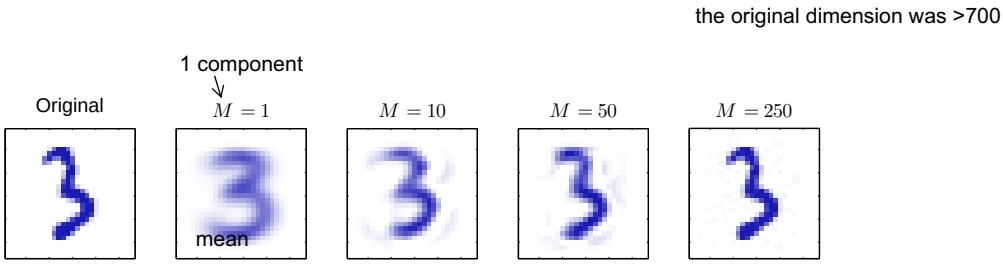


Eigenvalue spectrum

Sum of discarded eigenvalues (error)

PCA - Example

Reconstruction with a limited number of components



PCA for high-dimensional data

What if number of points is smaller than the dimensionality, i.e. $N < D$?
At least D-N+1 eigenvalues of S are zero.

Example: small set of high-resolution images.

In this case finding eigenvalues of S ($D \times D$ matrix) is inefficient.

PCA for high-dimensional data

Solution for $N < D$:

Let $\mathbf{X}$ be the $N \times D$ centered data matrix (i.e., n-th row is $(\mathbf{x}_n - \bar{\mathbf{x}})^T$)

and the corresponding covariance matrix:

$$S = \frac{1}{N}\mathbf{X}^T\mathbf{X}$$

The corresponding eigenvector equations is

$$\frac{1}{N}\mathbf{X}^T\mathbf{X}\mathbf{u}_i = \lambda_i\mathbf{u}_i$$

PCA for high-dimensional data

By left-multiplying by $\mathbf{X}$ we obtain

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{u}_i) = \lambda_i(\mathbf{X}\mathbf{u}_i)$$

By defining $\mathbf{v}_i = \mathbf{X}\mathbf{u}_i$ we have

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T\mathbf{v}_i = \lambda_i\mathbf{v}_i$$

$\mathbf{X}\mathbf{X}^T$ has the same $N-1$ eigenvalues of $\mathbf{X}^T\mathbf{X}$ (the others are 0).

$\mathbf{X}\mathbf{X}^T$ is an $N \times N$ matrix whose eigenvalues can be computed efficiently.

PCA for high-dimensional data

Given the eigenvalues λ_i of $\mathbf{X}\mathbf{X}^T$, to find the eigenvectors we left-multiply by $\mathbf{X}^T$

$$\left(\frac{1}{N}\mathbf{X}^T\mathbf{X}\right)(\mathbf{X}^T\mathbf{v}_i) = \lambda_i(\mathbf{X}^T\mathbf{v}_i)$$

This makes clear that $(\mathbf{X}^T\mathbf{v}_i)$ is an eigenvector of S with eigenvalue λ_i.

To find $\mathbf{u}_i$ we have to normalize these eigenvectors such that $\mathbf{u}_i^T\mathbf{u}_i = 1$

$$\mathbf{u}_i = \frac{1}{\sqrt{N\lambda_i}}\mathbf{X}^T\mathbf{v}_i$$

PCA for high-dimensional data

Summing up, when $N << D$:

- Consider the centered data matrix $\mathbf{X}$
- Compute (efficiently) the $N-1$ eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ of $\mathbf{X}\mathbf{X}^T$
- Let $\mathbf{u}_i = \frac{1}{\sqrt{N\lambda_i}}\mathbf{X}^T\mathbf{v}_i$

The so-obtained $N-1$ vectors $\mathbf{u}_i$ are the eigenvectors of $S = \mathbf{X}^T\mathbf{X}$, with non-null eigenvalue λ_i

Probabilistic PCA

Linear Latent Variable Model

- Represent data $\mathbf{x}$ with lower dimensional latent variables $\mathbf{z}$
- Assume linear relationship

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \boldsymbol{\mu}$$

- Assume Gaussian distribution of latent variables $\mathbf{z}$

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$

- Assume Linear-Gaussian relationship between latent variables and data

$$P(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$$

Probabilistic PCA

Marginal distribution

$$P(\mathbf{x}) = \int P(\mathbf{x}|\mathbf{z})P(\mathbf{z})d\mathbf{z} = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{C})$$

with

$$\mathbf{C} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}$$

Posterior distribution

$$P(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \mathbf{M}^{-1}\mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu}), \sigma^2\mathbf{M})$$

with

$$\mathbf{M} = \mathbf{W}^T\mathbf{W} + \sigma^2\mathbf{I}$$

Maximum likelihood PCA

Maximum likelihood: given data $\mathbf{X}$

$$\underset{\mathbf{W},\boldsymbol{\mu},\sigma}{\operatorname{argmax}} \ln P(\mathbf{X}|\mathbf{W}, \boldsymbol{\mu}, \sigma^2) = \sum_{n=1}^{N} \ln P(\mathbf{x}_n|\mathbf{W}, \boldsymbol{\mu}, \sigma^2)$$

