Neural Networks

Unsupervised learning techniques

(P-ITEEA-0011)

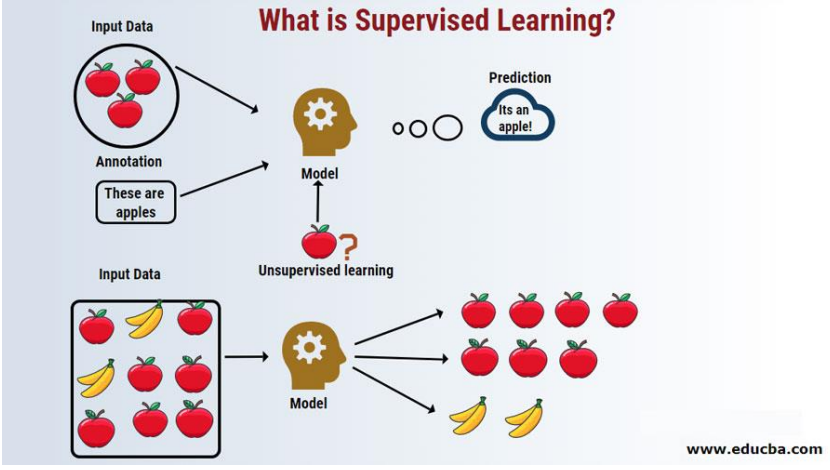
Akos Zarandy
Lecture 9
November 19, 2019

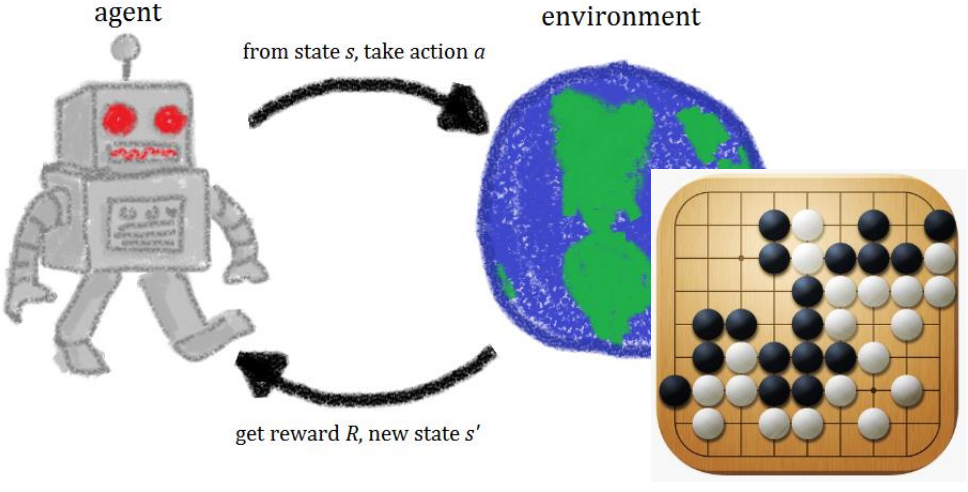
Contents

- Supervised vs unsupervised learning
- Unsupervised learning techniques
  - Curse of dimensionality
  - Principal component analysis (PCA)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Autoencoder

Typical Machine Learning Types

- **Supervised Learning**
  - Learning from labeled examples (for which the answer is known)
- **Unsupervised Learning**
  - Learning from unlabeled examples (for which the answer is unknown)
- **Reinforcement Learning**
  - Learning by trial and feedback, like the “child learning” example





Supervised vs Unsupervised learning

- Supervised learning
  - We have prior knowledge of the desired output
    - Always have data set with ground truth (like image data sets with labels)
  - Typical tasks
    - Classification
    - Regression

- Unsupervised learning
  - No prior knowledge of the desired output
    - Received radio signals from deep space
  - Typical tasks
    - Clustering
    - Representation learning
    - Density estimation

We wish to learn the inherent structure of (patterns in) our data.

Use cases for unsupervised learning

- Exploratory analysis of a large data set
  - Clustering by data similarity
  - Enables verifying individual hypothesizes after analyzing the clustered data

- Dimensionality reduction
  - Represents data with less columns
  - Allows to present data with fewer features
  - Selects the relevant features
  - Enables less power consuming data processing, and/or human analysis

Curse of dimensionality

- What is it?
  - A name for **various problems that arise** when analyzing data in high dimensional space.
  - Dimensions = independent features in ML
    - Input vector size (different measurements, or number of pixels in an image)
  - Occurs when d (# dimensions) is large in relation to n (number of samples).

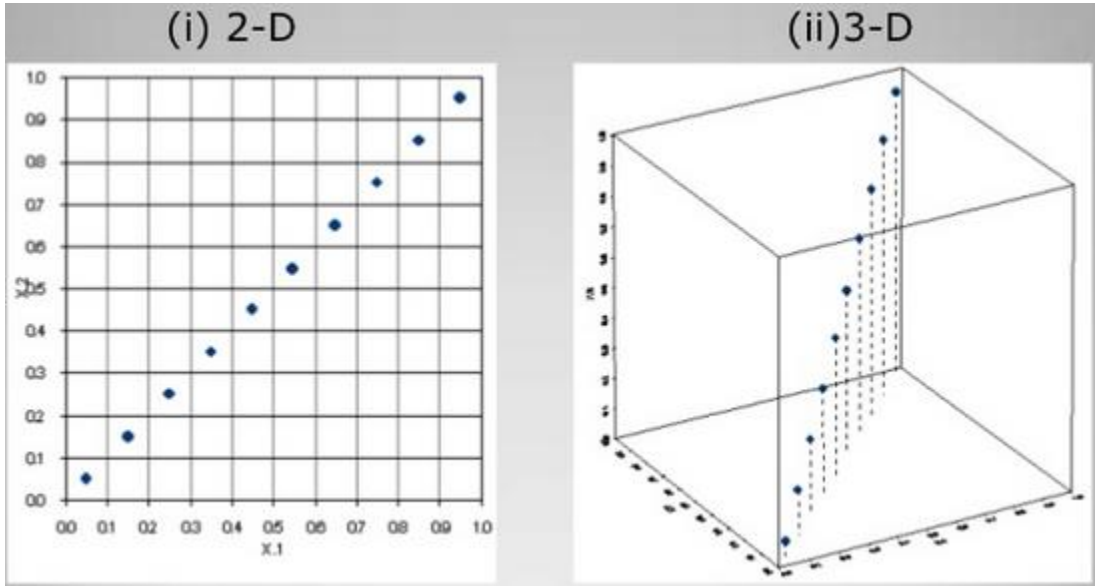
- Real life examples:
  - Genomics
    - We have ~20k genes, but disease sample sizes are often in the 100s or 1000s.

So what is this curse?

- **Sparse data:**
  - When the dimensionality d increases, the volume of the space increases so fast that the available data becomes **sparse, i.e. a few points in a large space**
  - Many features are not balanced, or are 'rarely occur' – sparse features
- **Noisy data**: More features can lead to increased noise → it is harder to find the true signal
- **Less clusters**: Neighborhoods with fixed k points are less concentrated as d increases.
- **Complex features**: High dimensional functions tend to have more complex features than low-dimensional functions, and hence harder to estimate

Data becomes sparse as dimensions increase

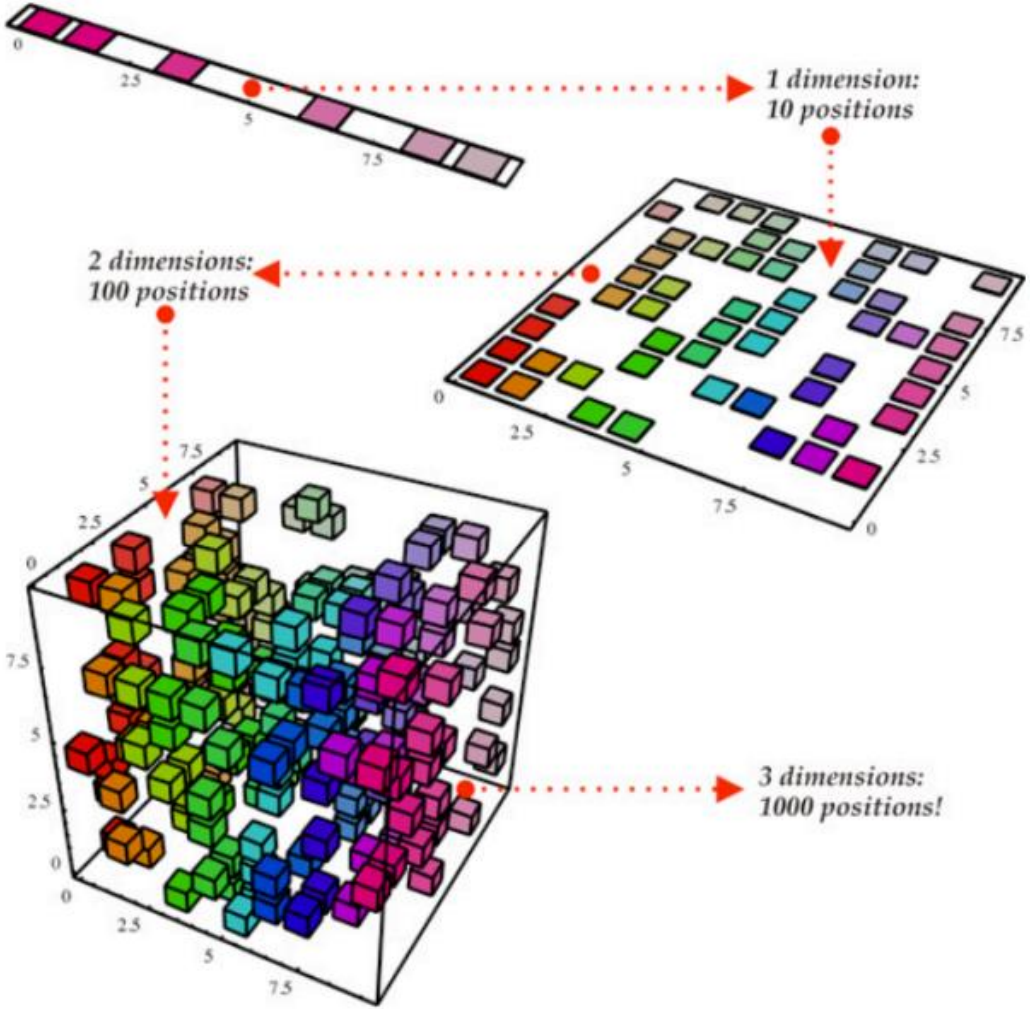
- A sample that maps 10% of the 1x1 squares in 2D represent only 1% of the 1x1x1 cubes in 3D



- There is an exponential increase in the search-space

Data sample number increase to avoid sparsity

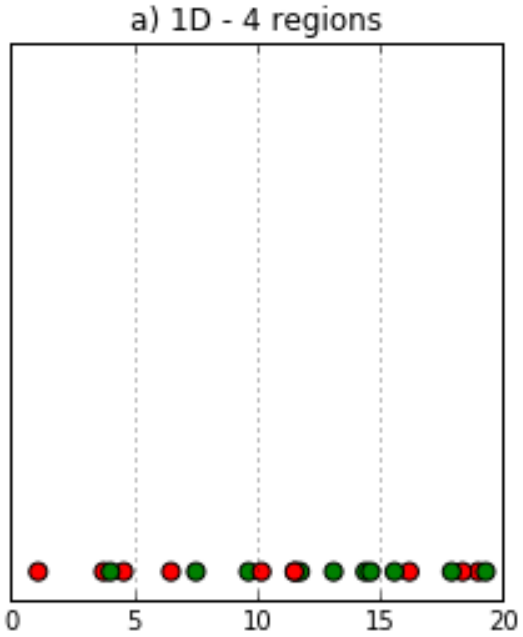
- e.g. 10 observations /dimension
  - 1D: 10 observations
  - 2D: 100 observations
  - 3D: 1000 observations
  - …

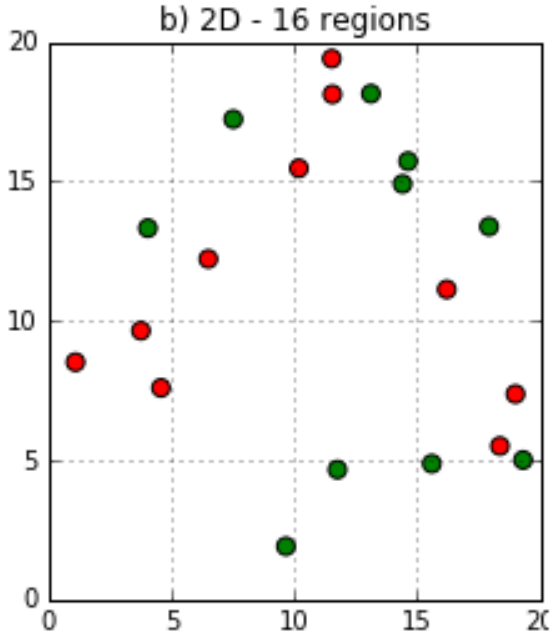


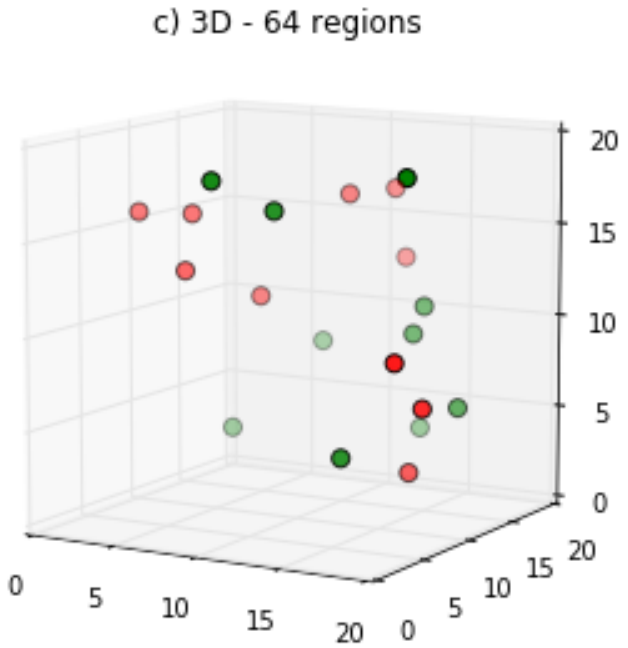
Curse of dim - Running complexity

- Many data points (labeled measurements) are needed
- Complexity (running time) increase with dimension ***d***
- A lot of methods have at least $O(n*d^2)$ complexity, where ***n*** is the number of samples
- As ***d*** becomes large, this complexity becomes very costly.
  - Compute = $

