Freie Universität Berlin

Lecture: Machine Learning for Data Science

Winter semester 2021/22

Lecture 21: High dimensionality (dimensionality reduction)

Prof. Dr. Eirini Ntoutsi

(Part of the slides thanks to M.Sc. Yi Cai)

High-dimensional data: outline

- Introduction and challenges of high dimensionality *(last week)*
- How we deal with high dimensionality?
  - Feature Selection *(last week)*
    - Find a subset $F' \subset F$ of features that are the most relevant for learning.
  - Dimensionality reduction *(this lecture)*
    - Find a lower dimensional data representation *F'* that still preserves properties of the data. F' consists of "combinations" of the original features
  - Learning in subspaces (not covered)
    - E.g., Clustering in high-dimensional data

Outline

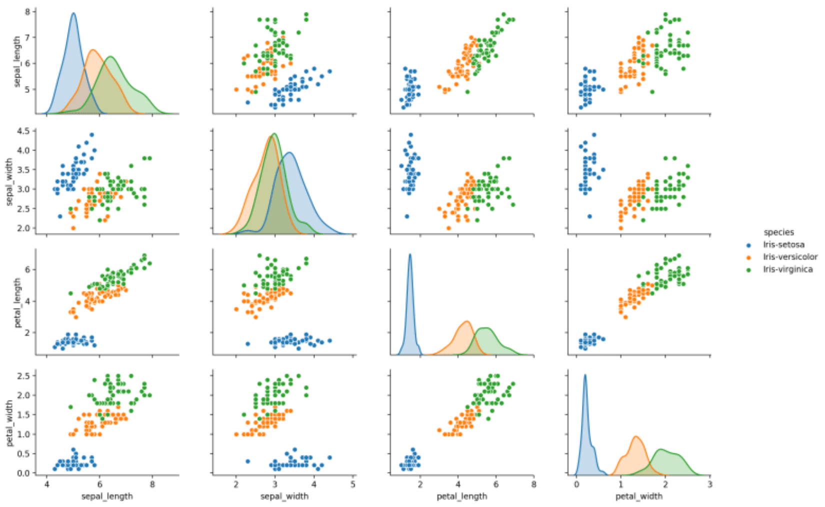
- Introduction
- Principal Component Analysis (PCA)
- Autoencoders
- Things you should know from this lecture & reading material

Dimensionality reduction: goal

- Given a problem in the original high-dimensional feature space $F = \{f_1, .., f_d\}$
- The dimensionality reduction task is to find a low dimensional feature space F' that can "reconstruct" the original space F as accurately as possible
  - *Redundant* features are ''summarized''
  - *Irrelevant* features contribute with a small weight

- Data are re-represented in the new feature space F'
  - We can proceed with our analysis/learning in F'

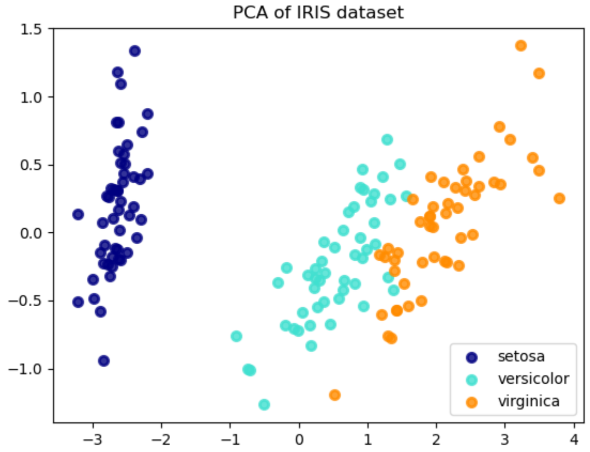
Why is dimensionality reduction useful



Source

- Handles high-dimensional data
  - If data has thousands of dimensions, can be difficult for ML methods to deal with.
  - Often, the intrinsic dimensionality is small, i.e., data can be described by much smaller representations

- Useful for
  - Visualization
  - Preprocessing
  - Compression
  - (Machine) Learning
    - Data are re-represented in the new feature space and further analysis/ learning can be applied in the new space



Source

Dimensionality reduction vs feature selection

- In feature selection, the new feature space consists of a subset of the original features
  - The goal is to find a new feature space F' $\subseteq$ F, where all "useless" features from F have been removed.
  - F' is interpretable
- In dimensionality reduction, the new feature space F' consists of "artificial" variables/features which can be:
  - linear combinations of the original variables as in PCA
  - non-linear combinations of the original variables as in autoencoders
  - F' is often not interpretable
- Most of the feature selection methods are supervised (need class labels), dimensionality reduction is typically unsupervised

Dimensionality reduction methods

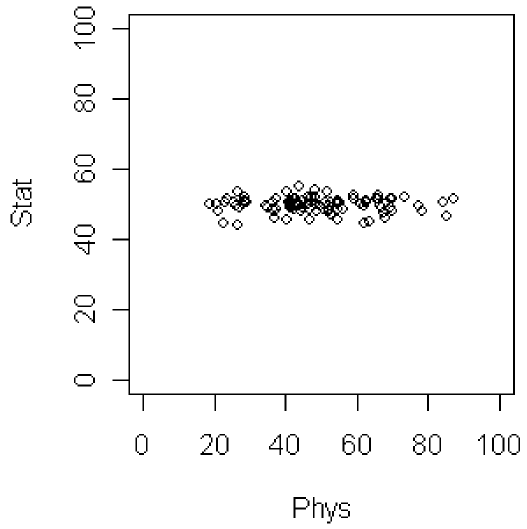
- A variety of different methods for dimensionality reduction:
  - Reference point embedding
  - Principal component analysis (PCA): unsupervised
  - Singular value decomposition (SVD)
  - Linear Discriminant Analysis (LDA): supervised
  - Autoencoders
  - ....

Outline

- Introduction
- Principal Component Analysis (PCA)
- Autoencoders
- Things you should know from this lecture & reading material

Principal Component Analysis (PCA): A simple example

- Consider the grades of students in Physics and Statistics.
- If we want to compare among the students, which grade should be more discriminative? Statistics or Physics?



Physics since the variance along that axis is larger.

Detour (see lecture 2): Variance is a measure of the spread of the data along a dimension *X*

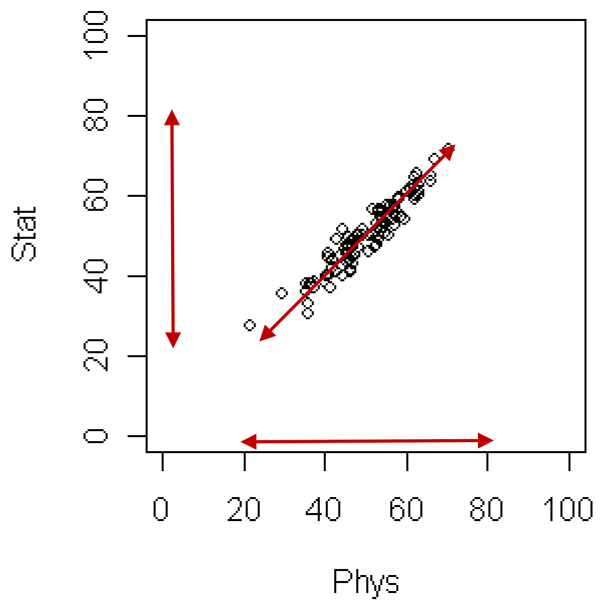
$$VAR(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

it measures how far the values of X are spread out from the average X value (μ)

Based on: http://astrostatistics.psu.edu/su09/lecturenotes/pca.html

Principal Component Analysis (PCA): A simple example

- Suppose now the plot looks as below.
- What is the best way to compare students now?



We should take (linear) combination of the two grades to get the best results.

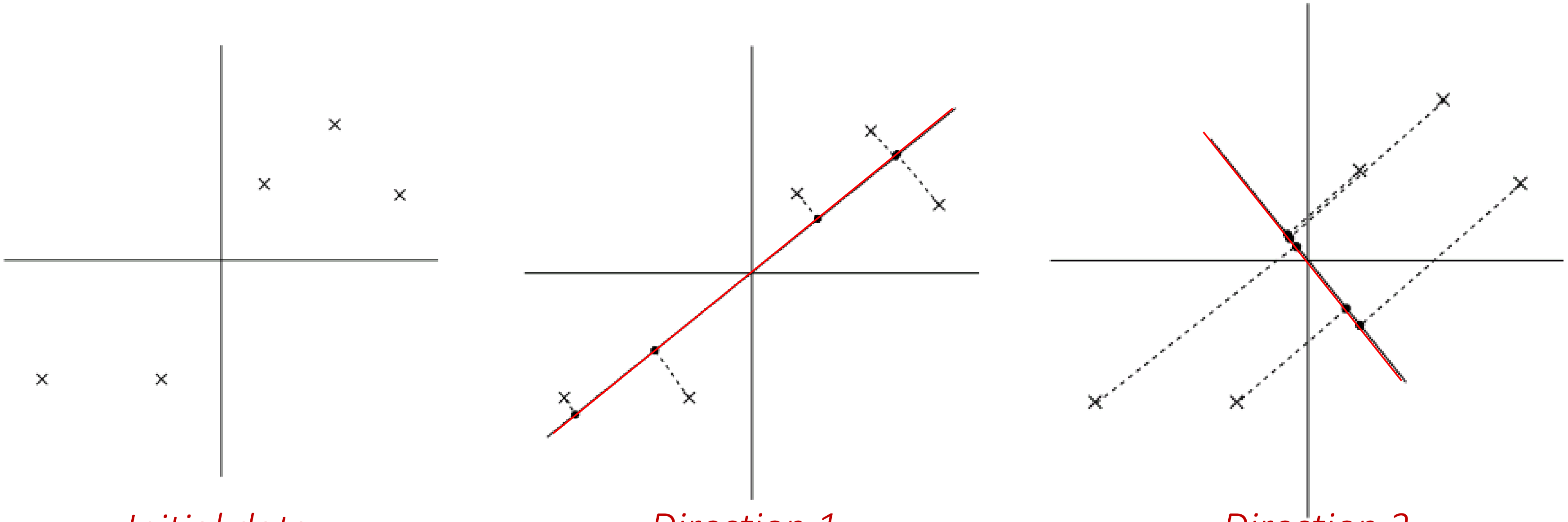
Here the direction of maximum variance is clear.

In general (for high dimensional data we cannot inspect visually our data) → PCA

Based on: http://astrostatistics.psu.edu/su09/lecturenotes/pca.html

Principal Component Analysis (PCA): Intuition

- The data has some amount of variance/information. We would like to choose a direction *u* so that if we were to approximate the data as lying in the direction/subspace corresponding to *u*, as much as possible of this variance is still retained.

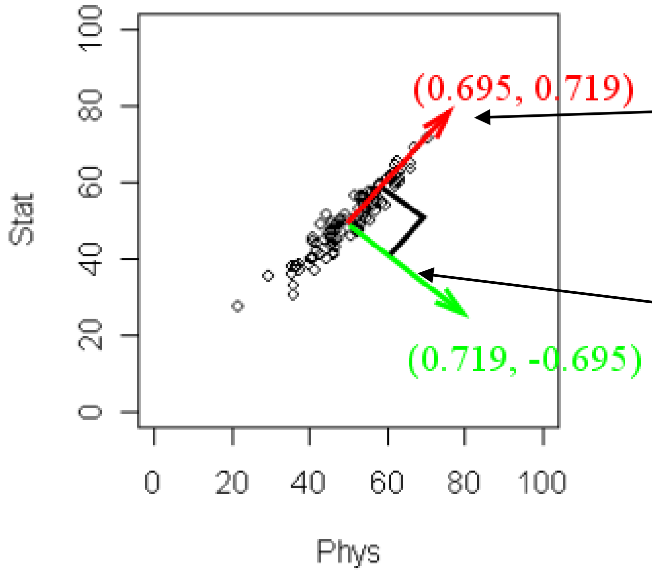


Initial data *Direction 1* *Direction 2*

- Idea: Choose the direction that maximizes the variance of the projected data

Principal Component Analysis (PCA): Back to our simple example

- PCA returns two principal components for our example (in general, as many as the dimensions)
  - The first gives the direction of the maximum spread of the data.
  - The second gives the direction of maximum spread perpendicular to the first direction
  - The principal components are orthogonal/ uncorrelated



