Neural Networks

Unsupervised learning techniques

(P-ITEEA-0011)

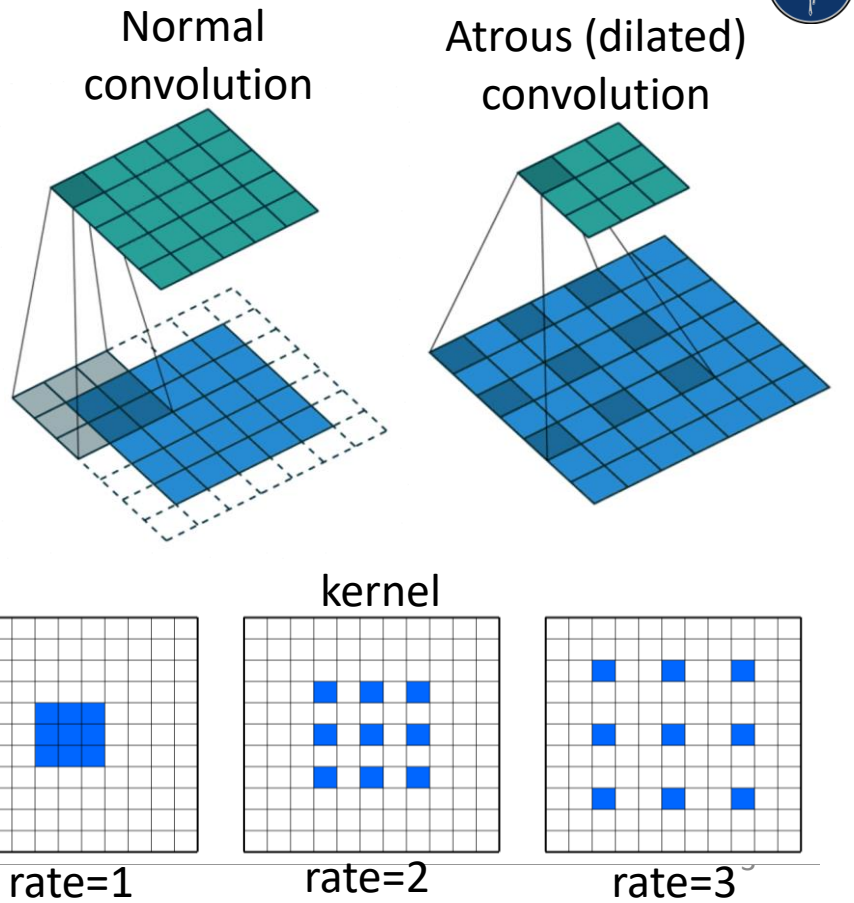
Akos Zarandy
Lecture 8
November 12, 2018

Contents

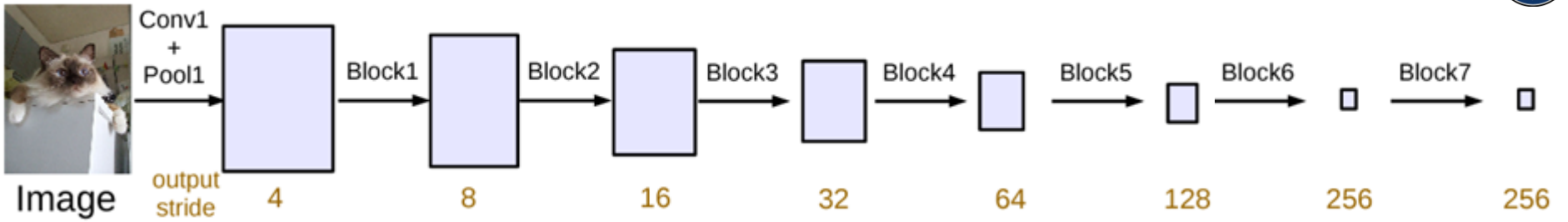
- Resolution controlling
  - Atrous convolutions, sub-pixel image combination
- Supervised vs unsupervised learning
- Unsupervised learning techniques
  - Curse of dimensionality
  - Principal component analysis (PCA)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
  - Autoencoder

Atrous convolution

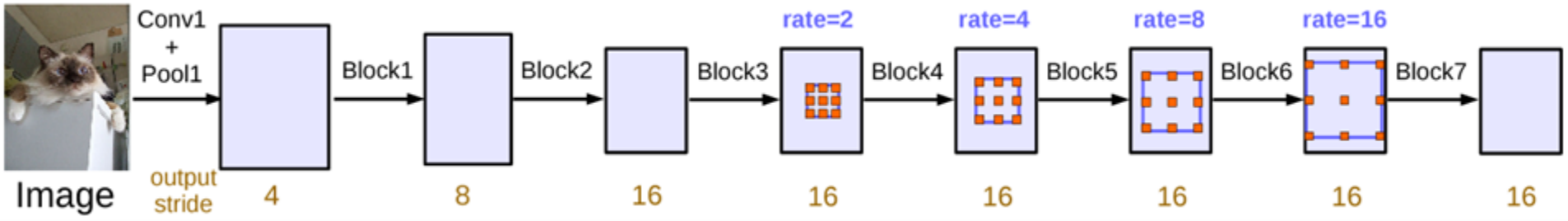
- How it works?
  - Blows up the kernel
  - Filling up the holes with zeros
    - Atrous means very dark (like the wholes between the values)
- Properties
  - Not doing downsampling
  - Not increasing computational load
  - But reaches larger neighborhood
  - Combines information from larger neighborhood



Going deep



Normal convolution goes deeper with reducing resolution



Atrous convolution goes deeper without further reducing resolution

Filter size considerations

- **Small** field-of-view → accurate **localization**
- **Large** field-of-view → context **assimilation**
- Effective filter size increases (enlarge the field-of-view of filter)

$$n_o\colon k \times k \quad \rightarrow \quad n_a\colon \big(k + (k-1)(r-1)\big) \times \big(k + (k-1)(r-1)\big)$$

n_o : original kernel size

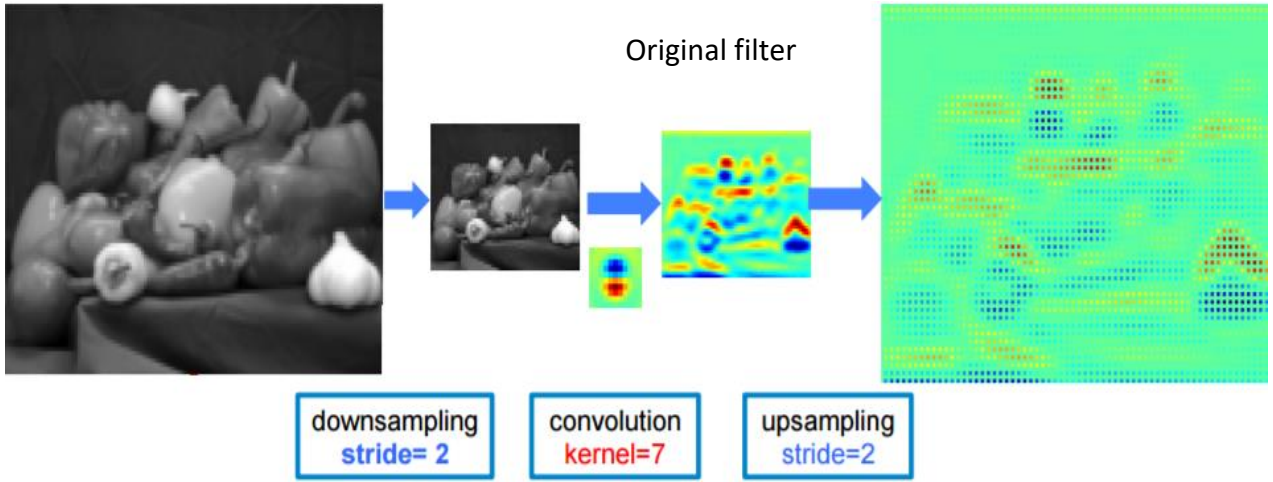
n_a : original kernel size

r: rate

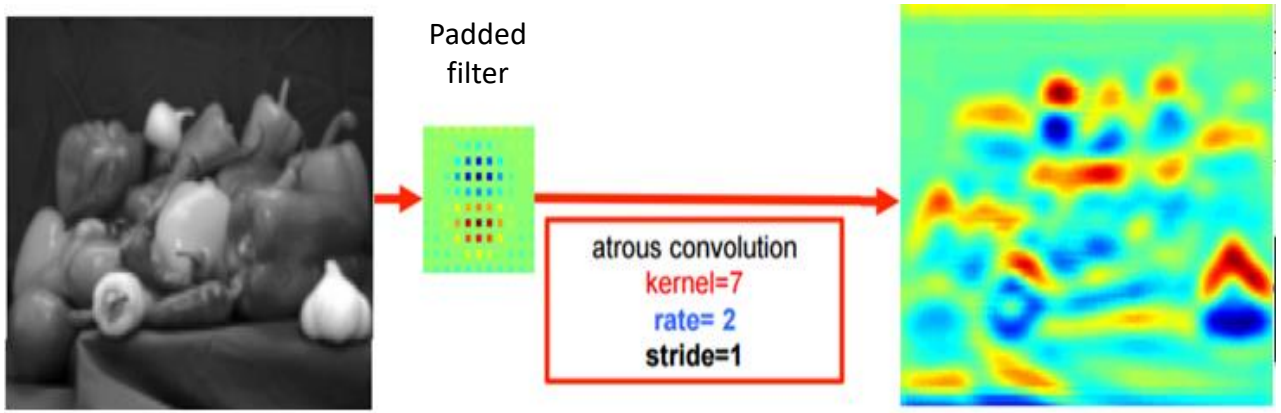
- However, we take into account only the non-zero filter values:
  - Number of filter parameters is the same
  - Number of operations per position is the same