Sparisty increase: More regions with the same number of data points

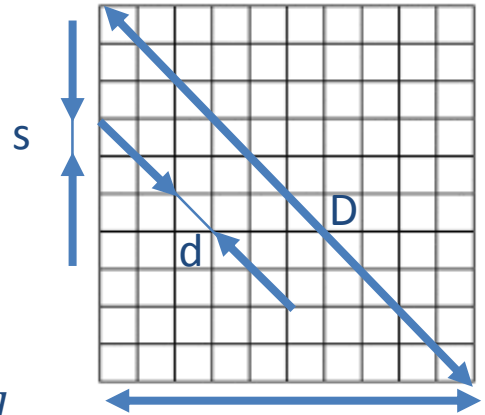






Distances in high dimension

- Assume, we have a unit side (2D) square, what we divided to 100 equal small squares
  - Calculate the ratio of the largest distance in a small square and the largest distance of the big square (in 2D)



$S_2=1$

$s_2=\sqrt[2]{\frac{1}{100}}=0.1$

$D_2=\sqrt{2}$

$d_2=0.1\sqrt{2}$

$$R_2=\frac{d_2}{D_2}=0.1$$

- Assume, we have a unit side 100D cube, what we divided to 100 equal small 100D cubes
  - Calculate the ratio Ratio of the largest distance in a small cube and the largest distance of the big cube (in 100D)
  - The average nearest neighbor distance is 95% of the largest distance!!!
  - Euclidian distance becomes meaningless, most two points are “far” from each others

$S_{100}=1$ $\quad s_{100}=\sqrt[100]{\frac{1}{100}}=0.95$

$D_{100}=\sqrt{100}=10$ $\quad d_{100}=\sqrt{100*0.95^2}=9.5$

$$R_{100}=\frac{d_{100}}{D_{100}}=0.95$$

Curse of dim - Some mathematical (weird) effects

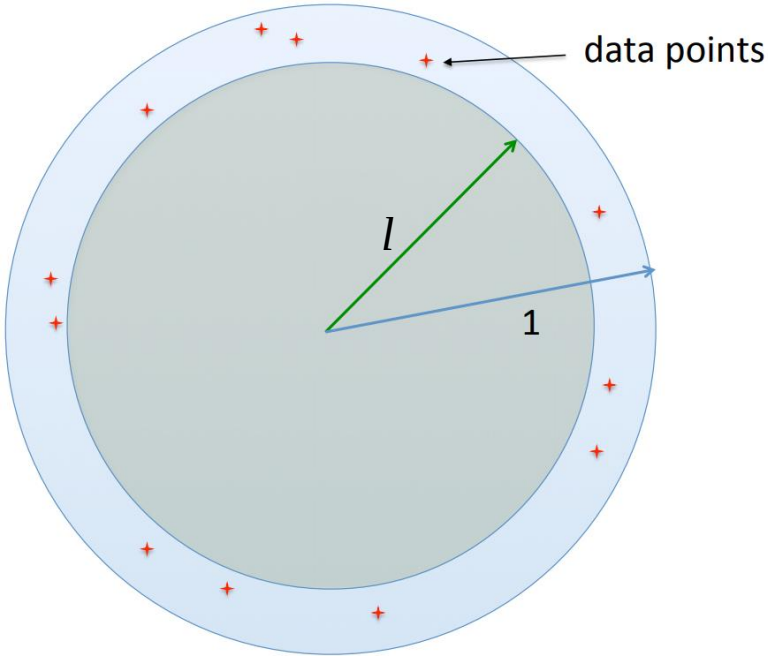
- Ratio between the volume of a sphere and a cube for **d**=3: $\frac{(\frac{4}{3})\pi r^3}{(2r)^3} \approx \frac{4r^3}{8r^3} \approx 0.5$
- When **d** tends to infinity the volume of the sphere (this ratio) tends to zero

d	3	5	10	20	30	50
ratio	0.52	0.16	0.0025	2.5E-08	2.0E-14	1.5E-28

- Most of the data is in the corner of the cube
  - Thus, Euclidian distance becomes meaningless, most two points are “far” from each others
- Very problematic for methods such as k-NN classification or k-means clustering because most of the neighbors are equidistant

The nearest neighbor problem in a sphere

- Assume randomly distributed points in a sphere with a unit diameter
- The median of the nearest neighbors is l
- As dimension tends to infinity
  - The median of the nearest neighbors converges to 1



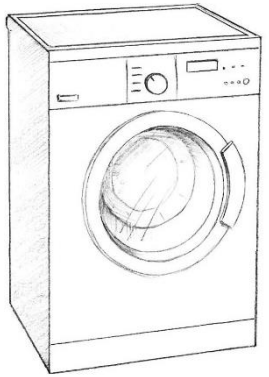
"The Curse of Dimensionality" by Raúl Rojas
https://www.inf.fu-berlin.de/inst/ag-ki/rojas_home/documents/tutorials/dimensionality.pdf

How to calculate dimensionality?

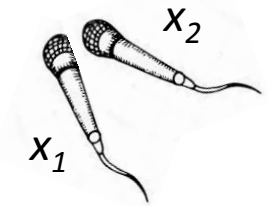
observations (d)

feature vectors (x)

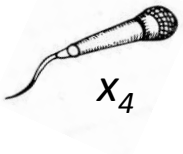
	x_1	x_2	x_3	x_4
d_1	1	2	1	1
d_2	2	4	3.5	1
d_3	3	6	17	1



- How many dimensions does the data intrinsically have here?
  (How many independent coordinates?)

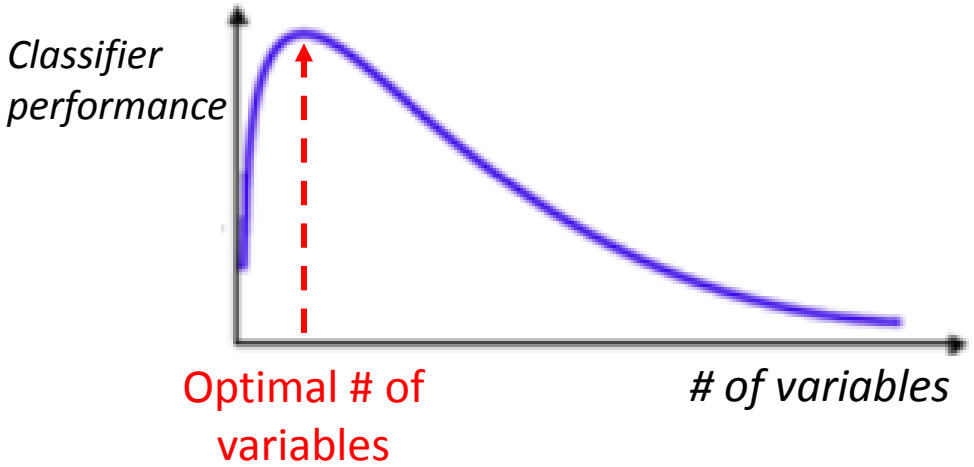


  - Two!
    - x1 = ½ * x2 (no additional information, correlated, not independent)
    - x4 is constant (carries no information at all!)



How to avoid the curse?

- Reduce dimensions
  - **Feature selection** - Choose only a subset of features
  - Use algorithms that transform the data into a lower dimensional space (example – PCA, t-SNE)
    **Both methods often result in information loss*
- Less is More
  - In many cases the information that is lost by discarding variables is made up for by a more accurate mapping/sampling in the lower-dimensional space



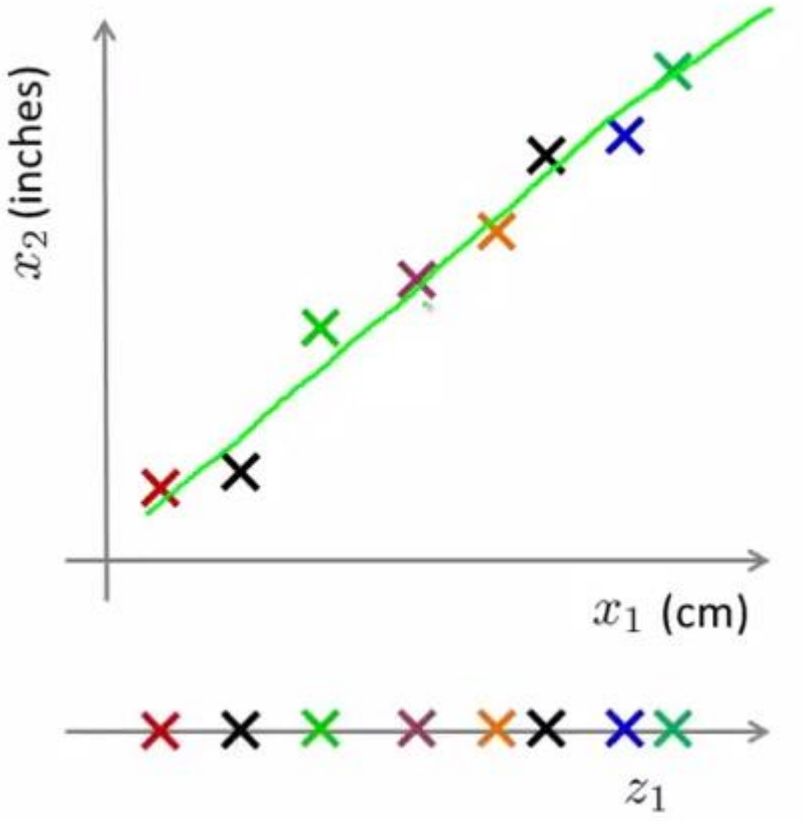
Principal component analysis

(PCA)

Dimensionality reduction goals

- Improve ML performance
- Compress data
- Visualize data (you can't visualize >3 dimensions)
- Generate new complex features
  - Loosing the meaning of a feature
  - Combining temperature, sound and current to one feature will be meaningless for human (non-physical)

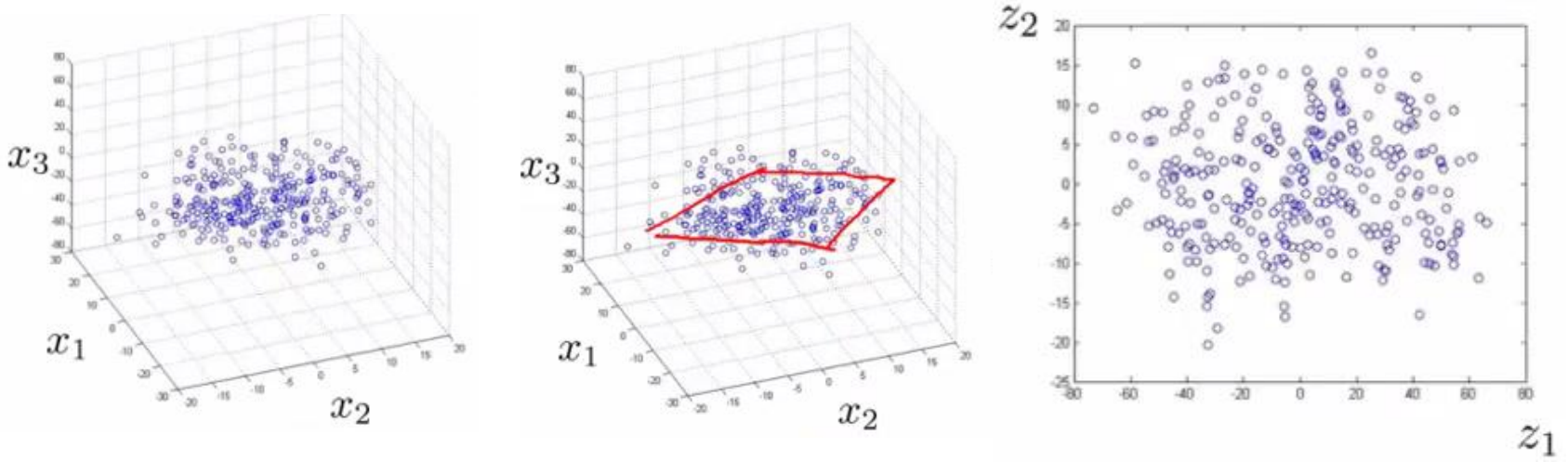
Example – reducing data from 2d to 1d



- X1 and x2 are pretty redundant. We can reduce them to 1d along the green line
- This is done by projecting the points to the line (some information is lost, but not much)

Example – 3D to 2D

- Despite having 3D data most of it lies close to a plane



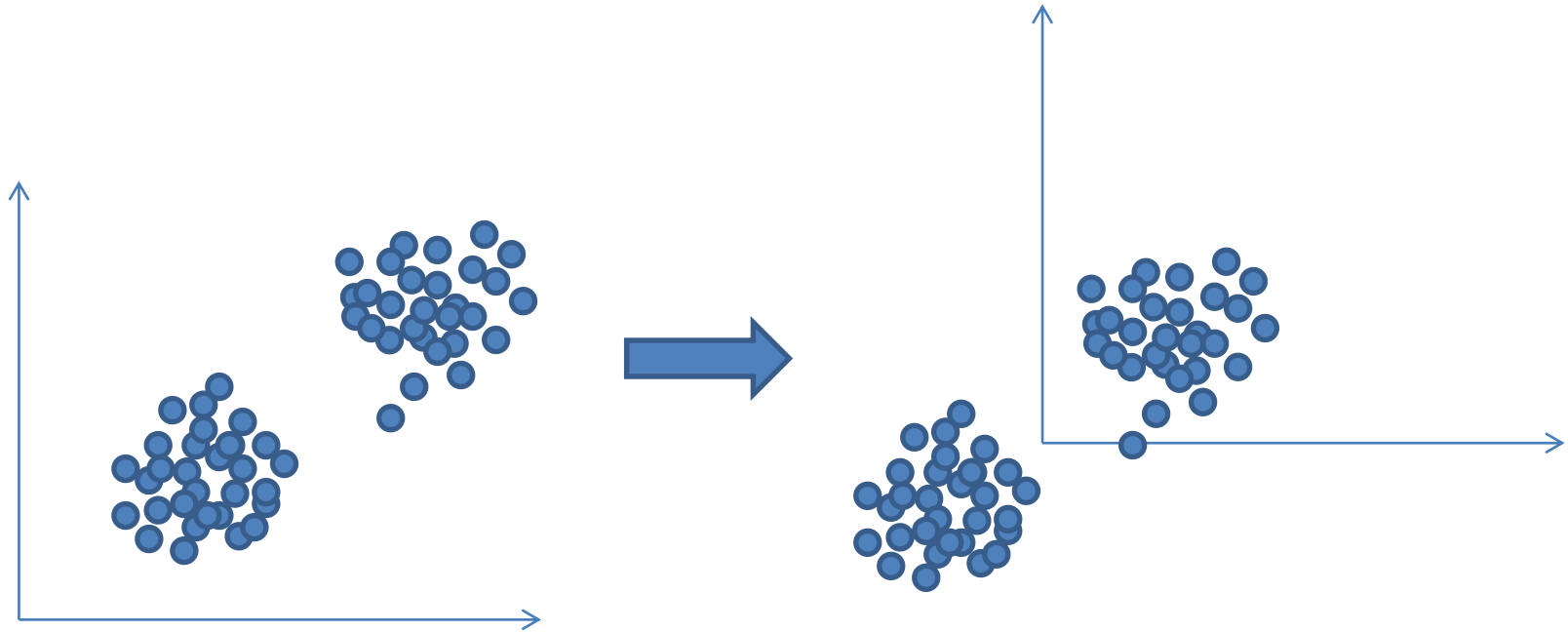
- If we were to project the data onto a plane we would have a more compact representation
- So how do we find that plane without loosing too much of the **variance** in our data? → PCA