1st principal component

Orthogonal 2nd principal component

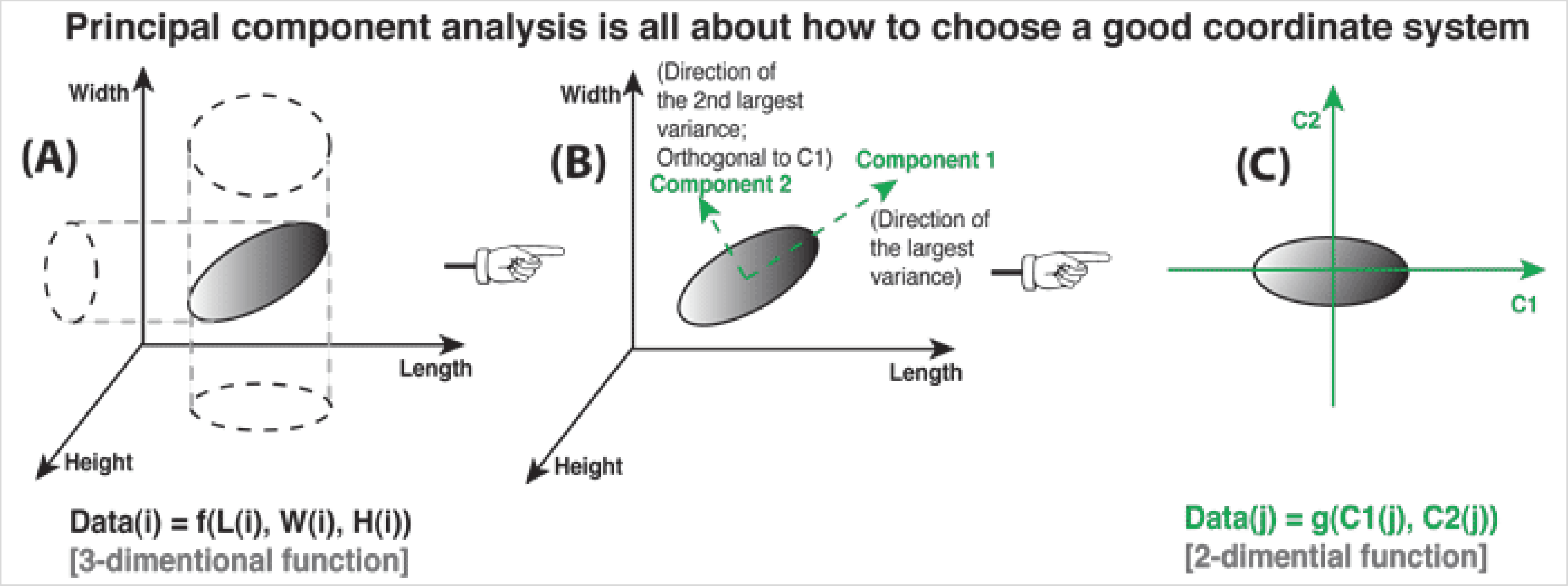
//Orthogonal= not linear correlated

Based on:
http://astrostatistics.psu.edu/su09/lecturenotes/pca.html

PCA is a coordinate system transformation

- PCA is nothing but coordinate system transformation
  - Can we find a simplest way to express our data?

Is there another basis, which is a linear combination of the original basis, that best represents our dataset?



- After transformation we get
  - A simpler way to describe our data
  - No information loss (The relative geometric positions of all data points remain unchanged.)

Source: http://mengnote.blogspot.de/2013/05/an-intuitive-explanation-of-pca.html

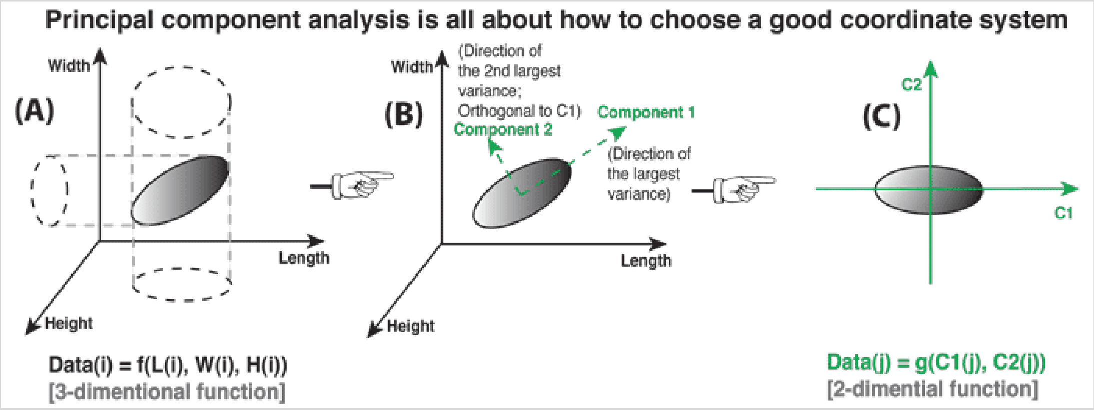
Principal components analysis (PCA): Formulation

- Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables *d* into a set of values of linearly uncorrelated variables *k* called principal components.

- The goal is to reduce the dimensionality from *d* to *k* ($k<d$) while retaining most of the information in the data

- How should we choose *k* to represent the data well?

Principal components analysis (PCA): Formulation



- General form: DV = D' where
  - D_{nxd} is the original dataset
  - V_{dxk} is a linear transformation
  - D'_{nxk} the re-representation of the dataset
  - *d*:original dimensionality, *k* reduced dimensionality
- So, V is a matrix that transforms D into D'
- Geometrically, V is a rotation and a stretch which transforms D into D'
  - The eigenvectors are the rotations to the new axes
  - The eigenvalues are the amount of scaling that needs to be done
  - The eigenvectors (principal components) determine the directions of the new feature space
  - The eigenvalues explain the variance of the data along the new feature axes.

- Roughly speaking, PCA computes the eigenvalues and eigenvectors of the covariance matrix

(Detour) Computing PCA basics: Data matrix

- Given n instances $v_i \in IR^d$, $n \times d$ matrix D is called data matrix

$$D = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} v_{1,1} & \cdots & v_{1,d} \\ \vdots & \ddots & \vdots \\ v_{n,1} & \cdots & v_{n,d} \end{pmatrix}$$

(Detour) Computing PCA basics: Variance

- Variance is a measure of the spread of the data along a dimension *X*

$$VAR(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

  - it measures how far the values of X are spread out from the average X value (μ)

- Variance refers to a single dimension, e.g., height
  - i.e., how a single dimension varies

ID	Height
1	100
2	100
3	100
4	100
5	100

Variance =0

ID	Height
1	100
2	100
3	105
4	100
5	100

Small variance (4)

ID	Height
1	100
2	100
3	100
4	200
5	100

Large variance (1600)

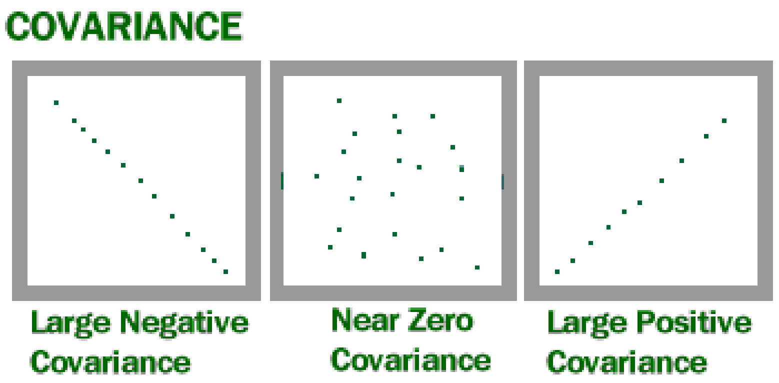
(Detour) Computing PCA basics: Covariance

- Covariance provides a measure of the strength of the correlation between two variables *X*,*Y*

$$COV(X,Y)=\frac{1}{n}\sum_{i=1}^{n}(x_i-\mu_x)(y_i-\mu_y)$$

  - μ_x,μ_y the means of *X*, *Y*

- What the covariance values mean
  - Zero value: the variables are uncorrelated.
  - Positive values: both dimensions move together (increase or decrease)
  - Negative values: while one dimension increases the other decreases



(Detour) Computing PCA basics: Covariance matrix

- Describes the variance of all features (in the diagonal) and feature pairwise correlations/covariances

$$\Sigma_D = \begin{pmatrix} VAR(X_1) & \cdots & COV(X_1, X_d) \\ \vdots & \ddots & \vdots \\ COV(X_d, X_1) & \cdots & VAR(X_d) \end{pmatrix}$$

- Properties:
  - For *d*-dimensional data, *dxd* covariance matrix
  - symmetric matrix as $COV(X,Y)=COV(Y,X)$

Computing PCA basics: Vector/ Matrix basics

- Inner (dot) product of vectors x, y:

$$x \cdot y = x^T \cdot y = \begin{pmatrix} x_1 & \cdots & x_d \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_d \end{pmatrix} = \langle x, y \rangle = \sum_{i=1}^{d} x_i \cdot y_i$$

- Outer product of vectors x, y:

$$x \otimes y = x \cdot y^T = \begin{pmatrix} x_1 \\ \vdots \\ x_d \end{pmatrix} \cdot \begin{pmatrix} y_1 & \cdots & y_d \end{pmatrix} = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_d \\ \vdots & \ddots & \vdots \\ x_d y_1 & \cdots & x_d y_d \end{pmatrix}$$

- Matrix multiplication:

$$A = [a_{ij}]_{m \times p};\ B = [b_{ij}]_{p \times n};$$

$$AB = C = [c_{ij}]_{m \times n}, where\ c_{ij} = row_i(A) \cdot col_j(B)$$

  - An example:

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \times \underset{B}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}} = \underset{C}{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}$$

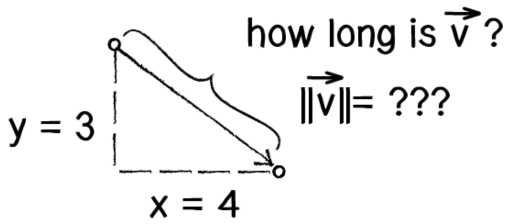
Computing PCA basics: Vector/ Matrix basics

Source: https://www.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-vectors/a/vector-magnitude-normalization

- Length (also known as magnitude) of a vector

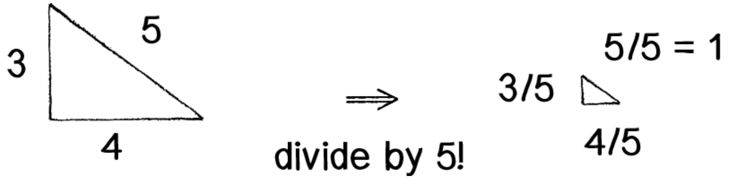
$$\|a\| = \sqrt{a^T \cdot a} = \sqrt{\sum_{i=1}^{n} a_i^{\,2}}$$

  - For example,



  - Unit vector: if $||a||=1$
  - For a given vector u, its unit vector is calculated as follows (normalization)
    - We divide each component by the length

$$\hat{u} = \frac{\vec{u}}{||\vec{u}||}$$

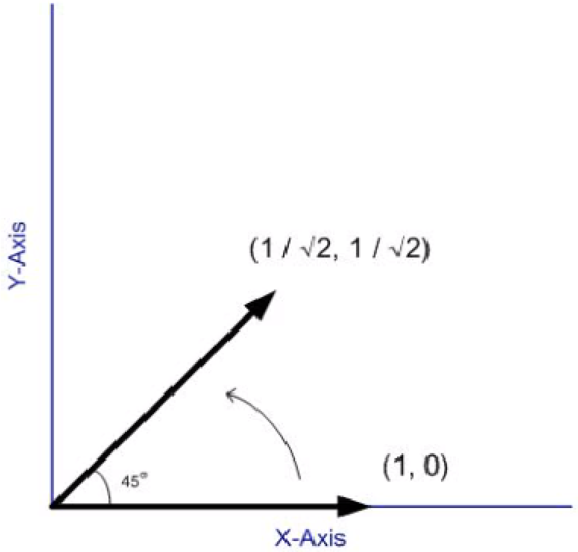


Computing PCA basics: Eigenvectors and eigenvalues

- Consider a matrix D_{nxn} and a vector x
- If we multiply D with x, we get a new vector $y = Dx$
- The matrix D acting on x does two things to x:
  - It scales the vector
  - It rotates the vector

Example 1: Only rotation

$$\begin{bmatrix} \cos(45°) & -\sin(45°) \\ \sin(45°) & \cos(45°) \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$



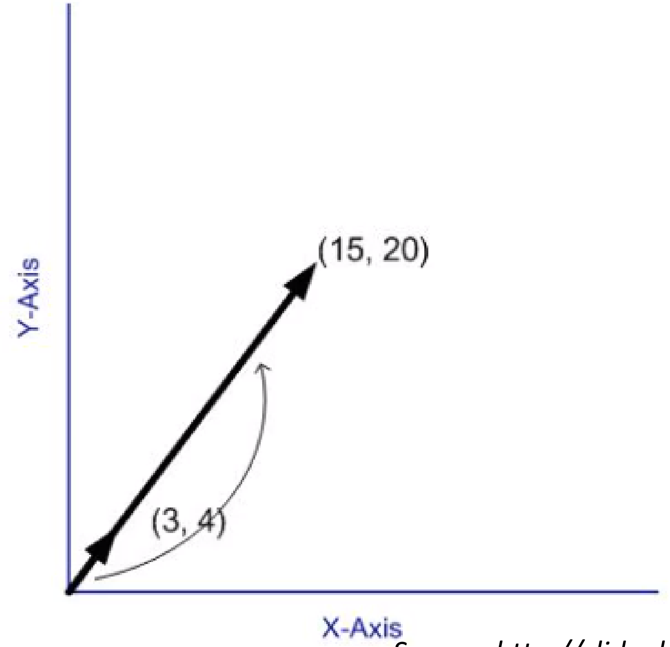
Source: http://slideplayer.com/slide/4377087/