Setting derivatives to 0, we have a closed form solution

$$\boldsymbol{\mu}_{ML} = \bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$$

$$\mathbf{W}_{ML} = \ldots$$

$$\sigma^2_{ML} = \ldots$$

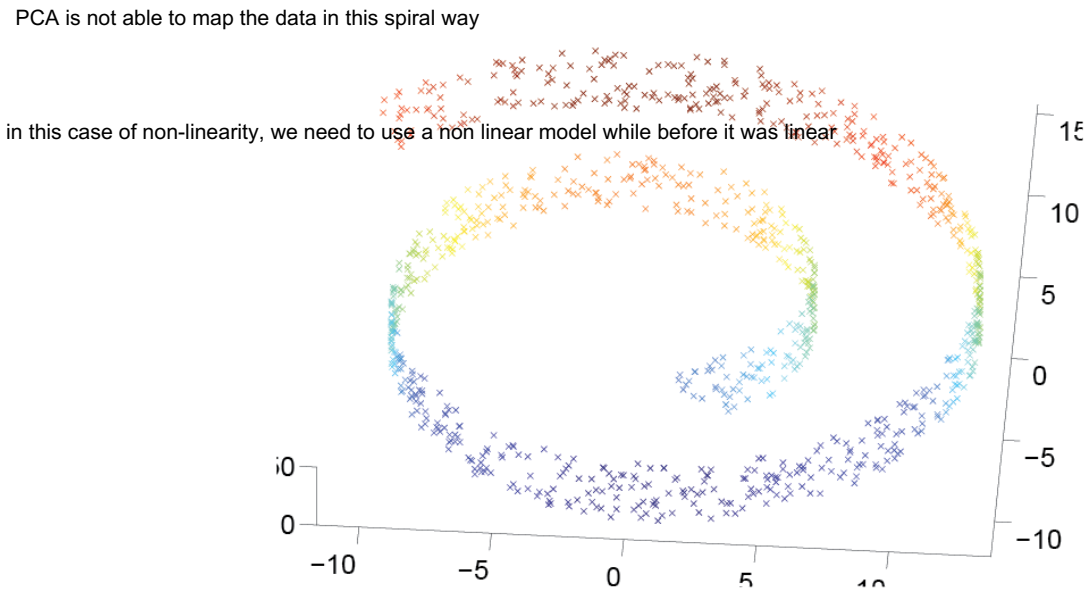
$\mathbf{W}$ depends on the eigenvalues and eigenvectors of S (not trivial proof)

Maximum likelihood PCA

Maximum likelihood solution for the probabilistic PCA model can be obtained also with EM algorithm.

Non-Linear Latent Variable Models

Motivation: Linear representations are not sufficient for complex data

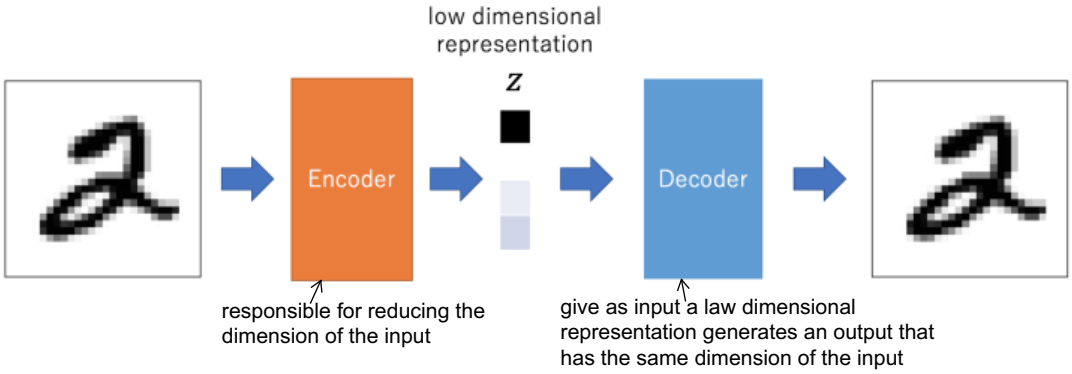


The 'Swiss Roll' dataset. 2D manifold embedded in 3D space.

Autoassociative Neural Networks (Autoencoders)

What is an autoencoder?