Principal component analysis (PCA)

- Technique for dimensionality reduction
- Invented by Karl Pearson (1901)
- Linear coordinate transformation
  - converts a set of observations of possibly correlated variables
  - into a set of values of linearly uncorrelated orthogonal variables called principal components
- Deterministic algorithm

PCA algorithm

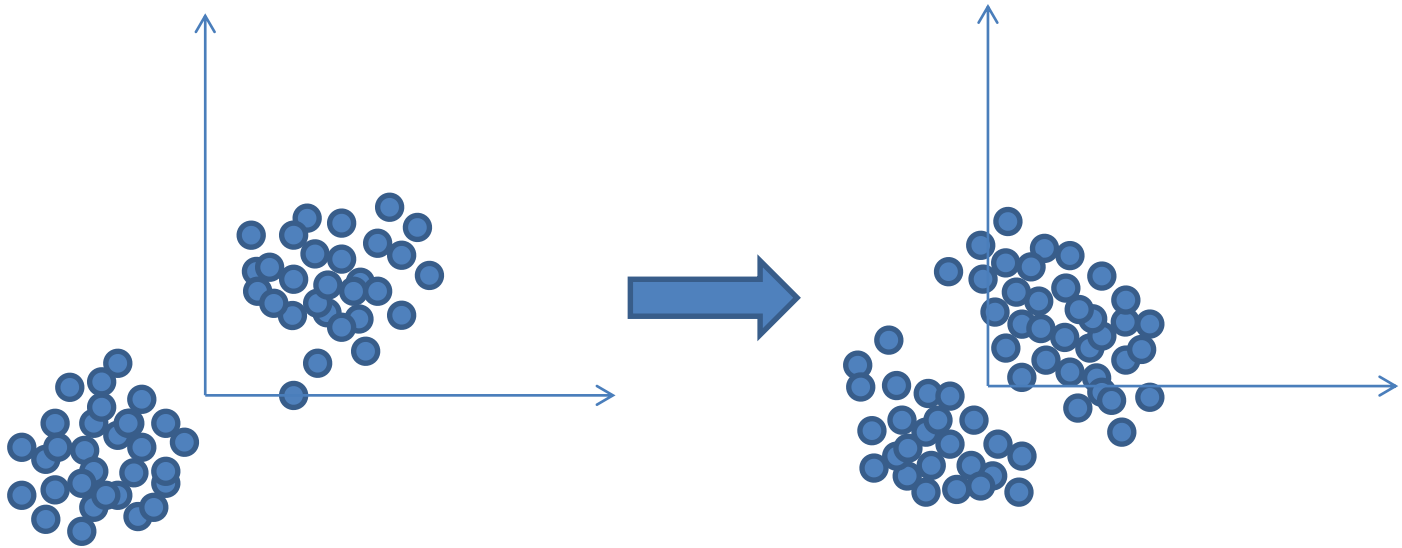
1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.



https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

PCA algorithm

1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.
2. Standardization (optional): Do it, if you want to have each of your features the same variance.



https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

PCA algorithm

1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.
2. Standardization (optional): Do it, if you want to have each of your features the same variance.
3. Covariance matrix: Calculate the covariance matrix

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

Covariance (formal definition)

- Assume that $\mathbf{x}$ are random variable vectors
- We have *n* vectors

$$\text{Variance(x)} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})$$

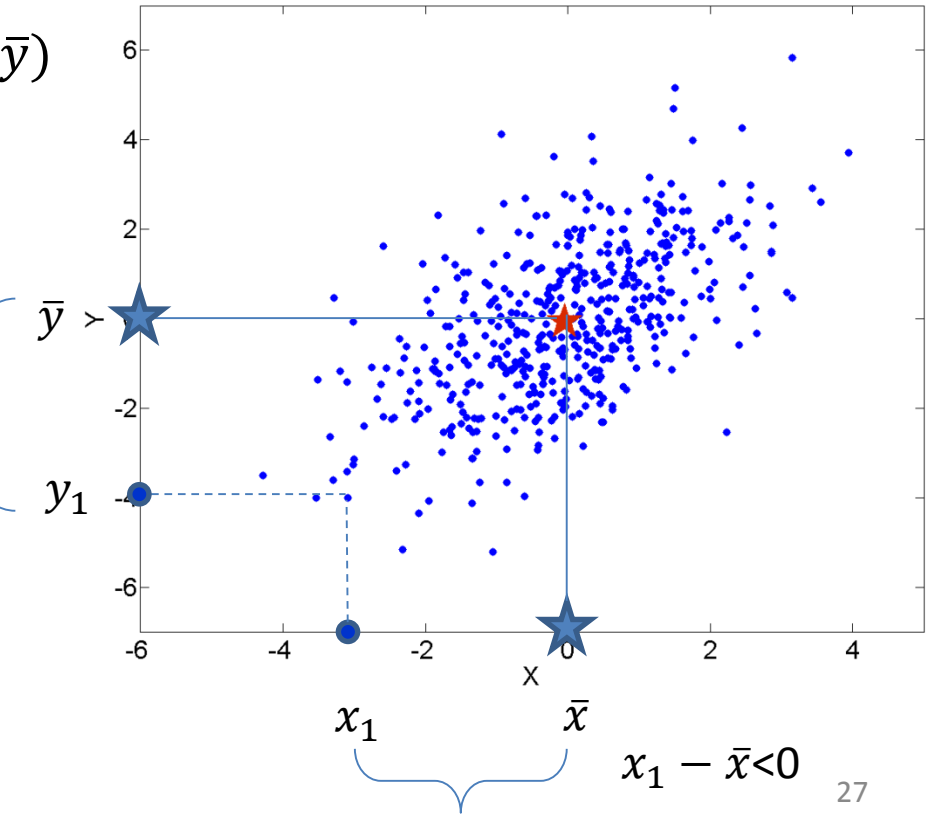
$$\text{Covariance}(\text{x}, \text{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- $\text{Covariance}(\text{x}, \text{x}) = \text{var}(\text{x})$
- $\text{Covariance}(\text{x}, y) = \text{Covariance}(\text{y}, \text{x})$

Covariance example for 2D

$$\text{Covariance}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- Positive covariance between the two dimensions

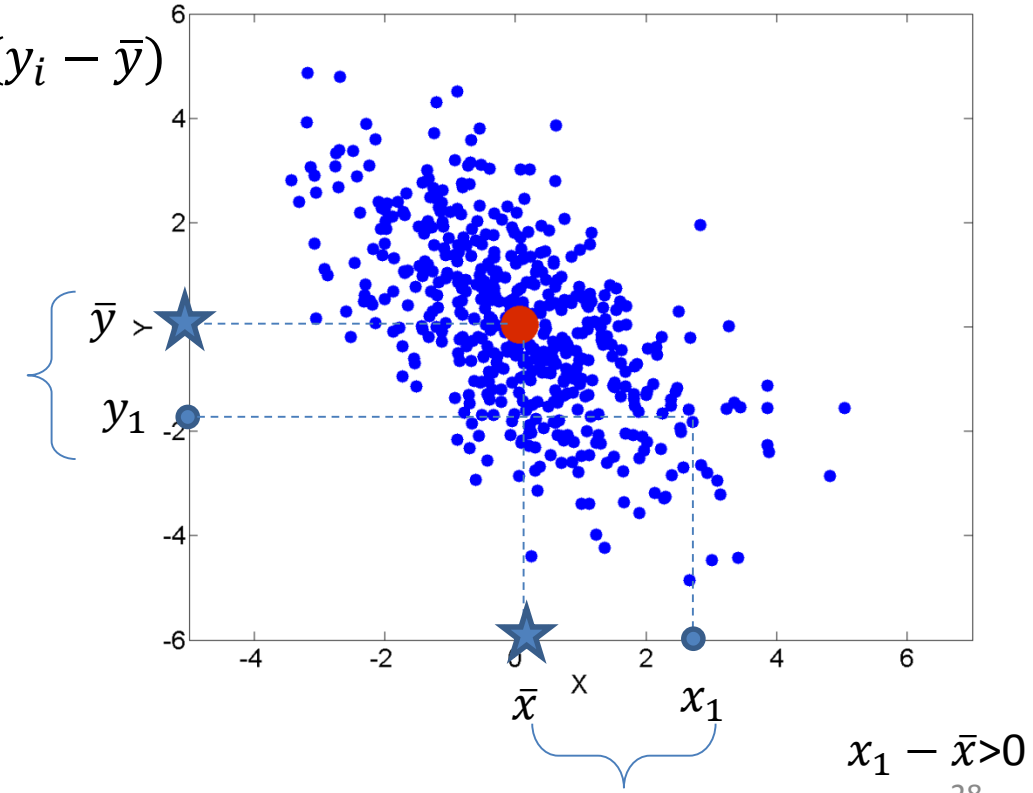


Covariance example for 2D

$$\text{Covariance}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- Negative covariance between the two dimensions

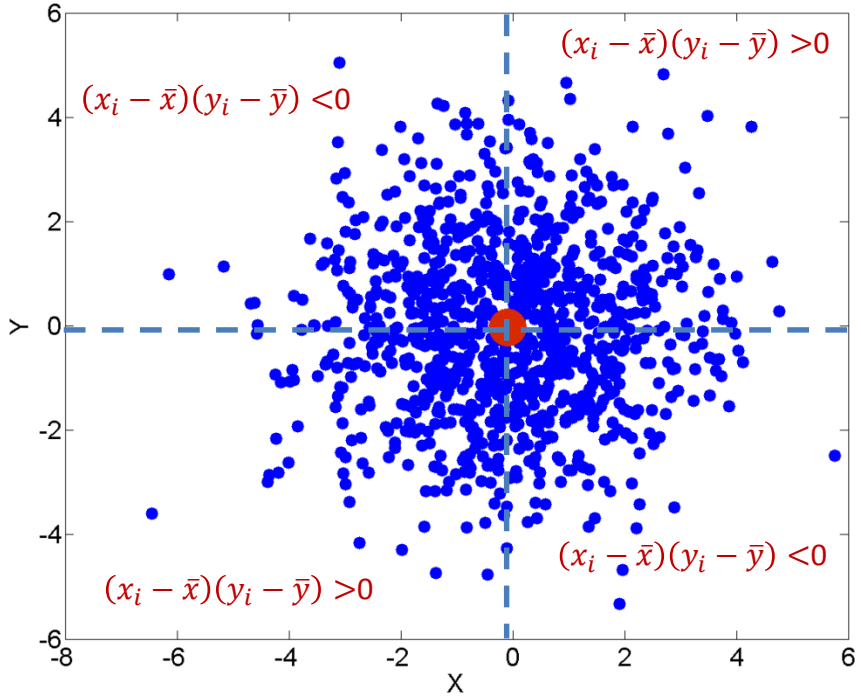
$y_1 - \bar{y}<0$



Covariance example for 2D

$$\text{Covariance}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

- No covariance between the two dimensions



Covariance matrix

- Diagonal elements are variances, i.e. $\mathrm{Cov}(x,x)=var(x)$
  - n is the number of the vectors
  - m is the dimension

$$Cov\,(\sum) = \begin{bmatrix} cov(x_1,x_1) & cov(x_1,x_2) & \cdots & cov(x_1,x_m) \\ cov(x_2,x_1) & cov(x_2,x_2) & \cdots & cov(x_2,x_m) \\ \vdots & \vdots & \vdots & \vdots \\ cov(x_m,x_1) & cov(x_m,x_2) & \cdots & cov(x_m,x_m) \end{bmatrix}$$

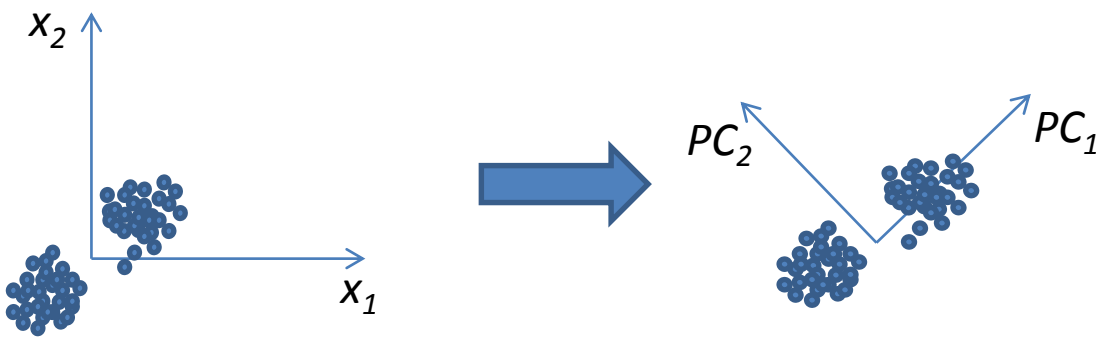
$$Cov\,(\sum) = \frac{1}{n}(X-\bar{X})(X-\bar{X})^T; where\ X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

- Covariance Matrix is symmetric
  - commutative

$$Cov\,(\sum) = \begin{bmatrix} var(x_1,x_1) & cov(x_1,x_2) & \cdots & cov(x_1,x_m) \\ cov(x_2,x_1) & var(x_2,x_2) & \cdots & cov(x_2,x_m) \\ \vdots & \vdots & \vdots & \vdots \\ cov(x_m,x_1) & cov(x_m,x_2) & \cdots & var(x_m,x_m) \end{bmatrix}$$

PCA algorithm

1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.
2. Standardization (optional): Do it, if you want to have each of your features the same variance.
3. Covariance matrix: Calculate the covariance matrix
4. Eigenvectors and eigenvalues of the covariance matrix
   - Note: Each new axis (PC) is an eigenvector of the data. The standard deviation of the data variance on the new axis is the eigenvalue for that eigenvector.