Visualizing atrous convolution

Standard convolution

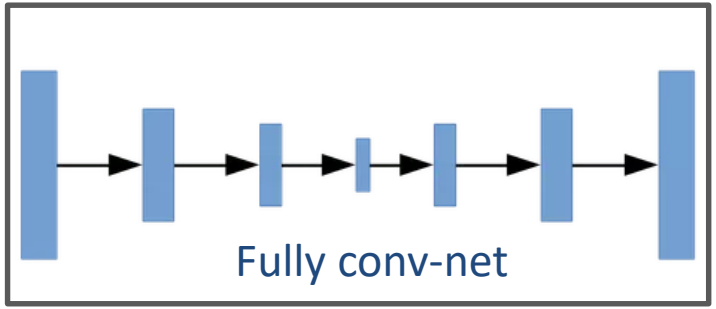


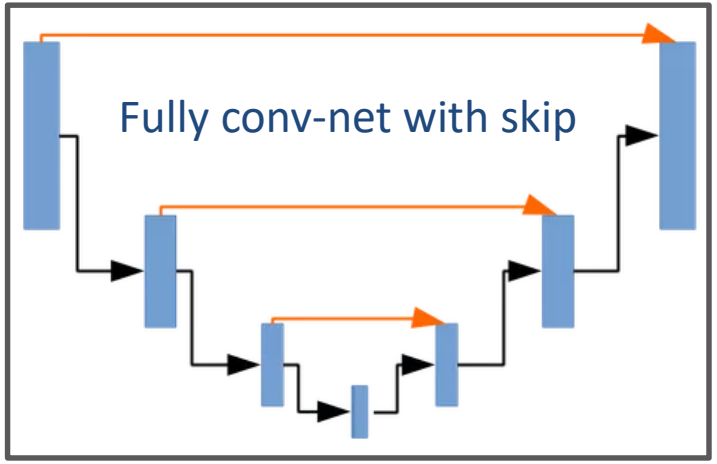
Atrous convolution

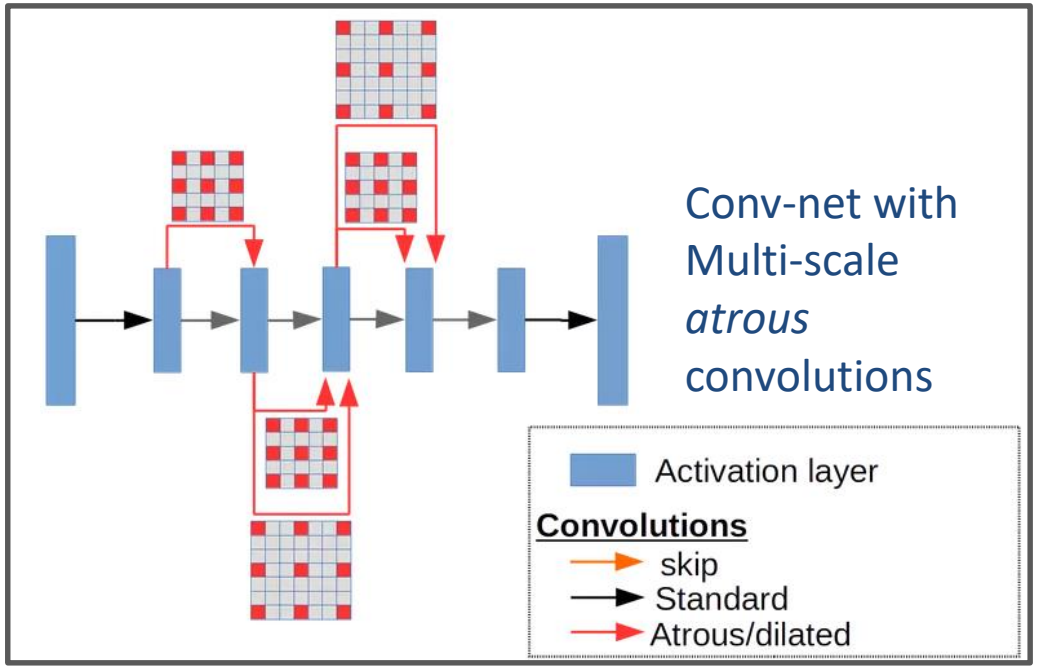


Chen, Liang-Chieh, et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs." *arXiv preprint arXiv:1606.00915* (2016).

Semantic segmentation CNN arrangements





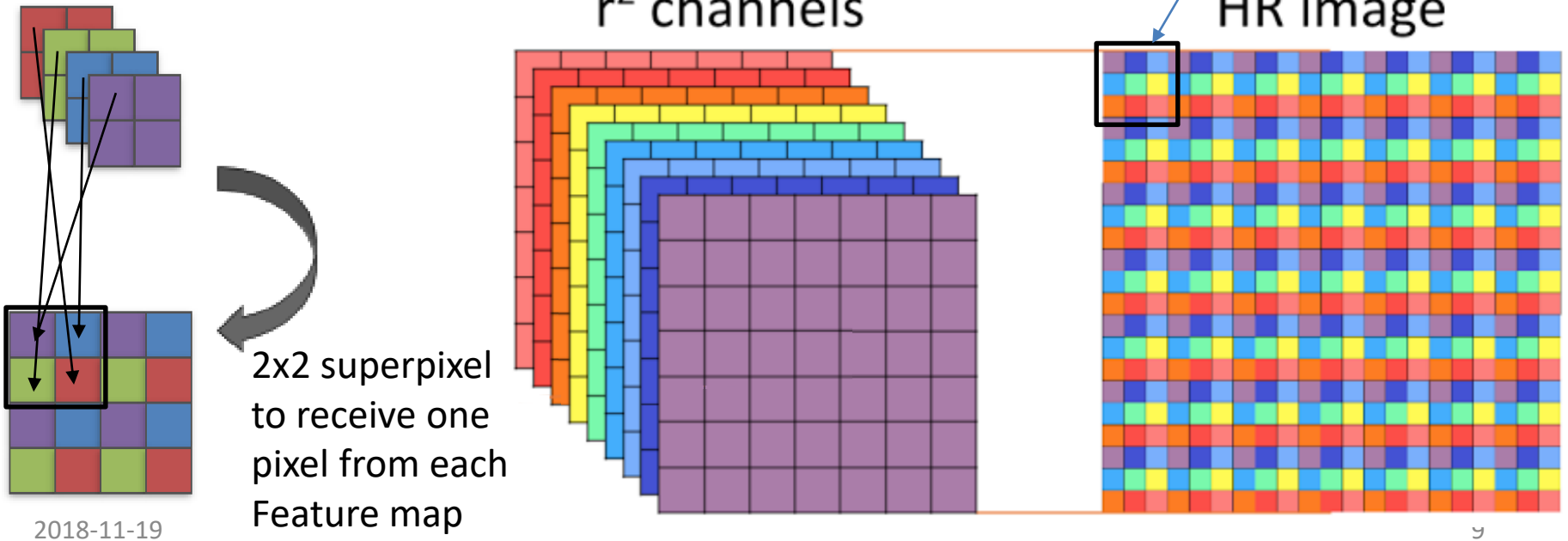


- How to solve reduced resolution?
  - Do not downsample !!!
    - Convolution on large images ⇒ Small FOV Enlarge kernel
    - Size $O(n^2)$ more parameters ⇒ getting close to fully
    - Connected layer, slow training, overfitting
  - Atrous Convolution.
    - Large FOV with little parameters → Kill two birds with one stone!

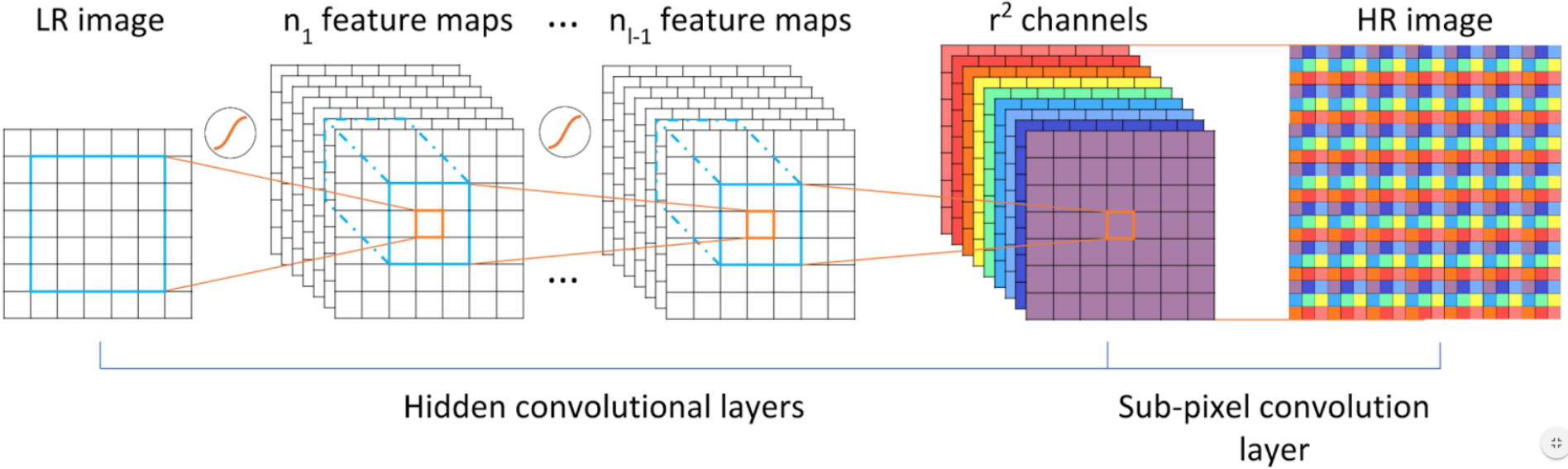
Sub-pixel upsampling

Sub-pixel upsampling

- Technique for combining multiple low resolution layers to a high resolution layer
- Avoids introducing zeros pixels or duplicated pixels
- Avoids checkerboard pattern with transpose convolution



Sub-pixel upsampling



- Each feature map is holding information for a sub-pixel position
- During the training, each feature map learns how to calculate the subpixels

Unsupervised learning

Typical Machine Learning Types

No two machine learning tasks are identical, but still there are common prototypes:

- **Supervised Learning**
  - Learning from labeled examples (for which the answer is known)
- **Unsupervised Learning**
  - Learning from unlabeled examples (for which the answer is unknown)
- Semi-supervised Learning
  - Learning from both labeled and unlabeled examples
- 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 (number of pixels in an imaga)
  - Occurs when d (# dimensions) is large in relation to n (number of samples).