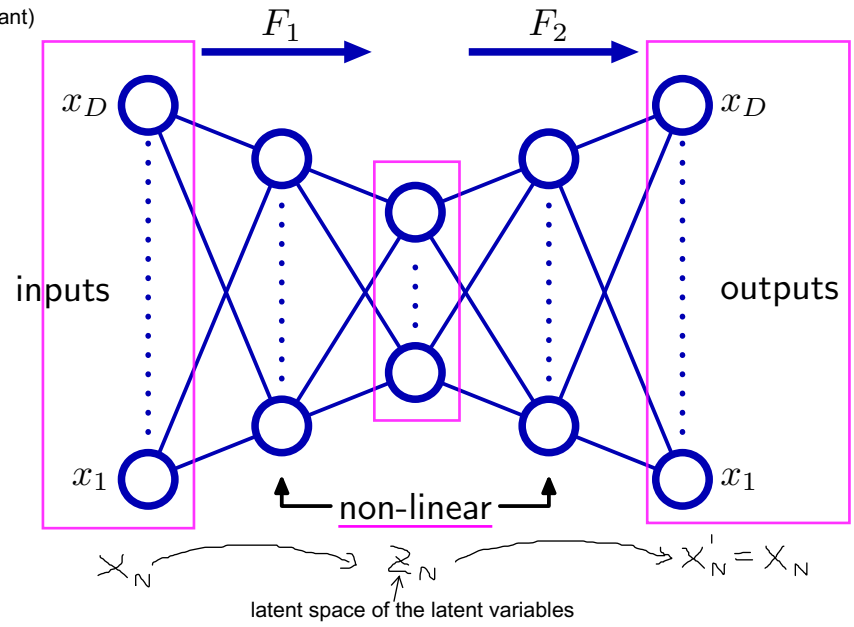
- a combination of two NN: an encoder and a decoder
- trained based on reconstruction loss
- provides low-dimensional representation



Autoencoders

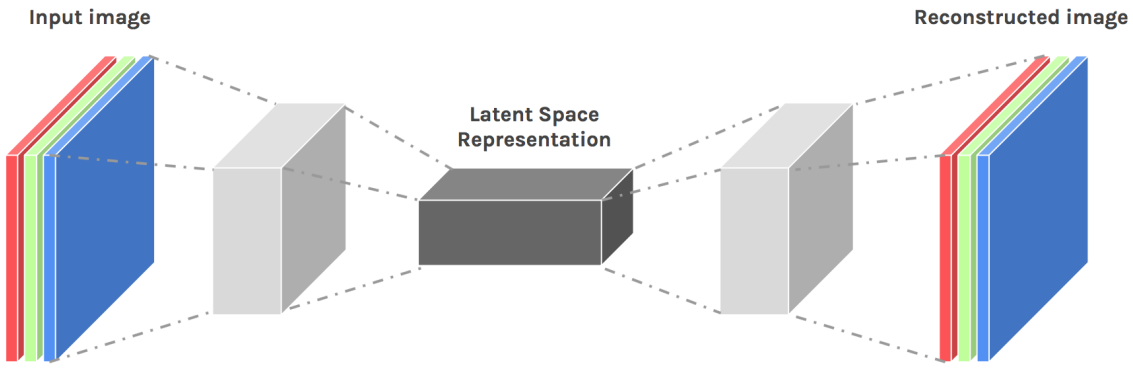
Neural networks with reduced sized hidden layers (bottleneck) which learn to reconstruct their input by minimizing a loss function.

the number of input is equal to the number of outputs (this is important)



Convolutional Autoencoders

AE for images based on Convolutional and Convolutional Transposed layers



Autoencoders

SUMMARY

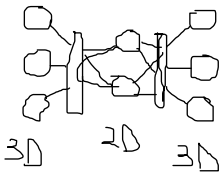
Given a dataset $\{x_n\}$, Autoencoders are trained with the same sample x_n both in input and in output.

Autoencoders learn how to encode/decode the samples in a dataset in a low-dimensional space.

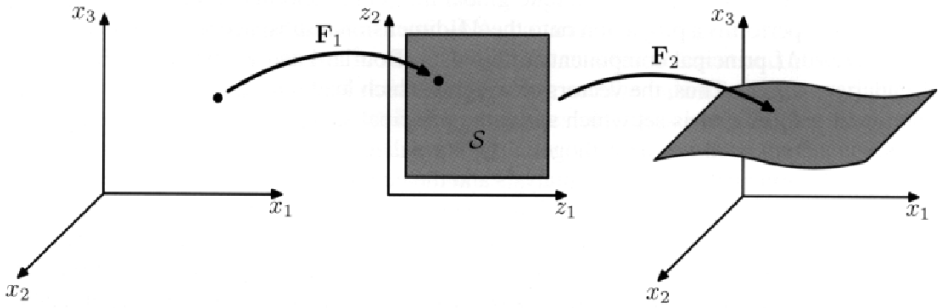
Autoencoders can be seen as a method for non-linear principal component analysis.

Autoencoders

Autoencoder example:



Input: 3-D, Hidden layer: 2-D, Output: 3-D



The 3D reconstruction from 2D hidden layer generates a 2D manifold (surface) in 3D.

Autoencoder Example

8 values autoencoder

Data set: $\{0, \ldots, 7\}$ with 1-out-of-8 encoding:
$\{\langle 1, 0, \ldots, 0\rangle, \langle 0, 1, \ldots, 0\rangle, \ldots, \langle 0, 0, \ldots, 1\rangle\}$

Autoencoder with 1 hidden layer with 3 nodes.

Analysis of latent space: see Exercise 14.