Principal components will be orthogonal. Uncorrelated, independent!

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

PCA algorithm

1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.
2. Standardization (optional): Do it, if you want to have each of your features the same variance.
3. Covariance matrix: Calculate the covariance matrix
4. Eigenvectors and eigenvalues of the covariance matrix
   - Note: Each new axis (PC) is an eigenvector of the data. The standard deviation of the data variance on the new axis is the eigenvalue for that eigenvector.
5. Rank eigenvectors by eigenvalues
6. Keep top k eigenvectors and stack them to form a feature vector
7. Transform data to PCs:
   - New data = feature vectors (transposed) * original data

https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c

From k original variables: $x_1,x_2,\ldots,x_k$:

Produce k new variables: $y_1,y_2,\ldots,y_k$:

$y_1 = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1k}x_k$

$y_2 = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2k}x_k$

...

$y_k = a_{k1}x_1 + a_{k2}x_2 + \ldots + a_{kk}x_k$

y_k's are **Principal Components**

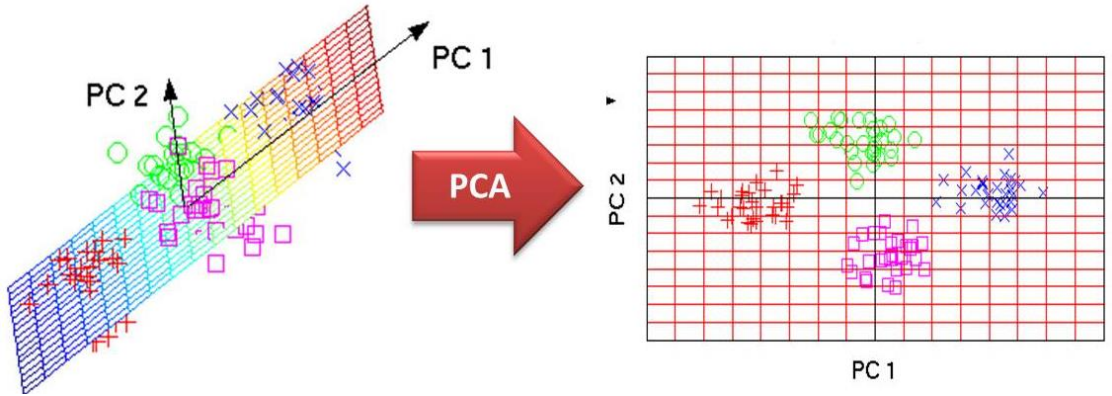
$\{a_{11},a_{12},\ldots,a_{1k}\}$ is 1st **Eigenvector** of of first principal component

$\{a_{21},a_{22},\ldots,a_{2k}\}$ is 2nd **Eigenvector** of of 2nd principal component

$\{a_{k1},a_{k2},\ldots,a_{kk}\}$ is kth **Eigenvector** of of kth principal component

Principal Component Analysis (PCA)

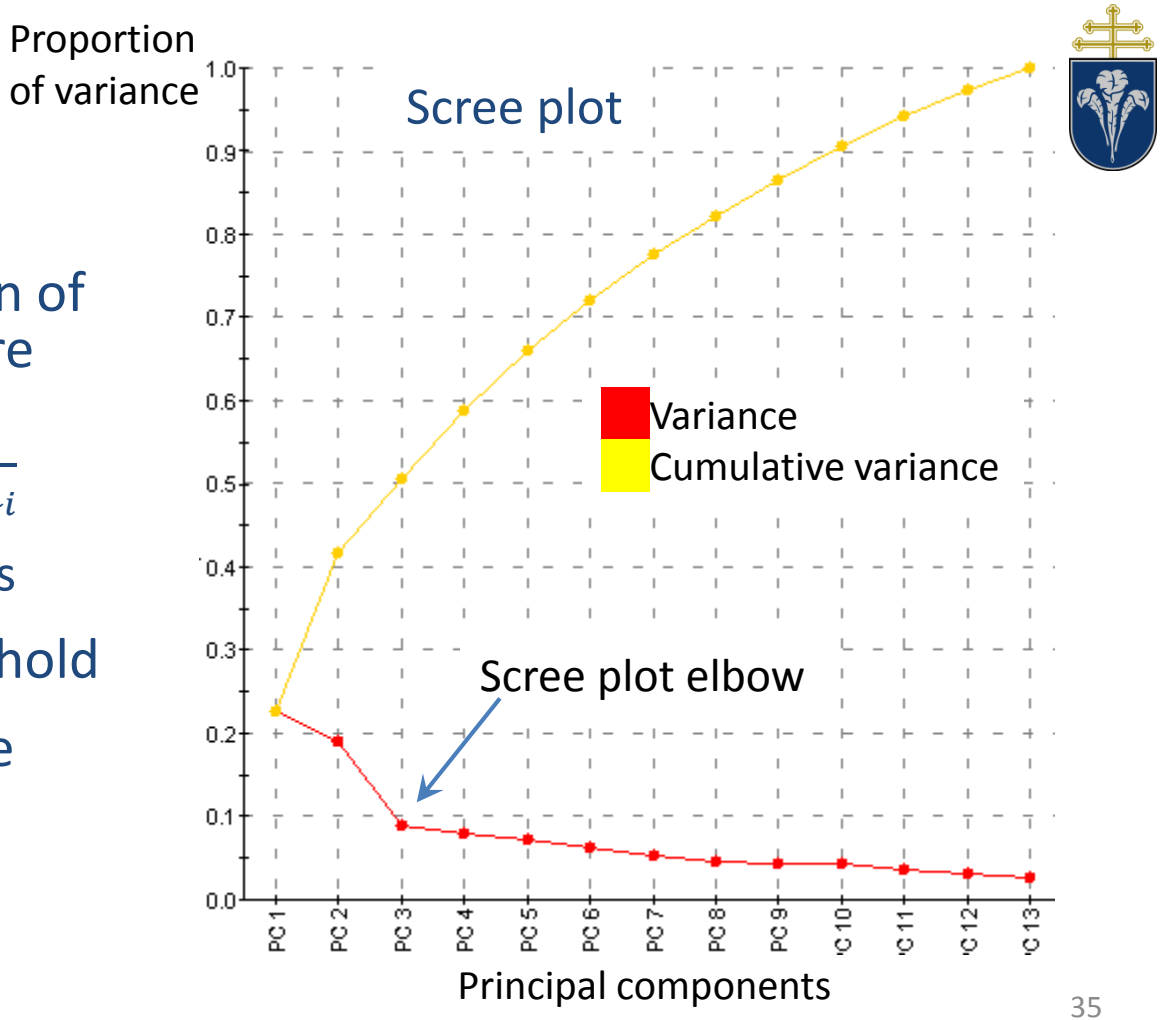
- The idea is to project the data onto a subspace which compresses most of the variance in as little dimensions as possible.
- Each new dimension is a **principle component**
- The principle components **are ordered** according to how much **variance in the data** they capture
  - Example:
    - PC1 – 55% of variance
    - PC2 – 22% of variance
    - PC3 – 10% of variance
    - PC4 – 7% of variance
    - PC5 – 2% of variance
    - PC6 – 1% of variance
    - PC7 - ….



We have to choose how many PCs to use from the top

How many PCs to use?

- Calculate the proportion of variance for each feature
  - $prop.of\ var. = \frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}$
  - λ_i are the eigen values
- Rich a predefined threshold
- Or find the elbow of the Scree plot



PCA Example

- Weekly food consumption of the four countries
  - food types: variables
  - countries: observations
- Clustering the countries:
  - Needs visualization in 17 dimension
- PCA: reduce dimensionality

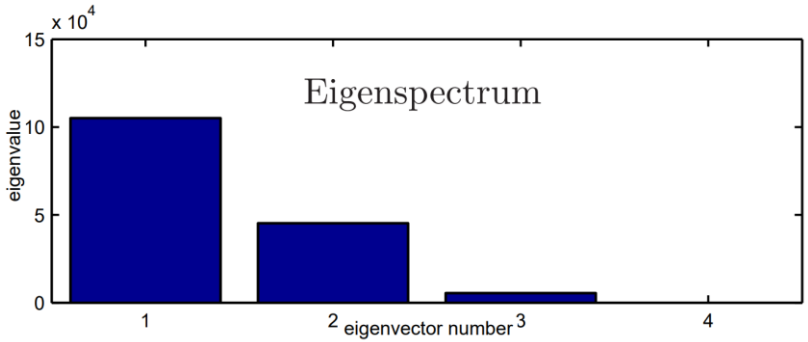
http://www.sdss.jhu.edu/~szalay/class/2016-oldold/SignalProcPCA.pdf

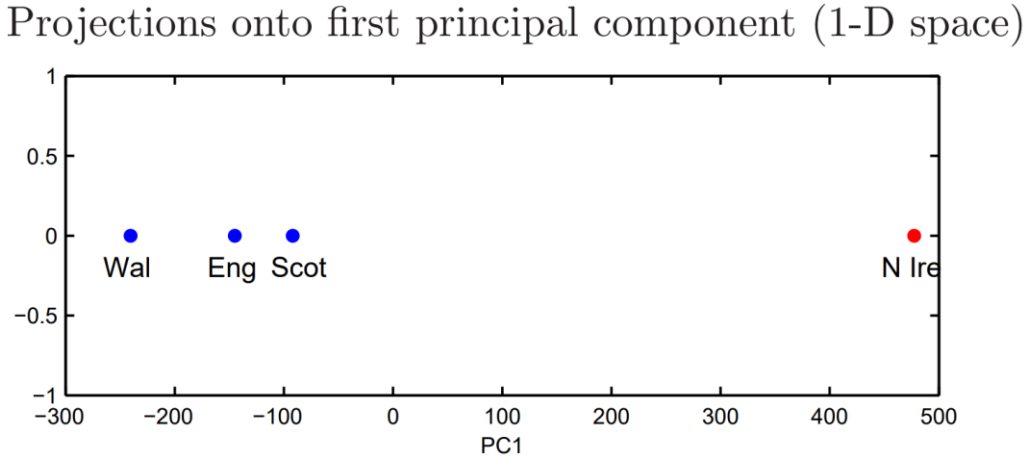
	England	Wales	Scotland	N Ireland
Cheese	105	103	103	66
Carcass meat	245	227	242	267
Other meat	685	803	750	586
Fish	147	160	122	93
Fats and oils	193	235	184	209
Sugars	156	175	147	139
Fresh potatoes	720	874	566	1033
Fresh Veg	253	265	171	143
Other Veg	488	570	418	355
Processed potatoes	198	203	220	187
Processed Veg	360	365	337	334
Fresh fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft drinks	1374	1256	1572	1506
Alcoholic drinks	375	475	458	135
Confectionery	54	64	62	41

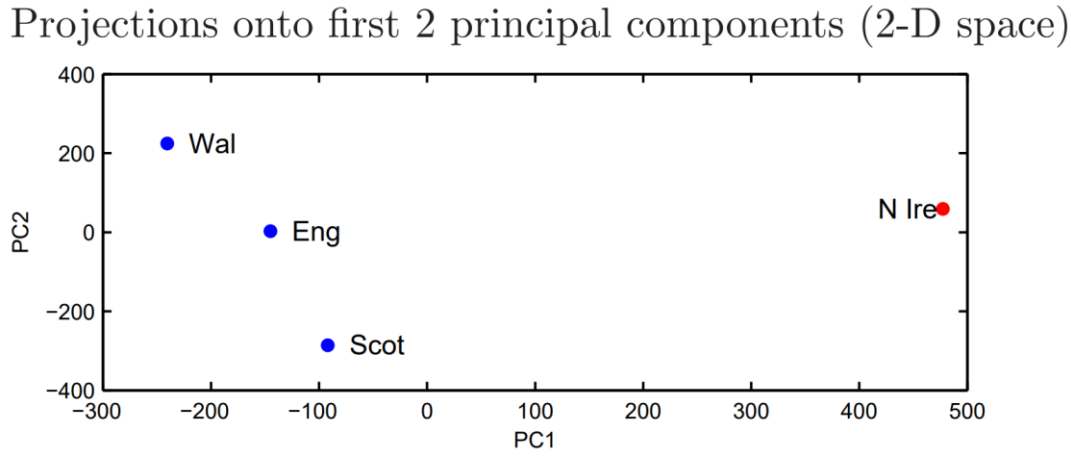
UK food consumption in 1997 (g/person/week). Source: DEFRA

PCA Example

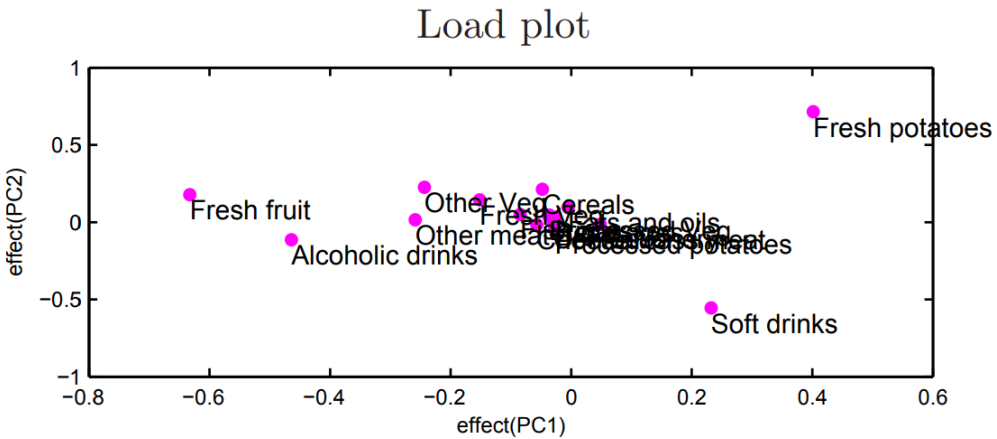
- From PC1, two clusters are well separable
- Including PC2, the four clusters can be well separated

