

# Graphical Models, Inference and Learning Lecture 7

-

## Learning / Convolutional Neural Networks

Yuliya Tarabalka

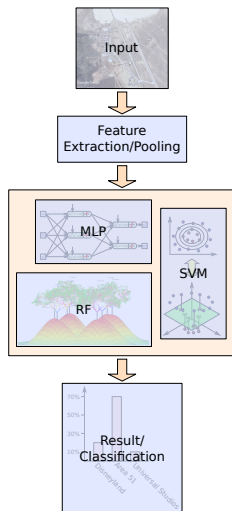
Inria Sophia Antipolis-Méditerranée - TITANE team  
Université Côte d'Azur - France

LUXCARTA Technology



# Classical Learning

- Features extract very basic, low level information
- We want very high level information (e.g. class of objects)
- Classical Learning: Learn the mapping between low level features and high level information





# Methods

- Choice of method not always rational
- Different pros/cons
- Speed, memory, scalability of training data, ease of implementation, ease of hyper parameter tuning, ...
- First intuitive understanding of the problems, then identifying methods

# Decision based on features

## Toy example

Task: Classify fruits into either bananas or apples

### Extracted Feature Vector

- Hue (yellow to red)
- Elongation (max extend over min extend)

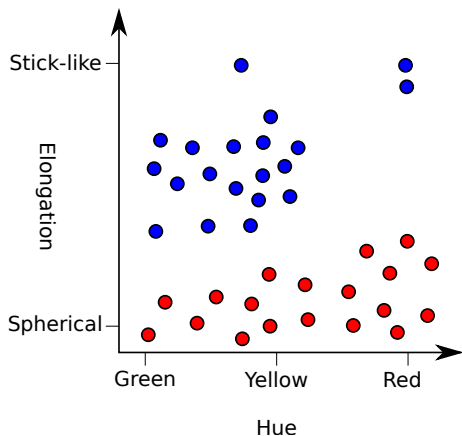


image by Abhijit Tembhekar licensed under CC BY 2.0

image by Darkone licensed under CC BY SA 2.0

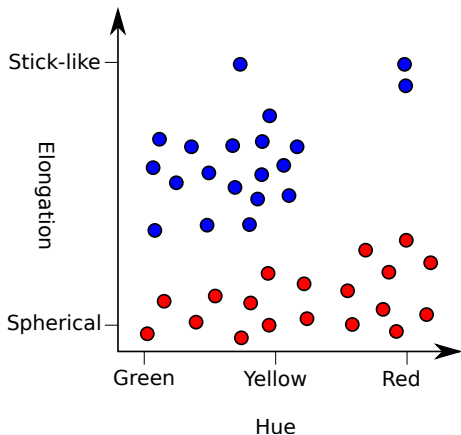
# Some training data

- Feature space is just 2D
- Datapoints can be plotted as a scatter plot



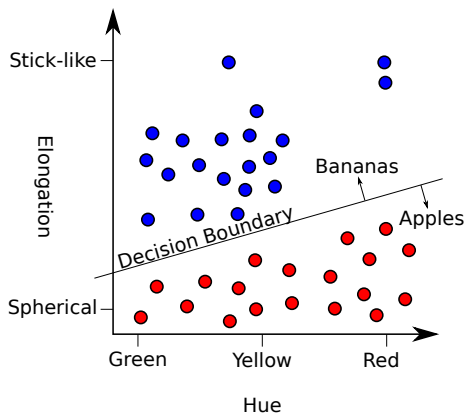
# Some training data

- Feature space is just 2D
- Datapoints can be plotted as a scatter plot
- Can we “learn”, which part of the feature space is bananas/apples?



# Decision boundary

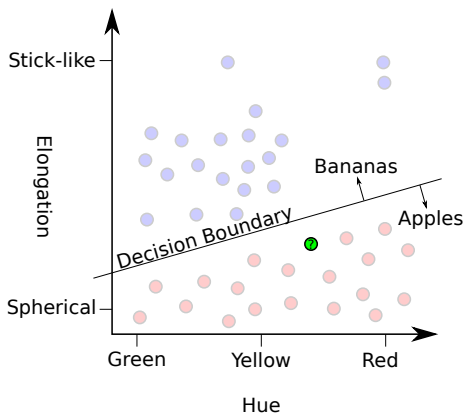
- (Very) simple idea: Split the feature space into two half spaces





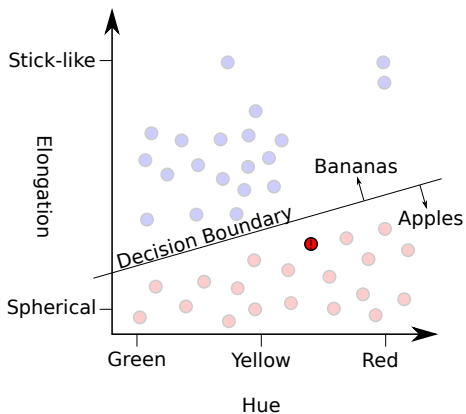
# Decision boundary

- (Very) simple idea: Split the feature space into two half spaces
- During application, classify data based on this decision boundary



# Decision boundary

- (Very) simple idea: Split the feature space into two half spaces
- During application, classify data based on this decision boundary

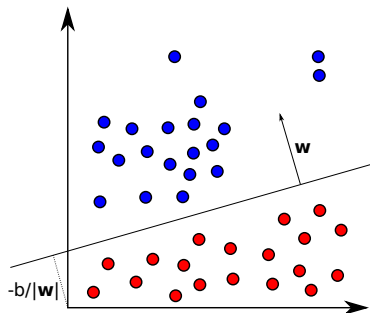


# Perceptron

## Perceptron

$$y = \text{sign}(\mathbf{w}^T \mathbf{x} + b) \quad (1)$$

- $y \in \{-1, 1\}$ : Predicted class
- $\mathbf{x} \in \mathbb{R}^2$ : Feature vector
- $\mathbf{w} \in \mathbb{R}^2$ : “Weight vector” (needs to be learned)
- $b \in \mathbb{R}$ : “Bias” (needs to be learned)



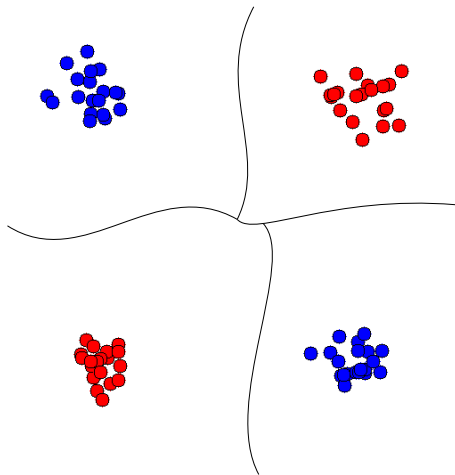
# Linear Separability

- What if no such line exists?
- Quite often, problem not linearly separable
- Needs non-linear decision boundary



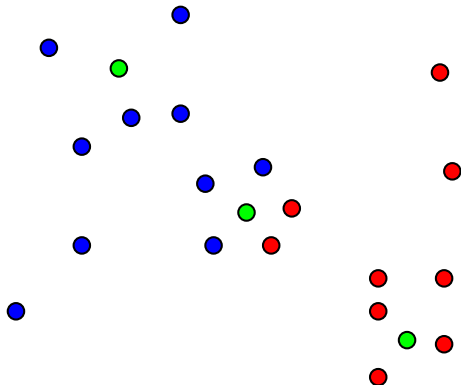
# Non-linear Decision Boundary

- Decision boundaries of more complex ML techniques usually non-linear
- Regions need not be connected



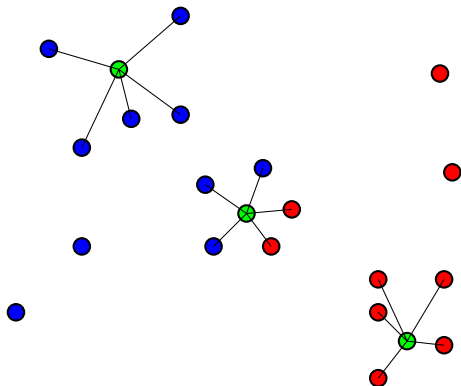
# kNN

- Very simple idea:  
k-Nearest-Neighbors for  
classification



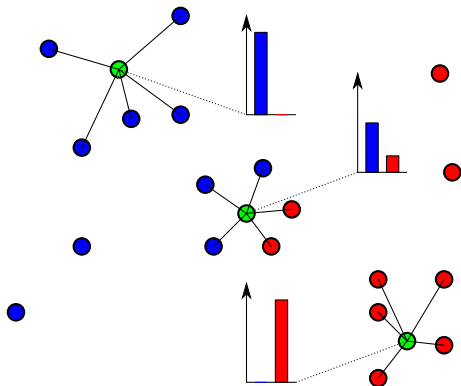
# kNN

- Very simple idea:  
k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset



# kNN

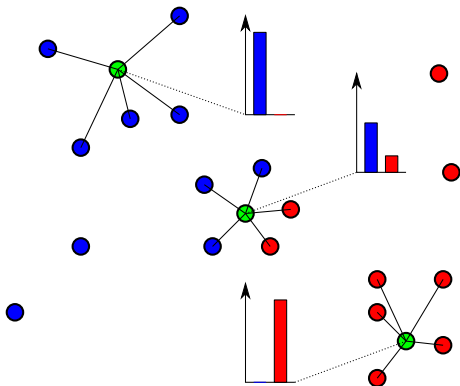
- Very simple idea:  
k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset
- Look at the labels of those neighbors





## kNN

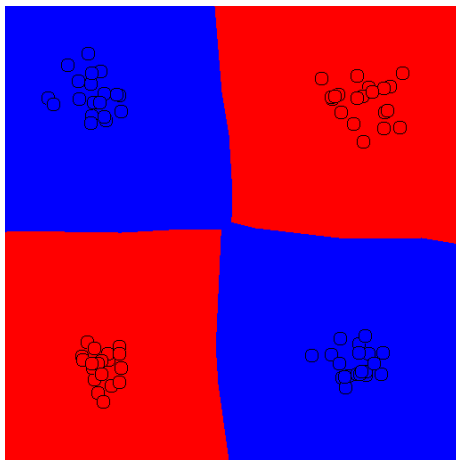
- Very simple idea:  
k-Nearest-Neighbors for classification
- For a sample find the k (e.g. 5) closest data points in the training dataset
- Look at the labels of those neighbors
- Fast lookup through trees/approximate methods
- Needs to keep all training data around