Autoencoders for anomaly detection

there are also cases in which we want to understand if the input is not part of the dataset

Autoencoders are very useful for anomaly detection (one-class classification)

Problem

learn $f : X \to \{n, a\}$ with data set $D = \{(x_n, n)\}$

Train with only samples from normal class (x_n, n)

Given test $x' \notin D$, predict $\hat{f}(x') \in \{n, a\}$ (normal vs. abnormal)

very often in anomaly detection we want to distinguish between nrmal and abnormal

Autoencoders for anomaly detection

Solution

1. Train an autoencoder AE with $\{x_n\}$ (i.e, learn a latent representatinon for normal samples), compute final train $loss$
2. Determine a threshold δ, e.g. $\delta = mean(loss) + std(loss)$
2. Given $x' \notin D$, reconstruct x' (new sample) with AE and compute $loss'$
3. If $loss' < \delta$ return *normal*, otherwise return *abnormal*

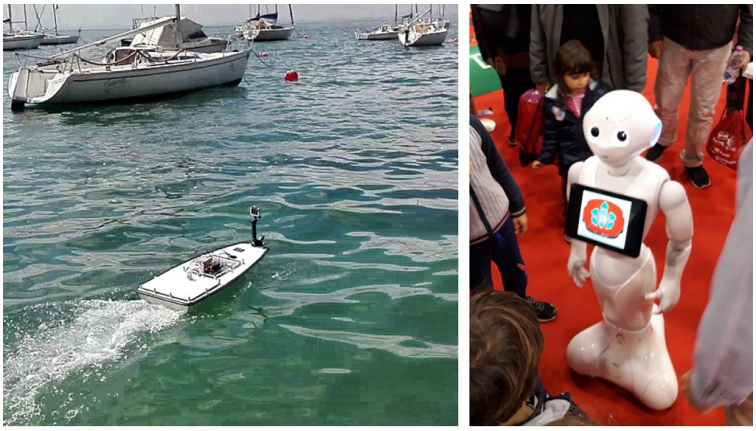
this works better if we use more encoders

Note: works even better with ensembles of autoencoders.

Autoencoders for anomaly detection

Example

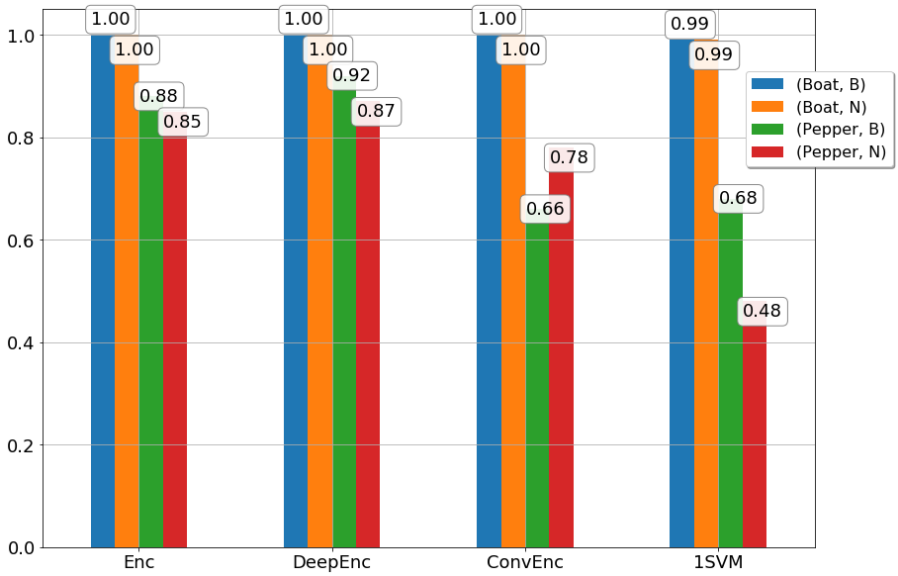
- Take logs of nominal behaviors during development
- Train an anomaly detection system
- Use anomaly detection during deployment



(a) (b)

Autoencoders for anomaly detection

Results



Matteo Olivato, Omar Cotugno, Lorenzo Brigato, Domenico Bloisi, Alessandro Farinelli, Luca Iocchi:
A Comparative Analysis on the use of Autoencoders for Robot Security Anomaly Detection. IROS 2019: 984-989

Generative models

this part is related with dimensionality reduction but the goal is not only reducing the dimension but also understand the probability distribution of the dataset in order to generate new samples that are similar as much as possible with the samples in the dataset

Variational Auto-Encoders (VAEs)

- focus on learning latent space structure

Generative Adversarial Networks (GANs)

→ to generate data that are similar to a distribution

- focus on learning a distribution
- no latent space (in general)