Computing PCA basics: Eigenvectors and eigenvalues

- Consider a matrix D_{nxn} and a vector x
- If we multiply D with x, we get a new vector $y = Dx$
- The matrix D acting on x does two things to x:
  - It scales the vector
  - It rotates the vector

Example 2: Only scaling

$$\begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \times \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 20 \end{bmatrix}$$



Source: http://slideplayer.com/slide/4377087/

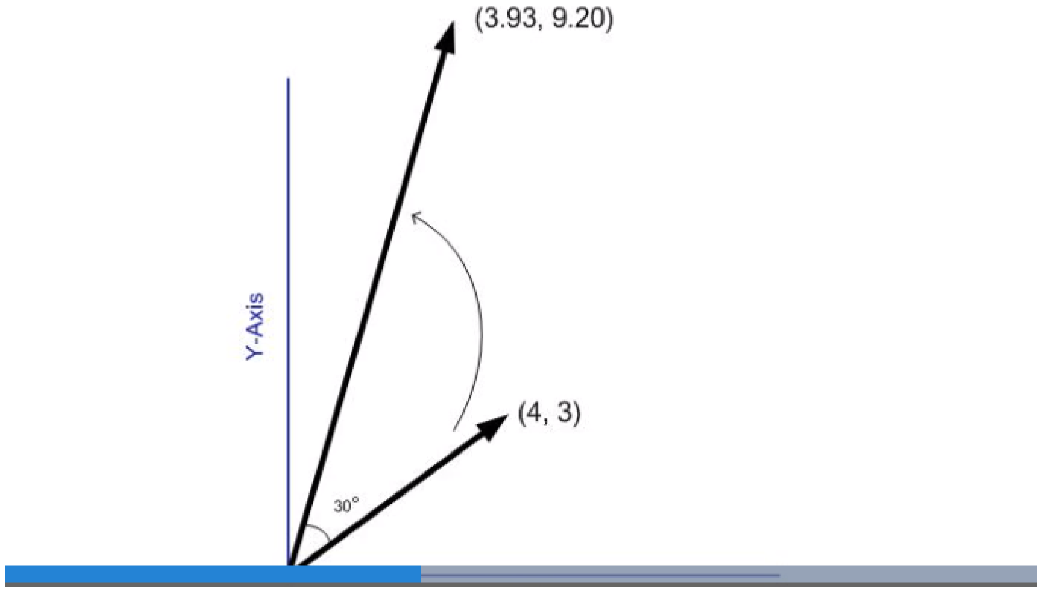
Computing PCA basics: Eigenvectors and eigenvalues

- Consider a matrix D_{nxn} and a vector x
- If we multiply D with x, we get a new vector $y = Dx$
- The matrix D acting on x does two things to x:
  - It scales the vector
  - It rotates the vector

- However, for any matrix D, there are some favored vectors/directions. When the matrix acts on these favored vectors, the action essentially results in just scaling the vector. There is no rotation.
- These favored vectors are precisely the eigenvectors and the amount by which each of these favored vectors is scaled (stretched or compressed) is the eigenvalue.

Example 3: Both

$$\begin{bmatrix} 2\cos(30^\circ) & -2\sin(30^\circ) \\ 2\sin(30^\circ) & 2\cos(30^\circ) \end{bmatrix} \times \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 3.93 \\ 9.20 \end{bmatrix}$$



Source: http://slideplayer.com/slide/4377087/

Computing PCA: Eigenvectors and eigenvalues formally

- Let D be a square dxd matrix.
- A non-zero vector v is called an eigenvector of D if and only if there exists a scalar (i.e., a single number) λ such that:

$$Dv = \lambda v$$

- That is, the multiplication by D alters only the scale of v, but does not change its direction
  - If such a number λ_i exists it is called an eigenvalue of D.
  - The vector v_i is called the eigenvector associated with λ_i.
  - For example, C_3 is an eigenvector for A associated with the eigenvalue 3 as

$$AC_3 = 3C_3$$

$$A = \begin{bmatrix} 1 & 2 & 1 \\ 6 & -1 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad C_3 = \begin{bmatrix} 2 \\ 3 \\ -2 \end{bmatrix}$$

Computing PCA basics: Eigenvectors and eigenvalues

- How to find the eigenvalues/eigenvectors of *D*?
  - By solving: $Dv = \lambda v$, which can be rewritten as $(D-\lambda I)v=0$
  - I is the identity matrix (dxd matrix), $\mathit{0}$ is a vector of all zeros
  - From linear algebra, in order for $(D-\lambda I)v=0$ to hold, the determinant should be zero, i.e.,

$$\det(D-\lambda I)=0$$

    - where $det(A)$ or $|A|$ is the determinant of matrix A

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

  - By solving $\det(D-\lambda I)=0$ we get the eigenvalues λ_i
    - is a d^{th}-degree polynomial in λ
  - For each eigenvalue λ_i, we can find its eigenvector by solving then the equation

$$Dv_i=\lambda_i v_i \qquad \text{or, } (D-\lambda_i)v_i=0$$

Computing PCA basics: Eigenvectors decomposition

- Let *D* be *dxd* square matrix.
- Eigenvalue decomposition of the data matrix

$$D = V \Lambda V^T$$

$$V = (v_1, \cdots, v_d) \quad with \underset{i \neq j}{\forall} \langle v_i, v_j \rangle = 0 \quad and \overset{d}{\underset{i=1}{\forall}} \|v_i\| = 1$$

$$\Lambda = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_d \end{pmatrix}$$

Each eigenvector is a unit vector

The eigenvectors are linearly independent

The corresponding eigenvalues

- The columns of *V* are the eigenvectors of *D*, ordered as largest eigenvalue first.
- The diagonal elements of *Λ* are the eigenvalues of *D* (largest first in the diagonal, elements not in diagonal are 0)

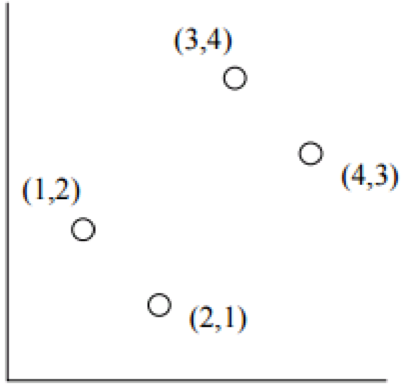
PCA steps

1. Compute the covariance matrix S of D
2. Compute the eigenvalues and the corresponding eigenvectors of S
3. Select the k biggest eigenvalues and their eigenvectors (V)
4. The k selected eigenvectors represent an orthogonal basis; the rest are ignored.
5. Transform the original $n \times d$ data matrix D with the $d \times k$ basis V:
   - V is the transformation we were looking for

$$D' = D \cdot \mathrm{V} = \begin{pmatrix} \mathrm{x}_1 \\ \vdots \\ \mathrm{x}_\mathrm{n} \end{pmatrix} (v_1, \cdots, v_k) = \begin{pmatrix} \langle \mathrm{x}_1, v_1 \rangle & \cdots & \langle \mathrm{x}_1, v_k \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathrm{x}_\mathrm{n}, v_1 \rangle & \cdots & \langle \mathrm{x}_\mathrm{n}, v_k \rangle \end{pmatrix}$$

Example of transformation

- Original



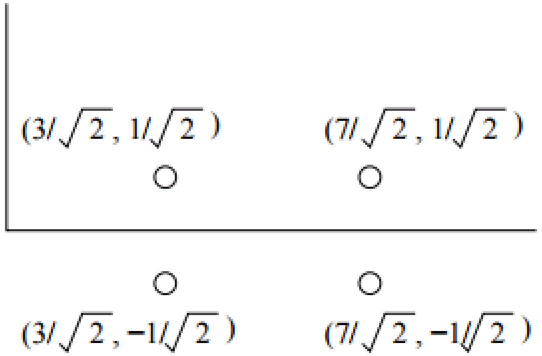
Eigenvectors

$$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \qquad \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

- Transformed data

$$\begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 3/\sqrt{2} & 1/\sqrt{2} \\ 3/\sqrt{2} & -1/\sqrt{2} \\ 7/\sqrt{2} & 1/\sqrt{2} \\ 7/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

In the rotated coordinate system



Source: http://infolab.stanford.edu/~ullman/mmds/ch11.pdf

Percentage of variance explained by PCA

- Let k be the number of top eigenvalues out of d (d is the number of dimensions in our dataset)
- The percentage of variance in the dataset explained by the k selected eigenvalues $\lambda_1,\ldots,\lambda_k$ is:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i}$$

- Similarly, you can find the variance explained by each principal component
- Rule of thumb: keep enough to explain **85%** of the variation

PCA results interpretation

- Example: iris dataset (d=4), results from R
- 4 principal components

	PC1	PC2	PC3	PC4
Sepal.Length	0.5038236	-0.45499872	0.7088547	0.19147575
Sepal.Width	-0.3023682	-0.88914419	-0.3311628	-0.09125405
Petal.Length	0.5767881	-0.03378802	-0.2192793	-0.78618732
Petal.Width	0.5674952	-0.03545628	-0.5829003	0.58044745

Importance of components:

	PC1	PC2	PC3	PC4
Proportion of Variance	0.7331	0.2268	0.03325	0.00686
Cumulative Proportion	0.7331	0.9599	0.99314	1.00000

- PC1 is the most significant dimension, followed by PC2 and so on and so forth.
- With PC1 and PC2 ~96% of the variance is explained

Inverse transformation of PCA (reconstruction)

- Transformation is done by:

$$D' = D \cdot V$$

- To reconstruct data from the transformed version, we multiply besides by the **transpose** of V:

$$D' \cdot V^T = D \cdot V \cdot V^T$$

$VV^T = I$, because V consists of normalized eigenvectors which are orthogonal to each other

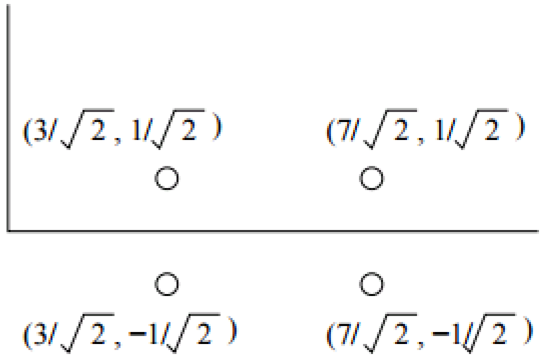
$$D = D \cdot I = D \cdot (VV^T) = D' \cdot V^T$$

$$D = D' \cdot V^T = \begin{pmatrix} x_1' \\ \vdots \\ x_k' \end{pmatrix} (v_1, \cdots, v_k)^T$$

Note: A perfect reconstruction (without any errors) is possible only if the number of selected eigenvectors k equals to the original dimensionality d.