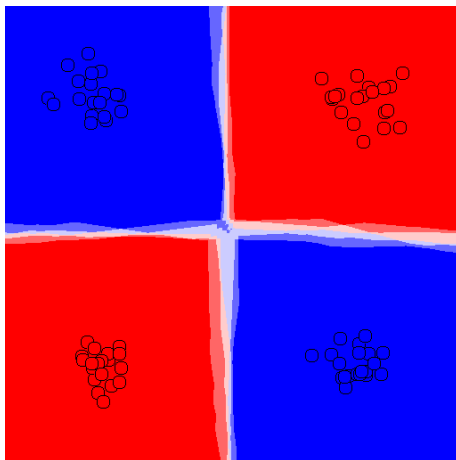
# kNN Example - Simple



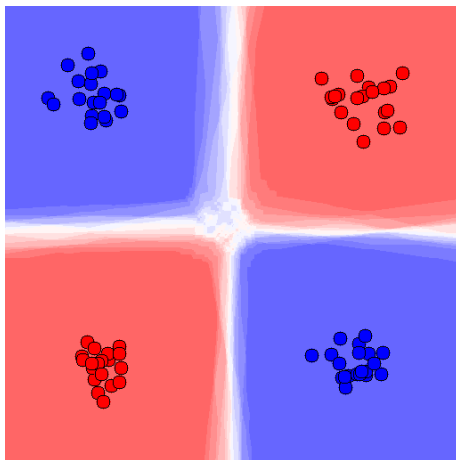
# kNN Example - Simple - kNN $K=1$



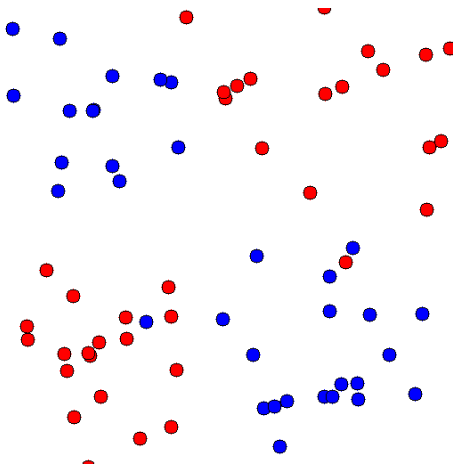
# kNN Example - Simple - kNN $K=5$



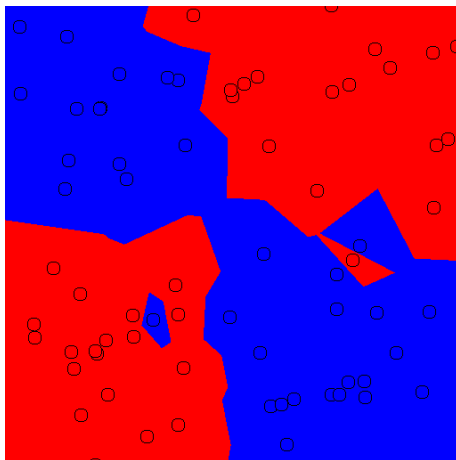
# kNN Example - Simple - kNN $K=25$



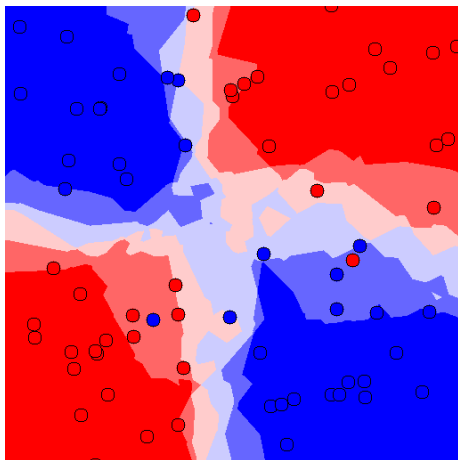
# kNN Example - Hard



# kNN Example - Hard - kNN $K=1$

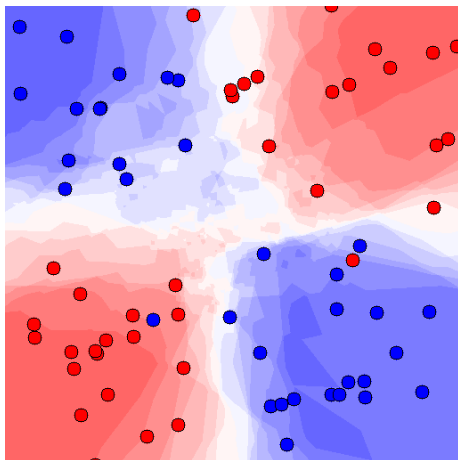


# kNN Example - Hard - kNN K=5

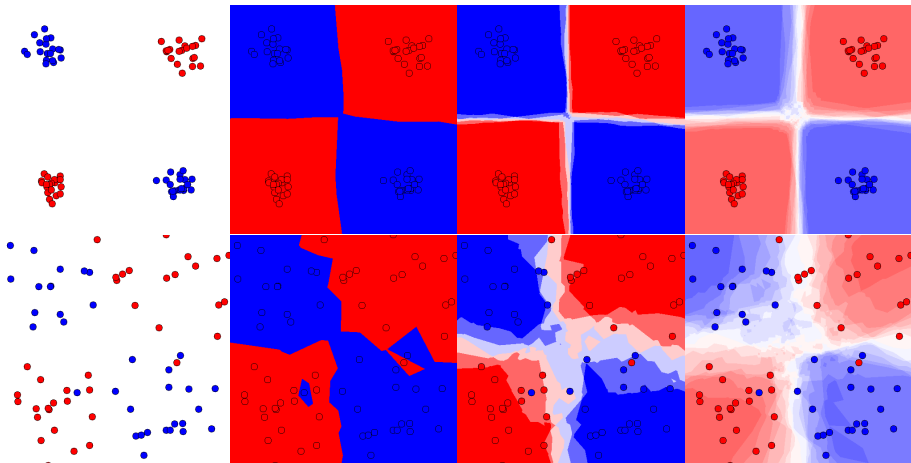




# kNN Example - Hard - kNN $K=25$

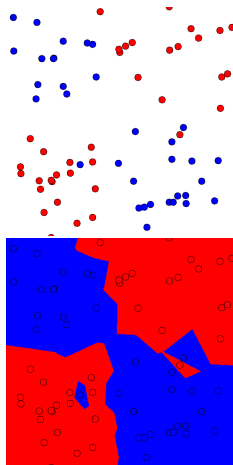


# kNN Example



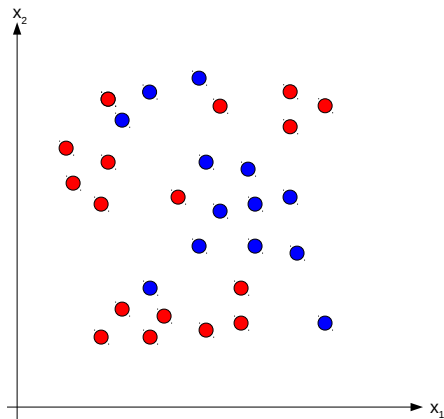
# Model Complexity vs Overfitting

- With sufficient model complexity, it is often easy to get ZERO training error
- Generalization is what matters!
- **Test on data not used during training**
  - Disjoint train and test set
  - Non-overlapping samples if spatial features are used
  - Semi-manual parameter tuning (grid-search, etc.) needs third independent data set



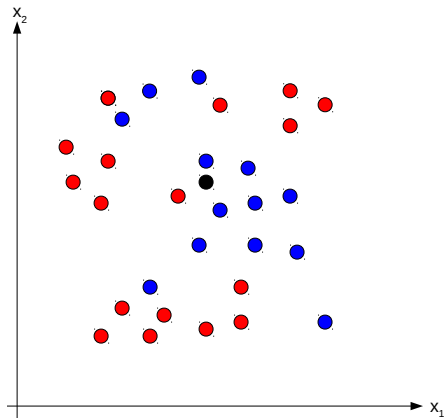
# From kNN to Search Trees

- Data samples  $\mathbf{x}$ 
  - Pixel information, image patch, feature vector, etc.
  - Often  $\mathbf{x} \in \mathbb{R}^n$
- Classification:  
 $\Rightarrow$  Estimate class label
- Training data: Values of target variable given e.g. class label



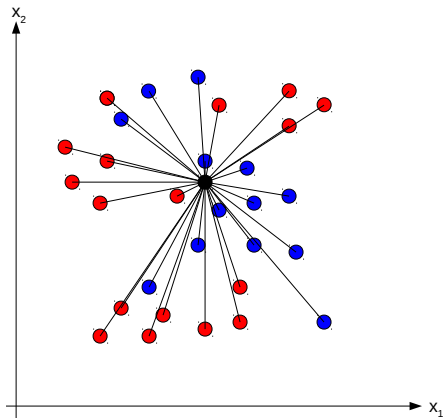
# From kNN to Search Trees

- Task: Given training data, estimate label of query sample



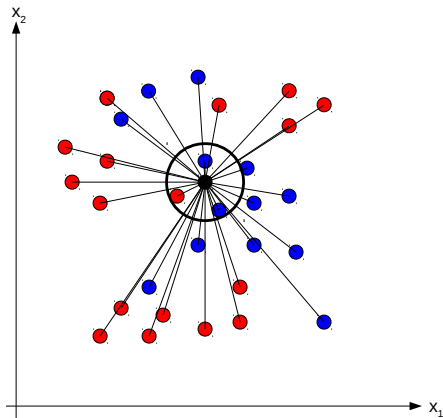
# From kNN to Search Trees

- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
  - Compute distance to **all** samples



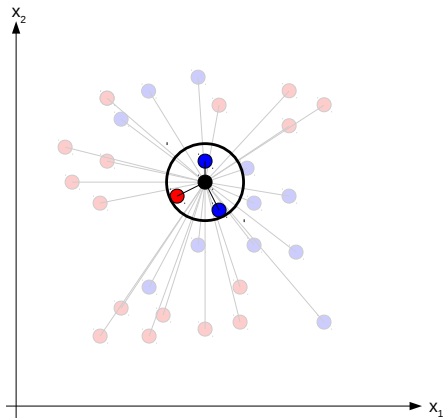
# From kNN to Search Trees

- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
  - Compute distance to **all** samples
  - Select samples within window of given size (Parzen)



# From kNN to Search Trees

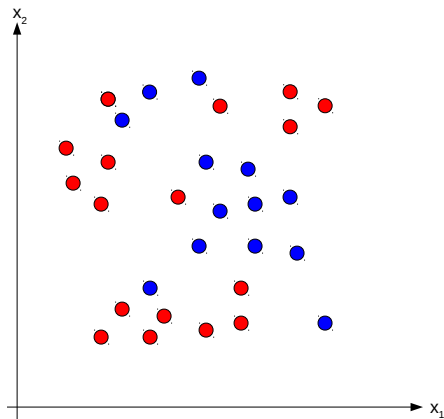
- Task: Given training data, estimate label of query sample
- kNN/Parzen Window:
  - Compute distance to **all** samples
  - Select samples within window of given size (Parzen)
  - Use these samples to estimate target variable, e.g. class label
- Problem: Computationally expensive (exhaustive search)





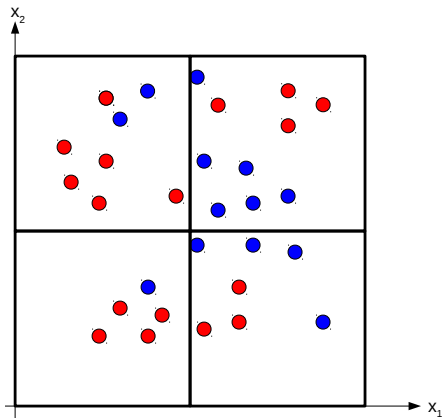
# From kNN to Search Trees

- Search trees  
→ Quad/Octree, KD tree, etc.



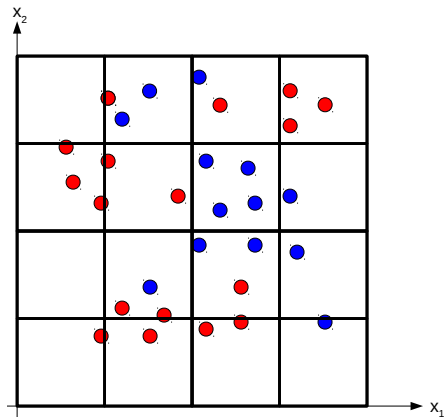
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells



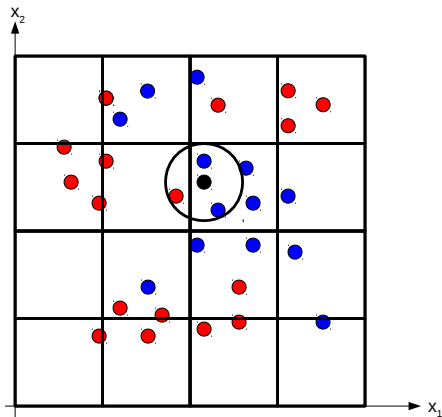
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells



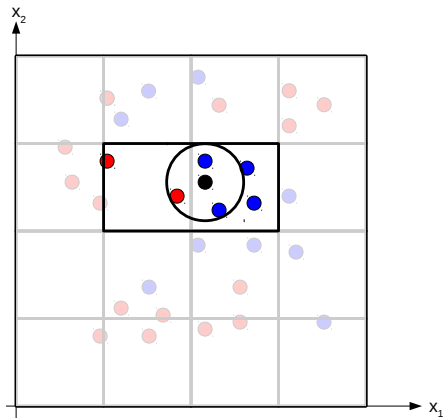
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells
    - Given a query, find relevant cells



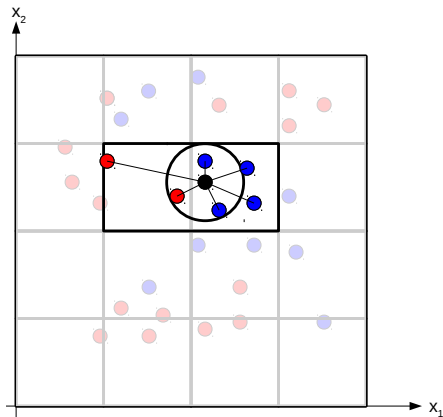
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells
    - Given a query, find relevant cells



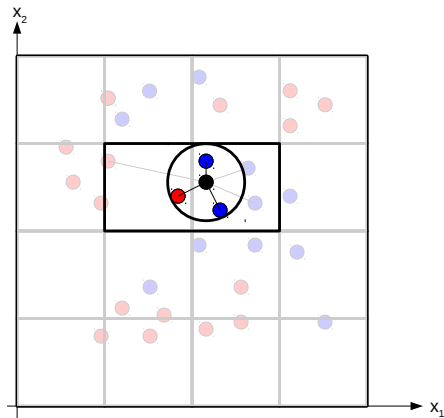
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
  - Divide space recursively into cells
  - Given a query, find relevant cells
  - Perform exhaustive search in these cells ONLY