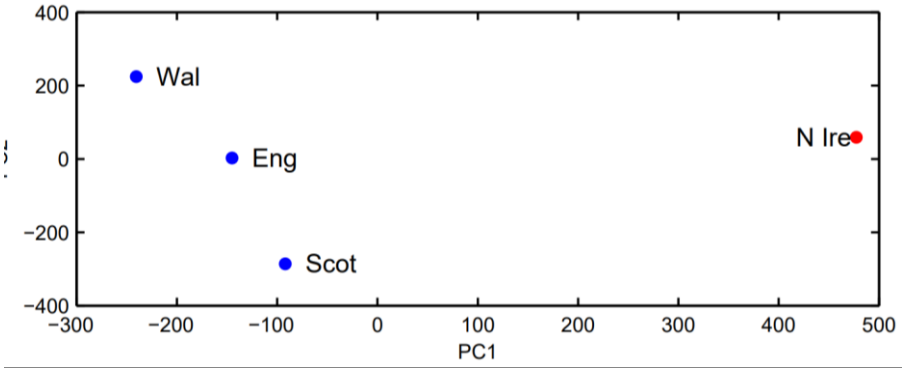






Coefficients of the Principal Components





Load plot shows the coefficients of the original feature vectors to the principal components

t-Distributed Stochastic Neighbor Embedding

(t-SNE)

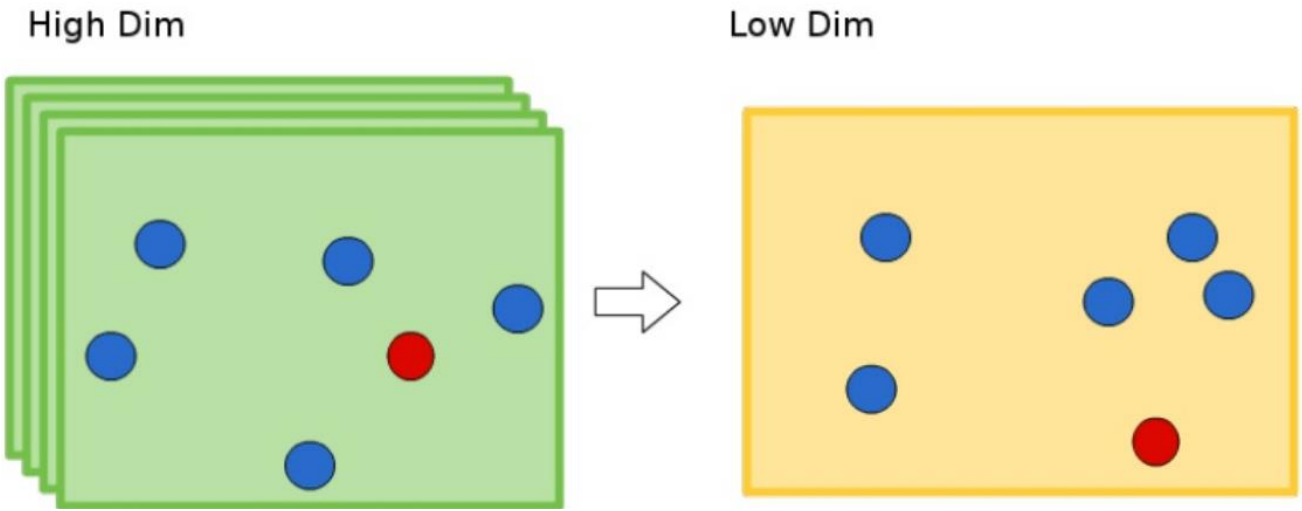
t-Distributed Stochastic Neighbor Embedding (t-SNE)

- Introduced by Laurens Van Der Maaten (2008)
- Generates a low dimensional representation of the high dimensional data set iteratively
- Aims to minimize the divergence between two distributions
  - Pairwise similarity of the points in the higher-dimensional space
  - Pairwise similarity of the points in the lower-dimensional space
- Output: original points mapped to a 2D or a 3D data space
  - similar objects are modeled by nearby points and
  - dissimilar objects are modeled by distant points with high probability
- Unlike PCA, it is stochastic (probabilistic)

t-SNE implementation I

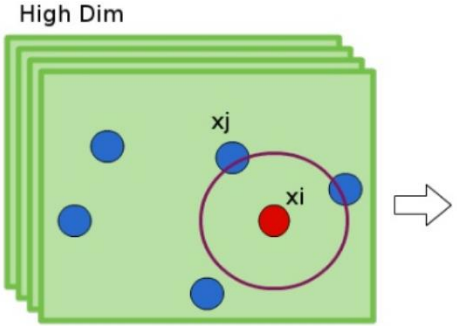
Step 1: *Generate the points in the low dimensional data set (2D or 3D)*

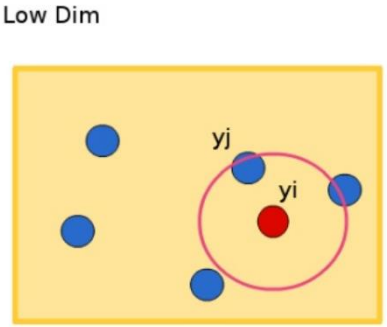
- random initialization
- First two or three components of PCA



t-SNE implementation II

Step 2: *Calculate the pair-wise similarities measures between data pairs (probability measure)*





The similarity of datapoint x_j to datapoint x_i means the conditional probability p_{ji} that x_i would pick x_j as its nearest neighbor.

$$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_l - x_k||^2/2\sigma^2)}$$

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

Exponential normalization of the Euclidian distances are needed due to the high dimensionality. (Curse of dimensionality)

t-SNE implementation III

Step 3: *Define the cost function*

- Similarity of data points in High dimension: $$p_{ij} = \frac{exp(-||x_i - x_j||^2/2\sigma^2)}{\sum_{k \neq l} exp(-||x_l - x_k||^2/2\sigma^2)}$$
- Similarity of data points in Low dimension: $$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$
- Cost function (called Kullback-Leiber divergence between the two distributions): $$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$
  - Large p_{ji} modeled by small q_{ji} → Large penalty
  - Large p_{ji} modeled by large q_{ji} → Small penalty
  - Local similarities are preserved

t-SNE implementation IV

Step 4: *Minimize the cost function using gradient descent*

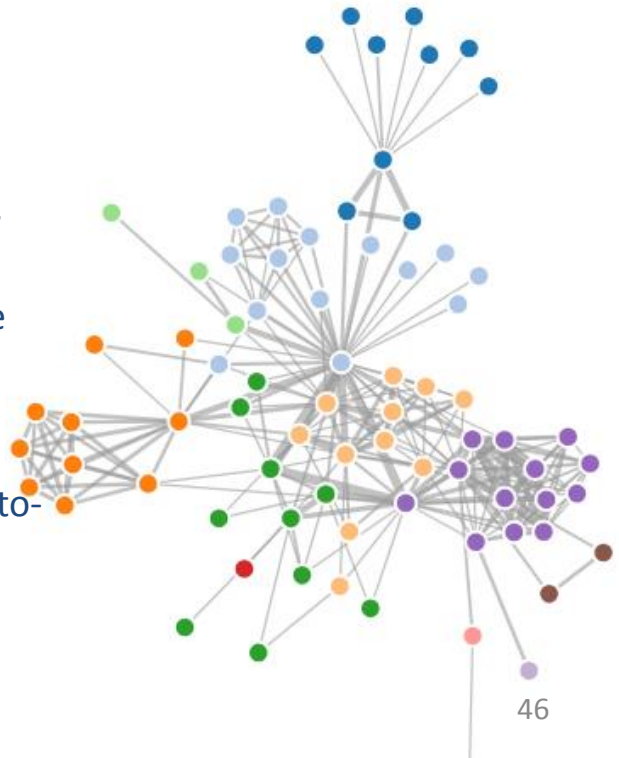
- Gradient has a surprisingly simple form:

$$\frac{\partial C}{\partial y_i} = 4 \sum_{j \neq i} (p_{ij} - q_{ij})(1 + ||y_i - y_j||^2)^{-1}(y_i - y_j)$$

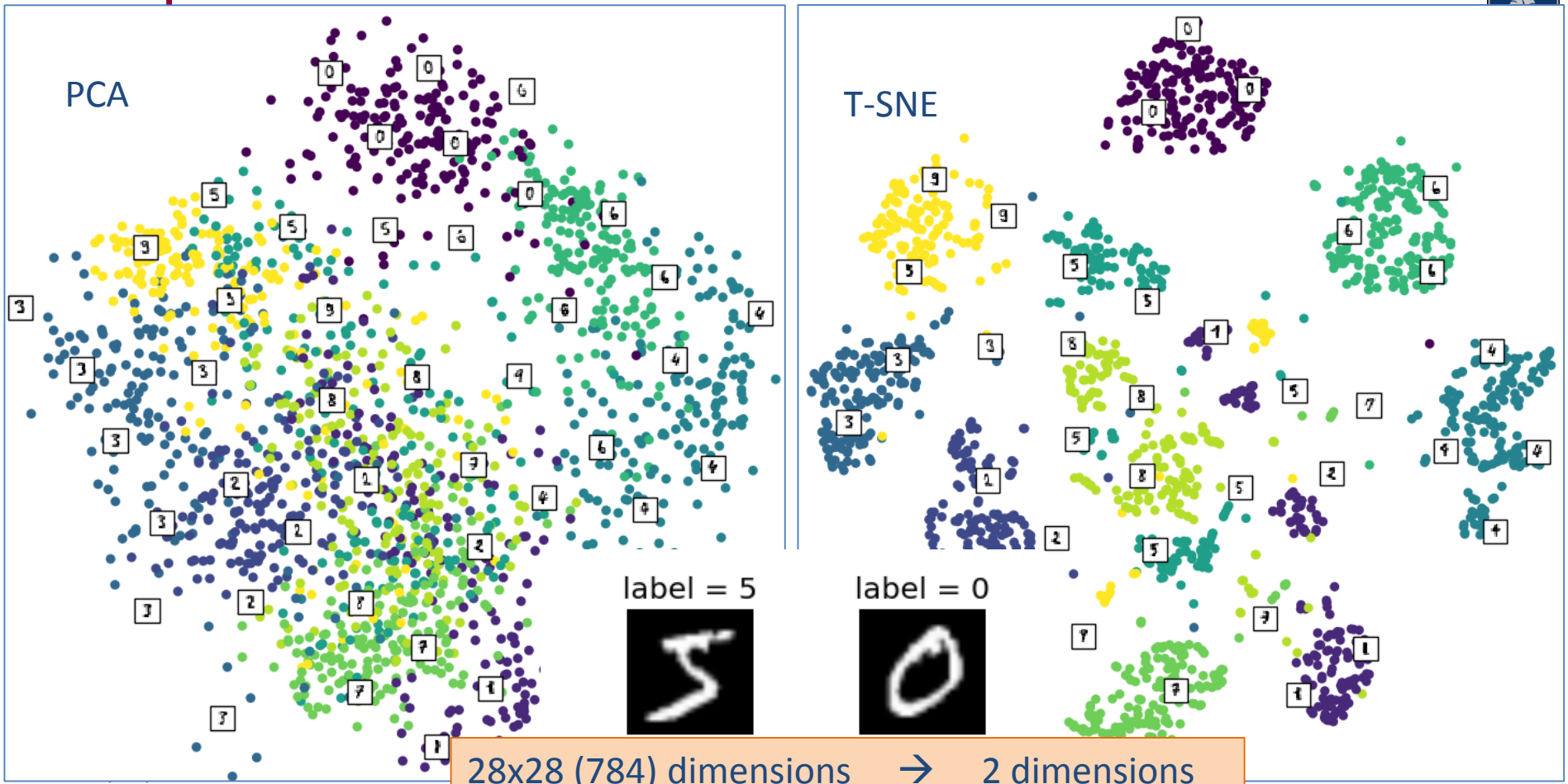
- Optimization can be done using momentum method

Physical analogy

- Our map points are all connected with springs in the low dimensional data map
- Stiffness of the springs depends on $p_{j|i} - q_{j|i}$
- Let the system evolve according to the laws of physics
  - If two map points are far apart while the data points are close, they are attracted together
  - If they are nearby while the data points are dissimilar, they are repelled.
- Illustration (live)
  - https://www.oreilly.com/learning/an-illustrated-introduction-to-the-t-sne-algorithm



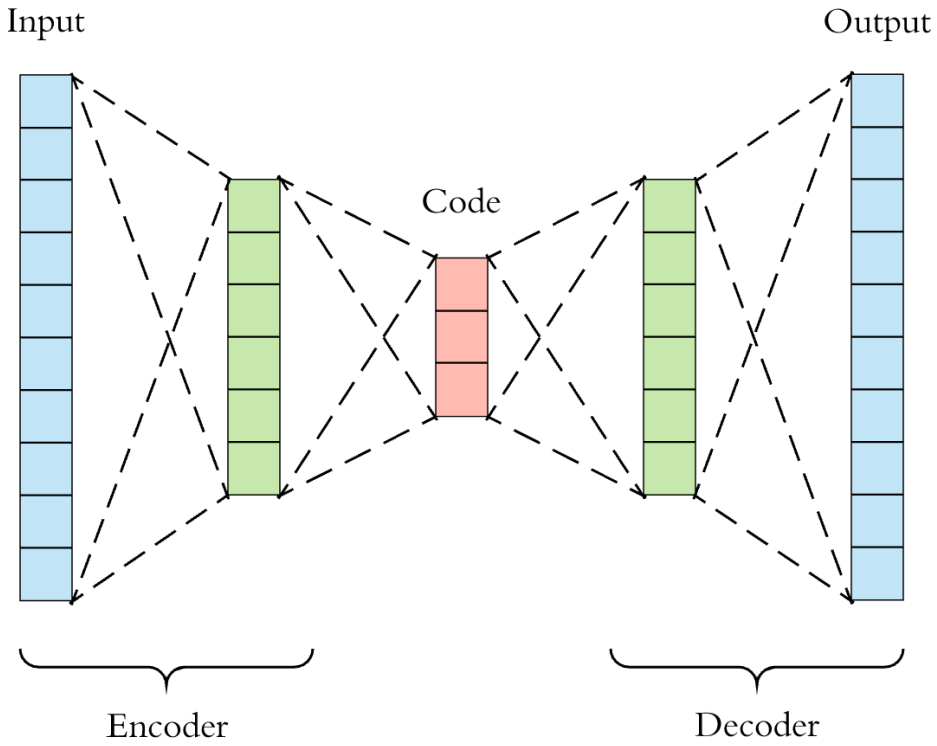
Comparison of PCA and t-SNE on MNIST database



