# Visual Object Recognition

## Lecture 7: Voting and Learning

**Per-Erik Forssén, docent**
**Computer Vision Laboratory**
**Department of Electrical Engineering**
**Linköping University**

# Exam

Exam times:

April 16, 9-11
April 29, 13-15

# Lecture 7: Voting and Learning

- Memory based learning
  Generalized Hough Transform, Meanshift Voting

- Learning
  SVM, Random Forests, Boosting, Deep Learning, Convolutional Neural Networks

# Motivation

- Recognition by storing all observations in a memory and looking up good matches is called ***memory based learning***.
- Efficient indexing techniques from LE6 are essential here.

# Motivation

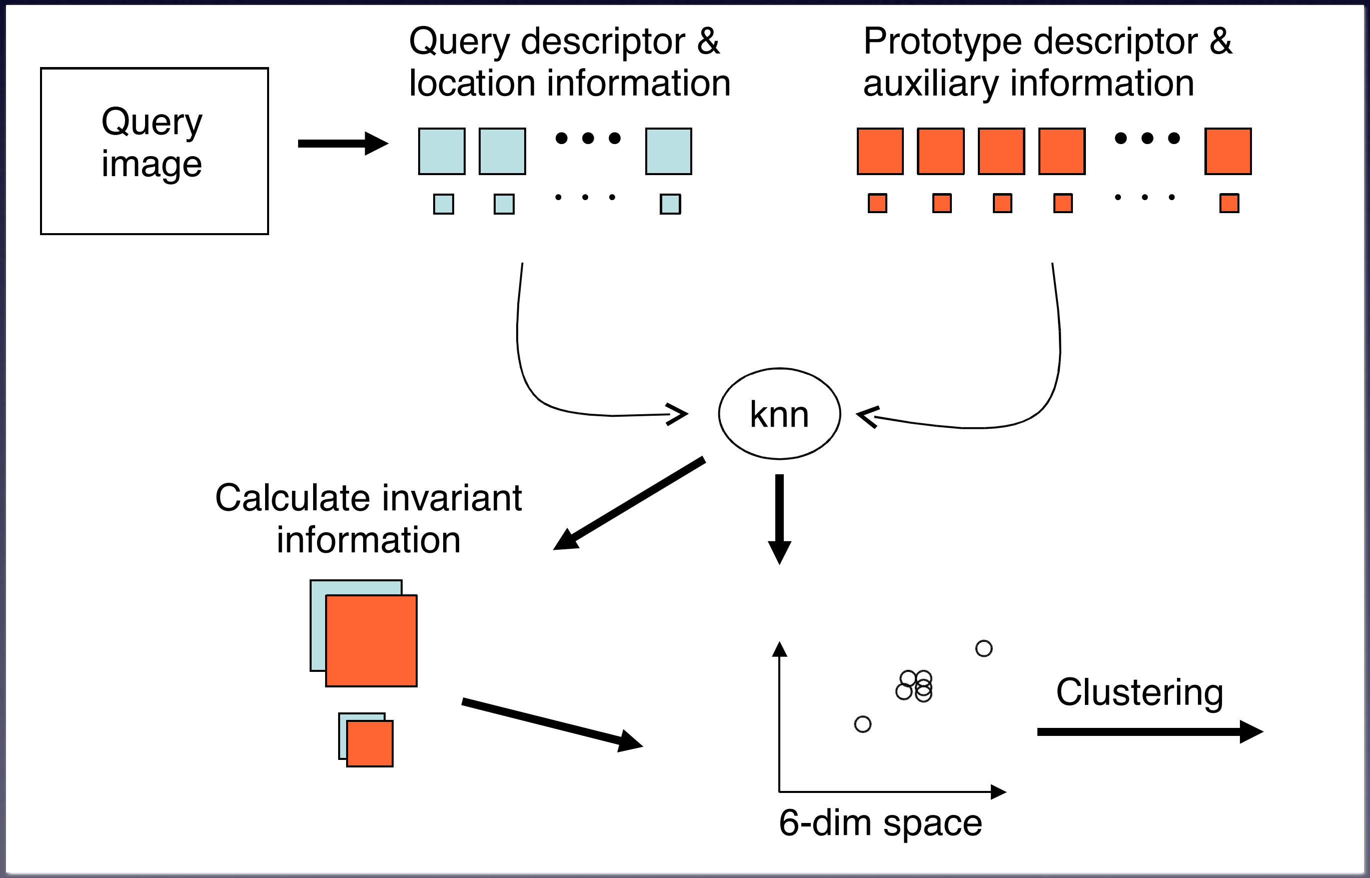
- Recognition by storing all observations in a memory and looking up good matches is called ***memory based learning***.
- Efficient indexing techniques from LE6 are essential here.
- When storing everything is not feasible, other machine learning techniques are necessary.
- Learning also adapts the matching metric.