VAEs

VARIATIONAL AUTOENCODERS

Goal

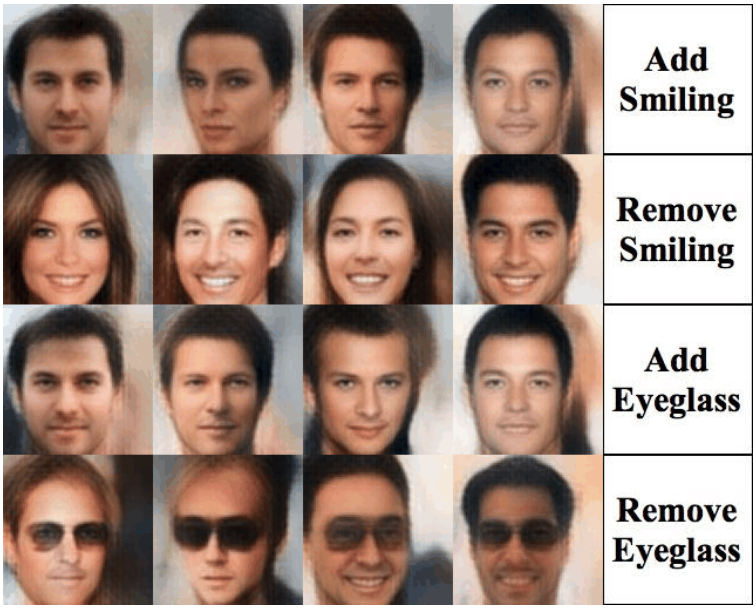
- modify data in specific directions
- identify meaningful directions in latent space

Examples

- 'distort' faces (change expression, add glasses)
- produce digits from different hand-writing styles
- distort 3D meshes

VAEs

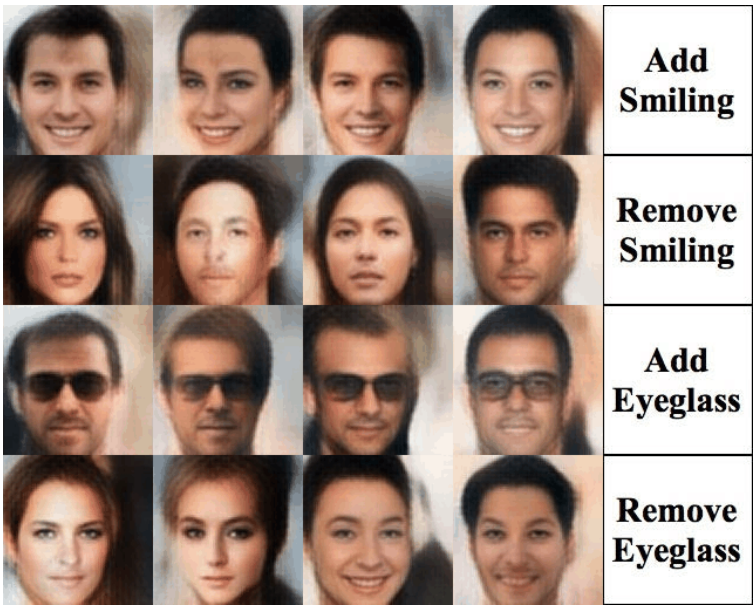
Example: faces



VAEs

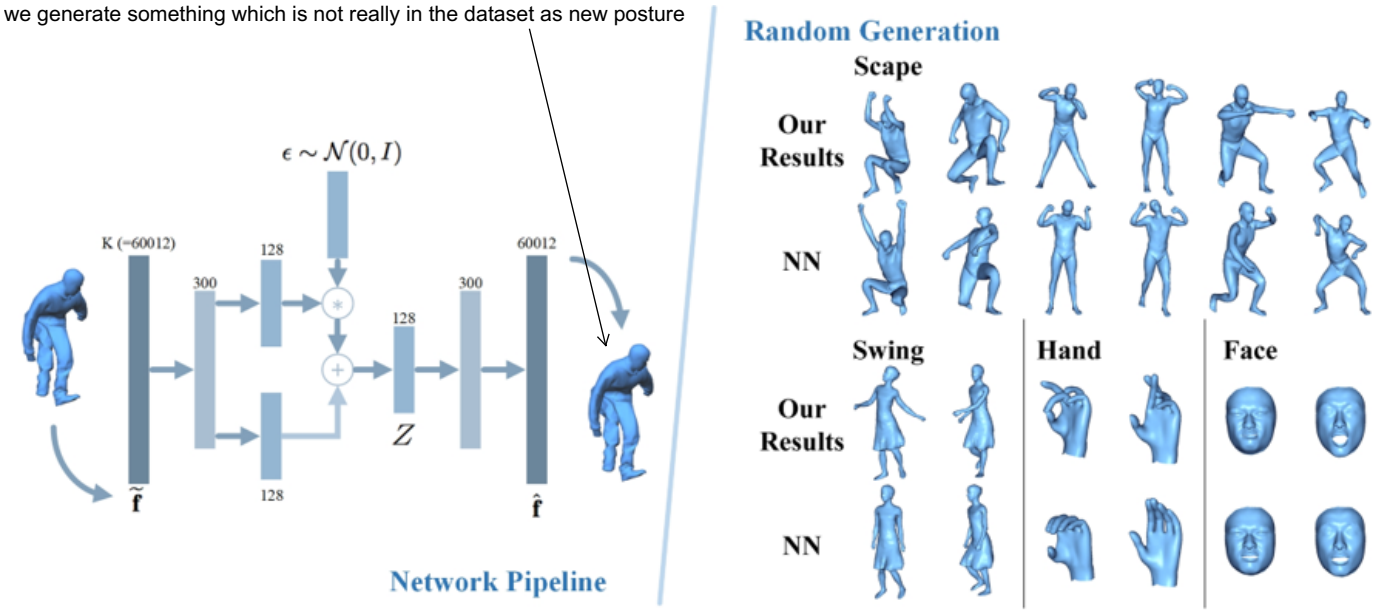
Example: faces

the model is able to generate a perrurbation of the faces that permits to modify them