Example of inverse transformation

- In the rotated coordinate system



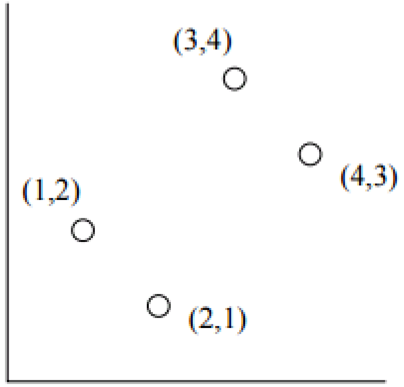
Eigenvectors

$$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \qquad \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

- Reconstruction

$$\begin{bmatrix} 3/\sqrt{2} & 1/\sqrt{2} \\ 3/\sqrt{2} & -1/\sqrt{2} \\ 7/\sqrt{2} & 1/\sqrt{2} \\ 7/\sqrt{2} & -1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \\ 4 & 3 \end{bmatrix}$$

Reconstruction



Source: *http://infolab.stanford.edu/~ullman/mmds/ch11.pdf*

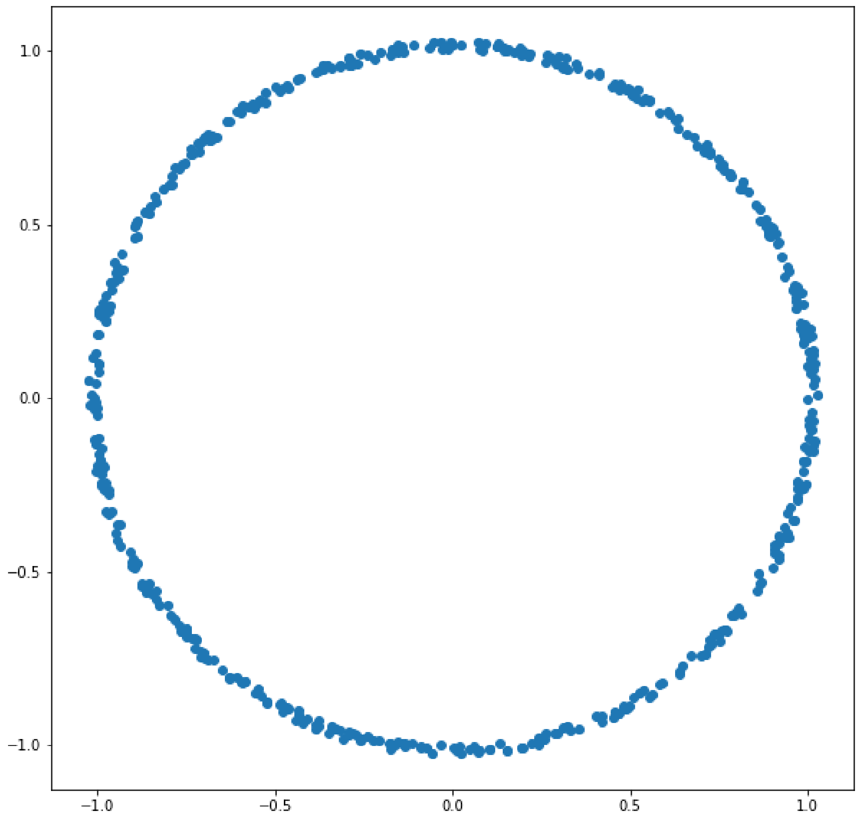
Outline

- Introduction
- Principal Component Analysis (PCA)
- Autoencoders
- Things you should know from this lecture & reading material

Autoencoders: Motivation

- PCA provides the optimal solution for linear dimensionality reduction.

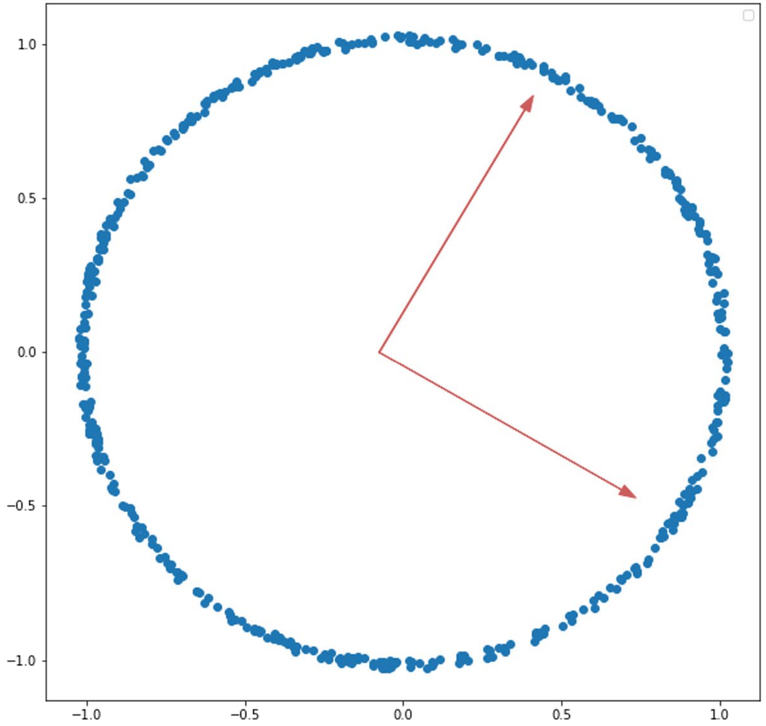
- But what if the data manifold is non-linear?



Autoencoders: Motivation

- PCA provides the optimal solution for linear dimensionality reduction.
- But what if the data manifold is non-linear?
- What should be the principle component for $k = 1$?

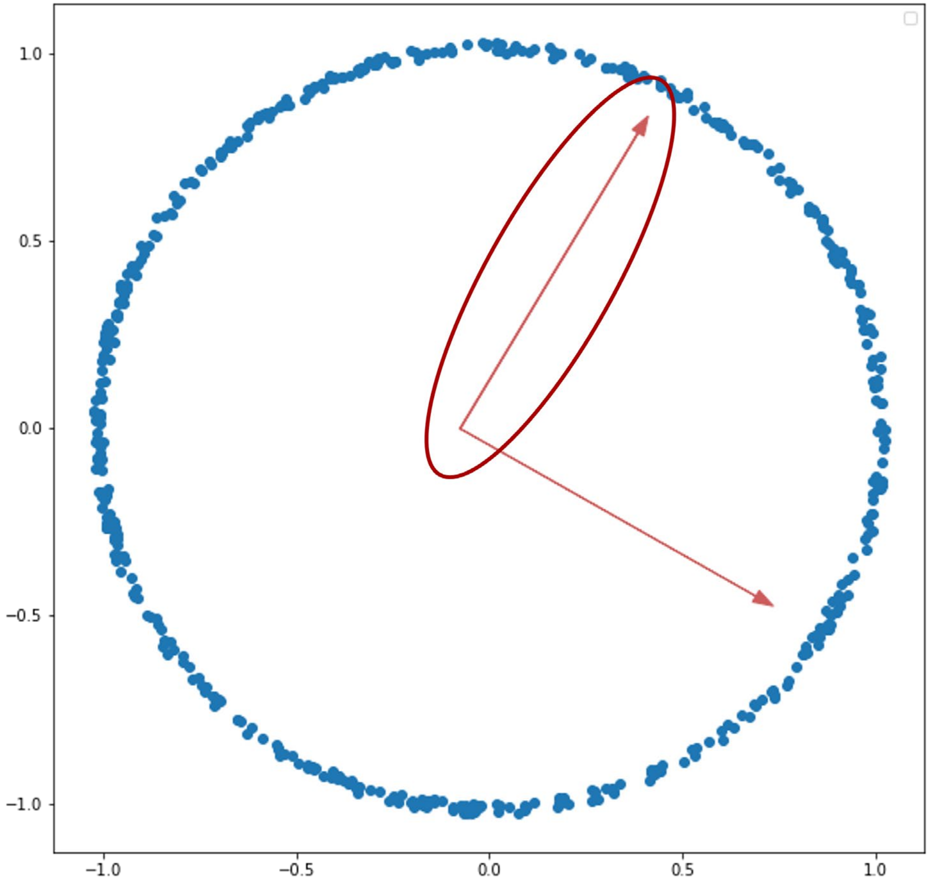
PCA



Autoencoders: Motivation

- PCA provides the optimal solution for linear dimensionality reduction.
- But what if the data manifold is non-linear?
- What should be the principle component for $k = 1$?
- We choose to use the first principle component

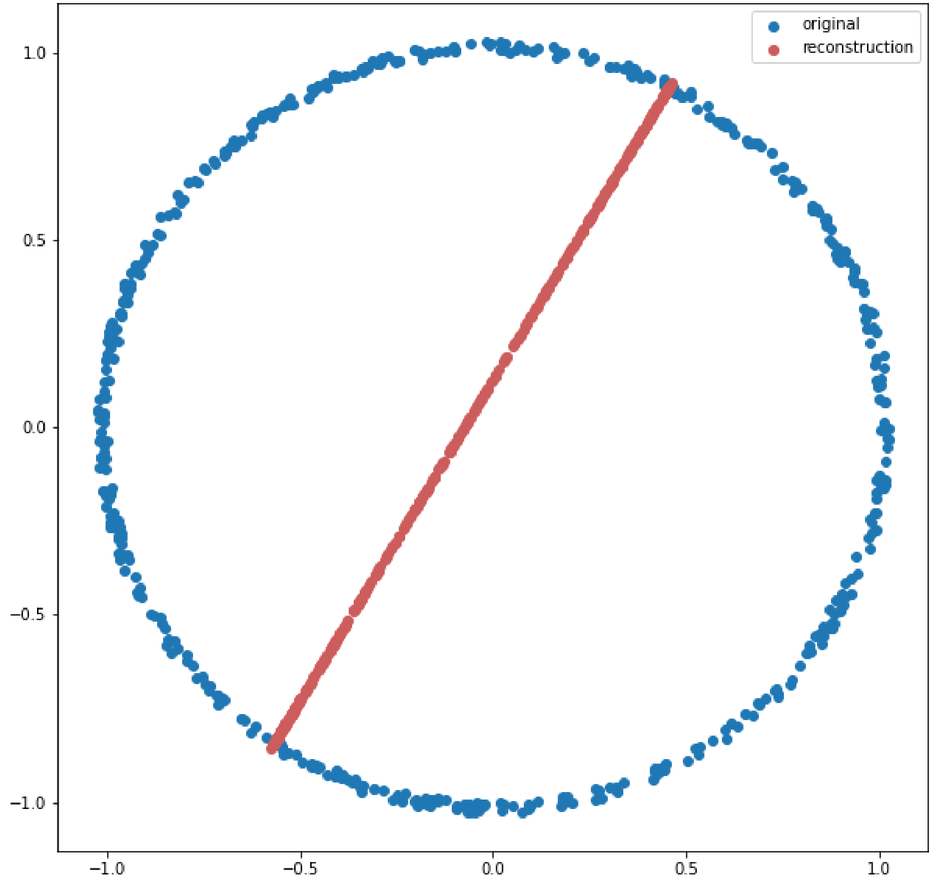
PCA



Autoencoders: Motivation

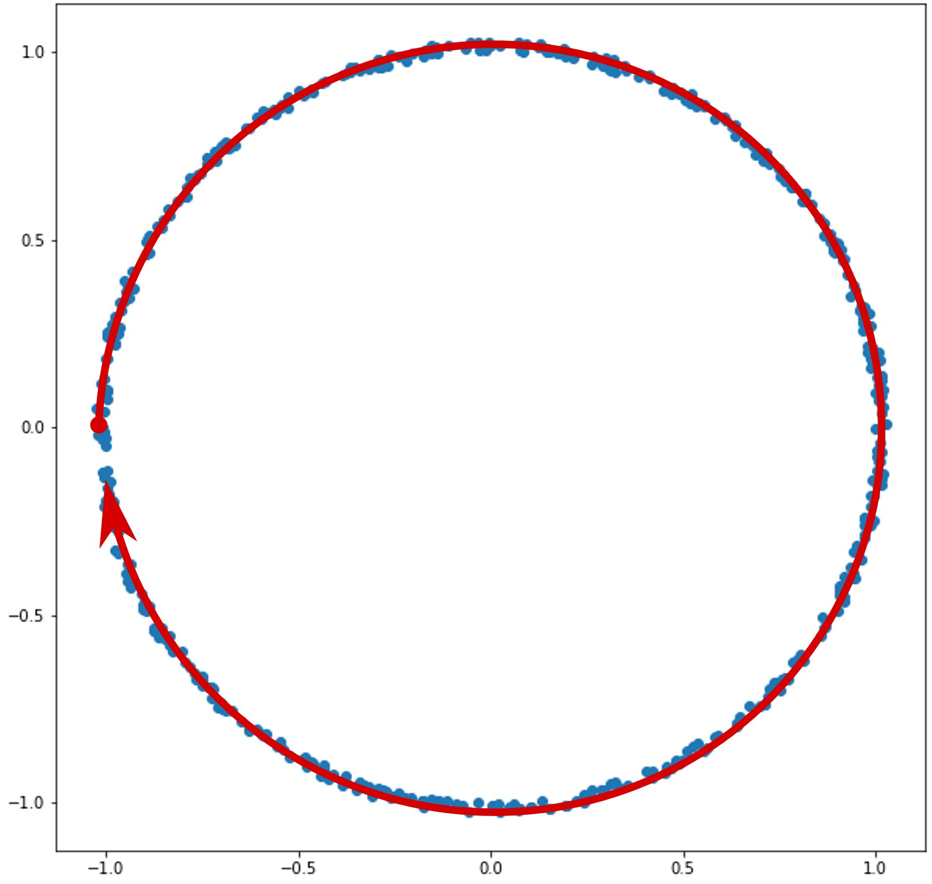
- PCA provides the optimal solution for linear dimensionality reduction.
- But what if the data manifold is non-linear?
- What should be the principle component for $k = 1$?
- We choose to use the first principle component
- Linear projection is not capable to capture information from both dimensions.