# Memory based learning

- Memory based learning is viable in object pose recognition [see e.g. Viksten LiU thesis 2010]

# Memory based learning

- Each training image generates a set of pairs (**x**,**y**) **x**=descriptor, **y**=(pose,id)
- At test time, descriptors are computed, and corresponding (pose,id) hypotheses are found in memory.



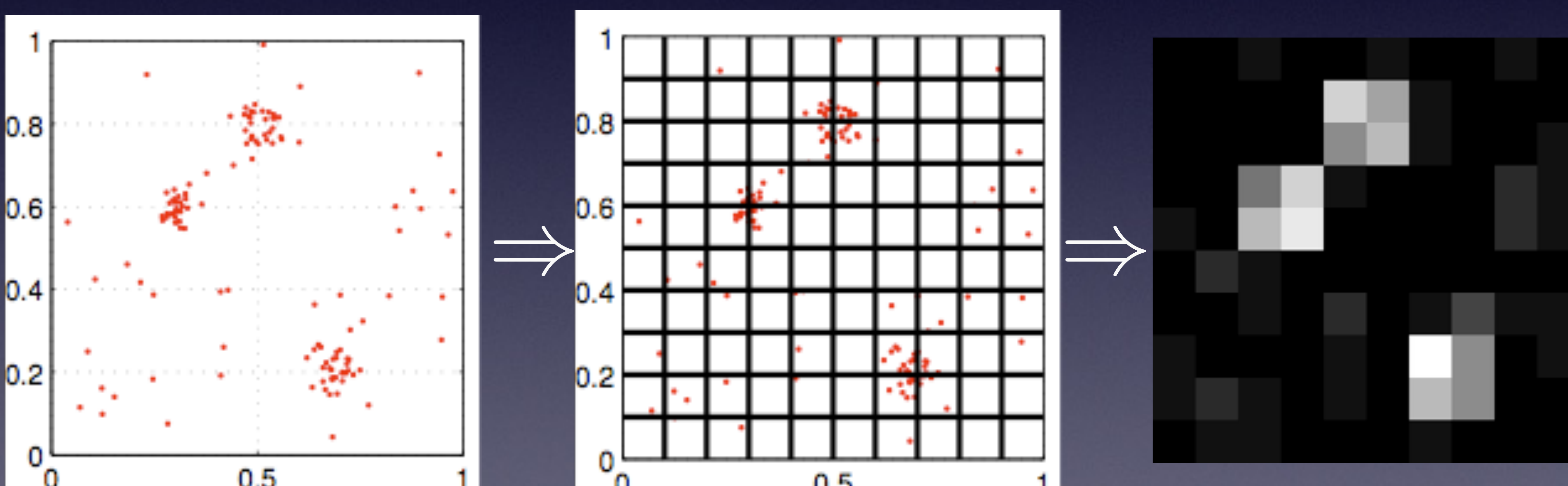

- Finally, hypotheses are combined using clustering.