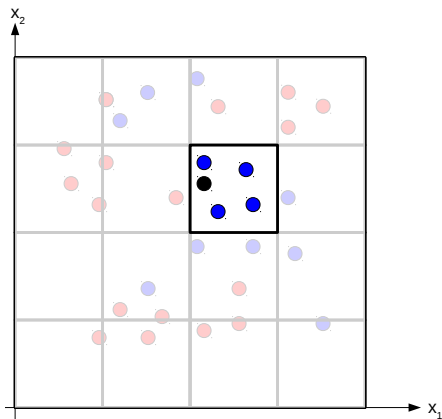
# From kNN to Search Trees

- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells
    - Given a query, find relevant cells
    - Perform exhaustive search in these cells ONLY
- Exact search: Leads to equivalent results



# From kNN to Search Trees

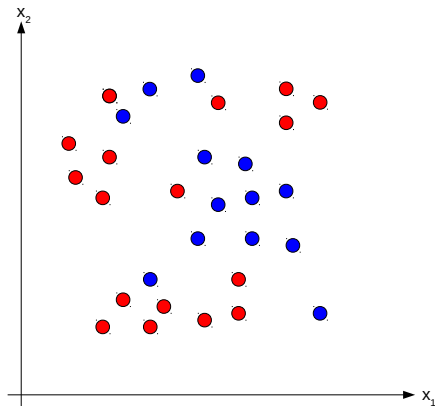
- Search trees
  - Quad/Octree, KD tree, etc.
    - Divide space recursively into cells
    - Given a query, find relevant cells
    - Perform exhaustive search in these cells ONLY
- Exact search: Leads to equivalent results
- Approximation: Use samples within query cell directly





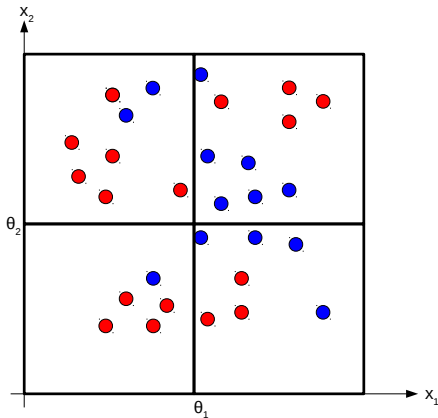
# From Search Trees to (Random) Decision Trees

- Cell construction



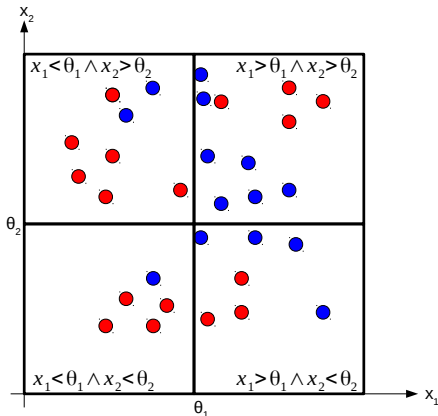
# From Search Trees to (Random) Decision Trees

- Cell construction



# From Search Trees to (Random) Decision Trees

- Cell construction
  - Simple threshold operation
  - Different threshold definitions (e.g. equi-sized cells, threshold as data median) lead to different search tree variants (e.g. quad-tree, k-D tree).

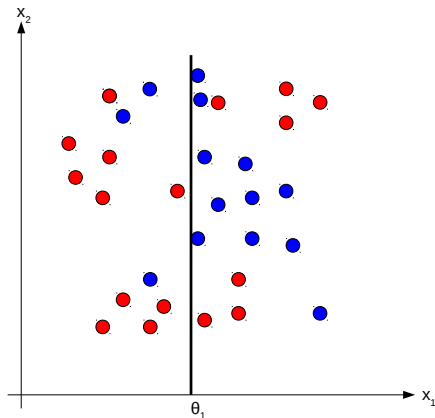
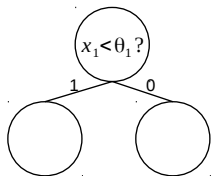


# From Search Trees to (Random) Decision Trees

- Cell construction  
→ Simple threshold operation

- Decision stump:

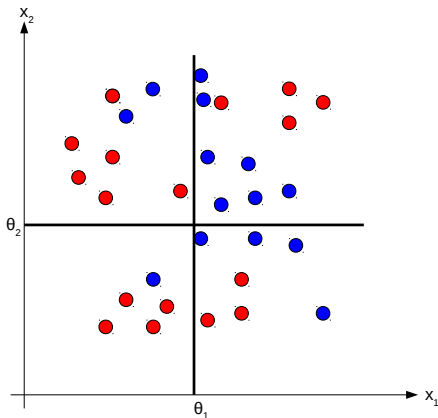
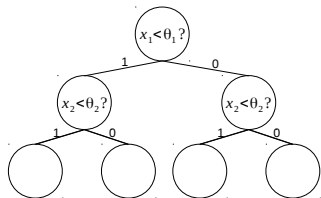
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$



# From Search Trees to (Random) Decision Trees

- Cell construction  
→ Simple threshold operation
- Decision stump:

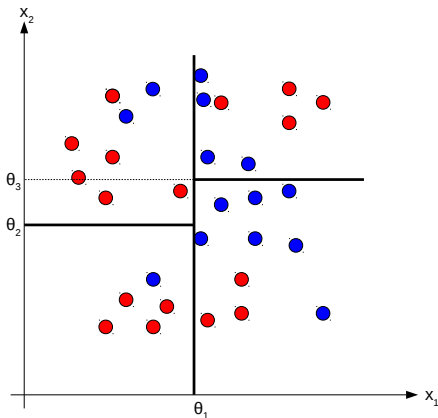
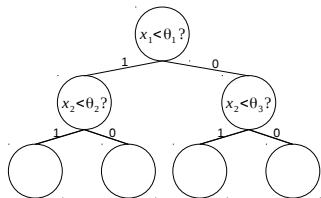
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$



# From Search Trees to (Random) Decision Trees

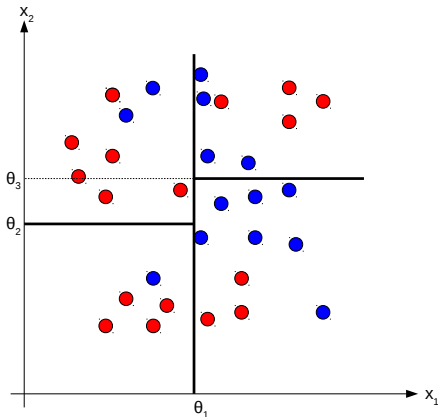
- Cell construction  
→ Simple threshold operation
- Decision stump:

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

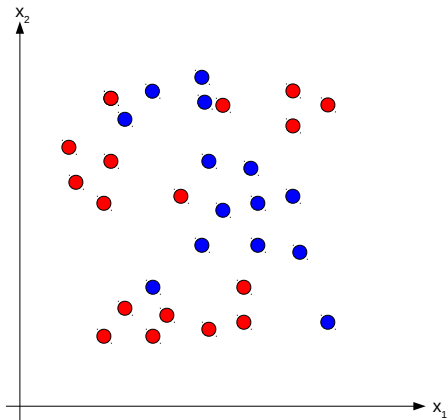


# From Search Trees to (Random) Decision Trees

- Cell construction  
→ Simple threshold operation
- Decision stump:  
$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$
- When to stop? Minimal resolution reached, purity, ...
- How to select split points?  
Randomly, optimized selection

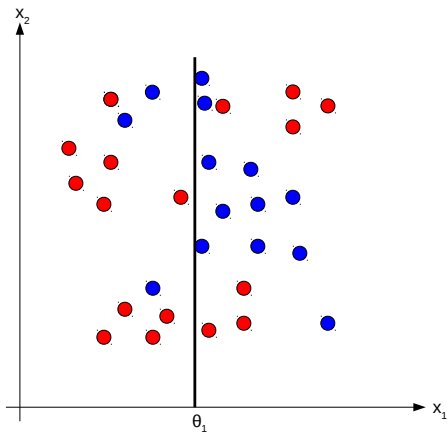
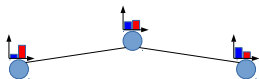


# From Search Trees to (Random) Decision Trees

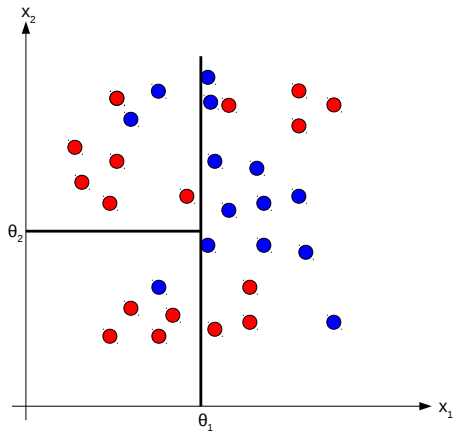
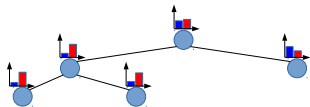




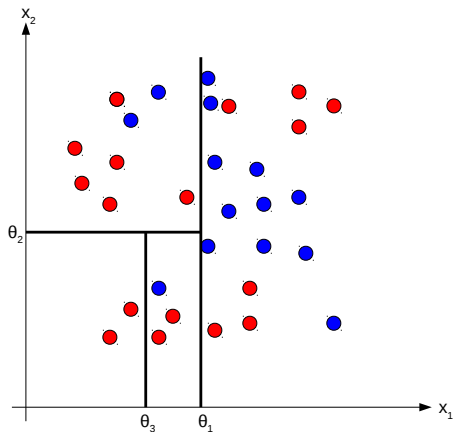
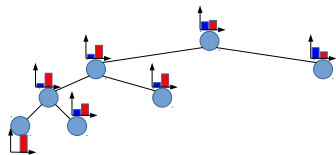
# From Search Trees to (Random) Decision Trees



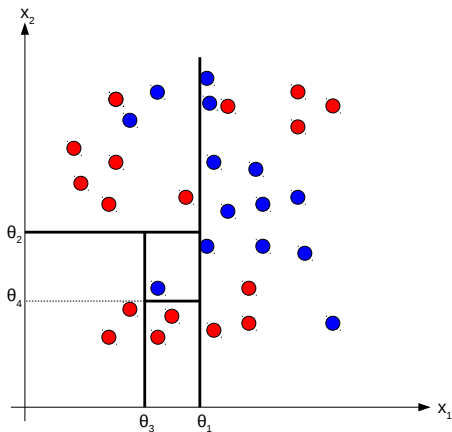
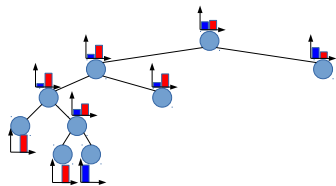
# From Search Trees to (Random) Decision Trees



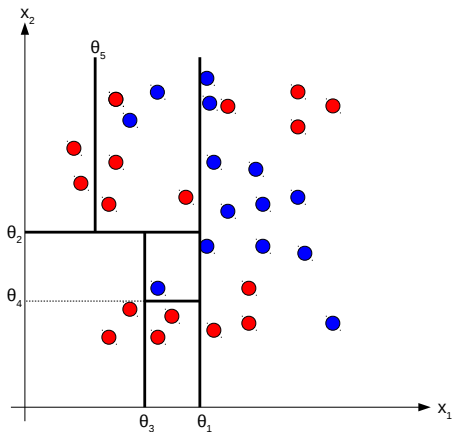
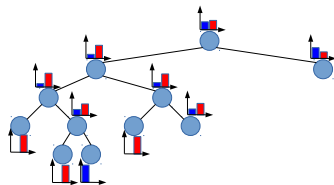
# From Search Trees to (Random) Decision Trees



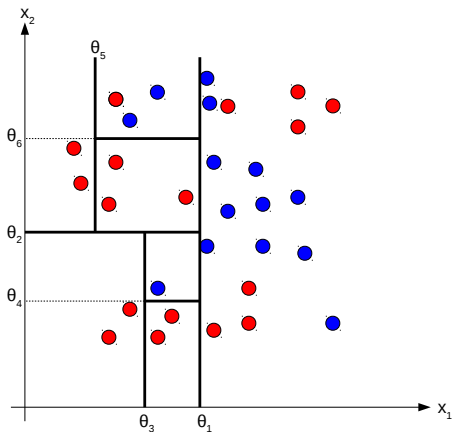
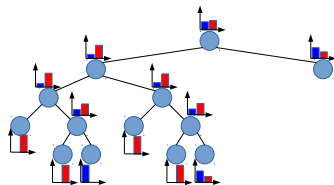
# From Search Trees to (Random) Decision Trees



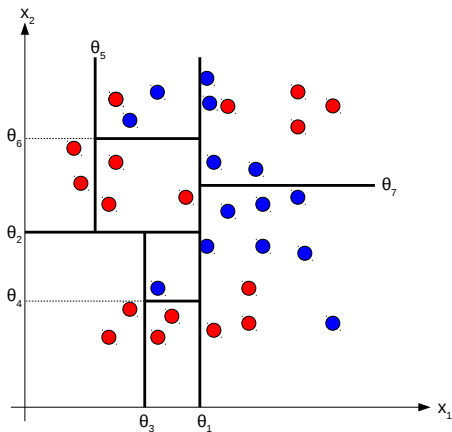
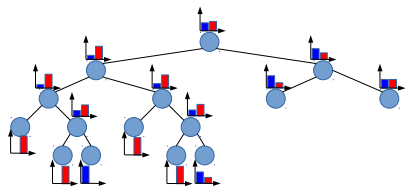
# From Search Trees to (Random) Decision Trees