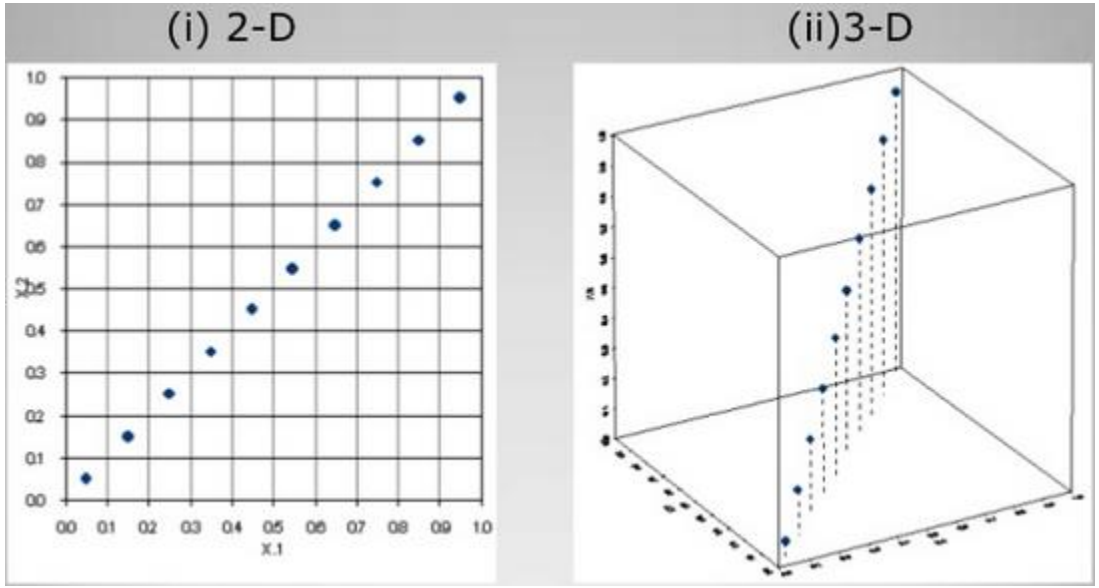
- Real life examples:
  - Genomics
    - We have ~20k genes, but disease sample sizes are often in the 100s or 1000s.

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

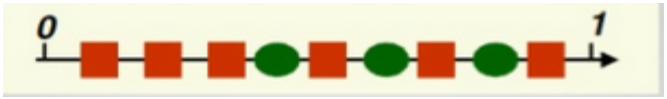
- A sample that maps 10% of the 1x1 squares in 2D represent only 1% of the 1x1x1 cubes in 3D

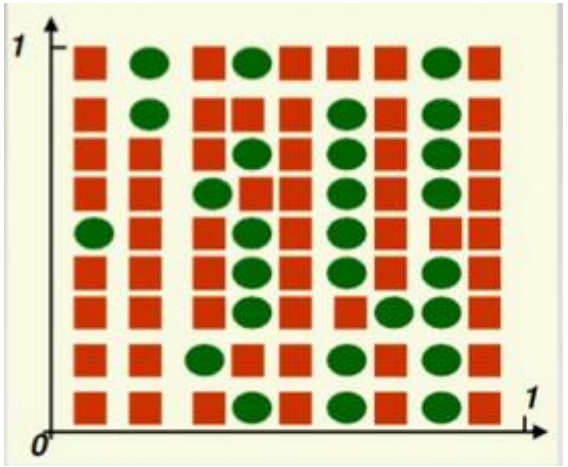


- There is an exponential increase in the search-space

Sparse example

- Suppose we want to discriminate between samples from two categories





72 new samples needed

- This one feature is not discriminative. We cannot classify well, thus we decide to add a second feature.
- To maintain the sample density of 9 samples per unit length (as above) how many samples do we need?
- We need 81 samples to maintain the same density as in 1D
- In d dimension we need 9^d samples!
  - Otherwise, the data becomes more sparse

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 = $

Curse of dim - Some mathematical (weird) effects

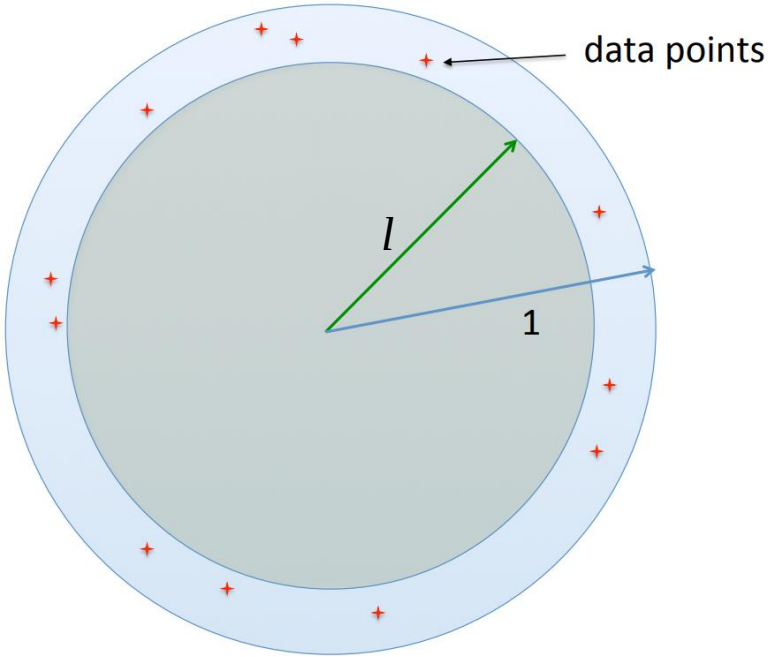
- Ratio between the volume of a sphere and a cube for **d**=3: $\frac{(\frac{4}{3})\pi r^3}{(2r)^3} \approx \frac{4r^3}{8r^3} \approx 0.5$
- When **d** tends to infinity the volume of the sphere (this ratio) tends to zero

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

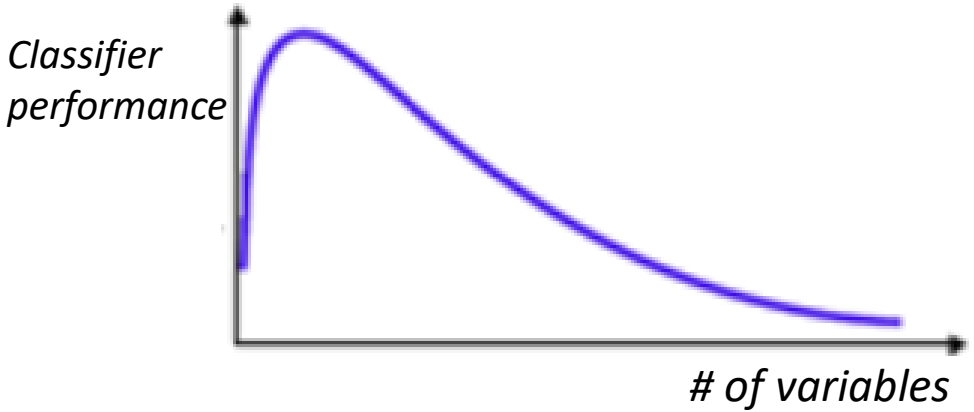
How to calculate dimensionality?

	x_1	x_2	x_3	x_4
d_1	1	2	1	1
d_2	2	4	0.5	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, 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