VAEs

Example: 3D Mesh deformation

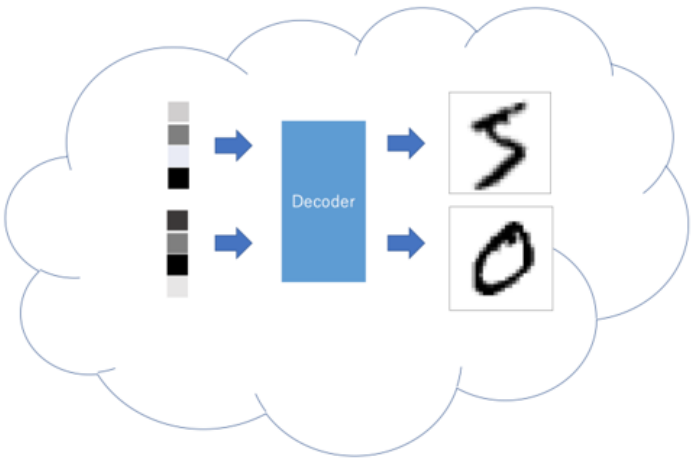


VAEs

Goal

Feed latent vectors and get realistic samples of $P(X)$

Similar vector values should produce similar generated instances.



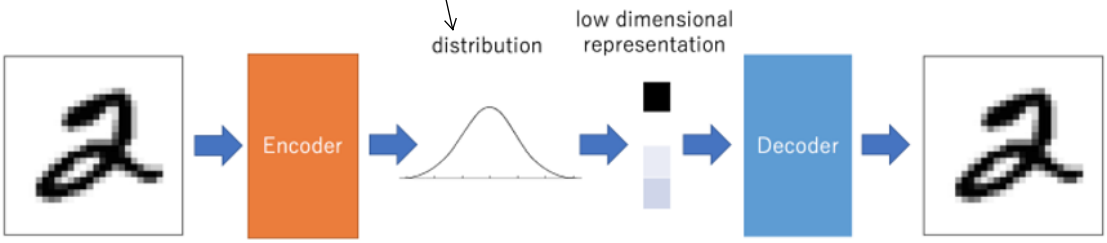
VAEs

Main idea

Encoder produces a *distribution* instead of a vector

Decoder operates on samples from this distribution

it has the same structure as an autoencoder, but here we have a gaussian distribution



VAEs

Problems

1. How to produce a distribution?
2. How to prevent degeneration?

Solutions

1. consider parametric distributions typically Gaussian
   - produce mean μ and variance Σ
2. add loss term based on Kullback-Leibler divergence (Evidence Lower Bound)

it measures the inequality or the distance between two distributions

loss = MSE + lambda*KL ← this part is not present in the autoencoder

lamda indicates how much important is the component KL

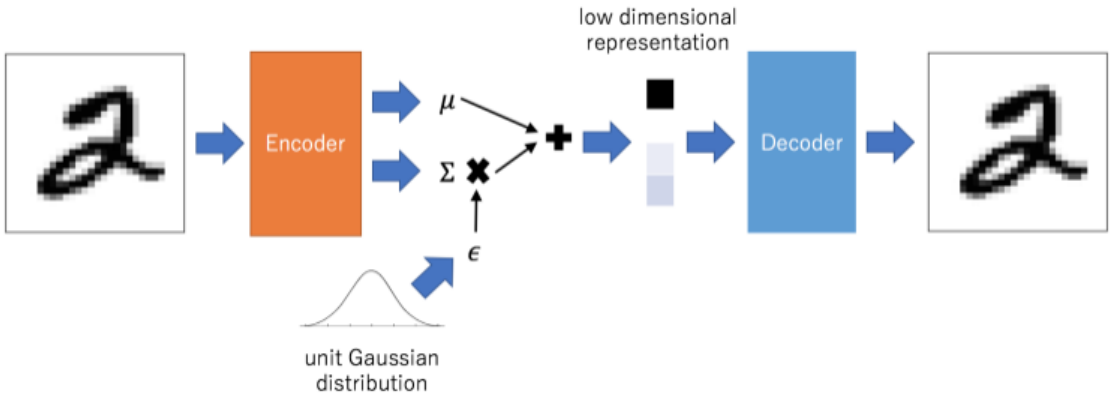
VAEs

One more problem:

- Sampling operation is not differentiable

Solution:

- re-parametrization



GANs

Goal

- Sample from the input data distribution $\mathcal{X}$

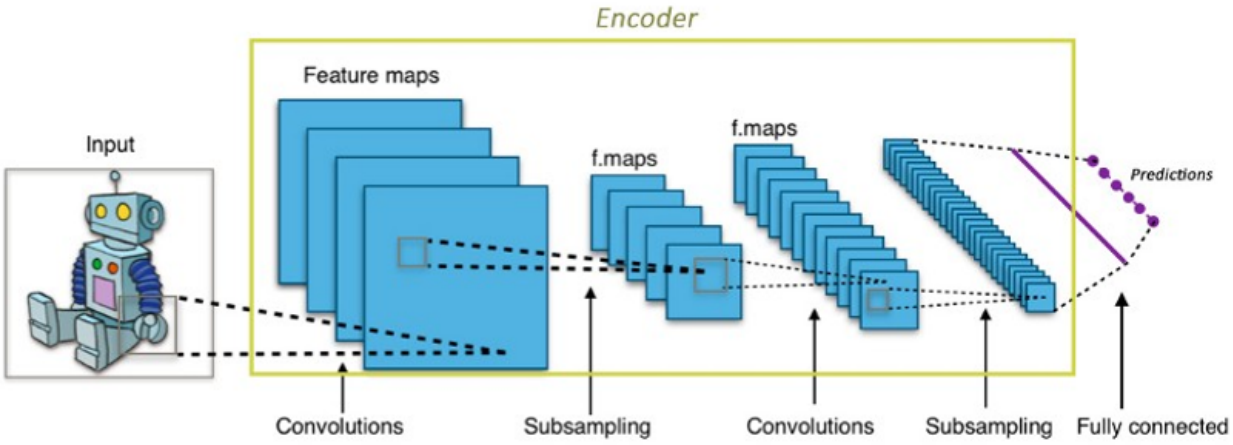
Idea

- Invert a (Convolutional) Neural Network
- Use adversarial training

GANs

Encoder (CNN)

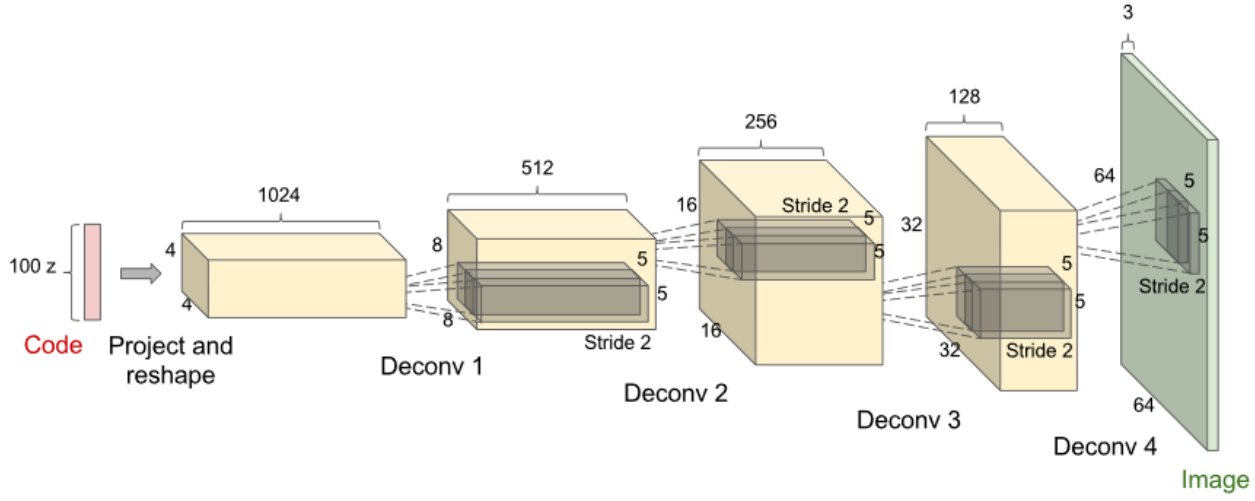
- processes an image and produces a vector/*code*



GANs

Decoder

- receives a code (random vector) and produces an image
- uses "deconvolutional" (transposed convolution) layers



GANs

Problem

How to train the decoder to produce meaningful data?

Idea

Use Adversarial Training

2 networks are competing each other in order to reach good results

GANs

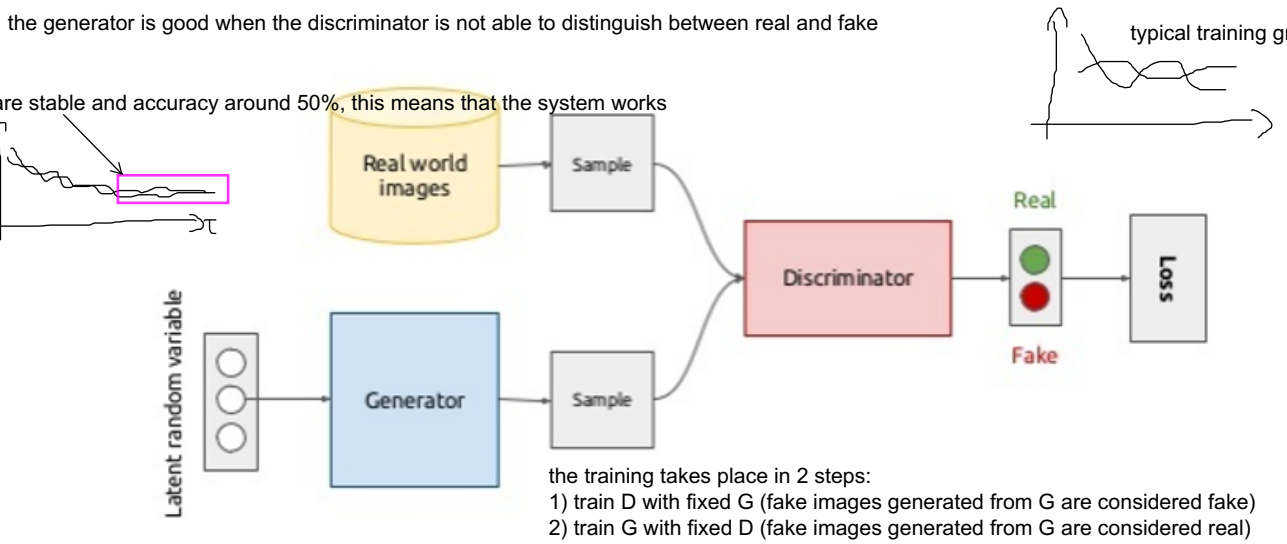
a trick is to use 0.1 and 0.9 instead of 0 and 1 for classifying real and fake images