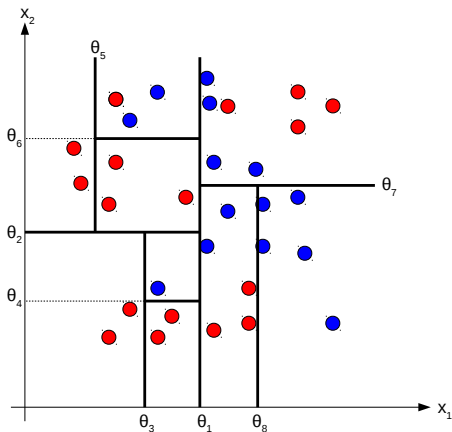
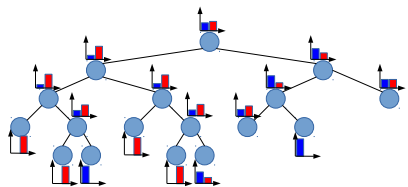
# From Search Trees to (Random) Decision Trees



# From Search Trees to (Random) Decision Trees

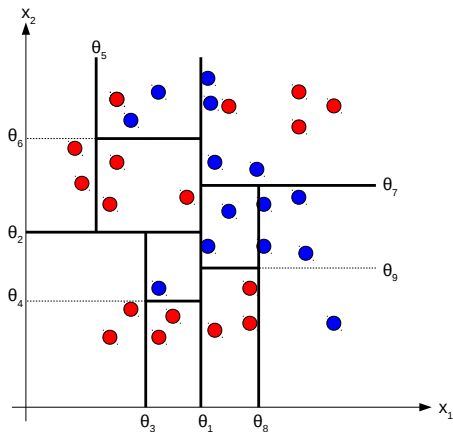
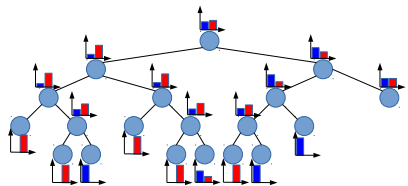


# From Search Trees to (Random) Decision Trees

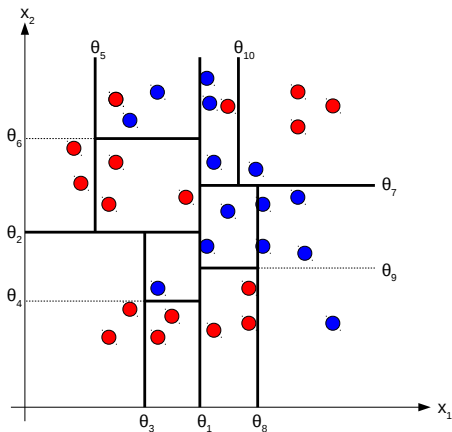
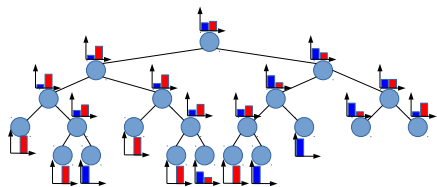




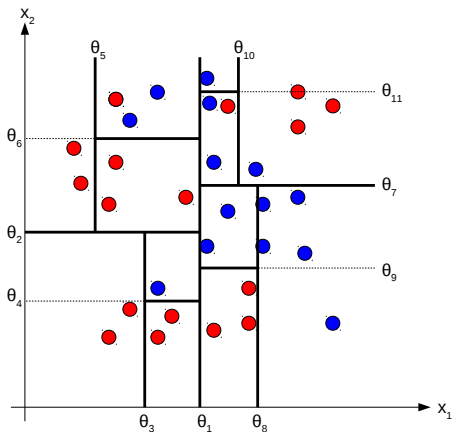
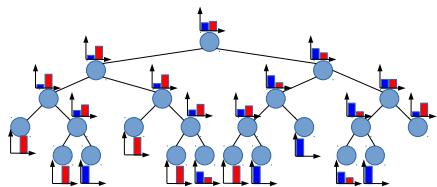
# From Search Trees to (Random) Decision Trees



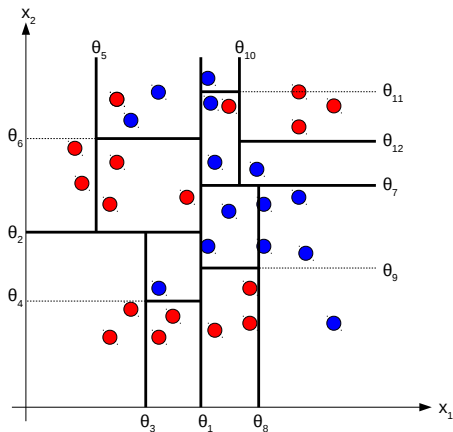
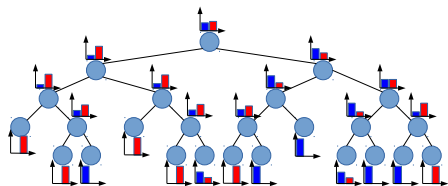
# From Search Trees to (Random) Decision Trees



# From Search Trees to (Random) Decision Trees

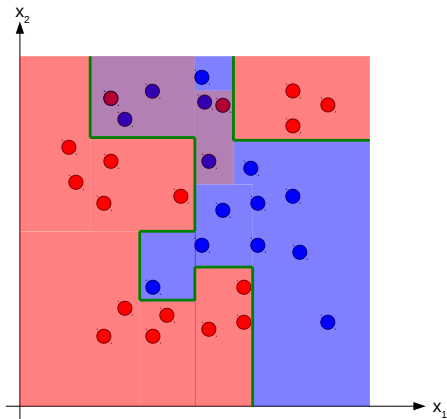


# From Search Trees to (Random) Decision Trees



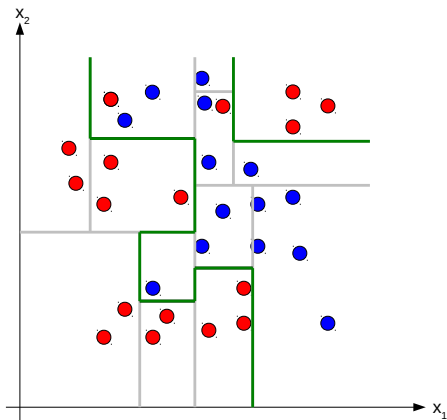
# From Search Trees to (Random) Decision Trees

- Local estimate of the target variable (e.g. class posterior) is assigned to cells



# From Search Trees to (Random) Decision Trees

- Local estimate of the target variable (e.g. class posterior) is assigned to cells
- Results in highly non-linear, even non-connected (but piece-wise constant) decision boundaries



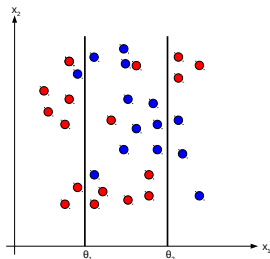
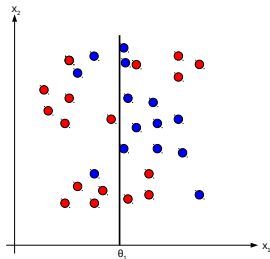
# From Search Trees to (Random) Decision Trees

Other node tests are possible:

- Axis-aligned:

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } x_1 < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < x_1 < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$



# From Search Trees to (Random) Decision Trees

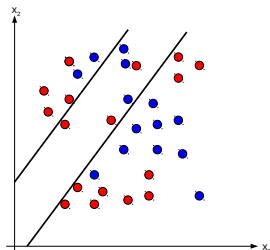
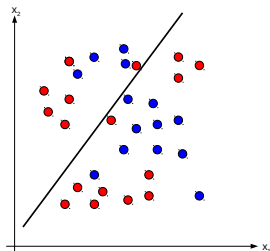
Other node tests are possible:

- Axis-aligned
- Linear:

$$\tilde{\mathbf{x}} = [\mathbf{x}, 1] \in \mathbb{R}^{d+1}, \psi \in \mathbb{R}^{d+1}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \psi^T \tilde{\mathbf{x}} < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < \psi^T \tilde{\mathbf{x}} < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$





# From Search Trees to (Random) Decision Trees

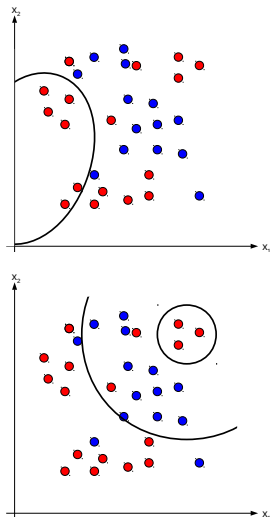
Other node tests are possible:

- Axis-aligned
- Linear
- Conic section:

$$\tilde{\mathbf{x}} = [\mathbf{x}, 1] \in \mathbb{R}^{d+1}, \psi \in \mathbb{R}^{(d+1) \times (d+1)}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \tilde{\mathbf{x}}^T \psi \tilde{\mathbf{x}} < \theta_1 \\ 1 & \text{otherwise.} \end{cases}$$

$$t(\mathbf{x}) = \begin{cases} 0 & \text{if } \theta_1 < \tilde{\mathbf{x}}^T \psi \tilde{\mathbf{x}} < \theta_2 \\ 1 & \text{otherwise.} \end{cases}$$



# From (Random) Decision Trees to Random Forests

## Advantages

- Can deal with very heterogeneous data
  - Different, data-specific types of node tests
- Not prone to the curse of dimensionality
  - Each node only works on a very limited set of dimensions
- Very efficient
  - Each sample passes maximal  $H$  nodes ( $H =$  maximal height)
- Easy to implement
  - Binary trees are one of the most basic data structures
- Easy to interpret
  - Path through tree is a connected set of decision rules
- Well understood
  - Theoretical and practical implications of design decisions have been researched for more than 4 decades

# From (Random) Decision Trees to Random Forests

## Disadvantages

- Optimized by greedy algorithms
  - A chain of individually optimal decisions, might not lead to an overall optimum
- The optimal solution (i.e. decision boundary) might not be part of the model class (e.g. piece-wise linear and axis-aligned functions)
- Prone to overfitting
- Model capacity depends on amount of data
  - Few samples lead to small trees: Only few questions can be asked.
  - Many samples (might) lead to very high trees: Long processing times, large memory footprint.

# From (Random) Decision Trees to Random Forests

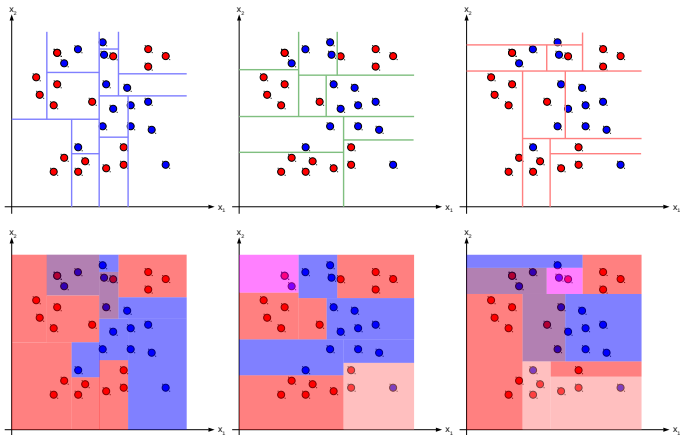
## Disadvantages

- Optimized by greedy algorithms
  - A chain of individually optimal decisions, might not lead to an overall optimum
- The optimal solution (i.e. decision boundary) might not be part of the model class (e.g. piece-wise linear and axis-aligned functions)
- Prone to overfitting
- Model capacity depends on amount of data
  - Few samples lead to small trees: Only few questions can be asked.
  - Many samples (might) lead to very high trees: Long processing times, large memory footprint.