Autoencoder

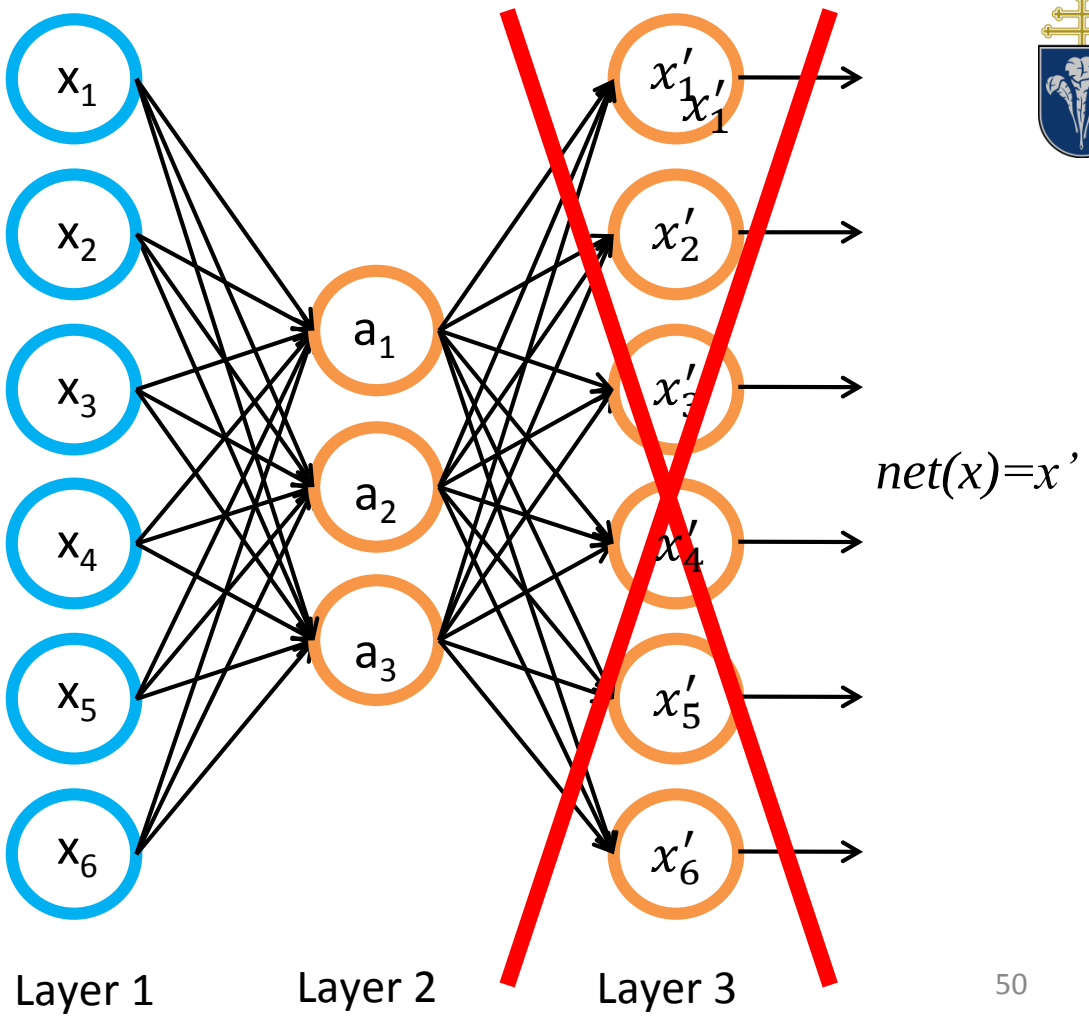
Autoencoder

- Neural network used for efficient data coding
- Uses the same vector for the input and the output
  - No labelled data set is needed
  - Unsupervised learning
- Two parts
  - Encoder: reduces data dimension
  - Decoder: reconstructs data
  - Middle layer: code



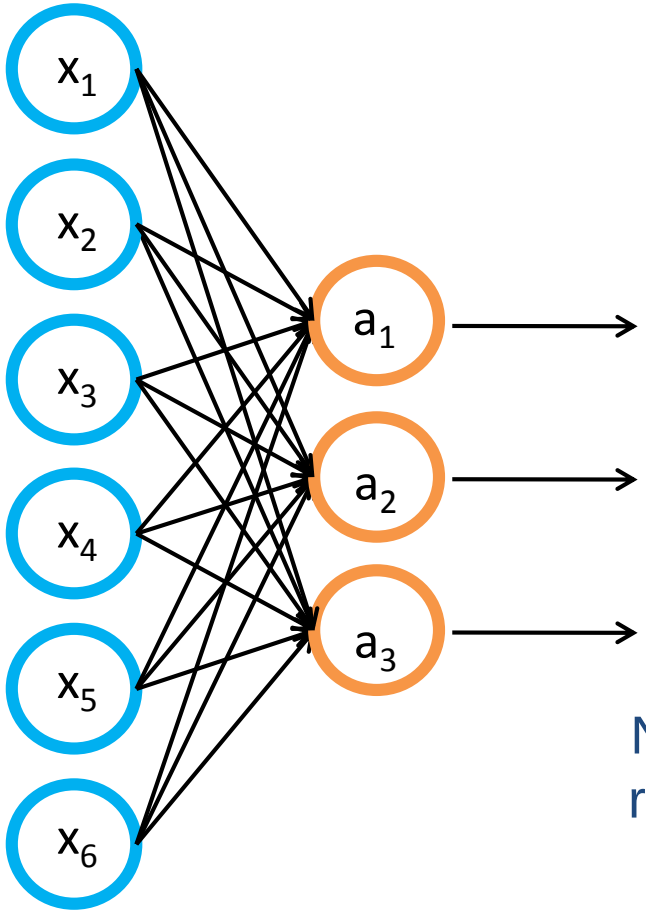
Operation

- The network is trained with the same input-output pairs
- Loss function:
  - MSE
  - Cross Entropy
- After network is trained, remove decoder part



Operation

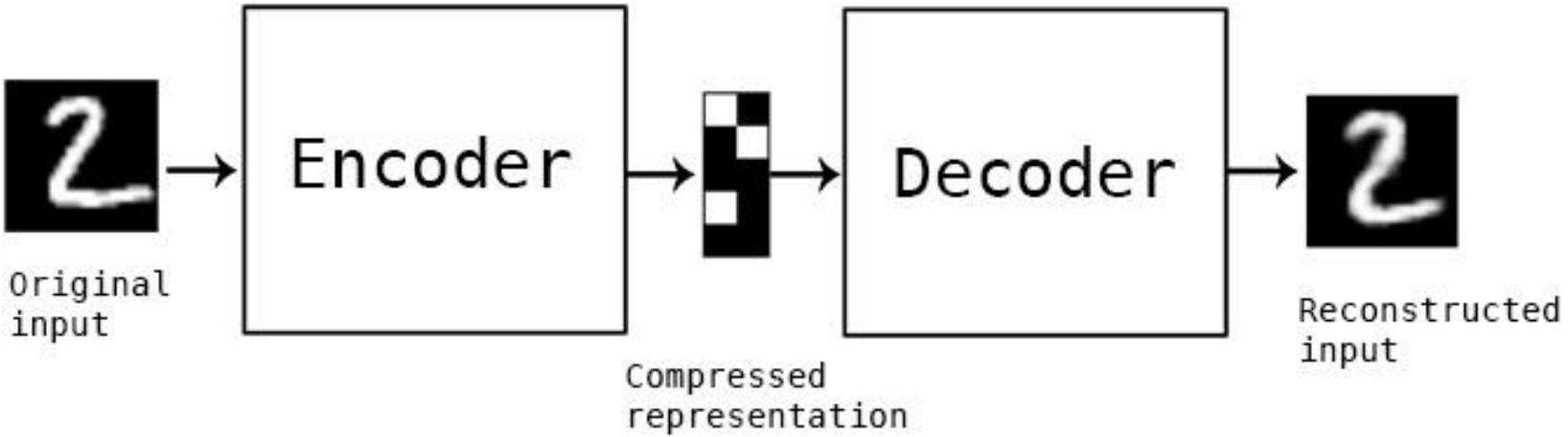
- The network is trained with the same input-output pairs
- Loss function:
  - MSE
  - Cross Entropy
- After network is trained, remove decoder part



$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

New compressed representation for input.

Example



- Coding MNIST data base
- 28x28 (784 dimensions) → 2x5 (10 dimensions)
- 78 times compression

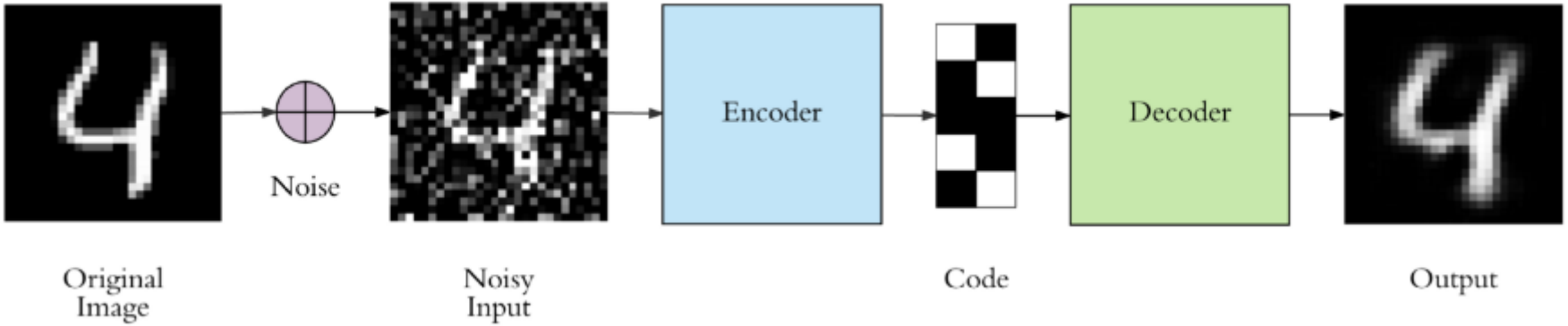
Autoencoder vs PCA

- Undercomplete autoencoder with
  - one hidden layer
  - linear output function
  - MSE loss
- Projects data on subspace of first K principal components

Undercomplete: width (dimension) of hidden layer is smaller than width input/output layer

Denoising

- Trick:
  - Adding noise to the input
  - The desired output is the original input