# Memory based learning

- Common clustering techniques include:
  - Generalized Hough transform [Ballard, PR'80]
    generate constraints in parameter space, and paint them in an accumulator array/vote space.
  - Mean-shift clustering [Cheng PAMI'95]
    continuous domain voting

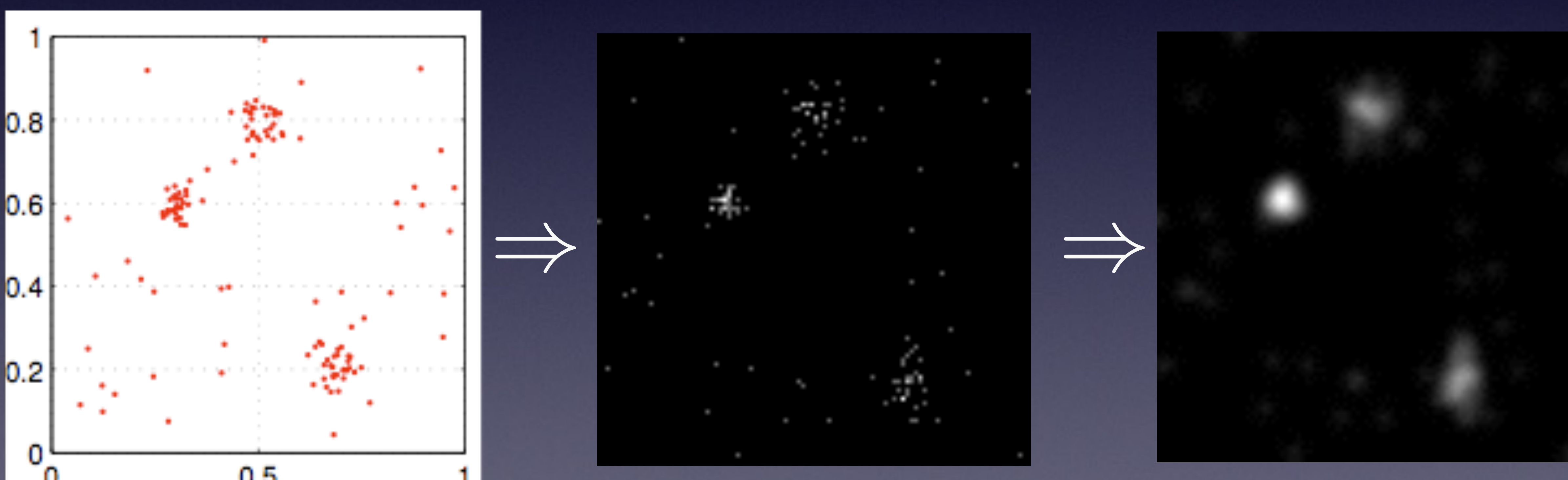
# Generalised Hough Transform

- Non-iterative $\Rightarrow$ constant time complexity.



# Generalised Hough Transform

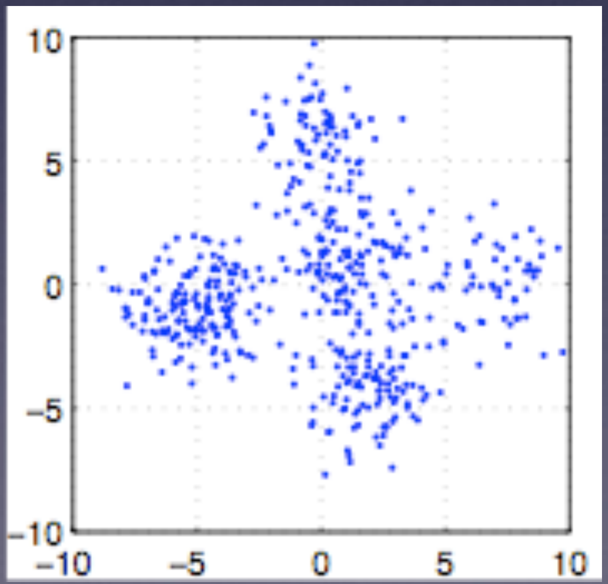
- Quantisation can be dealt with by increasing the number of cells, and blurring.