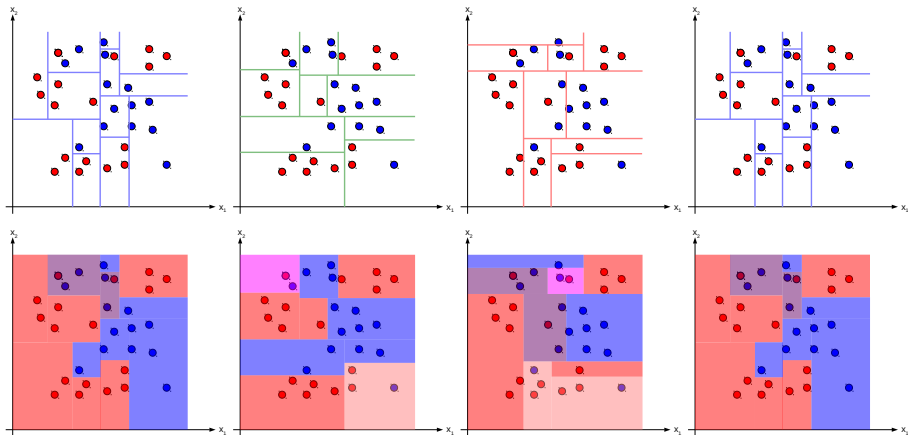
## How to

- **keep (most) of the advantages**
- **getting rid of (most) disadvantages?**

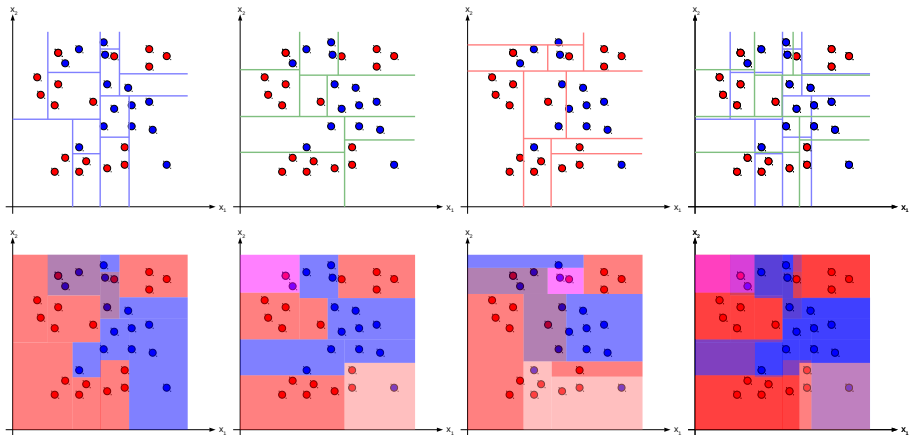
# From (Random) Decision Trees to Random Forests



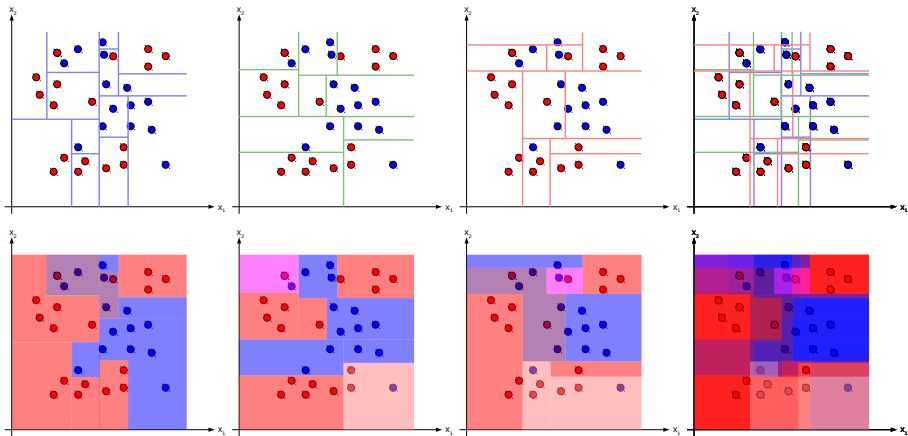
# From (Random) Decision Trees to Random Forests



# From (Random) Decision Trees to Random Forests



# From (Random) Decision Trees to Random Forests





# Random Forests

- Many (suboptimal) baselearners, i.e. decision trees
- Fusion of the individual output
- Minimization of the risk to use wrong model
- Extension of the model space
- Decreased dependence on initialization
- One name to rule them all
  - Bagged Decision Trees
  - Randomized Trees
  - Decision Forests
  - ERT, PERT, Rotation Forests, Hough Forests, Semantic Texton Forests, ...

# Random Forests - Randomization through Bagging

Given: Training set  $D \subset \mathbb{D}$  with  $|D| = N$  samples.

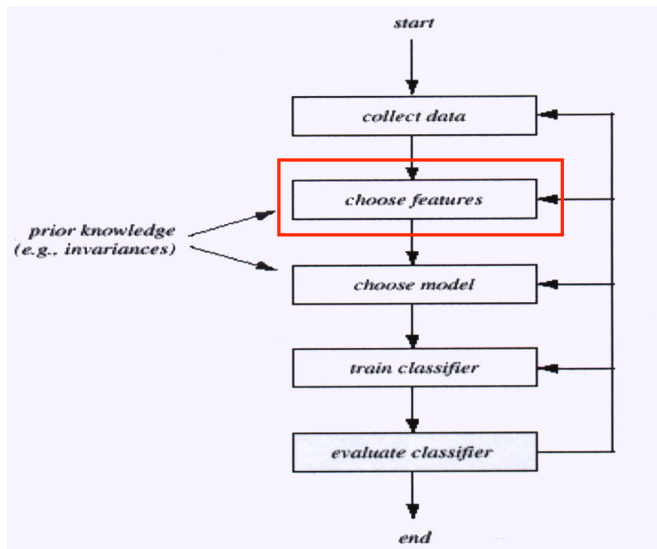
Bagging (**B**ootstrap **a**ggregating):

1. Randomly sample  $M$  data sets  $D_m$  with replacement ( $|D_m| = N$ ).
2. Train  $M$  models where  $m$ -th model has only access to  $m$ -th dataset.
3. Average all models.
  - Meta learning technique
  - Works if small change in input data leads to large model variation
  - Reduces variance (of final model), avoids overfitting.
  - Leads to diverse decision trees, even if all other parameters are fixed

# Overview

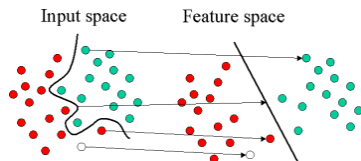
1. Classification based on Features
  - Decision Boundary
  - Linear Decision Boundary
  - Non-linear Decision Boundary
  - Random Forest (RF)
2. Feature extraction
3. Multi-Layer Perceptron (MLP)
4. ConvNets
  - Convolution
  - Auto Encoder
  - Frameworks

# Why do we need feature extraction?



# Motivation

- **Main motivation:** get out most of the data
- For classification task: find a space where samples from different classes are well separable

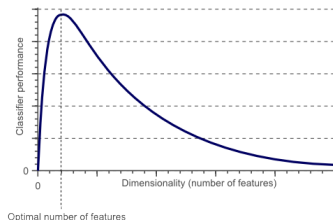
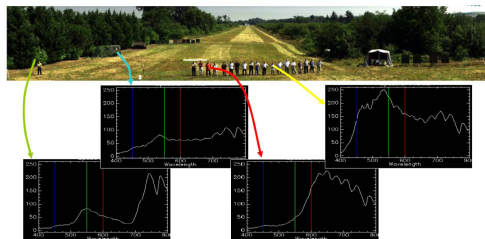


## Objectives:

- Reduce computational load of the classifier
- Increase data consistency
- Incorporate different sources of information into a feature vector: spectral, spatial, multisource, ...

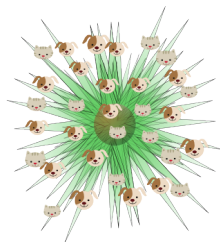
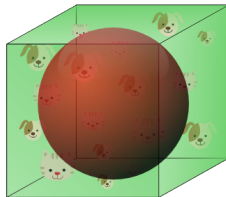
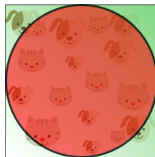
# Motivation - Curse of dimensionality

- Too few features do not allow to discriminate between classes
  - In the color image, both trees and a truck are green
- As the dimensionality of the feature space increases, the classifier's performance increases until the optimal number of features is reached
- Further increasing the dimensionality without increasing the number of training samples yields a performance decrease



# Motivation - Curse of dimensionality

- As the dimensionality increases:
  - The volume of the hypersphere tends to zero
  - A larger percentage of the training data resides in the corners of the feature space
  - Distance measures start losing their effectiveness
  - Gaussian likelihoods become flat and heavy tailed distributions



# How to reduce data dimensions?

## Principal component analysis

Convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables, called **principal components**

## Discriminant analysis

Find the best set of vectors which best separates the patterns



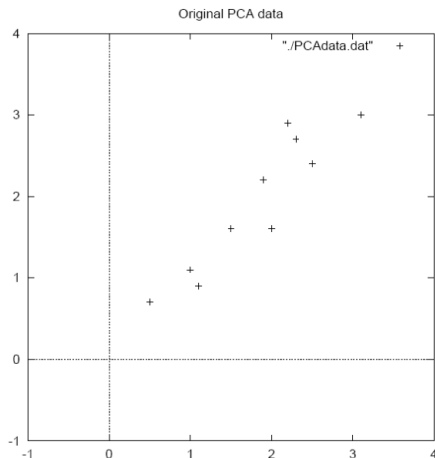
# Principal component analysis

- **Goal:** represent data in a space that best describes the variation in a sum-squared error sense
- Projection onto eigenvectors that correspond to the first few largest eigenvalues of the covariance matrix
  - $d$ -dimensional data are represented in a lower-dimensional space
  - Reduces the space and time complexities
- Intuitive introduction: <http://www.youtube.com/watch?v=BfTMmoDFXyE&feature=related>

# Principal component analysis

- **Step 1:** Get some data

	$x$	$y$
Data =	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9



# Principal component analysis

- **Step 2:** Subtract the mean
  - From each of the data dimensions (from  $x$ - and  $y$ -dimension)

	$x$	$y$
	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
Data =	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9




	$x$	$y$
	.69	.49
	-1.31	-1.21
	.39	.99
	.09	.29
DataAdjust =	1.29	1.09
	.49	.79
	.19	-.31
	-.81	-.81
	-.31	-.31
	-.71	-1.01

# Principal component analysis

- **Step 3:** Calculate the covariance matrix

	<i>x</i>	<i>y</i>
	2.5	2.4
	0.5	0.7
	2.2	2.9
	1.9	2.2
Data =	3.1	3.0
	2.3	2.7
	2	1.6
	1	1.1
	1.5	1.6
	1.1	0.9



$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$

# Principal component analysis

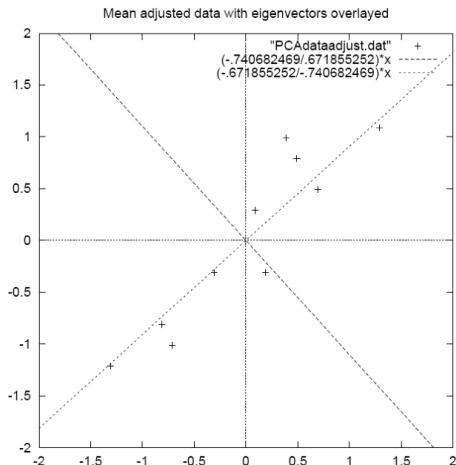
- **Step 4:** Calculate the unit eigenvectors and eigenvalues of the covariance matrix

$$cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$$



$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



# Principal component analysis

- The 1st eigenvector (principle component) shows how data in two dimensions are related along the eigenvector line
- The 2nd eigenvector shows that all the points are off to the side of the main line by some amount
- Eigenvectors are lines that characterize the data
- The next steps: transforming the data so that it is expressed in terms of these lines

# Principal component analysis

- **Step 5:** Choose components and form a feature vector
  - Order eigenvectors by eigenvalues
    - This gives the components in order of significance
    - You can decide to ignore the components of lesser significance  $\Rightarrow$  final data will have less dimensions ( $p < d$ )
  - Form a feature vector (matrix of vectors):

$$\text{FeatureVector} = (\text{eig}_1 \text{ eig}_2 \text{ eig}_3)$$

- For our example, two feature vectors are possible:

$$\text{eigenvalues} = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$\text{eigenvectors} = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



$$\begin{pmatrix} -.677873399 & -.735178656 \\ -.735178656 & .677873399 \end{pmatrix}$$

or

$$\begin{pmatrix} -.677873399 \\ -.735178656 \end{pmatrix}$$

# Principal component analysis

- **Step 6:** Derive the new dataset:

$$FinalData = FeatureVector^T \times RowDataAdjust$$

where *RowDataAdjust* is the mean-adjusted data transposed

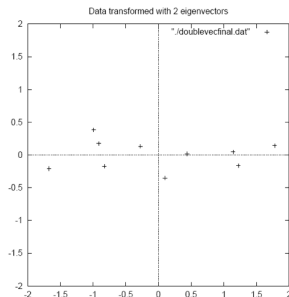
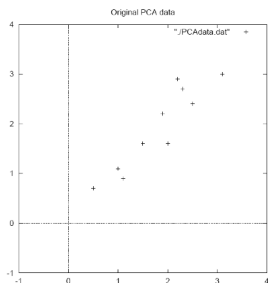
- It will give us the original data solely in terms of the vectors we chose