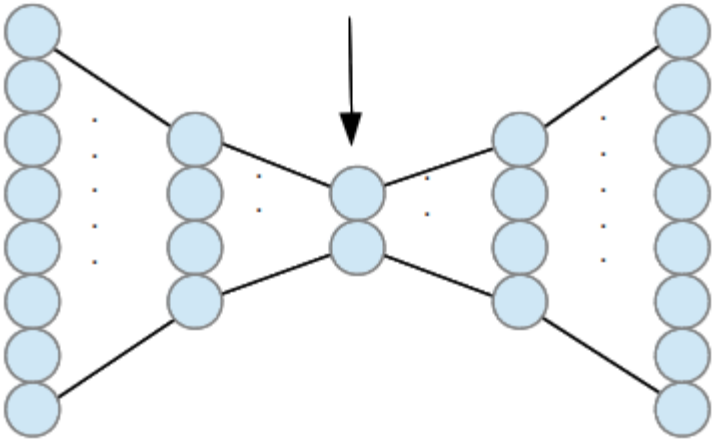


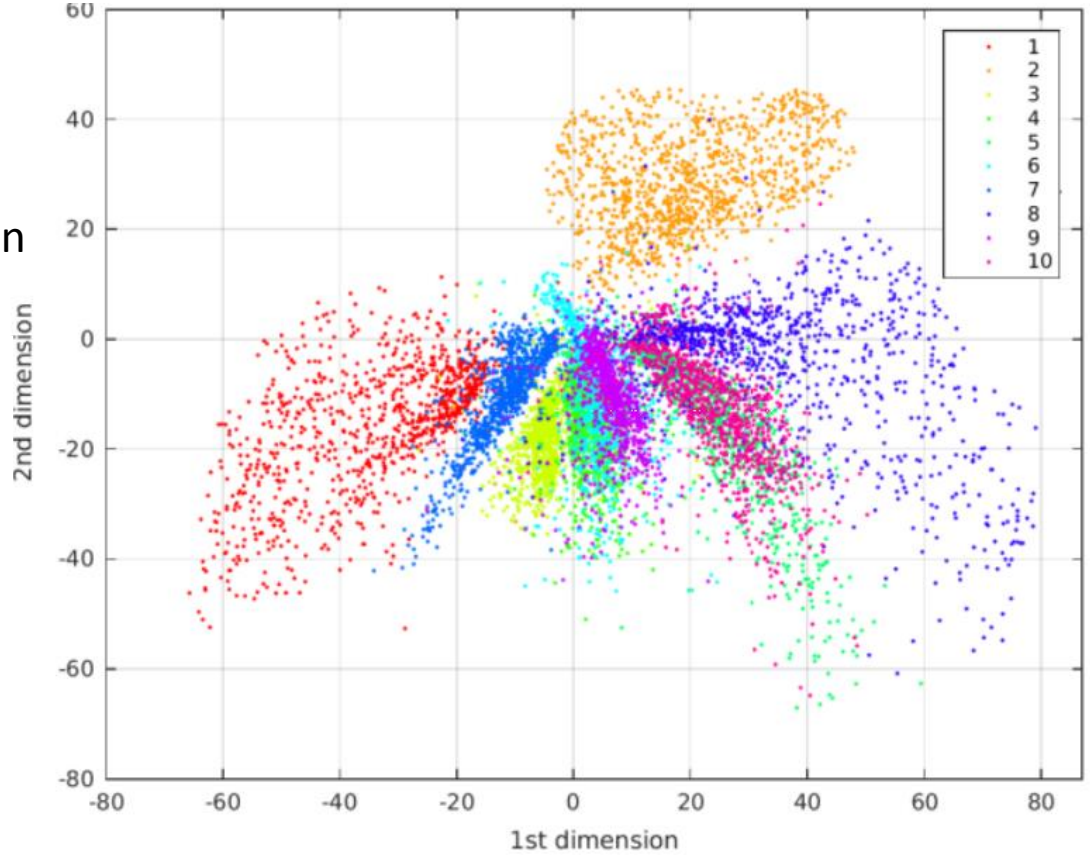
MNIST database coding to two dimension

label = 5

label = 0

Two neurons in the coding hidden layer



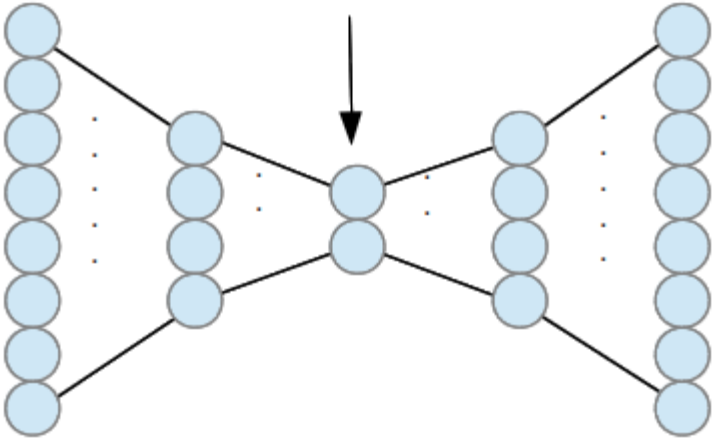


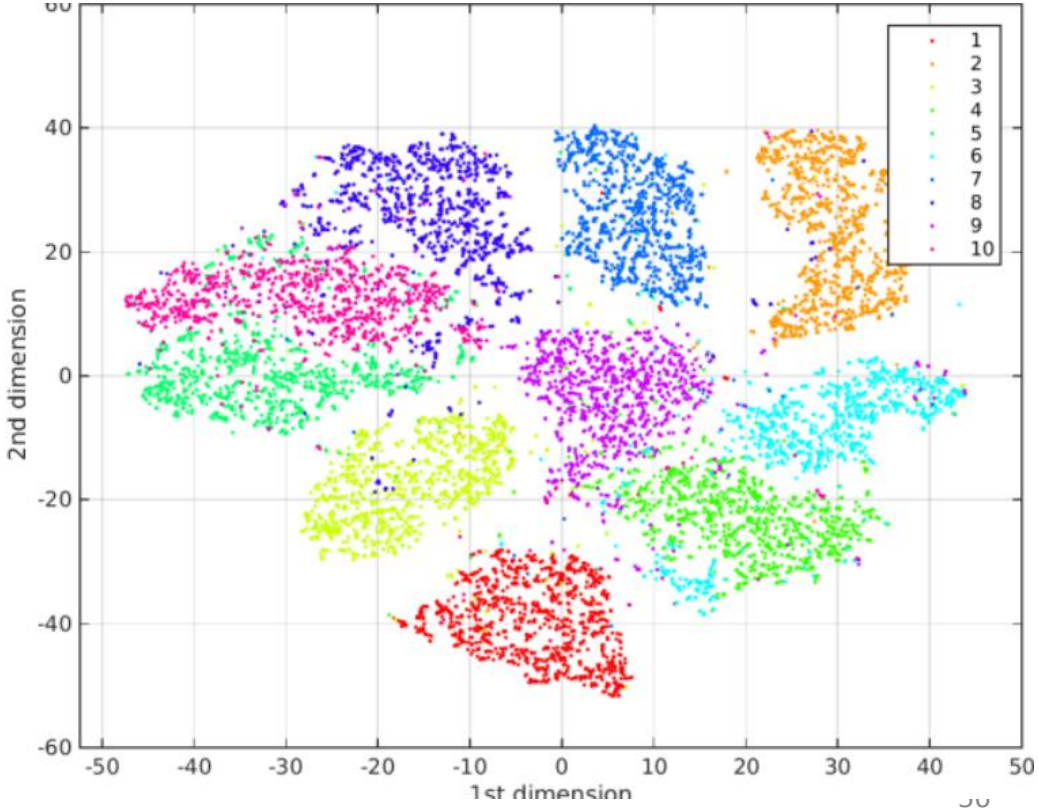
Autoencoder + t-SNE

label = 5

label = 0

Two neurons in the coding hidden layer





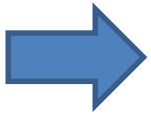
Recurrent Neural Networks

- How to handle sequential signals with Neural Networks?
- General Architecture of the Recurrent Networks

Static samples vs Data signal flow

AlexNet could recognize 1000s of images.
ResNet could reach better then human performance.

- Though human can recognize
  - Single letters
  - Single sounds
  - Single tunes
  - Single pictures



- But in real life we handle
  - Texts
  - Speech
  - Music
  - Movies

Story

(temporal analysis of sequential data)

Can feed-forward neural networks (perceptrons, conv. nets) solve these problems?

DATA MEMORY

Memory

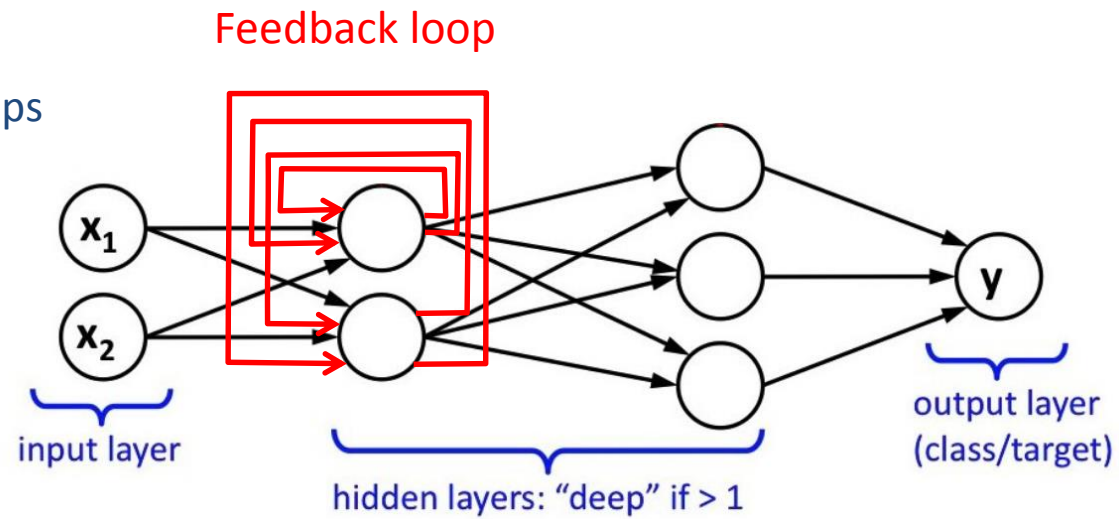
- Our feed-forward nets had so far
  - Program memory (for the weights)
  - Registers
    - For storing data temporally due to implementation and not matematical resasons
    - Registers were not part of the networks
- After each inferences the net was reset
  - All registers were deleted
  - No information remained in the net after processing an input vector
  - Therefore the order of a test sequence made no difference

Recurrent networks (RNN)

- Unlike traditional neural networks, the output of the RNN depends on the previous inputs
  - State
- RNN contains feedback
- Theoretically:
  - Directed graph with cyclic loops
- From now, time has a role in execution
  - Time steps, delays

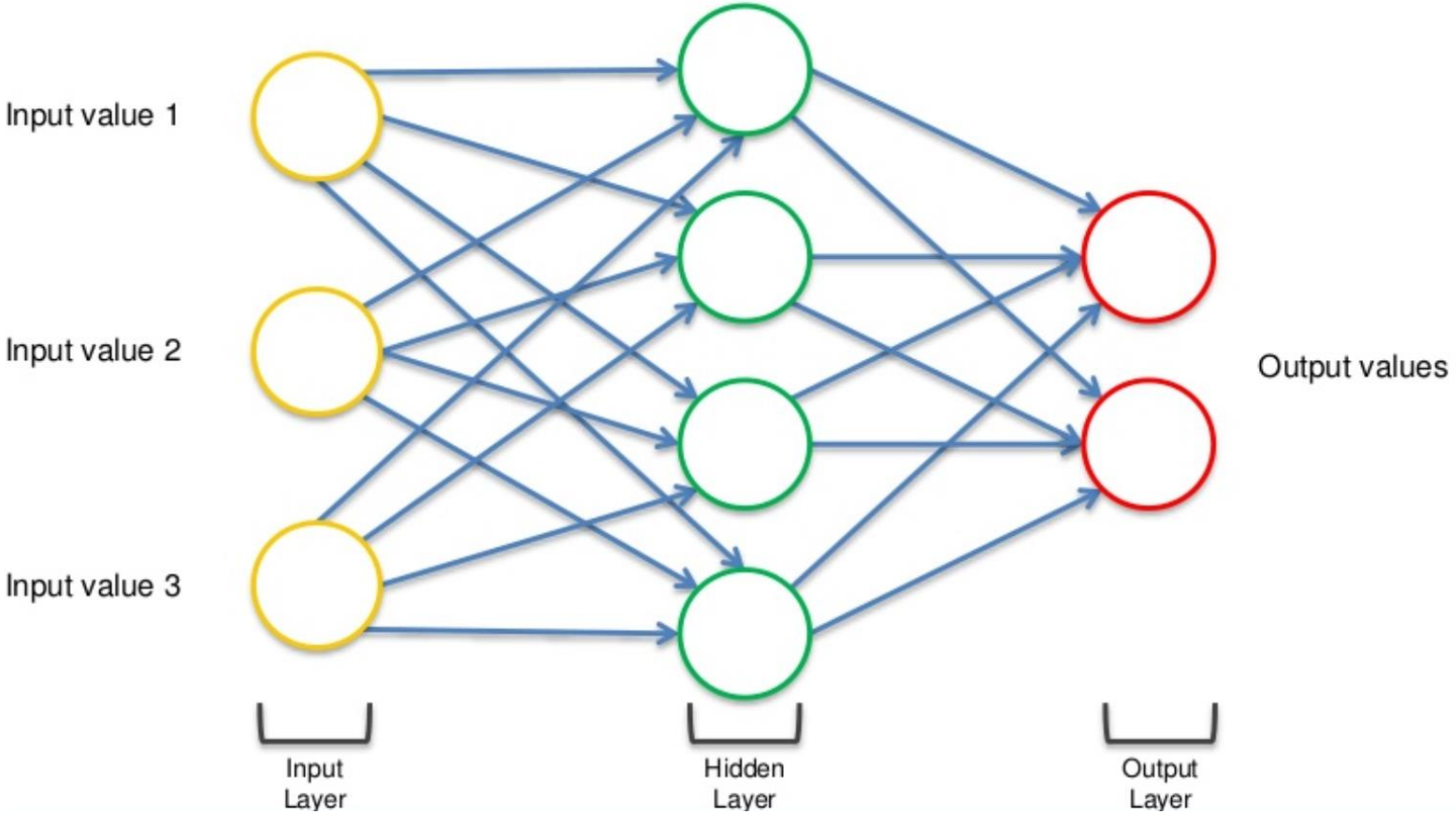
Jürgen lives in Berlin.

He speeks …….



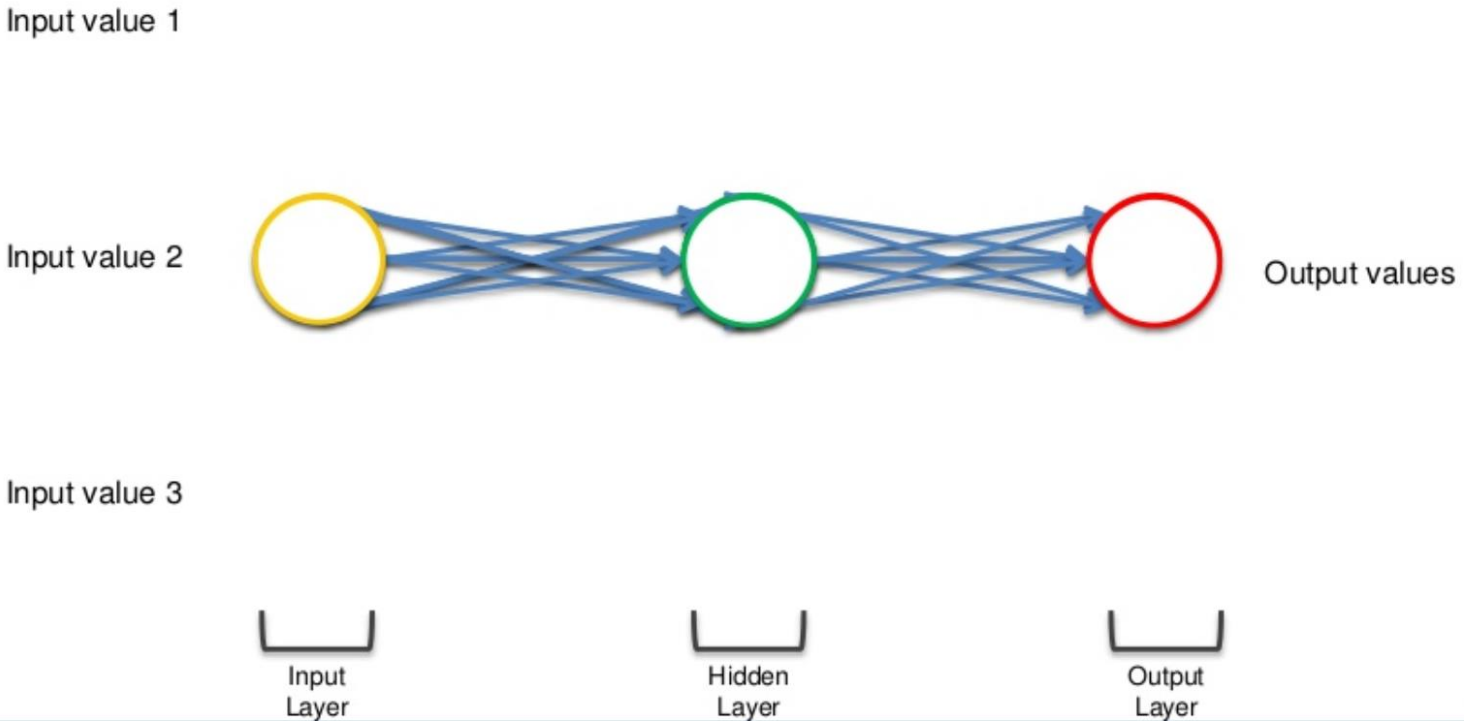
Steps towards vectorized data and parameters

- Weights (multiple arrows)