Inverse transform from PCA



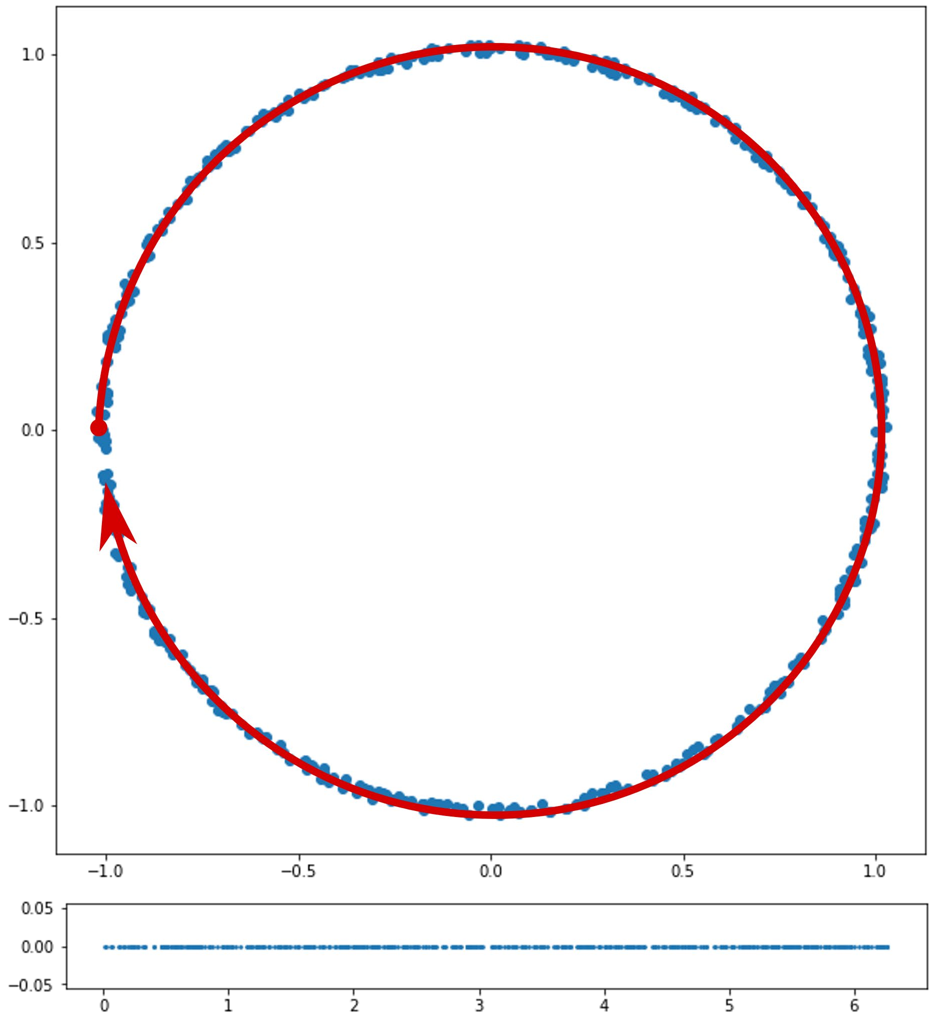
Autoencoders: Non-linear dimensionality reduction

- Imagine we can apply the compression along the direction of the ring.



Autoencoders: Non-linear dimensionality reduction

- Imagine we can apply the compression along the direction of the ring.
- Points in 2-D space can be compressed into 1-D space.
- And a (nearly) perfect reconstruction is accessible by keeping the manifold in memory.



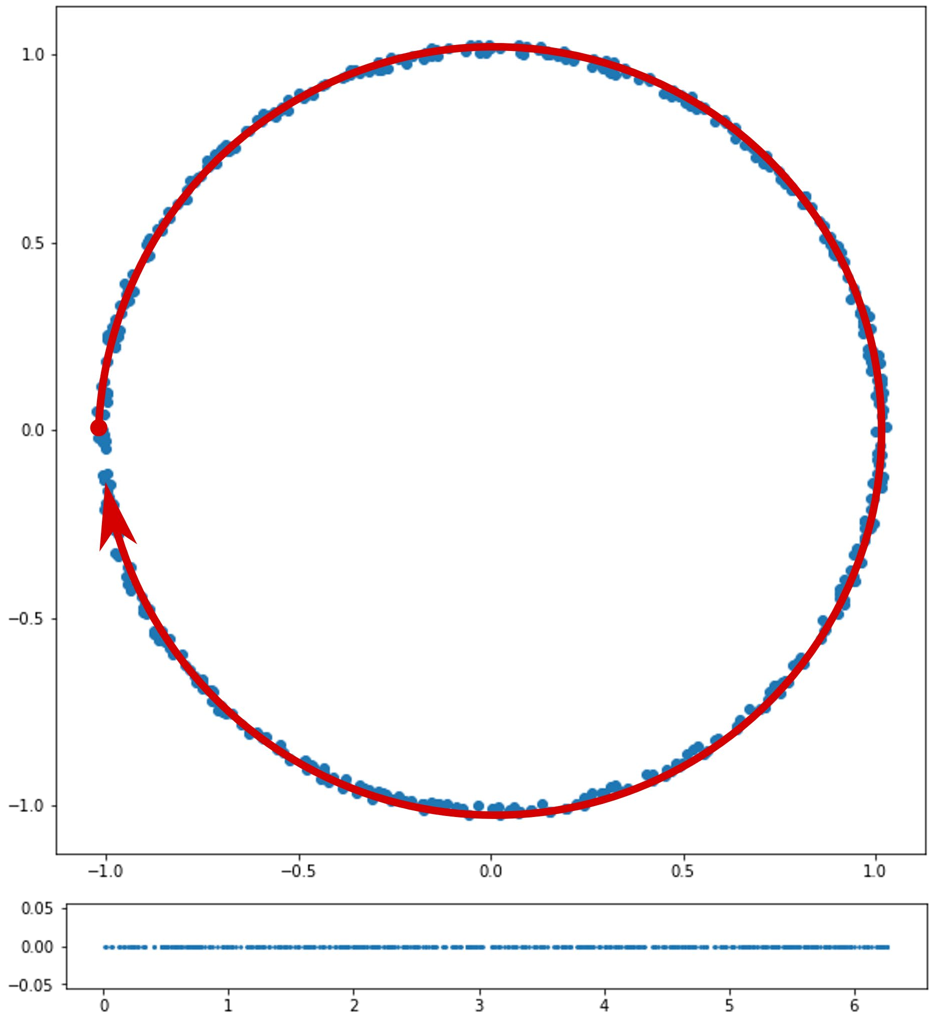
Autoencoders: Non-linear dimensionality reduction

- Imagine we can apply the compression along the direction of the ring.

- Points in 2-D space can be compressed into 1-D space.

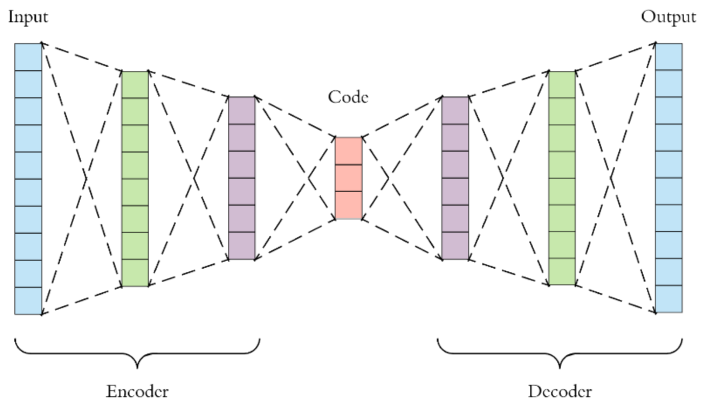
- And a (nearly) perfect reconstruction is accessible by keeping the manifold in memory.

- This can be achieved by autoencoders.



Autoencoders: Basic idea

- (Vague definition) An autoencoder is a type of artificial neural network whose outputs are its own inputs
  - (Differently from traditional NNs) An autoencoder is not trained to produce a class, f(x)=y but rather to reproduce its input at the output layer f(x)=x (identity function)

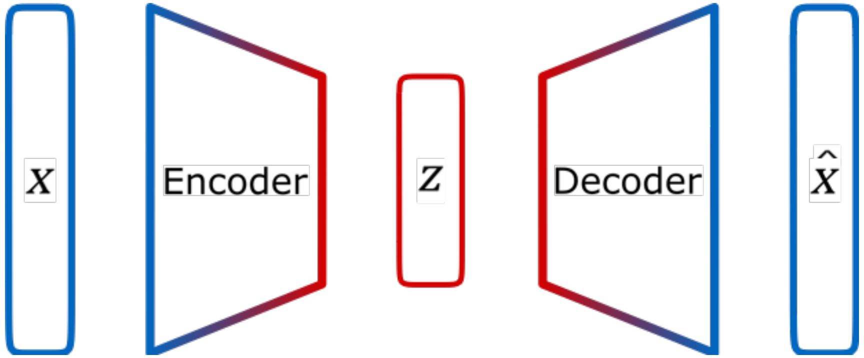


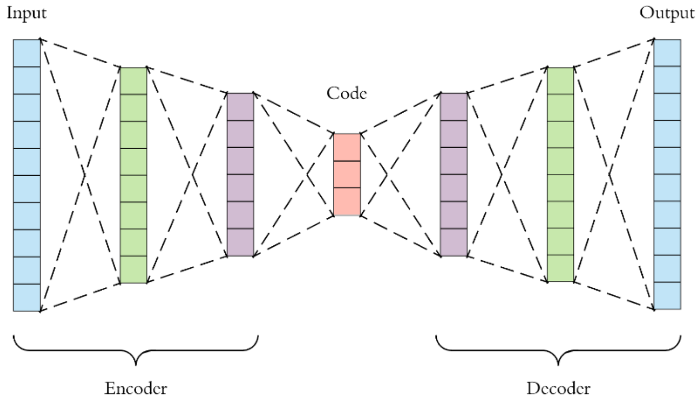
- What is special about autoencoders?
  - We want to compress the input into a lower-dimensional code

  (bottleneck layer z) and reconstruct the output from this representation
- Main idea:
  - Given instances X (no target outputs -> unsupervised learning)
  - Compress the input into a lower-dimensional code (bottleneck layer z).
    - A bottleneck constrains the amount of information that can traverse the full network, forcing a learned compression of the input data.
  - Reconstruct the output from this representation.
- (Better definition) An autoencoder is a type of artificial neural network for learning a lower-dimensional feature representation from unlabeled training data (unsupervised).

Autoencoders: Architecture

- An autoencoder consists of the following components:
  - Encoder: map the input x into a lower dimensional latent space z (bottleneck)
  - Latent representation: the compressed representation z of the input
  - Decoder: map the latent representation to a reconstruction of the input $\hat{x}$

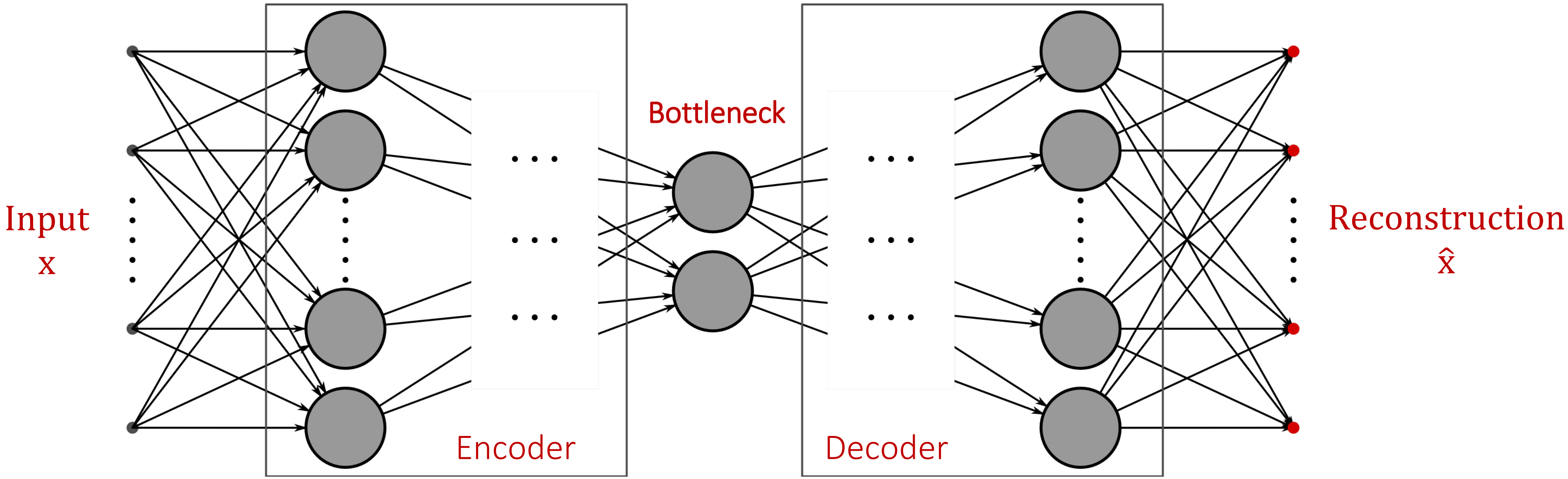




- How to achieve bottleneck? Fewer neurons, i.e., the intermediate layer (latent space) should be of much lower dimensionality