## Principal component analysis

$x$	$y$
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

$x$	$y$
-0.827970186	-0.175115307
1.77758033	.142857227
-0.992197494	.384374989
-0.274210416	.130417207
-1.67580142	-0.209498461
-0.912949103	.175282444
.0991094375	-0.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-0.162675287

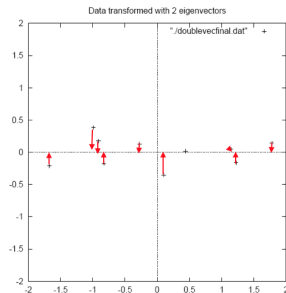


# Principal component analysis (PCA)

- If only one eigenvector was kept, the transformed data will have only one dimension

Transformed Data =

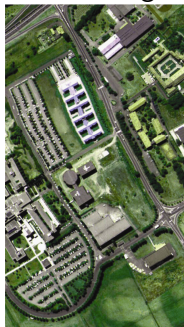
$x$	$y$
-0.827970186	-0.175115307
1.77758033	.142857227
-0.992197494	.384374989
-0.274210416	.130417207
-1.67580142	-0.209498461
-0.912949103	.175282444
.0991094375	-0.349824698
1.14457216	.0464172582
.438046137	.0177646297
1.22382056	-0.162675287



# Example of PCA for hyperspectral image analysis

- Principal component analysis in the spectral space
  - Principal components (PCs) 1-3 contain 97% of information from original 103 channels

Color image



PC1



PC2



PC3

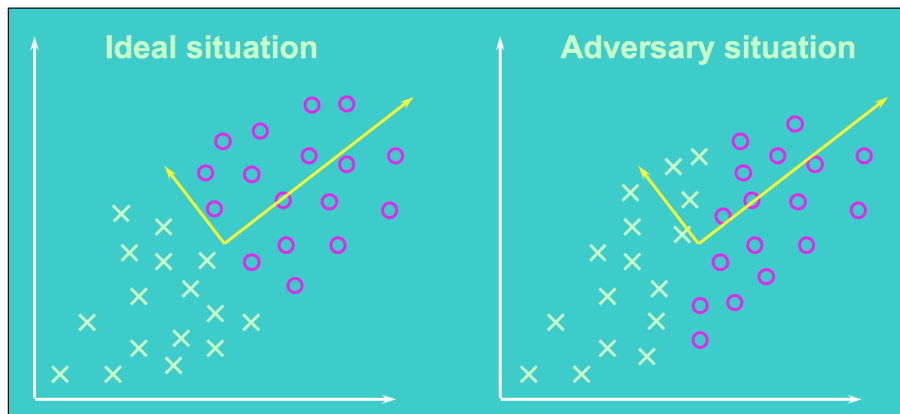


PC4



# Principal component analysis

- Projection onto eigenvectors that correspond to the first few largest eigenvalues of the covariance matrix

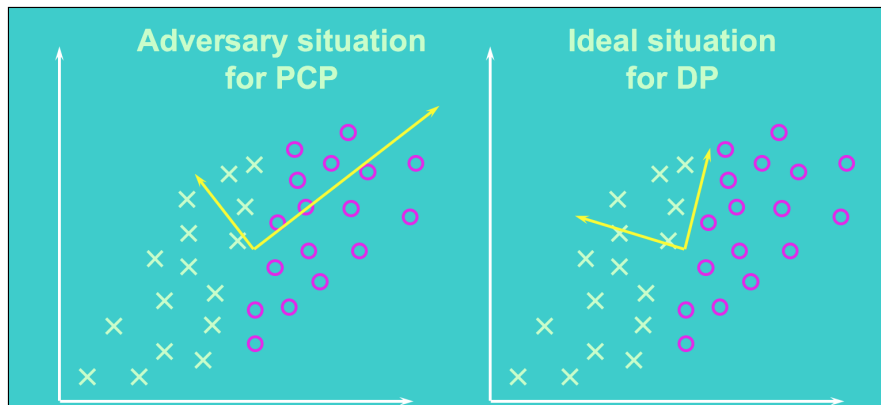


# Discriminant analysis

- PCA seeks directions that are efficient for **representation**
  - *Unsupervised technique*
- Discriminant analysis seeks directions that are efficient for **discrimination**
  - *Supervised technique*

# Discriminant analysis

- Projection onto directions that can best separate data of different classes



## Discriminant analysis

- Theory of Fisher linear discriminant: [http://www.csd.uwo.ca/~olga/Courses//CS434a\\_541a//Lecture8.pdf](http://www.csd.uwo.ca/~olga/Courses//CS434a_541a//Lecture8.pdf)
- Project on line in the direction  $v$  which maximizes:

**want projected means are far from each other**

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

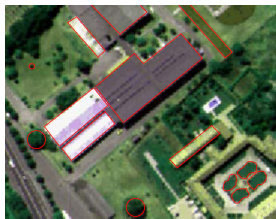
**want scatter in class 1 is as small as possible, i.e. samples of class 1 cluster around the projected mean  $\tilde{\mu}_1$**

**want scatter in class 2 is as small as possible, i.e. samples of class 2 cluster around the projected mean  $\tilde{\mu}_2$**

- Main drawback: in most real-life cases, projection to even the best line results in unseparable projected samples

# How to include spatial information for image classification?

- By simply looking at a grey pixel, we cannot say if it belongs to a *building* or a *road*
- We guess a category by considering spatial/contextual information
- How can a classifier consider this rich source of information?





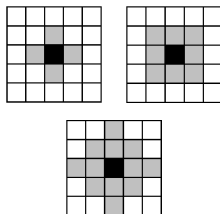
# Approaches to extract spatial info

## 1. Closest fixed neighborhoods

- Markov Random Field [Pony00, Jackson02, Farag05]
- Contextual features [Camps-Valls06]
  - Spectral content +
  - Spatial content (e.g. mean or standard deviation per spectral band)

+ Simplicity

- Imprecision at the border of regions



# Approaches to extract spatial info

## 1. Closest fixed neighborhoods

- Markov Random Field [Pony00, Jackson02, Farag05]
- Contextual features [Camps-Valls06]
  - Spectral content +
  - Spatial content (e.g. mean or standard deviation per spectral band)

+ Simplicity

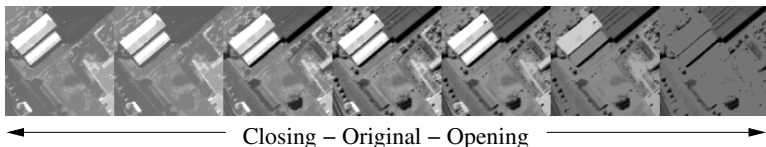
- Imprecision at the border of regions



# Approaches to extract spatial info

## 2. Morphological and area filtering

- Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
  - Self-complementary area filtering [Fauvel07]
  - Attribute profiles [Ghamisi15, Cavallaro17]
- + Neighborhoods are adapted to the structures
- + Non-linear operators  $\Rightarrow$  do not blur the edges as convolutions do
- Neighborhoods are scale dependent

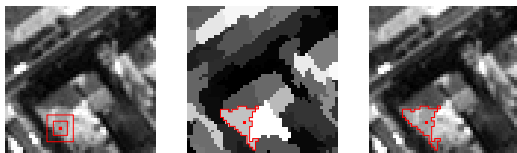


Course on mathematical morphology:  
<http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm>

# Approaches to extract spatial info

## 2. Morphological and area filtering

- Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
  - Self-complementary area filtering [Fauvel07]
  - Attribute profiles [Ghamisi15, Cavallaro17]
- + Neighborhoods are adapted to the structures
- + Non-linear operators  $\Rightarrow$  do not blur the edges as convolutions do
- Neighborhoods are scale dependent



---

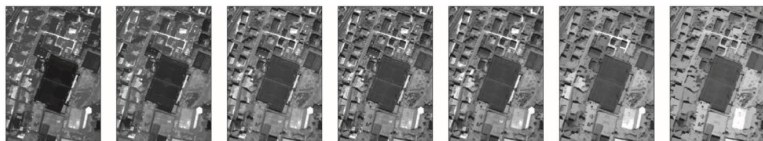
Course on mathematical morphology:

<http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm>

# Approaches to extract spatial info

## 2. Morphological and area filtering

- Morphological profiles [Pesaresi01, Dell'Acqua04, Benediktsson05]
  - Self-complementary area filtering [Fauvel07]
  - Attribute profiles [Ghamisi15, Cavallaro17]
- + Neighborhoods are adapted to the structures
- + Non-linear operators  $\Rightarrow$  do not blur the edges as convolutions do
- Neighborhoods are scale dependent



---

Course on mathematical morphology:

<http://www-sop.inria.fr/members/Yuliya.Tarabalka/teaching.htm>

# Approaches to extract spatial info

## 3. Superpixels derived from segmentation

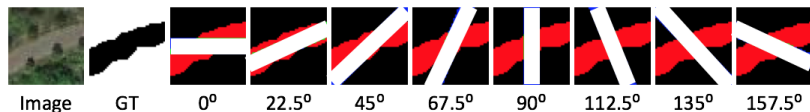
- Extraction and Classification of Homogeneous Objects [Kettig76]
- ...
- Multiscale segmentation, then features are derived from the regions [Linden07, Huang09]
  - + Flexible
  - Computationally demanding
  - Difficult to scale/parallelize



# Approaches to extract spatial info

## 4. Features handcrafted for a particular application

- Example 1: Line templates with different orientations for road detection [Jeong15]
  - Example 2: Rectangular templates for building detection [Garcin01]
- + Can model complex shape
  - Lack of genericity
  - Computationally demanding



# Approaches to extract spatial info

## 4. Features handcrafted for a particular application

- Example 1: Line templates with different orientations for road detection [Jeong15]
- Example 2: Rectangular templates for building detection [Garcin01]
  - + Can model complex shape
  - Lack of genericity
  - Computationally demanding

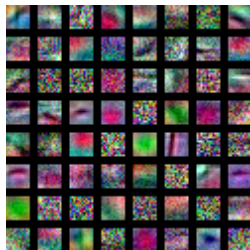




## Modern trend & Conclusions

### Deep learning:

- Automatically learn features if a lot of training data are available



### Advice:

- If for the considered application it is easy to hand-craft class-separable features, no need to learn them
- If it is not easy to discriminate between categories, learning features often helps

# Challenges in classification today?

- Increasing amount & openness of data
- Intra-class variability:



Chicago



Vienna



Austin

- Interest in semantic classes (e.g., *building*, *road*, *lane*)
  - ⇒ Need for high-level contextual reasoning (shape, patterns,...)
  - ⇒ Generalization to different locations

How to face these challenges?

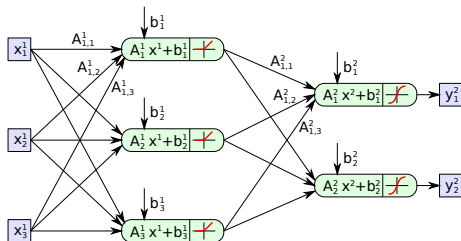


Deep learning



# What is multi-layer perceptron?

- Feed forward neural network
- Neural networks “inspired by biology”
  - But work quite differently
- Core idea: concatenate multiple simple mappings to get one powerful mapping
- Multiple simple steps more powerful than one complex step
- Keep everything (mostly) differentiable
- Train by doing gradient descent on classification error



# Building blocks

Standard Layers:

- Fully connected layer with...
- ... activation function

# Building blocks

## Standard Layers:

- Fully connected layer with...
- ... activation function

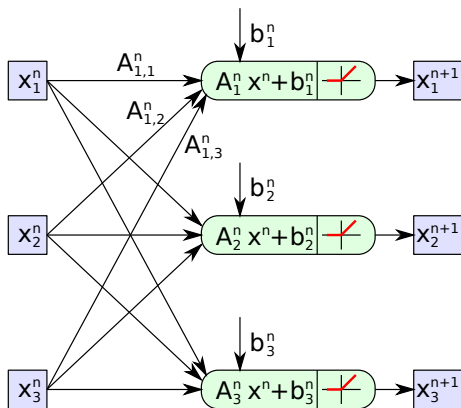
## Special Layers (selection):

- Dropout (for regularization)
- Normalization (Improves training)
- Softmax (Produces nice classification output)

## Fully connected layer

$$\mathbf{x}^{n+1} = \mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \quad (2)$$

- $\mathbf{x}^n$ : Layer input
- $\mathbf{y}^n = \mathbf{x}^{n+1}$ : Layer output
- $\mathbf{A}^n$ : Weights
- $\mathbf{b}^n$ : Bias
- $f(\cdot)$ : Activation function



# Activation functions

$$\mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \quad (3)$$

- Assume  $f(x) = x$
- Layer can assume any linear function (plus offset)



# Activation functions

$$\mathbf{y}^n = f(\mathbf{A}^n \cdot \mathbf{x}^n + \mathbf{b}^n) \quad (3)$$

- Assume  $f(x) = x$
- Layer can assume any linear function (plus offset)
- Stacked layers can't improve that
- Activation function must be non-linear