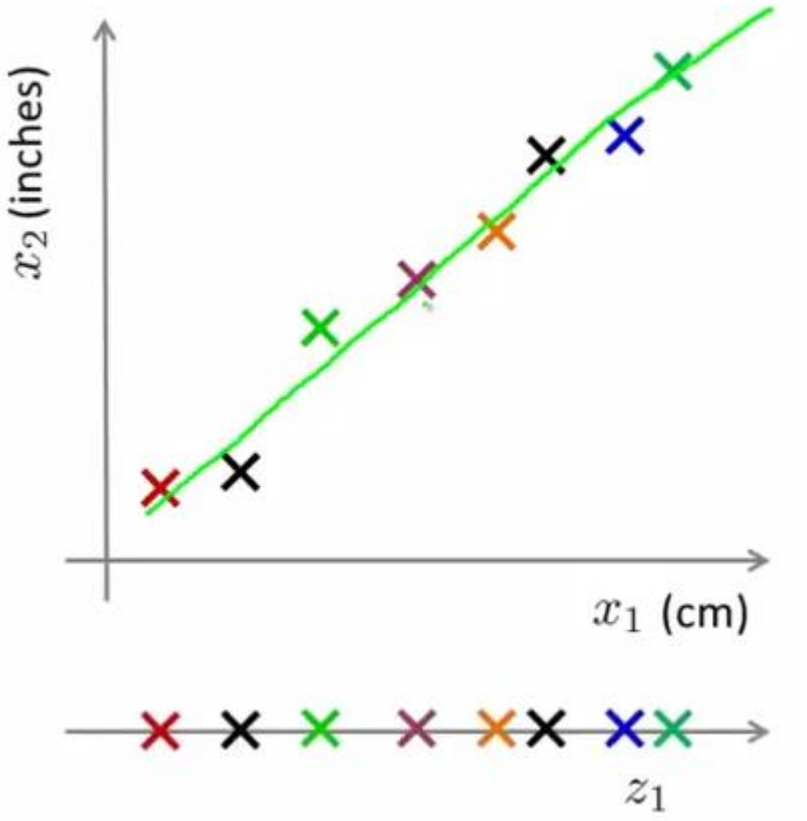
Principal component analysis

(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

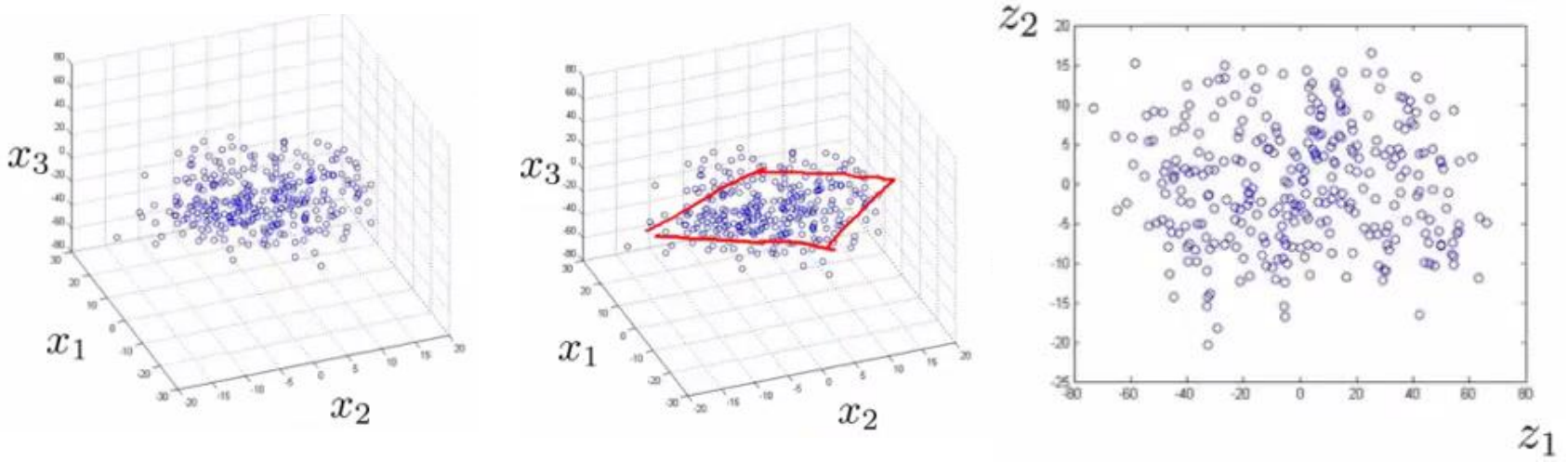
Example – reducing data from 2d to 1d



- 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



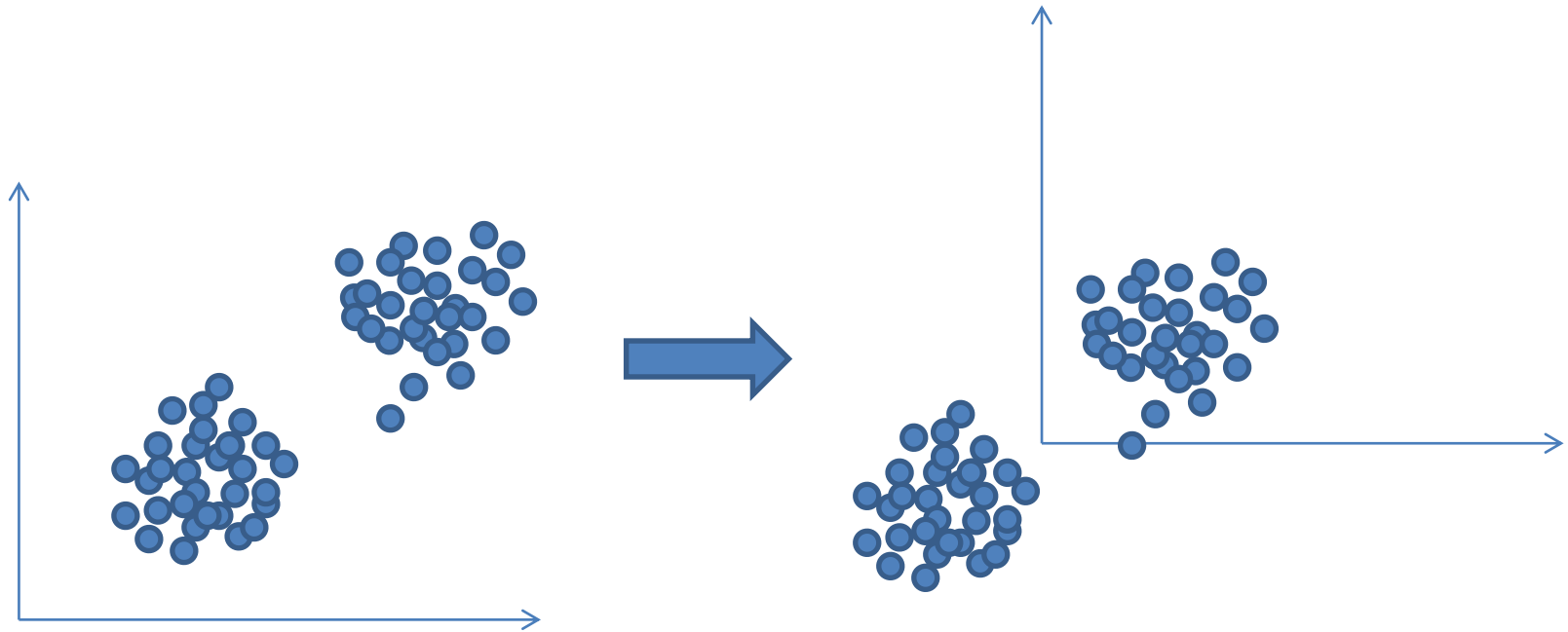
- If we were to project the data onto a plane we would have a more compact representation
- So how do we find that plane without loosing too much of the **variance** in our data? → PCA

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

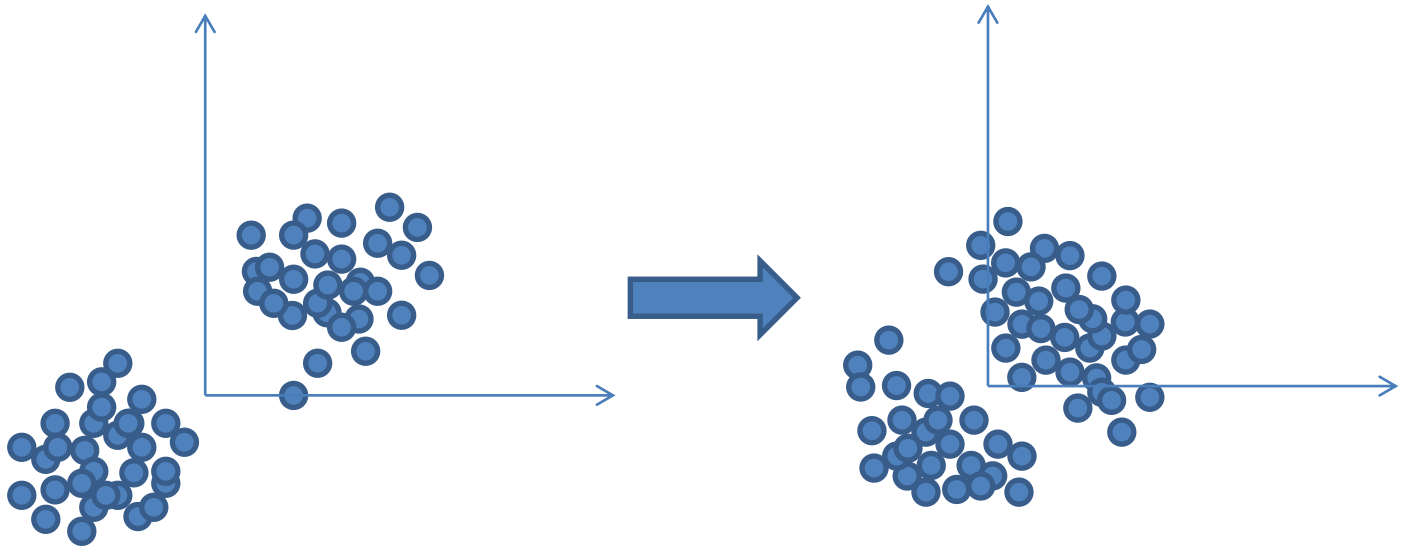
1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.



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

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

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

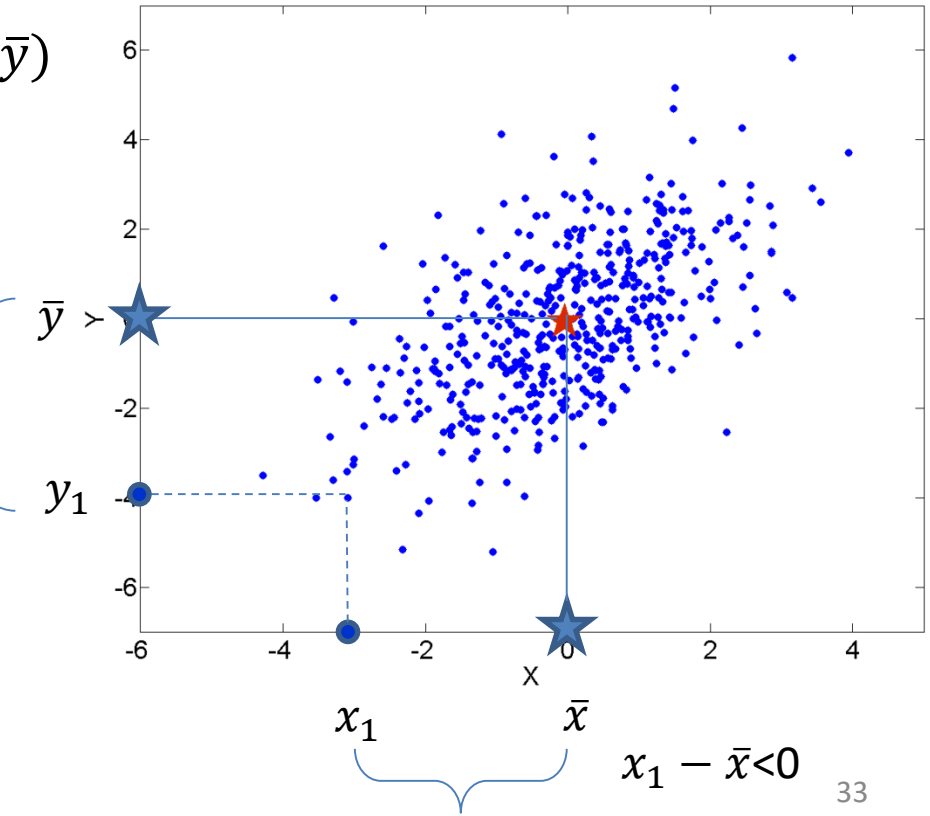
$$\text{Covariance}(\text{x}, \text{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