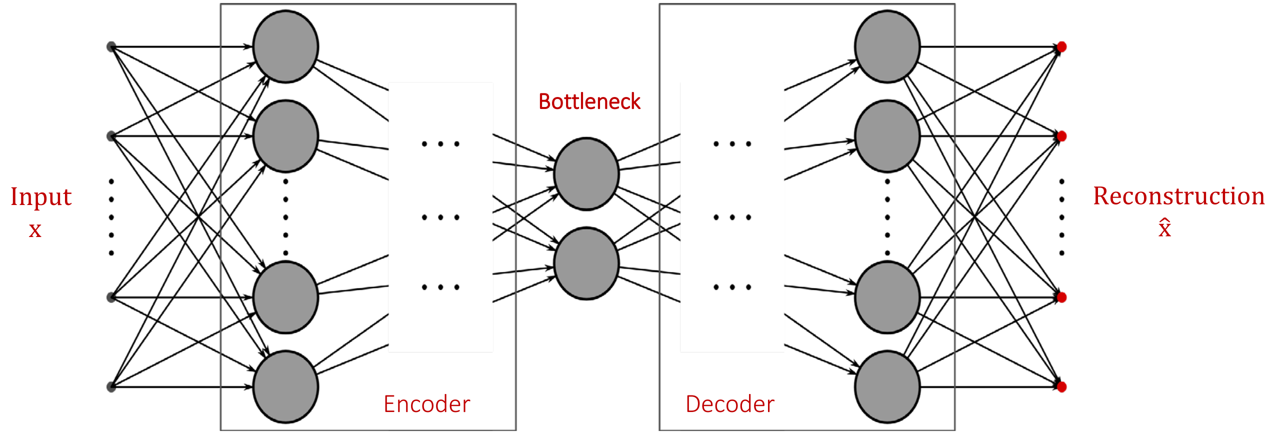
Autoencoders: Training

- The network is trained so that to minimize the reconstruction loss *L*(x, $\hat{x}$), e.g., squared loss
  - x is the original instance
  - $\hat{x}$ is the reconstructed one.
- So, the encoder is forced to capture an informative representation to benefit the reconstruction.
- The better reconstruction indicates the more informative latent representation.



Autoencoders: Training

- The network is trained so that to minimize the reconstruction loss *L(x, $\hat{x}$)*, e.g., squared loss
  - *x* is the original instance
  - $\hat{x}$ is the reconstructed one.



- The encoder is trained to learn a function f that maps the input into the latent space z (σ is the activation function):

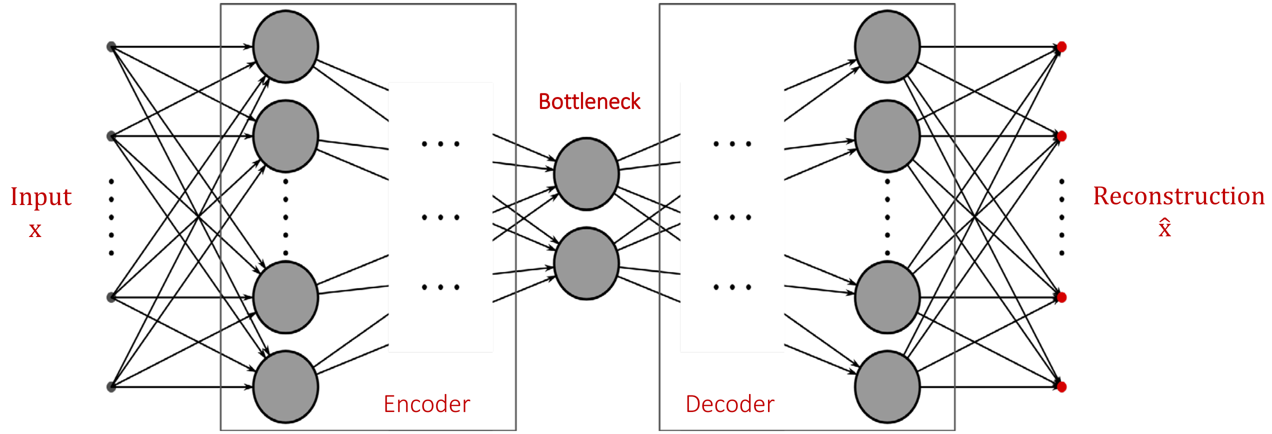
$$z = f(x) = \sigma(Wx + b)$$

- The decoder is trained to learn a function g that reconstructs the input from the latent representation z (σ is the activation function) :

$$\hat{x} = g(z) = \sigma'(W'z + b')$$

Autoencoders: Training

- The network is trained so that to minimize the reconstruction loss *L(x, $\hat{x}$)*, e.g., squared loss
  - *x* is the original instance
  - $\hat{x}$ is the reconstructed one.



- The goal of training is to find the optimal parameter set that minimize the reconstruction loss.

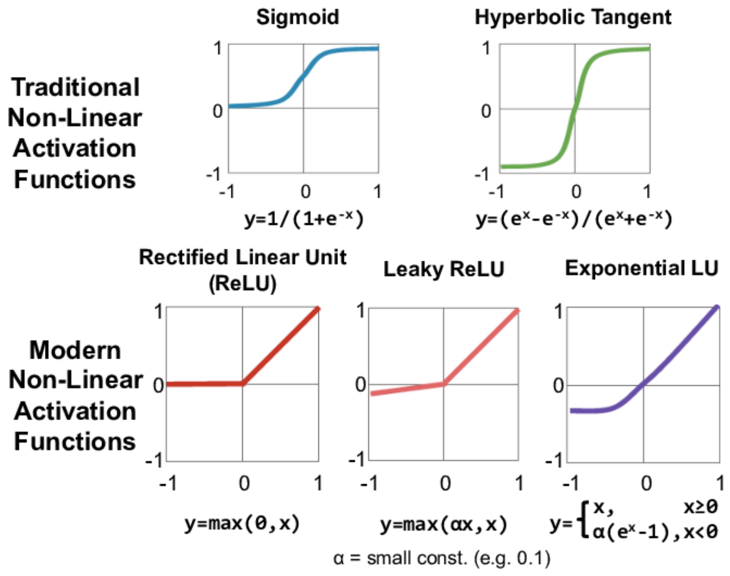
$$\underset{W,W',b,b'}{\arg\min} \left\| x - \sigma'\left(W'\big(\sigma(Wx+b)\big)+b'\right) \right\|$$

- Encoder and decoder are (typically) symmetric
  - This allows for e.g., weight sharing, makes hyperparameter tuning more efficient etc.

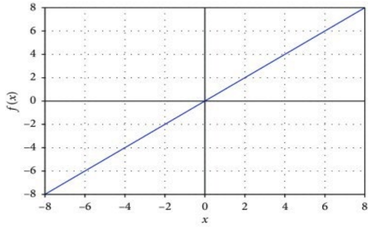
Autoencoders: Training

- To train an autoencoder for non-linear dimensionality reduction, the activation function σ must be non-linear (e.g. ReLU).

- Otherwise, it can only learn a linear function and ends up in approximating PCA.
  - But the autoencoders weights are not equal to the principle components, and are generally not orthogonal.