# Activation functions

Typical choices:

ReLU

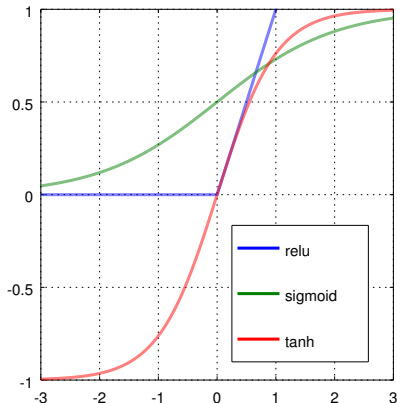
$$f(\mathbf{x}_i)_i = \max(\mathbf{x}_i, 0) \quad (4)$$

Sigmoid / Logistic

$$f(\mathbf{x}_i)_i = \frac{1}{1 + e^{-\mathbf{x}_i}} \quad (5)$$

TanH

$$f(\mathbf{x}_i)_i = \tanh(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i} - e^{-\mathbf{x}_i}}{e^{\mathbf{x}_i} + e^{-\mathbf{x}_i}} \quad (6)$$



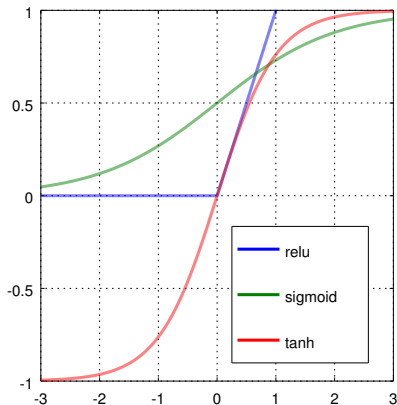
# Activation functions

Typical choices:

## ReLU

$$f(\mathbf{x}_i)_i = \max(\mathbf{x}_i, 0) \quad (7)$$

- ReLU (and variations of it) today the most common choice
- Better for deep networks
  - Derivative of activation function = 1 (in positive direction)
  - No saturation (in positive direction)
  - Gradients propagate better



# Training

- How to find correct model parameters  $\theta$ ?
  - weight values
  - bias values
  - sometimes aux parameters

# Training

- How to find correct model parameters  $\theta$ ?
  - weight values
  - bias values
  - sometimes aux parameters
- Setup/define energy function objective  $E(\theta)$
- Derive analytic gradients  $\frac{\partial E(\theta)}{\partial \theta}$
- Perform gradient descent  $\Delta \theta = -\lambda \cdot \frac{\partial E(\theta)}{\partial \theta}$

# Stochastic gradient descent

- Exact gradient usually not needed or wanted
- Just empirical average over  $N$  samples anyways
- Stochastic Gradient Descent: Split into batches of  $M < N$  samples and update weights after every batch

$$\Delta\theta = -\lambda \cdot \frac{\partial \hat{E}(\theta)}{\partial \theta} = \frac{\partial}{\partial \theta} \sum_{\alpha}^M e(\mathbf{y}^L(\mathbf{x}_{\alpha}, \theta), \hat{\mathbf{y}}_{\alpha}) \quad (8)$$

- Usually small batch sizes (eg. around 128) sufficient

# Parameter update rule

- $\Delta\theta = -\lambda \cdot \frac{\partial \hat{E}(\theta)}{\partial \theta}$  most simple update rule
- Momentum
  - Accumulate “momentum” over time
  - Pick up speed in the valley direction, average out noise
- Adam [Kingma and Ba, 2014]/Adagrad/Adadelata [Zeiler, 2012]
  - Normalize based on average gradient variance in the past

# Parameter initialization

- How to initialize  $\theta$ ?
- Random Gaussian
- Xavier (and some variants) [Glorot and Bengio, 2010]
  - Draw weights randomly
  - Choose variance per layer depending on input/output size
  - Balance variance to keep signal/gradient variance constant



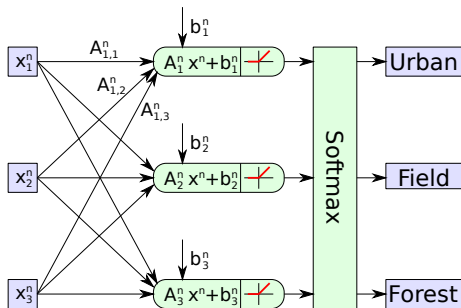
# Special layers

- Softmax
- Normalization
- Dropout

# Softmax

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (9)$$

- Special (last) layer/activation for classification
- Creates vector that sums to one (read probabilities), one element per class
- Usually together with a specific optimization objective: Cross-entropy loss
  - Comparing the predicted probability mass distribution to the ground truth one



## Cross-entropy loss function

Loss function quantifies the misclassification by comparing the target label vectors  $\mathbf{y}^{(i)}$  and the predicted label scores  $\hat{\mathbf{y}}^{(i)}$ , for  $n$  training samples

- **Cross-entropy loss:**

$$L_{CE} = -\frac{1}{n} \sum_{i=1}^n \sum_{k=1}^{|\mathcal{L}|} y_k^{(i)} \log \hat{y}_k^{(i)}$$

- has fast convergence rates when training neural networks
- numerically stable when coupled with softmax normalization

# Dropout

- [Srivastava et al., 2014]
- During training, randomly disable neurons with probability  $p$
- During application, scale output with  $1 - p$
- Prevents co-adaptation
- Fosters redundancy throughout the network
- Reduces overfitting and improves generalization

# Normalization

- Normalization can be important for learning
- Neither signal (forward) nor gradients (backward) must explode/shrink in magnitude
- Input Normalization
  - Normalize input to have zero mean and unit stddev
- Batch Normalization [Ioffe and Szegedy, 2015]
  - Special layer placed at strategic locations
  - Normalize mean and variance of activations across training batch (or accumulate running averages)
  - After learning, becomes fixed scale & offset

# Handling Overfitting

- Dropout

# Handling Overfitting

- Dropout
- Weight regularization
  - Penalize large weight values
  - e.g., add  $\lambda \cdot |\theta|^2$  to optimization objective

# Handling Overfitting

- Dropout
- Weight regularization
  - Penalize large weight values
  - e.g., add  $\lambda \cdot |\theta|^2$  to optimization objective
- Data Augmentation
  - Randomly modify training data
  - Based on what kind of invariances you want to have
    - Resistance to noise: add noise
    - Resistance to brightness/contrast/hue changes: Change those
    - Translation/Rotation (ex. for images)
    - Can also be applied to data before extracting features!



# Increasing depth

- Recent trend goes towards deeper networks
- Networks more powerful, but ...
- ... more difficult to train
  - Gradients collapse/explode/diffuse through the layers
- This is the book to read: Deep Learning [Goodfellow et al., 2016]

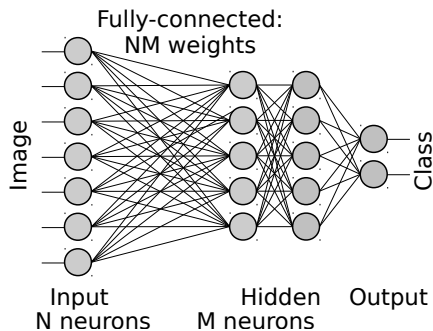
# MLP conclusion

## MLPs

- Provide a mapping from  $\mathcal{X} \rightarrow \mathcal{Y}$ , i.e. from a features space (usually  $\mathcal{X} \equiv \mathbb{R}^n$ ) to a label space  $\mathcal{Y}$
- Are based on concatenation of “simple” functions that depend on parameters (i.e. weights)
- Are optimized by gradient descent (and its modern extensions)
- Work great, BUT:

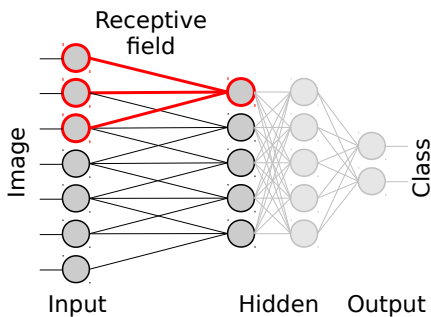


# From fully connected (MLP) to convolution (ConvNet)



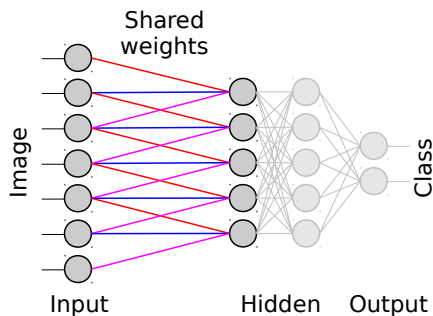
- Multiple layers of units
- All-to-all connection between two adjacent layers
- No lateral connections
- A tremendous amount of parameters in case of images  
→ Untrainable

# From fully connected (MLP) to convolution (ConvNet)



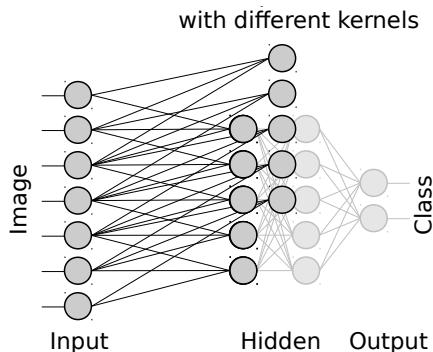
- Set most weights to zero and thus delete most connections and decrease parameters.

# From fully connected (MLP) to convolution (ConvNet)



- Set most weights to zero and thus delete most connections and decrease parameters.
- Use same values for weights of different neurons within a layer.
- The multiplication of the input with identical weights for different neurons corresponds to a convolution.
- The kernel of this convolution is automatically learned.

# From fully connected (MLP) to convolution (ConvNet)



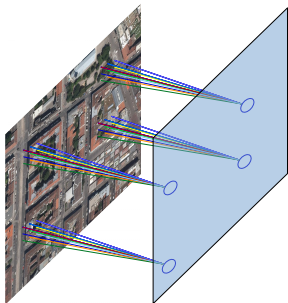
- Set most weights to zero and thus delete most connections and decrease parameters.
- Use same values for weights of different neurons within a layer.
- The multiplication of the input with identical weights for different neurons corresponds to a convolution.
- The kernel of this convolution is automatically learned.
- Use multiple convolutional layers to enable different kernels to be learned.

# Convolutional neural networks (CNNs)

- Input: the image itself
- $\{\text{Convolutional layers} + \text{pooling layers}\}^* + \text{MLP}$
- Jointly learn to extract features & conduct classification

## Convolutional layer

Learned convolution filters  $\rightarrow$  feature maps



Special case of fully connected layer:

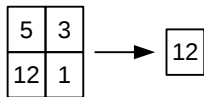
- Only local spatial connections
  - Location invariance
- $\Rightarrow$  Makes sense in image domain (or text, time series,...)

# Convolutional neural networks (CNNs)

## Pooling layers

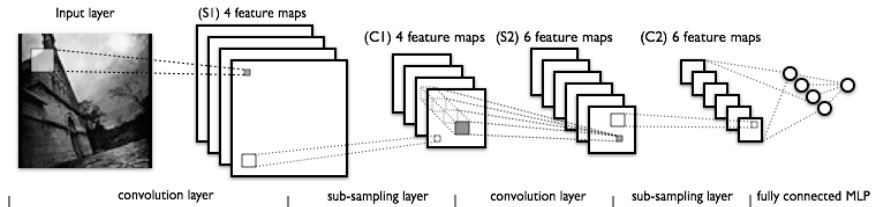
### Subsample feature maps

- Increase *receptive field* 😊
- Downgrade resolution
  - Robustness to spatial variation 😊
  - Not good for *pixelwise* labeling ☹️



Max pooling

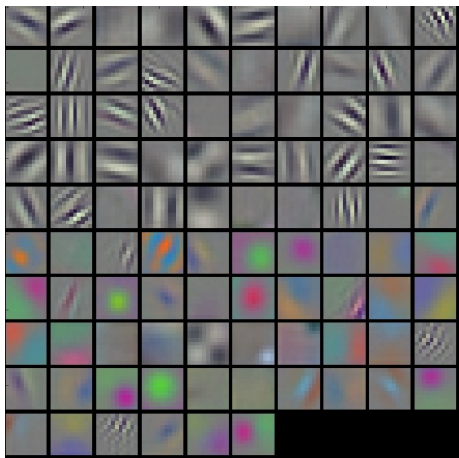
## Overall categorization CNN



Source: deeplearning.net

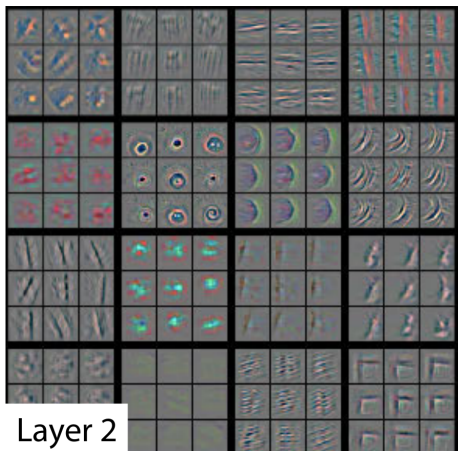


## Example of First Level Filters



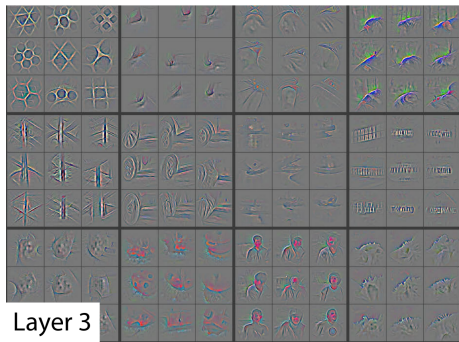
- Learned kernels of first convolutional layer of a ConvNet (AlexNet).
- Correspond mostly to edges and corners of different orientations.
- Note: Grouping is caused by network architecture (two independent streams were used to handle the large amount of data).