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

Covariance example for 2D

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

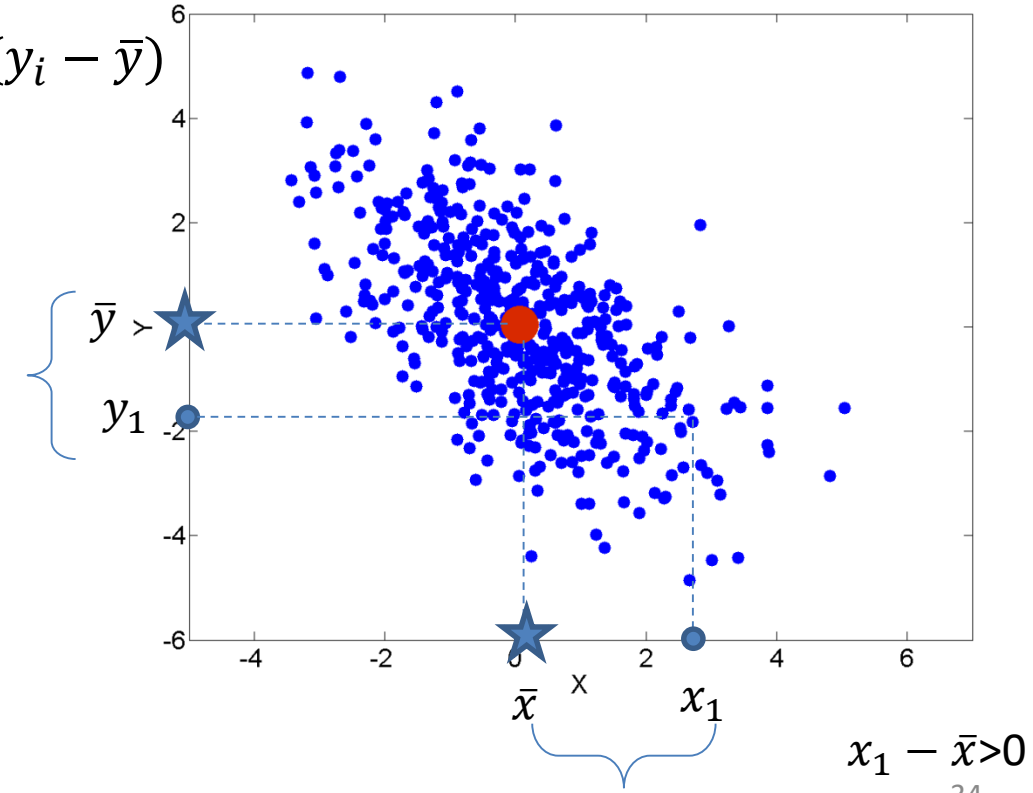
- Positive covariance between the two dimensions



Covariance example for 2D

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

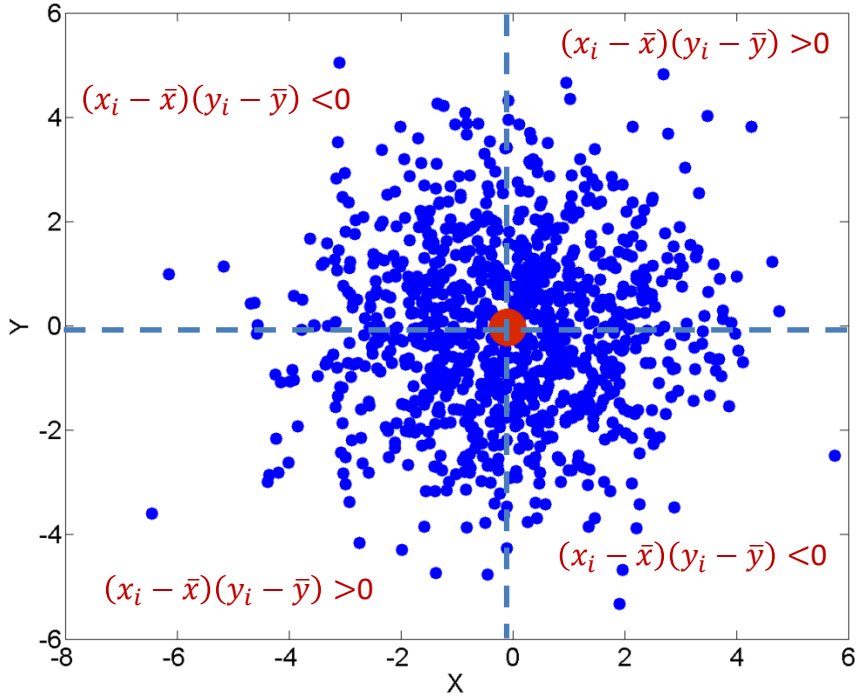
- Negative covariance between the two dimensions



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

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

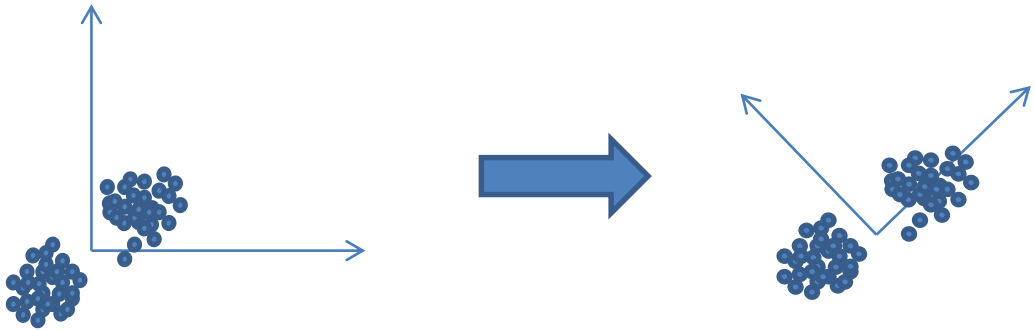
$$Cov\,(\sum) = \frac{1}{n}(X-\bar{X})(X-\bar{X})^T; where\ X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

- 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.



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

PCA algorithm

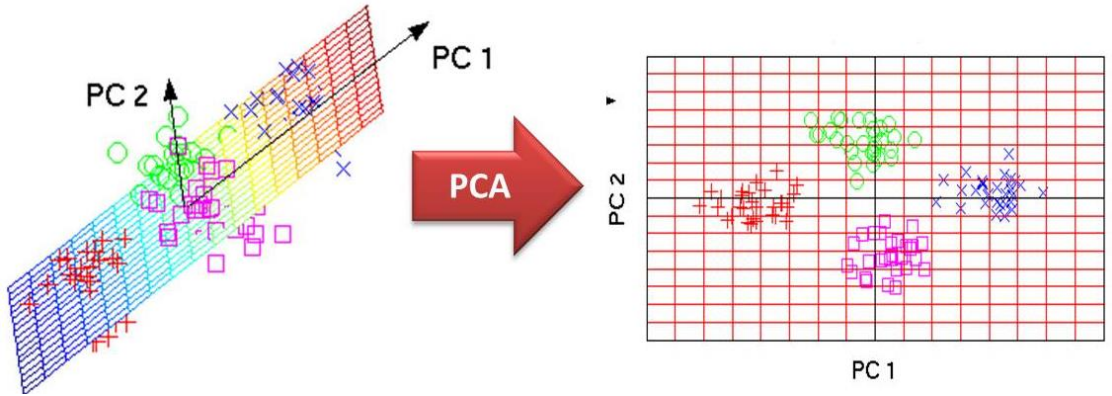
1. Mean normalization: For every value in the data, subtract its mean dimension value. This makes the average of each dimension zero.
2. Standardization (optional): Do it, if you want to have each of your features the same variance.
3. Covariance matrix: Calculate the covariance matrix
4. Eigenvectors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

$$\begin{pmatrix} y1 \\ \vdots \\ yK \end{pmatrix} = \begin{pmatrix} u11 & \cdots & uK1 \\ \vdots & \ddots & \vdots \\ u1n & \cdots & uKn \end{pmatrix}^T \begin{pmatrix} x1 \\ \vdots \\ xn \end{pmatrix}$$

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

Principal Component Analysis (PCA)

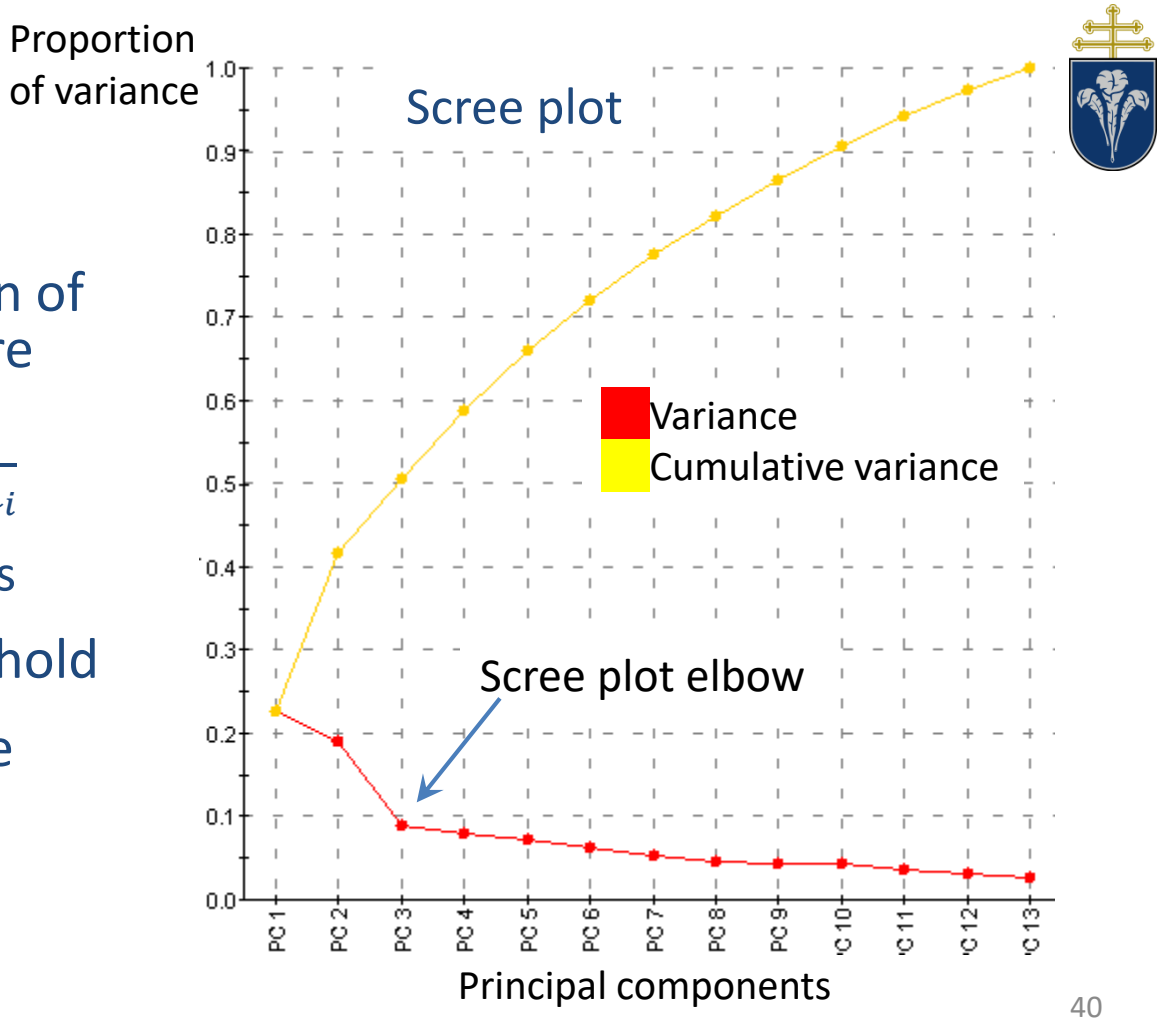
- The idea is to project the data onto a subspace which compresses most of the variance in as little dimensions as possible.
- Each new dimension is a **principle component**
- The principle components **are ordered** according to how much **variance in the data** they capture
  - Example:
    - PC1 – 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



t-Distributed Stochastic Neighbor Embedding

(t-SNE)