A GAN is a combination of two networks

1. a generator network (decoder) — part of the network that given a representation in latent space generates an image
2. a discriminator network (critic) — binary classifier with 2 classes that takes as input the images from the dataset (REAL images) and the images generated by the generator (FAKE images), and it have to discriminate if an image is REAL or FAKE

the generator is good when the discriminator is not able to distinguish between real and fake

typical training graph



the training takes place in 2 steps:
1) train D with fixed G (fake images generated from G are considered fake)
2) train G with fixed D (fake images generated from G are considered real)

GANs

Roles

Generator produces samples of the distribution $P(X)$

Discriminator identifies if a sample actually comes from the (unknown) $P(X)$ or not

Training

make the networks compete with each other

- generator tries to fool the discriminator in believing that the sample is 'real'
- discriminator tries to discriminate as good as possible 'real' from 'fake' samples

GANs

Training

Repeat:

- Train the discriminator with a batch of data $\{(x_n, Real)\} \cup \{(x'_m, Fake)\}$, where x_n comes from the data set, while x'_m are images generated from the generator with random values of the latent variable.
- Train the generator by using the entire model (generator + discriminator) with discriminator layers fixed (i.e., not trainable) with a bacth of data $\{(r_k, Real)\}$, where r_k are random values of the latent variable.

GANs

Example on CIFAR dataset (32x32 images)
Are these real images or generated?

GANs

Example on CIFAR dataset (32x32 images)
Are these real images or generated?

GANs

Example on CIFAR dataset (32x32 images)
Results at 300, 900 and 5700 iterations

at the first iteration the weights are randomly set and so the images are not good, but increasing the iterations so performing the training it becomes better

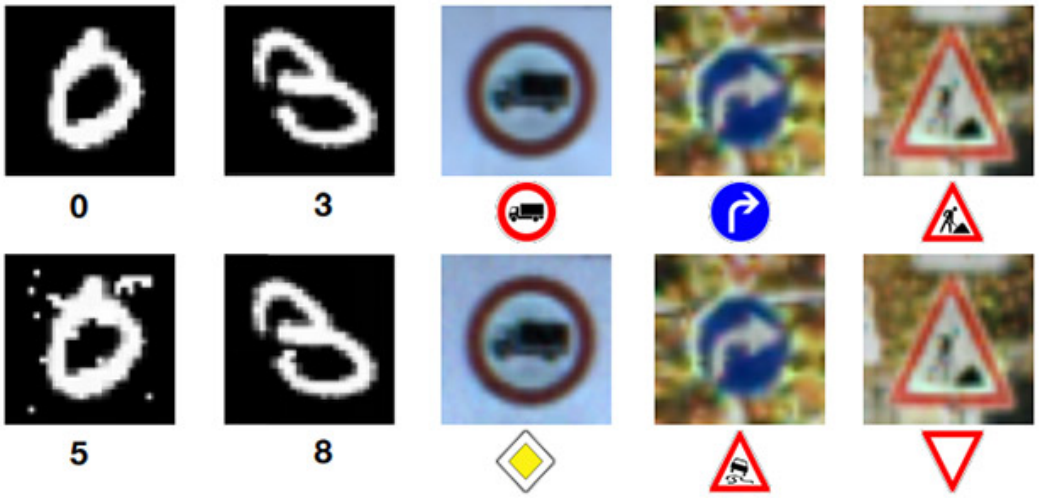
GANs

Example: Celebrity faces (1024x1024 images)

Adversarial attacks to ML models

Unfortunately, ML can be used to attack ML models.

Examples:



https://spectrum.ieee.org/cars-that-think/transportation/sensors/slight-street-sign-modifications-can-fool-machine-learning-algorithms

Summary

- Dimensionality reduction aims at identifying the real/intrinsic degrees of freedom of a data set
- Analysis of latent variables helps in understanding the variability of the input data
- Deep associative neural networks provide a general tool for non-linear PCA
- Special architectures can be used to sample the latent space for generating realistic samples of the data distribution