Steps towards vectorized data and parameters

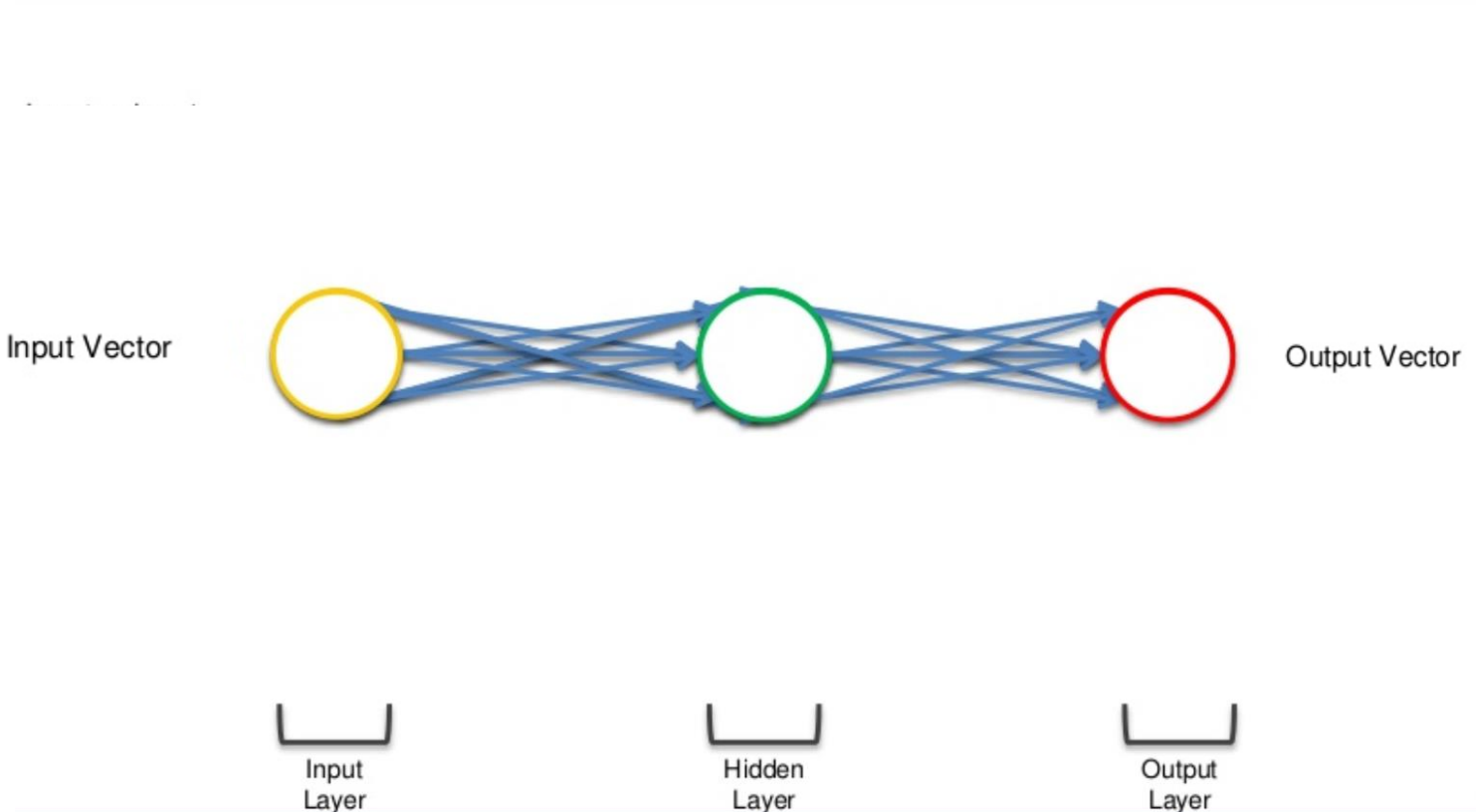
- Weights (multiple arrows)



Steps towards vectorized data and parameters

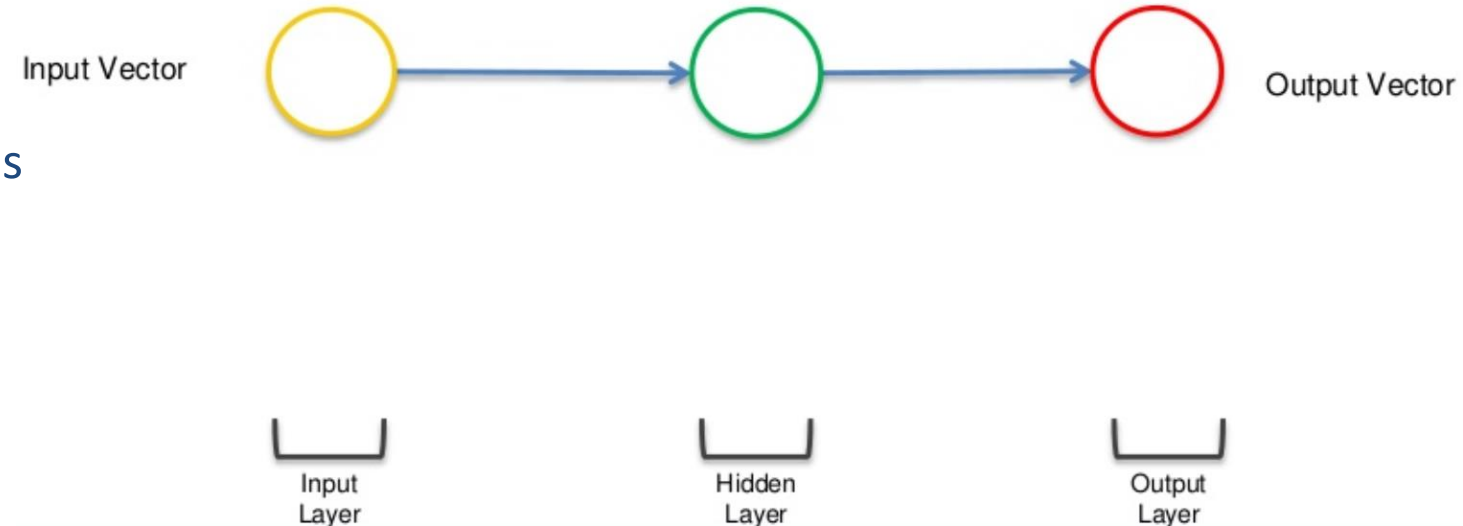
- Weights (multiple arrows)

  replaced with vectors (single arrows)



Steps towards vectorized data and parameters

- Single arrows indicate all interconnections between layers
- w_{ij} matrix matematically



Introducing feedback loop

$$h(0) = \begin{bmatrix} h_1(0) \\ \cdot \\ \cdot \\ h_l(0) \end{bmatrix}$$

$$x(1) = \begin{bmatrix} x_1(1) \\ \cdot \\ \cdot \\ x_k(1) \end{bmatrix}$$

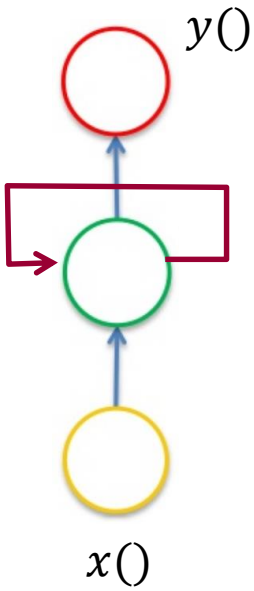
concatenation →

$$c(1) = \begin{bmatrix} h_1(0) \\ \cdot \\ \cdot \\ h_l(0) \\ x_1(1) \\ \cdot \\ \cdot \\ x_k(0) \end{bmatrix}$$

$$h(1) = f\big(h(0), x(1)\big) = \mathbf{W_h} c(1)$$

w: $l \times (k+l)$ *sized weight matrix*

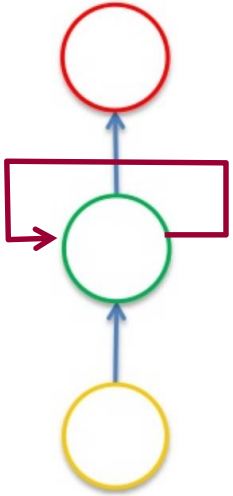
f() can be defined as a more complex function not only a matrix vector multiplication.

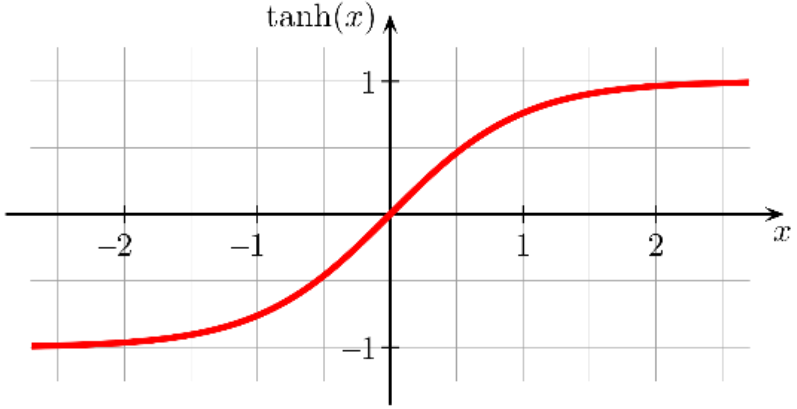


$$h(0) = 0$$

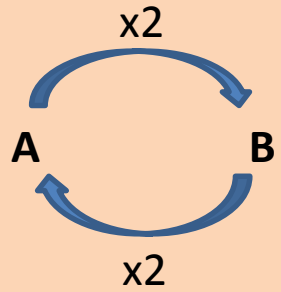
Activation function in feedback loop

- Activation function of the hidden layers is typically hyperbolic tangent
- It avoids large positive feedback
  - Keeps the output between -1 and +1
  - Avoids exploding the loop calculation
  - Gain should be smaller than 1 in the loop!





Positive feedback in a loop:
A *produces more of* ***B*** *which in turn produces more of* ***A.***
It leeds to increase beyond any limit.

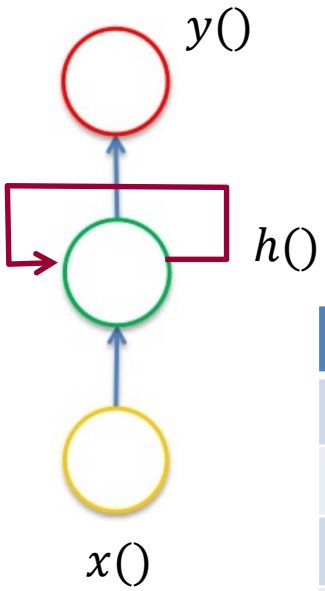


Timing of the RNN

- Discrete time steps are used
- Input vector sequence to apply
- Signals are calculated in a node, when all inputs exist
- State machine

Time	Input	State	output
t=1	$x(1)$	$h(1) = f\big(h(0), x(1)\big)$	$y(1) = g\big(h(1)\big)$
t=2	$x(2)$	$h(2) = f\big(h(1), x(2)\big)$	$y(2) = g\big(h(2)\big)$
t=3	$x(3)$	$h(3) = f\big(h(2), x(3)\big)$	$y(3) = g\big(h(3)\big)$
t=4	$x(4)$	$h(4) = f\big(h(3), x(4)\big)$	$y(4) = g\big(h(4)\big)$
		. . .	

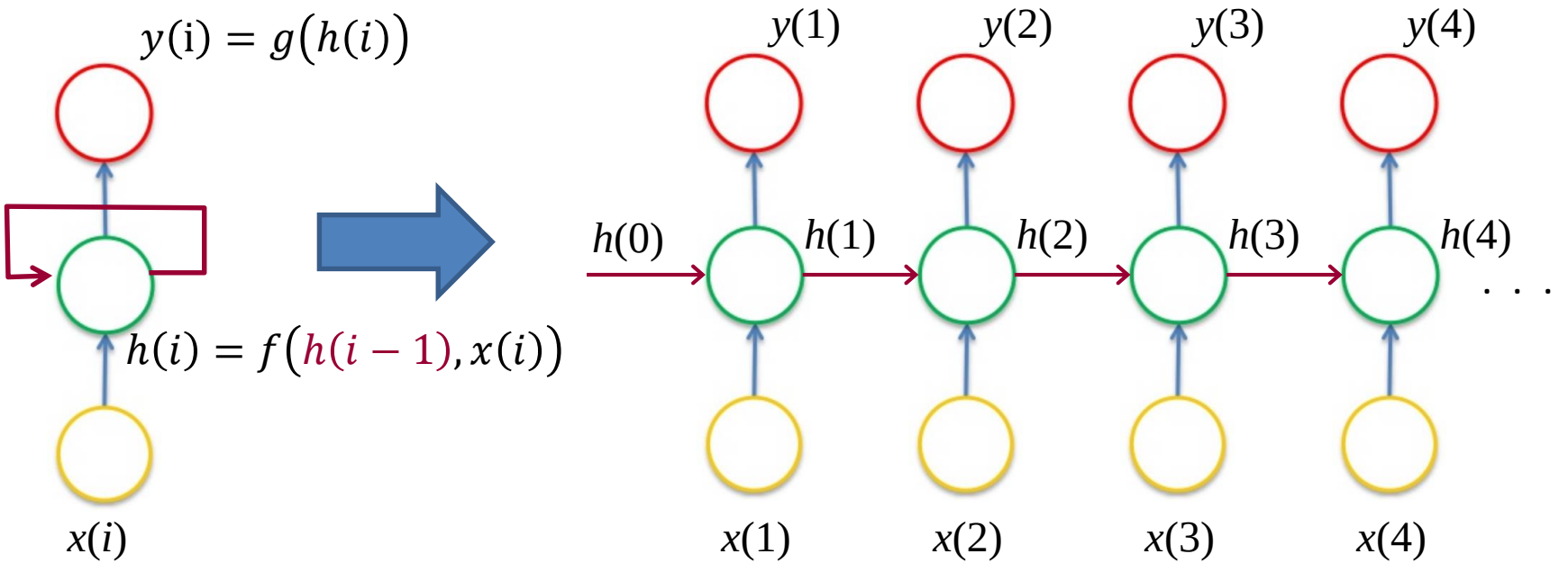
How to calculate back propagation?



$h(0) = 0$

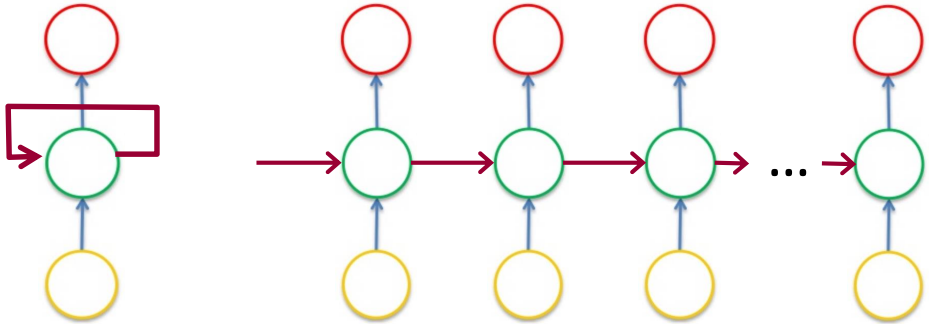
input
$x(1)$
$x(2)$
$x(3)$
$x(4)$
…

Unrolling



Unrolling

- Unrolling generates an acyclic directed graph from the original cyclic directed graph structure
- It generates a final impulse response (FIR) filter from the original infinite impulse response (IIR) filter
- Dynamic behavior



IIR filters may response to any finite length input with a infinite (usually decaying) response, due to their internal loop.

FIR filters response to any finite length input with a final response.

Weight matrix sharing

RNN re-uses the same weight matrix in every unrolled steps.