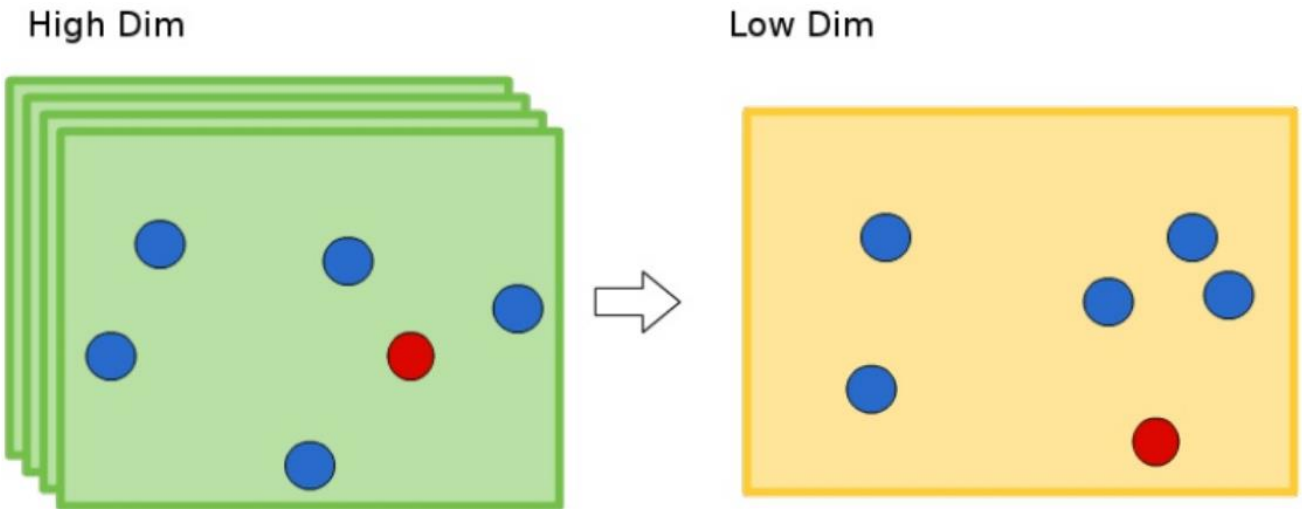
t-Distributed Stochastic Neighbor Embedding (t-SNE)

- 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

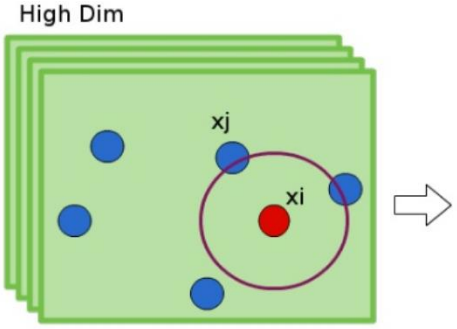
Step 1: *Generate the points in the low dimensional data set (2D or 3D)*

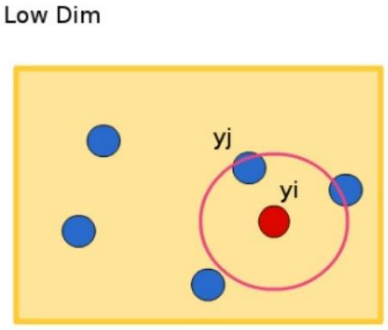
- random initialization
- First two or three components of PCA



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 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)

Student-t distribution is used to measure similarities between low-dimensional points in order to allow dissimilar objects to be modeled far apart in the map.

Student distribution

- We know a few samples of a large normally distributed population $N(\mu, \sigma^2)$ with expected mean and variance
- The mean value of these members are calculated normally

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

- The (Bessel corrected) variance of the samples is:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

- Normal variance function would underestimate the variance due to the limited number of samples

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*

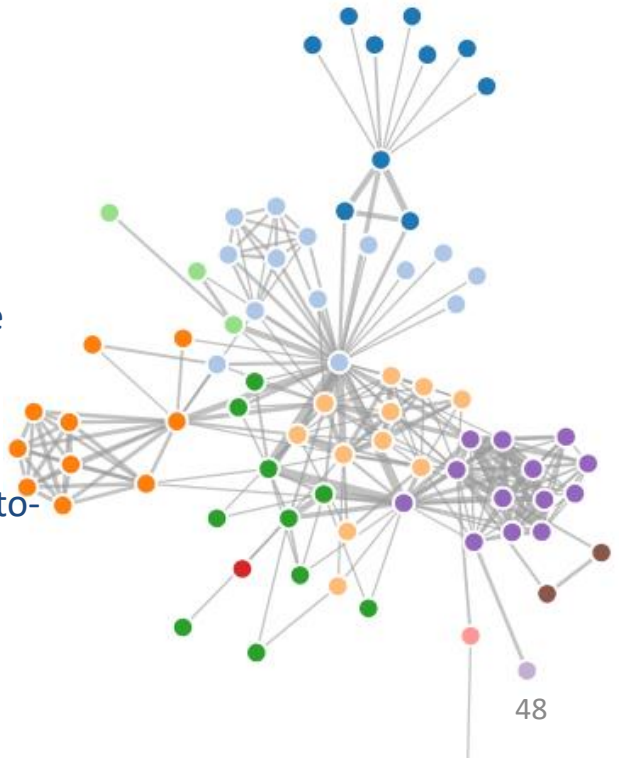
- Gradient has a surprisingly simple form:

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

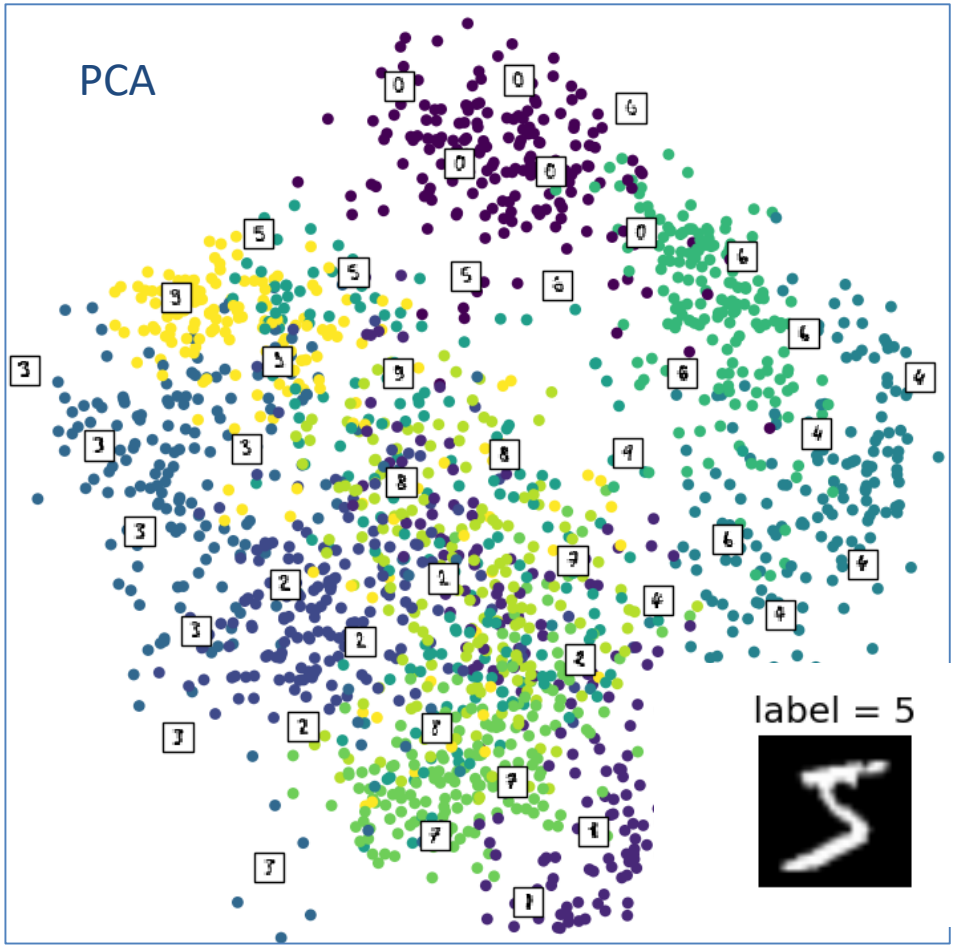
- Optimization can be done using momentum method