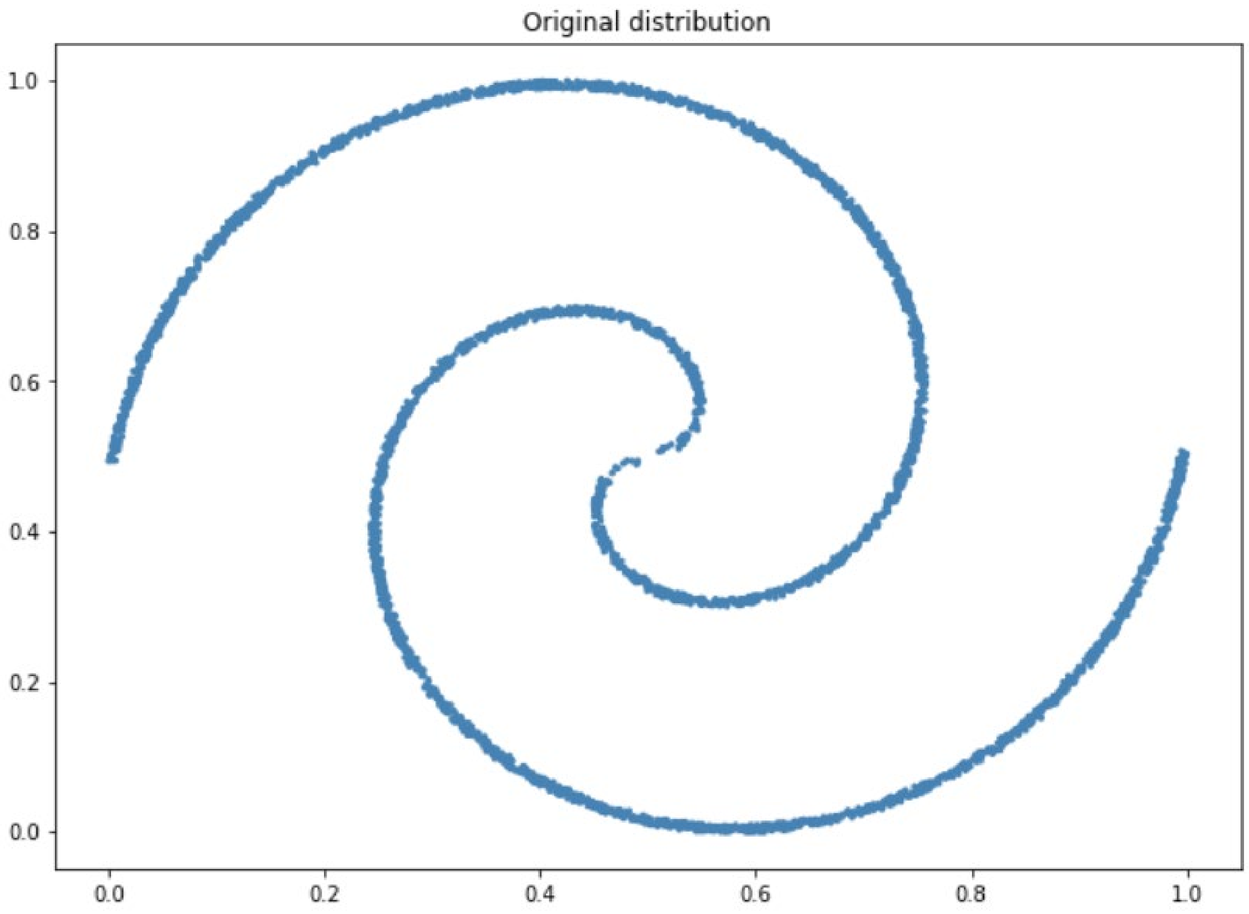


Source

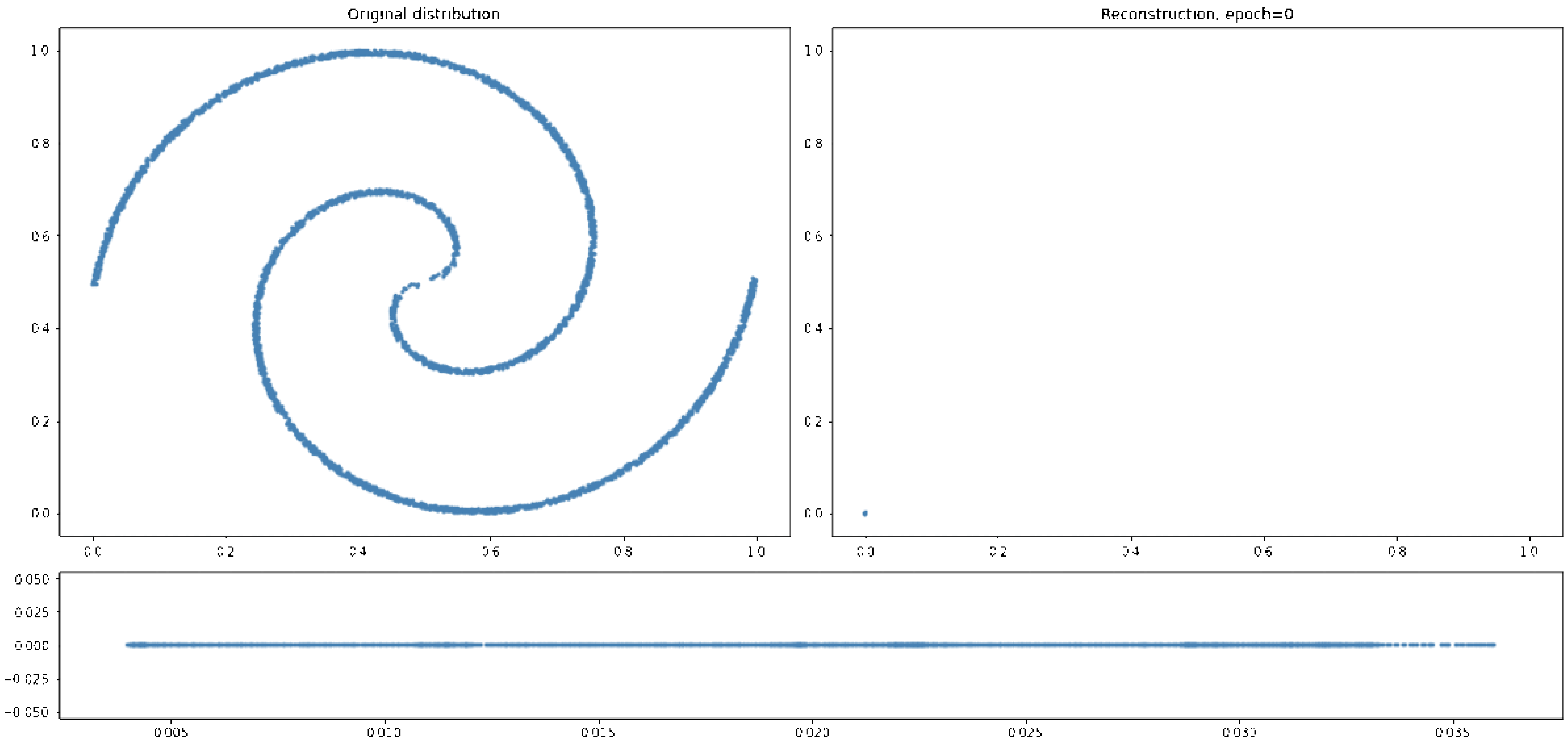


Linear function

Autoencoders: Training an autoencoder

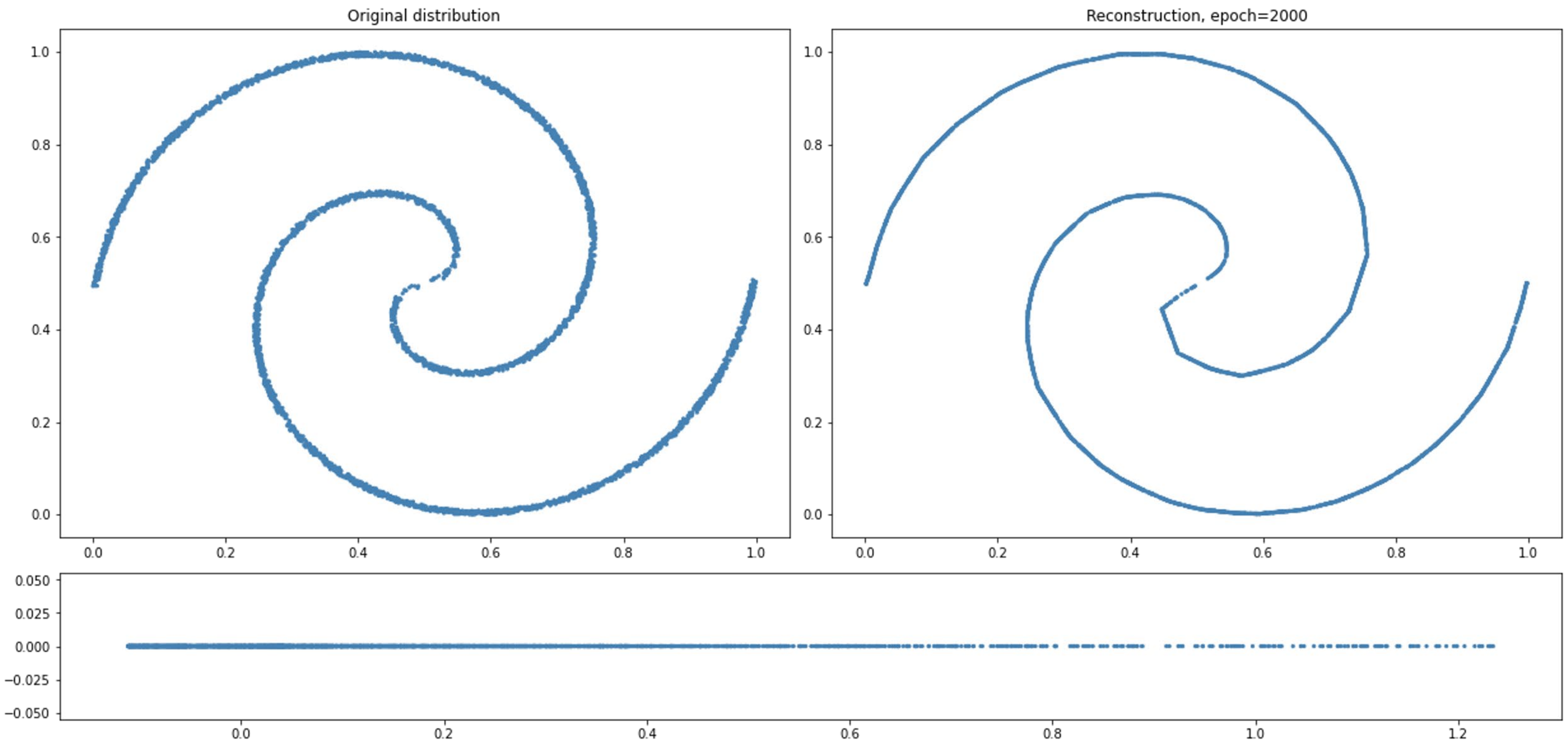


Autoencoders: Training an autoencoder



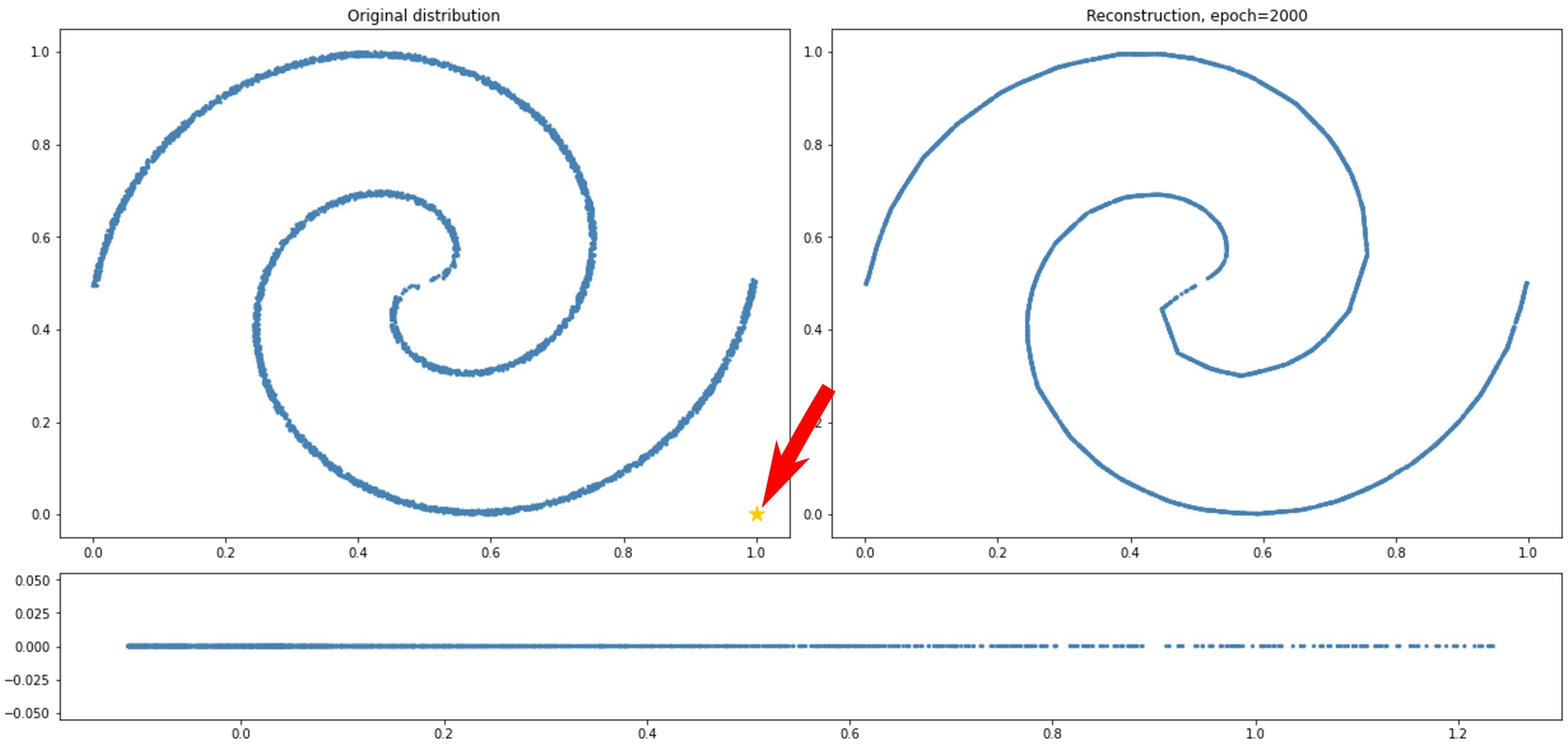
Autoencoders: Training an autoencoder

- The autoencoder learns the data manifold and achieves an accurate reconstruction.



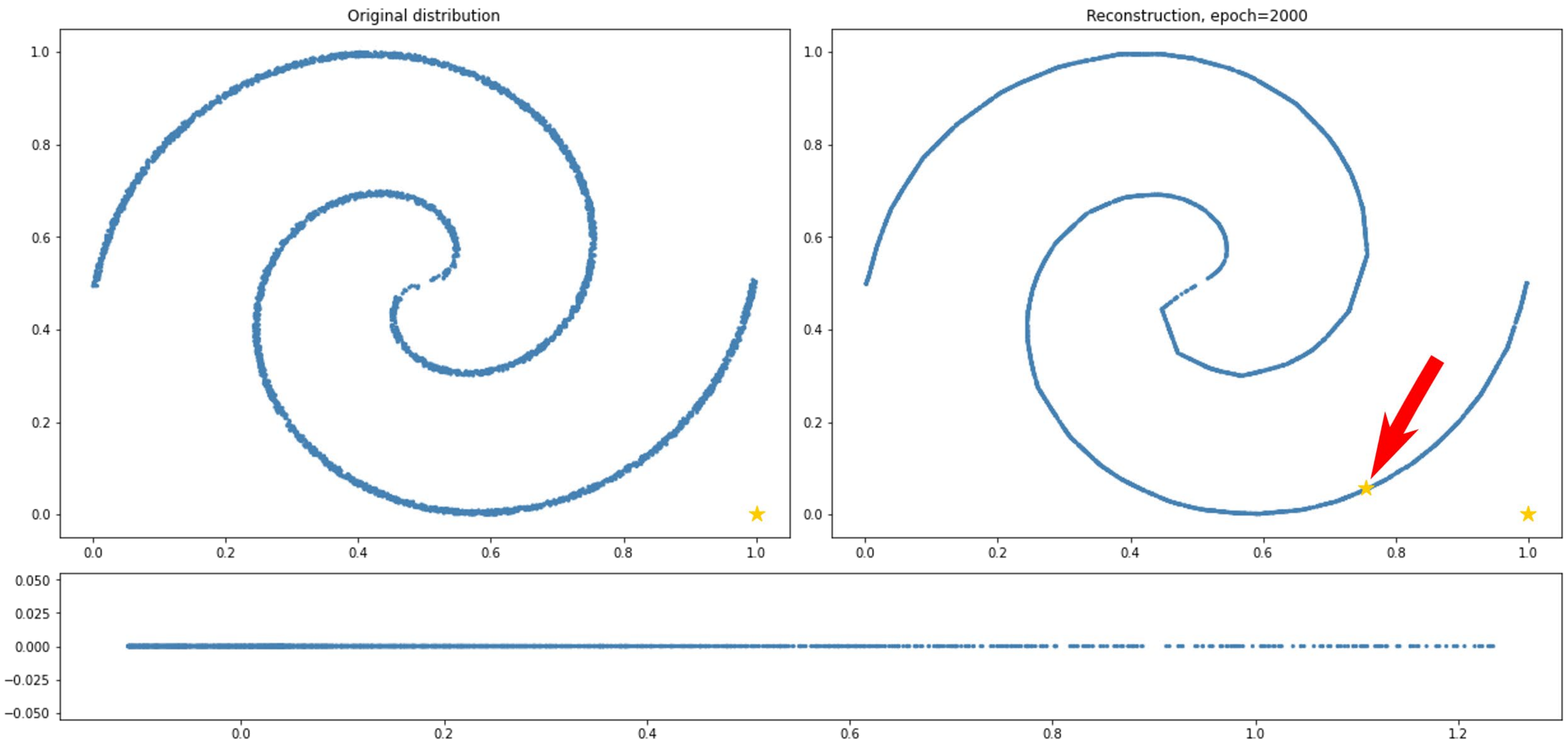
Autoencoders: Anomaly detection

- But how would the reconstruction of a point out of distribution be?