## Example of Higher Level Filters



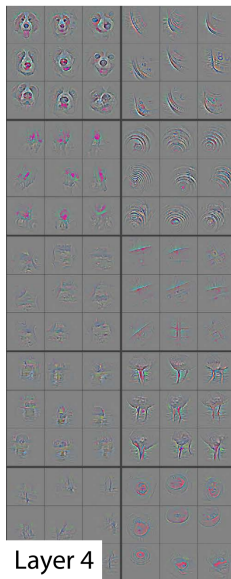
- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

## Example of Higher Level Filters



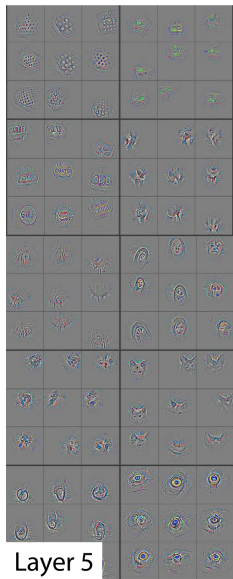
- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

# Example of Higher Level Filters



- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

## Example of Higher Level Filters



- Top nine activations in feature maps
- Projected to pixel space using a deconvolutional network
- Reconstructed patterns that cause high activations
- Note: Images taken from [Zeiler and Fergus, 2013].

# Examples of architectures

## LeNet (1998)

- One of the first successful applications of ConvNets
- Digital digit / character recognition

## VGG Net (2014)

- Simple and deep: Only 3x3 filters and 2x2 pooling
- Stacked conv-layers to increase effective receptive field size
- Used Caffe toolbox
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks

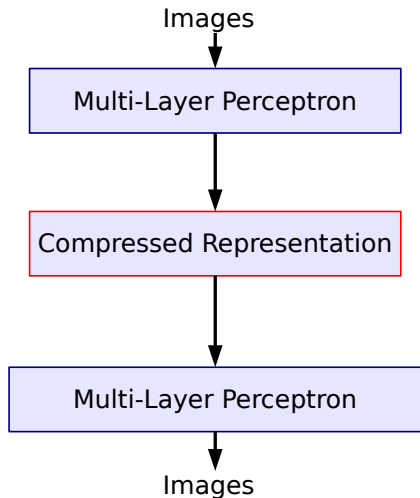
## Microsoft ResNet (2015)

- 152 layers
- Trained on an 8 GPUs for two to three weeks
- 3.6% error on ImageNet LSVRC (AlexNet: 15.4%)

# Common Architectures and Tricks

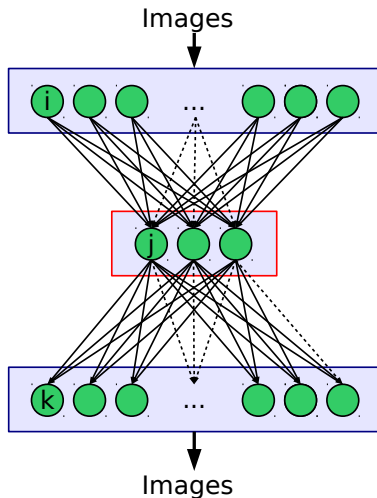
- Designing good architecture somewhat tricky
- Some designs, or parts of designs, exist that work well
- Usually a good idea to look at papers of common architectures
  - Most of the time, at least some intuition or motivation for choice of layers

# (Convolutional) Auto Encoder



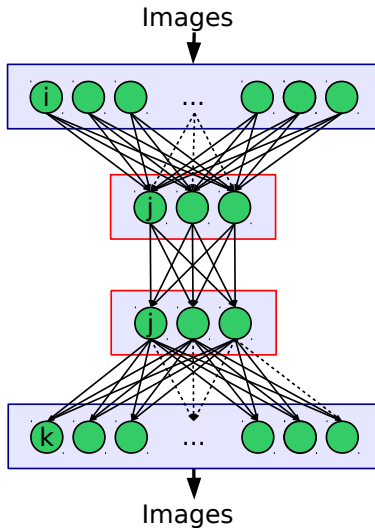


# (Convolutional) Auto Encoder



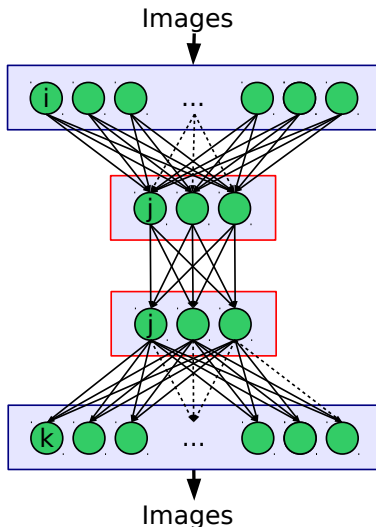
# (Convolutional) Auto Encoder

- Stacked Autoencoder



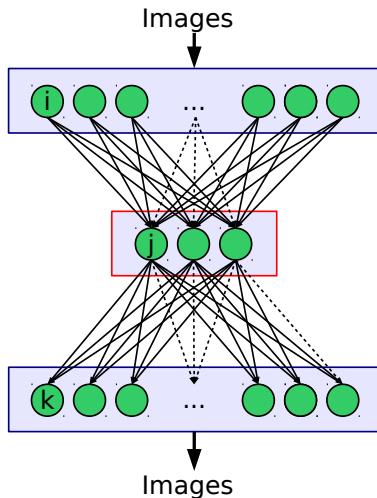
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients



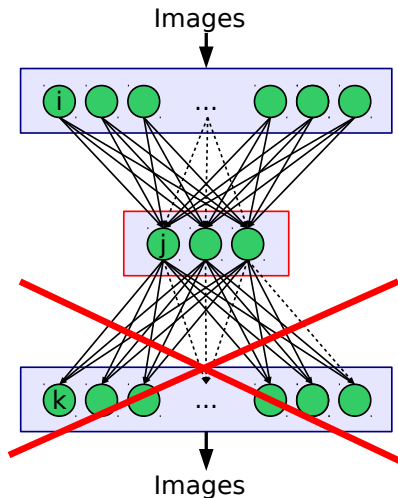
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



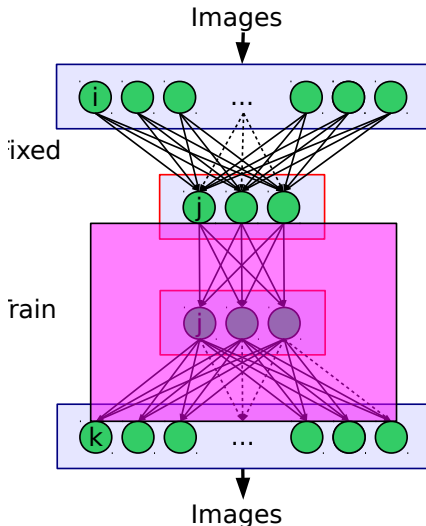
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



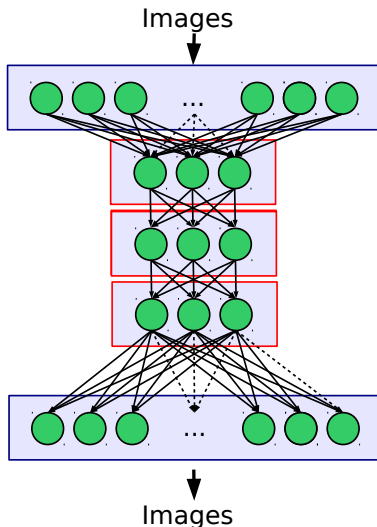
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



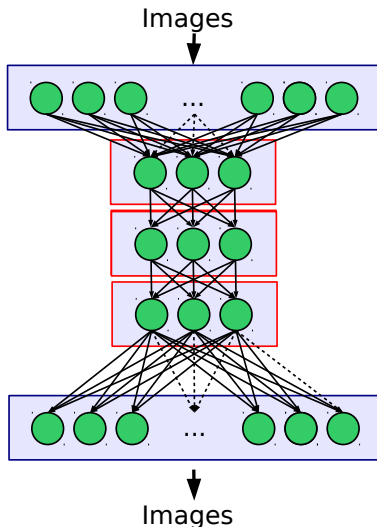
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training



# (Convolutional) Auto Encoder

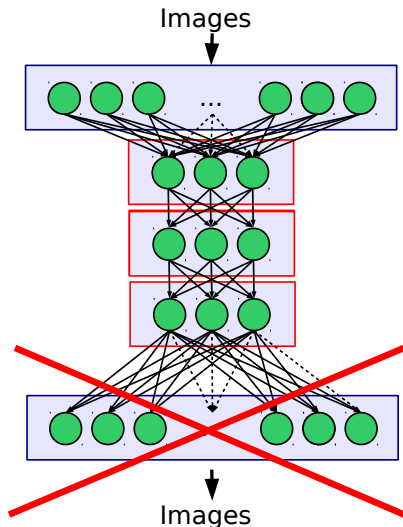
- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training
- Application: Deep Learning





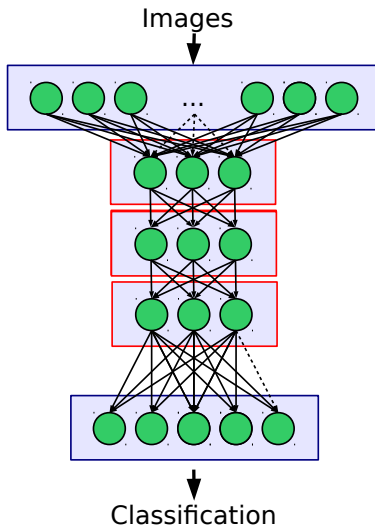
# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training
- Application: Deep Learning



# (Convolutional) Auto Encoder

- Stacked Autoencoder
- Problem: Vanishing gradients
- Solution: Pre-training  
→ Learn “reasonable” features from unlabeled data
- Application: Deep Learning  
→ Supervised learning (via Backpropagation) only as refinement



# Frameworks

- Implementing fast, multi-channel convolutions just as hard as implementing fast matrix multiplications
- *Use existing tools!*
  - Caffe
  - Tensorflow
  - Torch
- For larger datasets you want to use a (good) GPU!

# Caffe

## Caffe

Deep learning framework  
by BAIR

- Started by Yangqing Jia at UC Berkeley
- Maintained by Berkeley AI Research and many contributors
- Backend in C++, frontends for Python and Matlab
- <http://caffe.berkeleyvision.org/>

# Tensorflow



- Developed by Google Brain team
- Python frontend
- <https://www.tensorflow.org/>
- <https://github.com/tensorflow>

# Torch



- Lua frontend
- <http://torch.ch/>
- <https://github.com/torch/torch7>



Glorot, X. and Bengio, Y. (2010).

Understanding the difficulty of training deep feedforward neural networks.

*In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256.



Goodfellow, I., Bengio, Y., and Courville, A. (2016).

*Deep Learning*.

MIT Press.

<http://www.deeplearningbook.org>.



Ioffe, S. and Szegedy, C. (2015).

Batch normalization: Accelerating deep network training by reducing internal covariate shift.

*CoRR*, abs/1502.03167.



Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016).

On large-batch training for deep learning: Generalization gap and sharp minima.

*CoRR*, abs/1609.04836



Glorot, X. and Bengio, Y. (2010).

Understanding the difficulty of training deep feedforward neural networks.

*In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256.



Goodfellow, I., Bengio, Y., and Courville, A. (2016).

*Deep Learning*.

MIT Press.

<http://www.deeplearningbook.org>.



Ioffe, S. and Szegedy, C. (2015).

Batch normalization: Accelerating deep network training by reducing internal covariate shift.

*CoRR*, abs/1502.03167.



Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016).

On large-batch training for deep learning: Generalization gap and sharp minima.

*CoRR*, abs/1609.04836