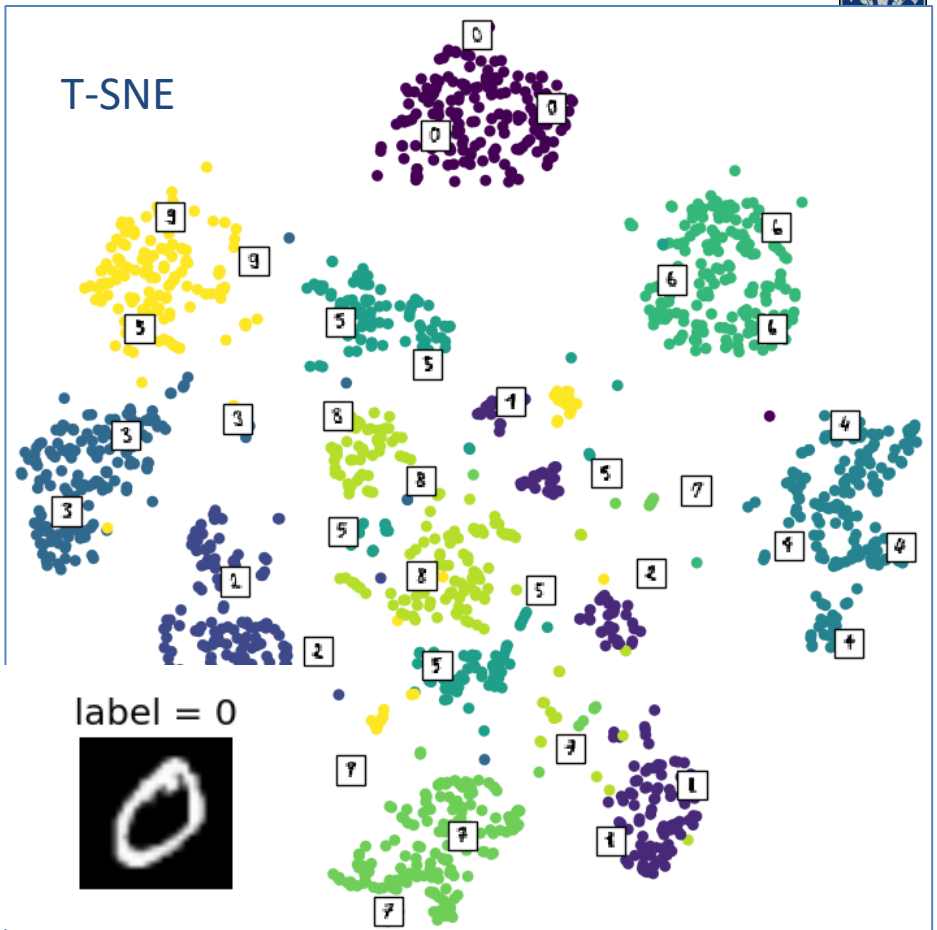
Physical analogy

- 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





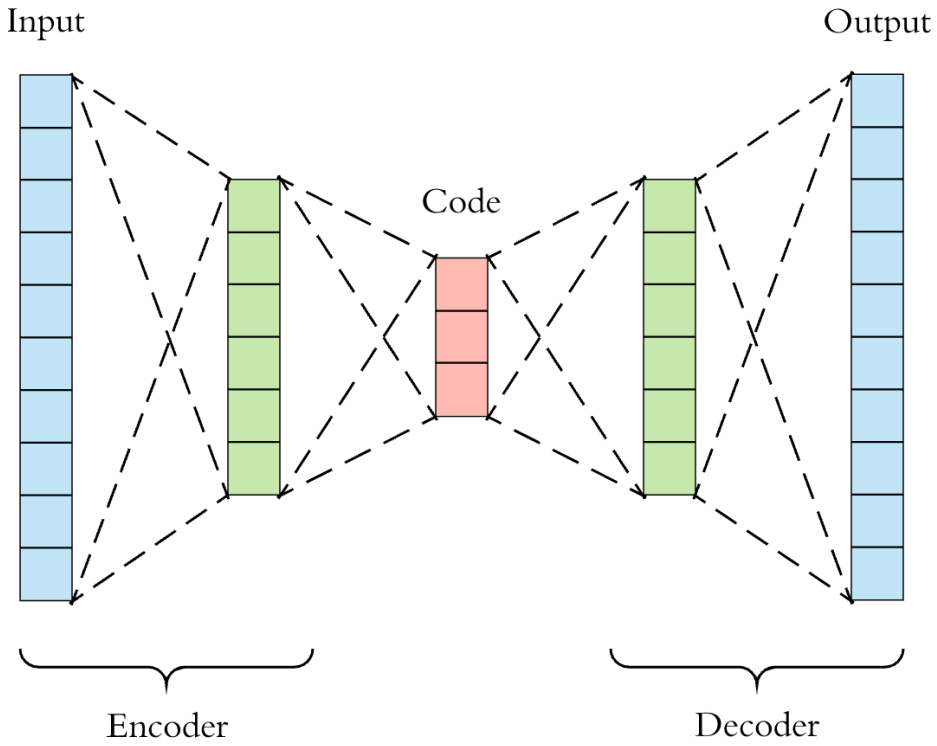
label = 5

label = 0

Autoencoder

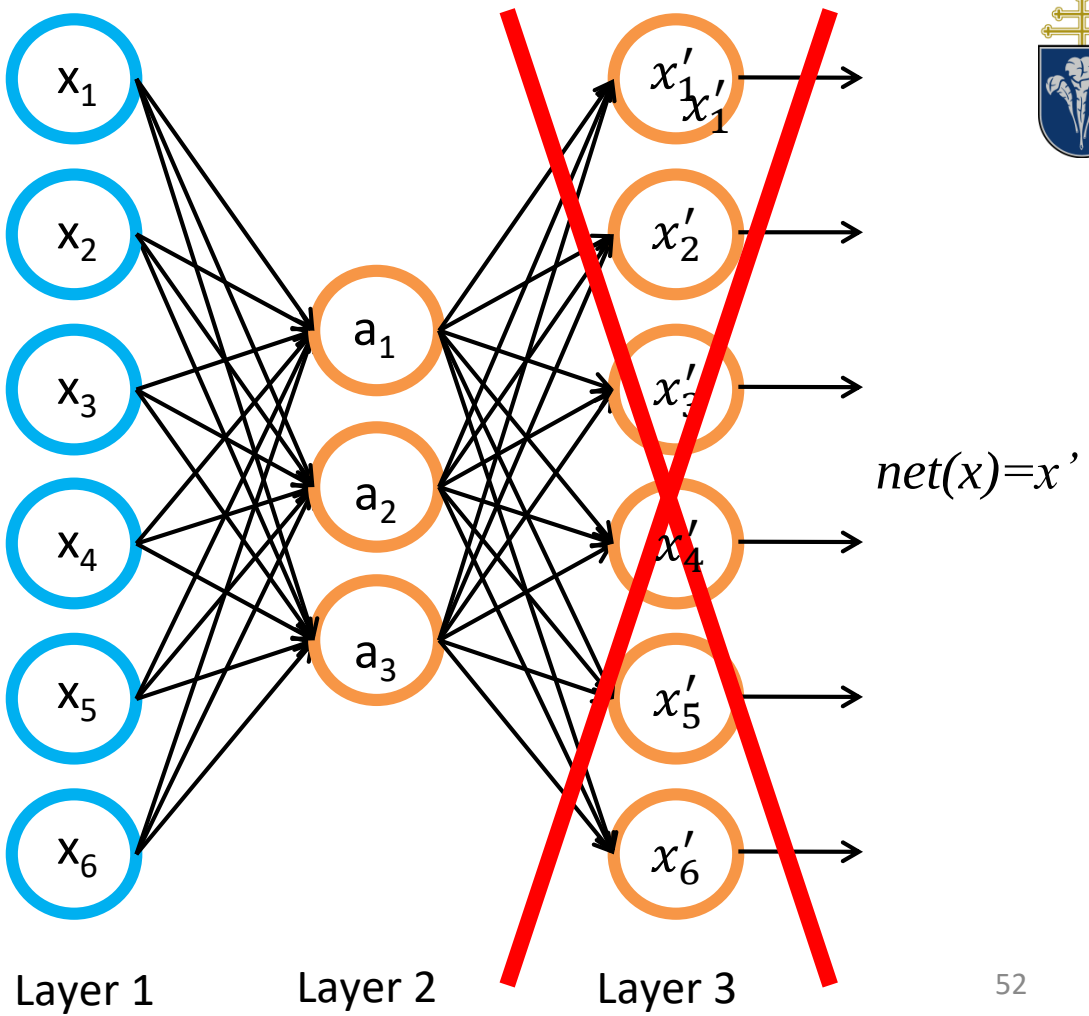
Autoencoder

- 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



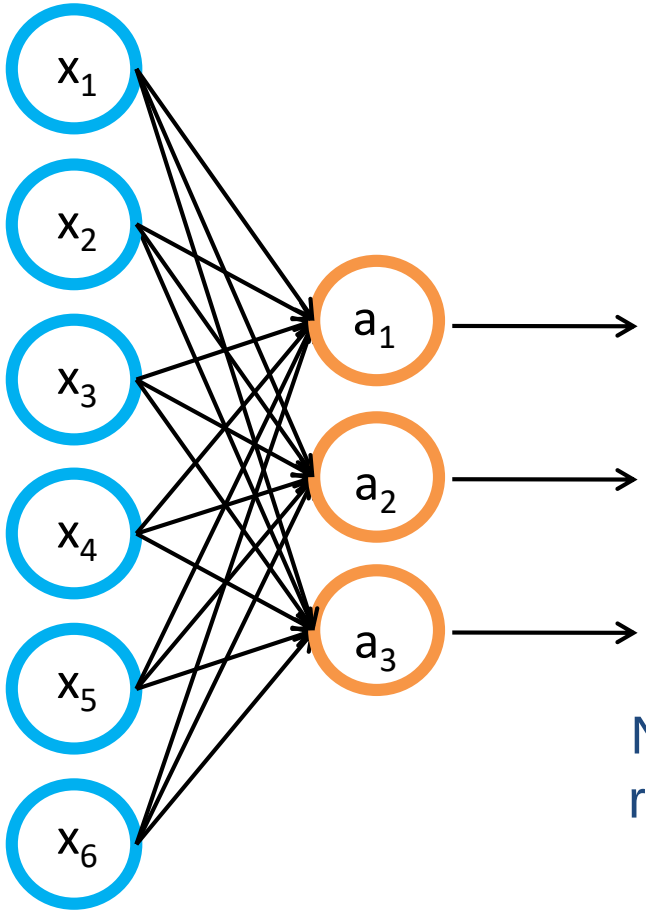
Operation

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



Operation

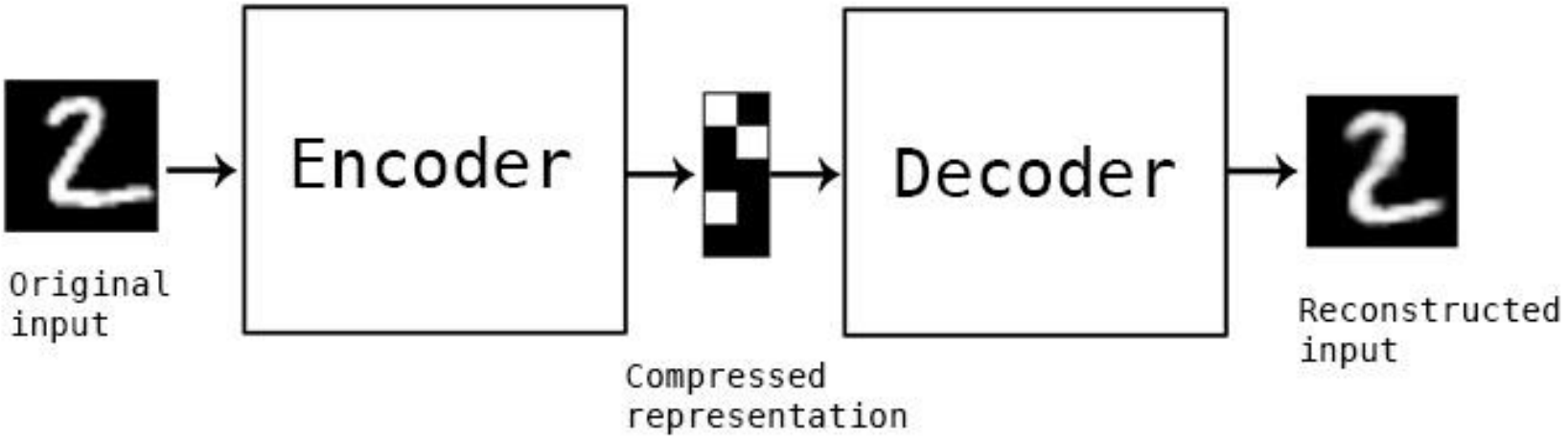
- The network is trained with the same input-output pairs
- Loss function:
  - MSE
  - Cross Entropy
- After network is trained, remove decoder part



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