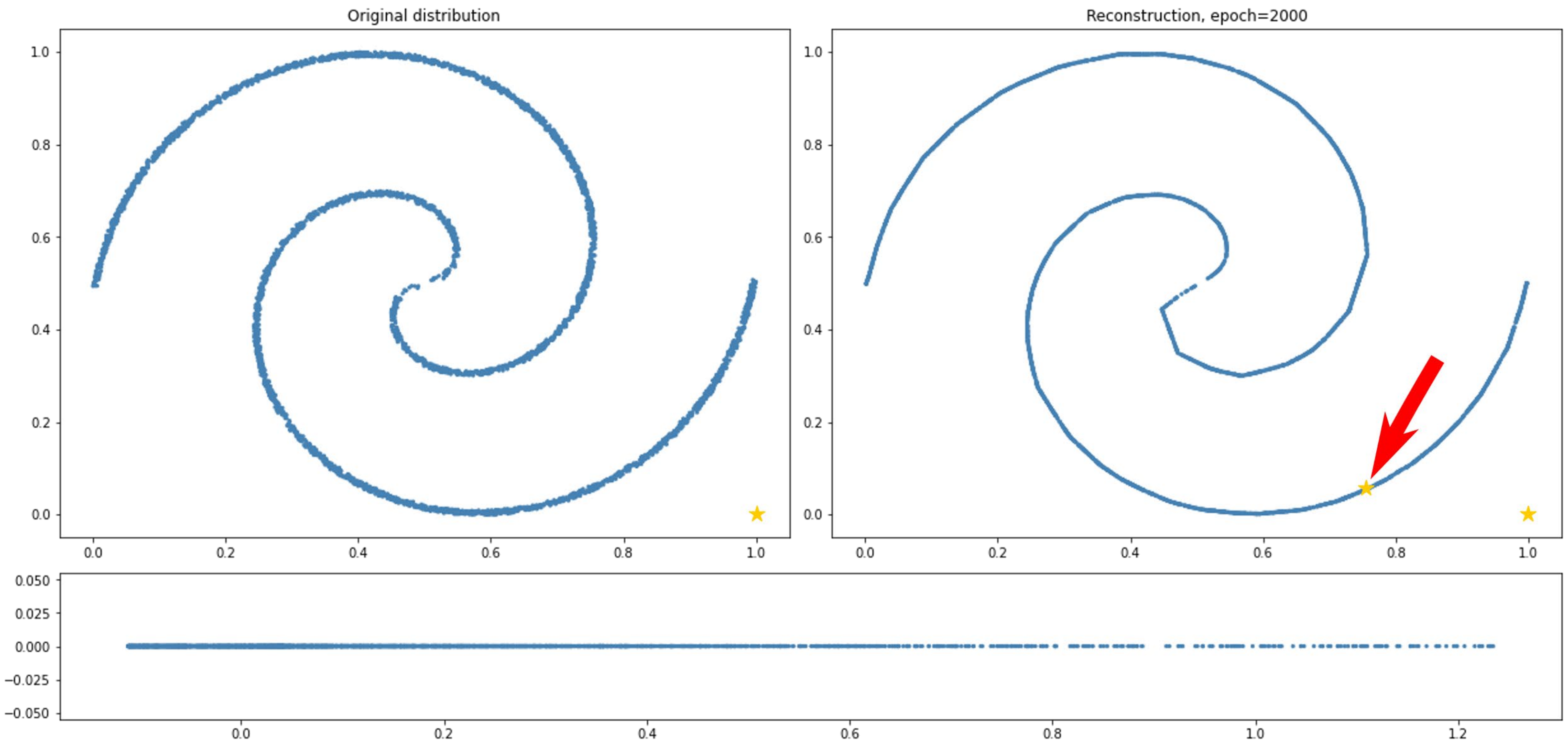
Autoencoders: Anomaly detection

- The reconstruction will still follow the manifold.



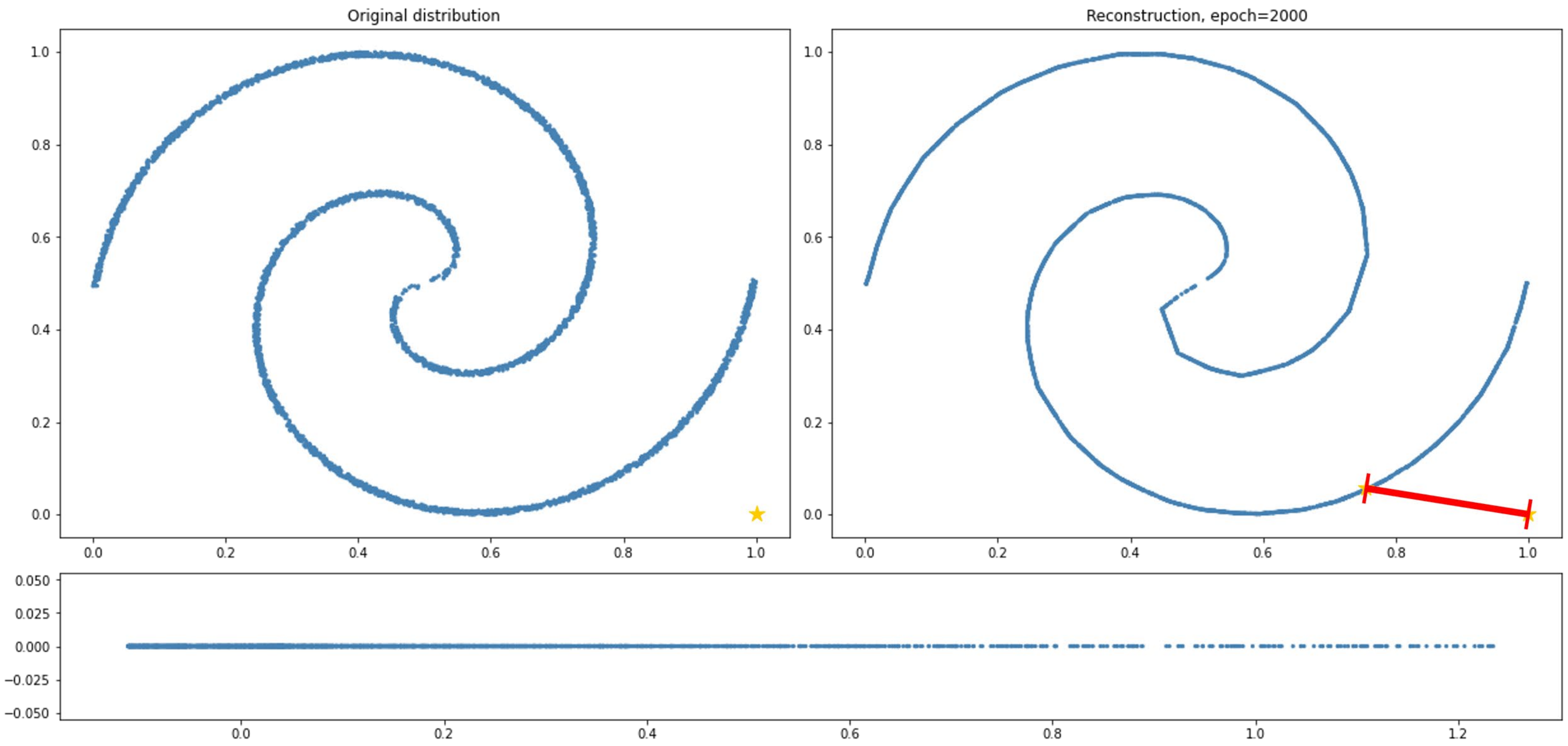
Autoencoders: Anomaly detection

- The reconstruction will still follow the manifold.
- This is because the autencoder learns to reproduce the most frequently observed charateristics.



Autoencoders: Anomaly detection

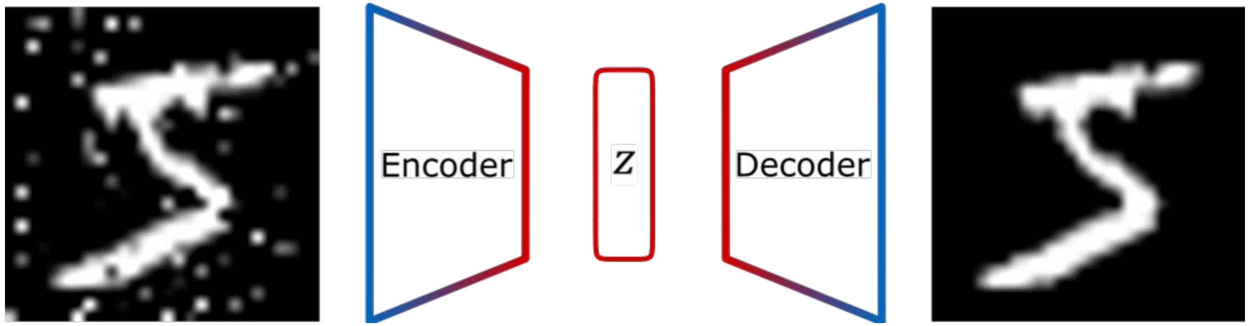
- The reconstruction loss of anomalies can be apparently higher.
- This property of autoencoders can be used for anomaly detection.



Autoencoders: Applications

- What we have mentioned:
  - Dimensionality Reduction
  - Anomaly detection

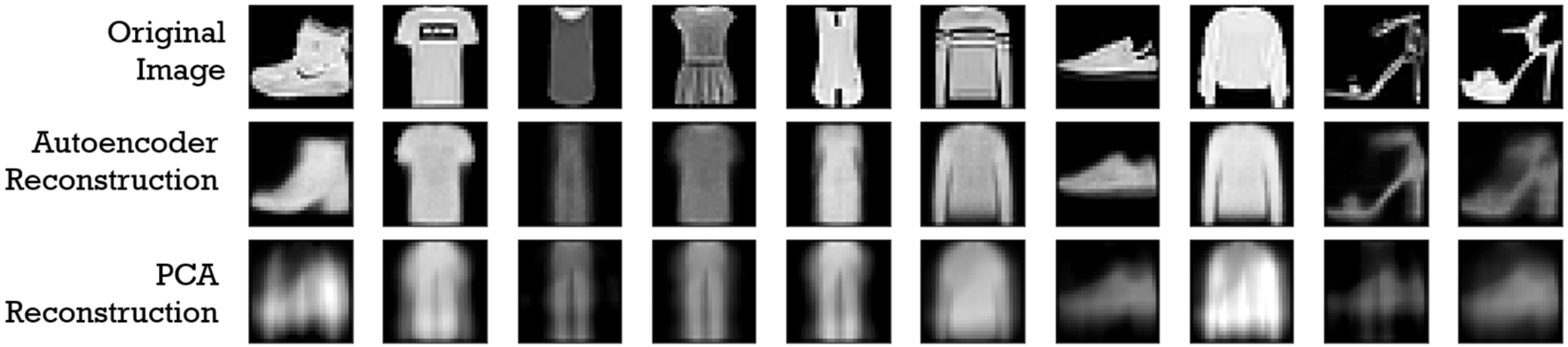
- Other applications:
  - Image compression
  - Image denoising



Autoencoder vs. PCA

- Autoencoder
  - Complex non-linear functions
  - Features might be correlated
  - Computationally expensive

- PCA
  - Linear transformation
  - Uncorrelated features
  - Faster



Source: https://en.wikipedia.org/wiki/Autoencoder

Outline

- Introduction
- PCA
- Autoencoders
- Things you should know from this lecture & reading material

Things you should know from this lecture

- Dimensionality reduction
- Linear dimensionality reducation → PCA
- Non-linear dimensionality reduction → autoencoders

Hands on experience

- Compare PCA vs Autoencoders (and experiment with different network architectures, etc) on your favorite dataset
  - For the comparison/evaluation you can use:
    - Reconstruction error
    - Efficiency
    - Performance in some downstream task

Thank you

Questions/Feedback/Wishes?

Reading material

- Deep Learning book, Chapter 14 https://www.deeplearningbook.org/contents/autoencoders.html
- Eigenvectors and eigenvalues | Chapter 14, Essence of linear algebra, https://www.youtube.com/watch?v=PFDu9oVAE-g
- Hugo Larochelle videos: https://www.youtube.com/watch?v=FzS3tMl4Nsc
- Hinton and Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, https://www.cs.toronto.edu/~hinton/science.pdf
- Hinton, “From PCA to autoencoders” https://www.youtube.com/watch?v=PSOt7u8u23w