New compressed representation for input.

Example



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

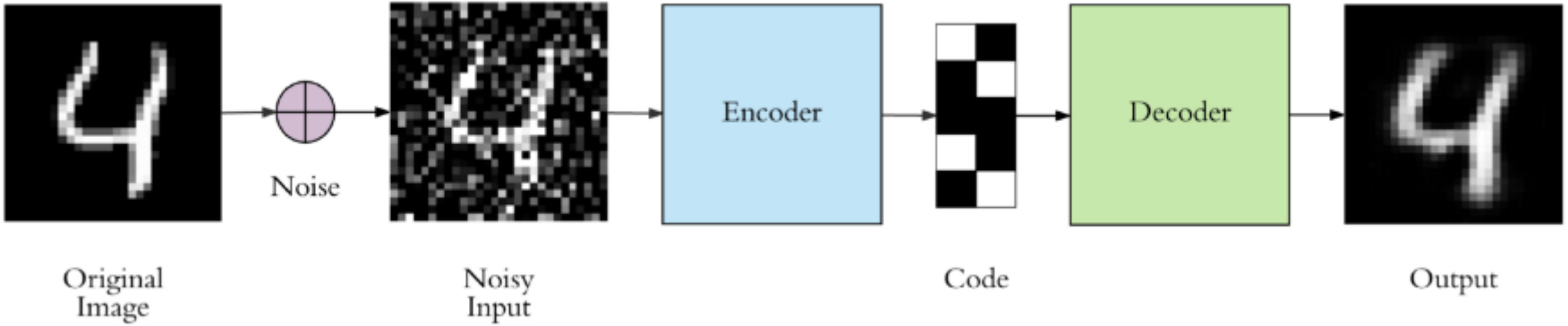
Autoencoder vs PCA

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

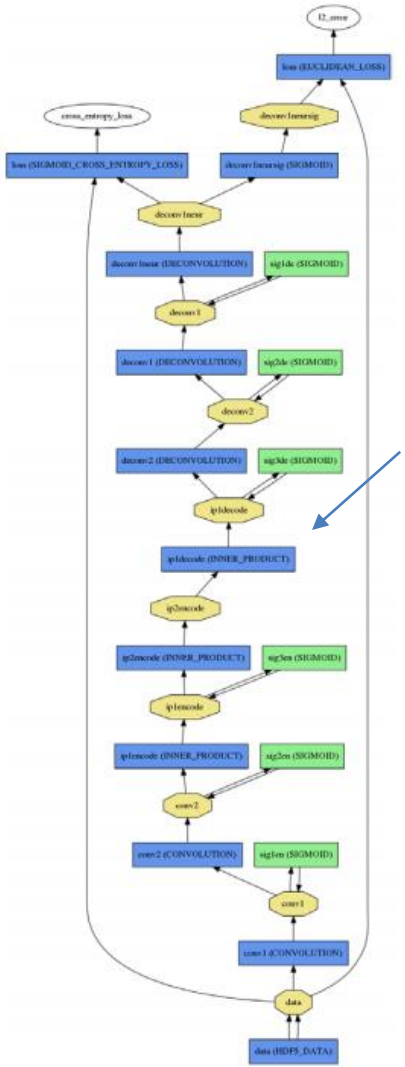
Undercomplete: width of hidden layer is smaller than width input/output layer

Denoising

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



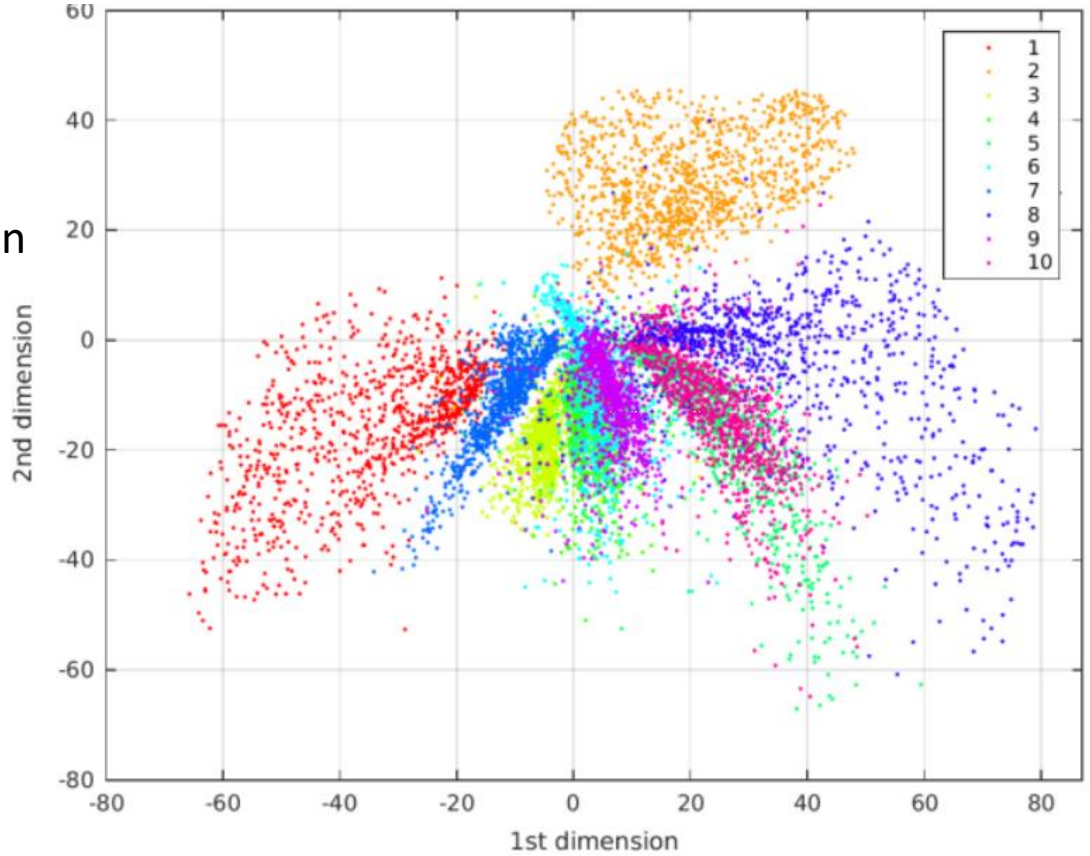
MNIST database coding to two dimension



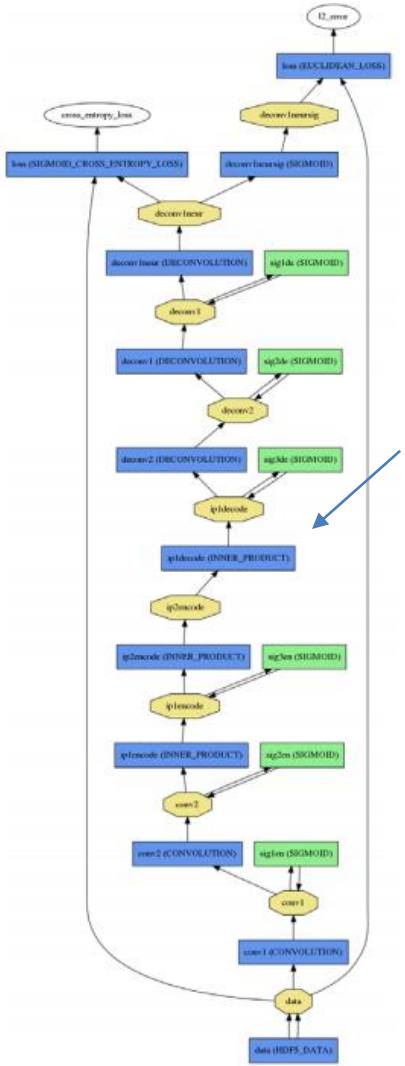
Two neurons in the coding hidden layer

label = 5

label = 0



Autoencoder + t-SNE



Two neurons in the coding hidden layer

label = 5

label = 0