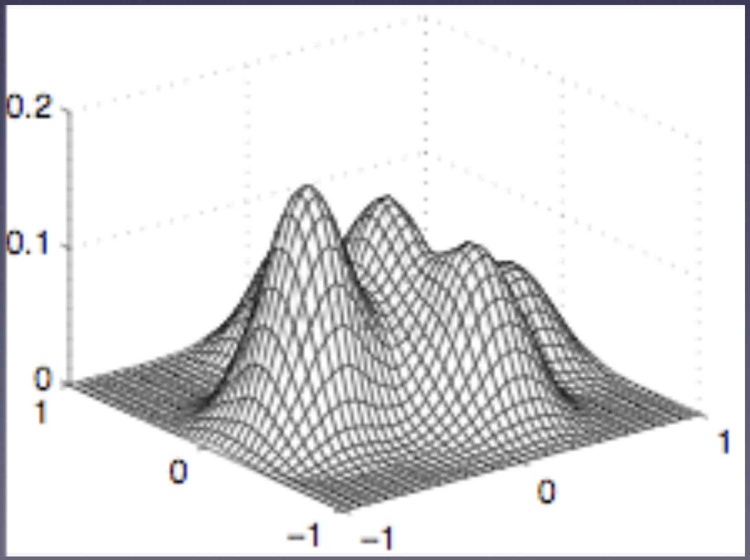


# Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$ we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$



# Kernel density estimate

- For a set of sample points $\{\mathbf{x}_n\}_1^N$
  we define a continuous PDF-estimate as:

$$p(\mathbf{x}) = \frac{1}{Nh^d} \sum_{n=1}^{N} K\left(\frac{\mathbf{x}_n - \mathbf{x}}{h}\right)$$

- K() is a kernel, e.g. $K(\mathbf{x}) = c\exp(-\mathbf{x}^T\mathbf{x}/2)$
- h is the kernel scale.

# Mode seeking

- By *modes* of a PDF, we mean the local peaks of the kernel density estimate.
  - These can be found by gradient ascent, starting in each sample.
  - If we use the Epanechnikov kernel,

$$K_E(\mathbf{x}) = \begin{cases} c(1-\mathbf{x}^T\mathbf{x}) & \text{if } \mathbf{x}^T\mathbf{x} \leq 1 \\ 0/\text{otherwise.} \end{cases}$$

    a particularly simple gradient ascent is possible.

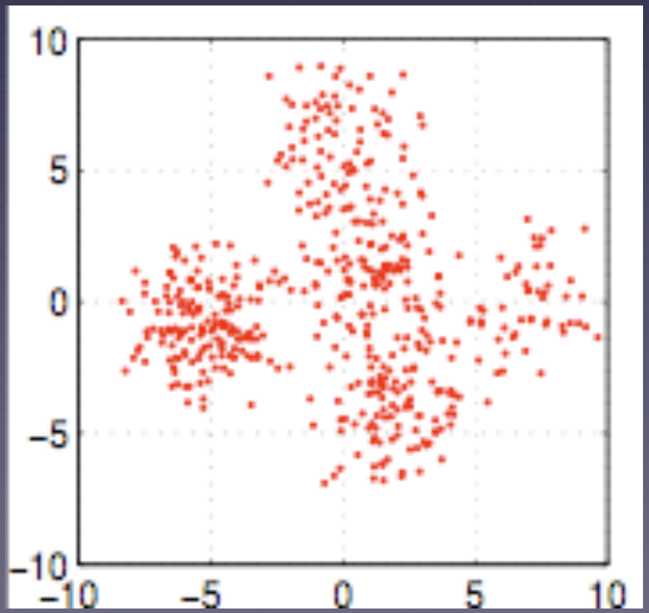
# Mean-shift filtering

1. Start in each data point, $\mathbf{m}_n = \mathbf{x}_n$

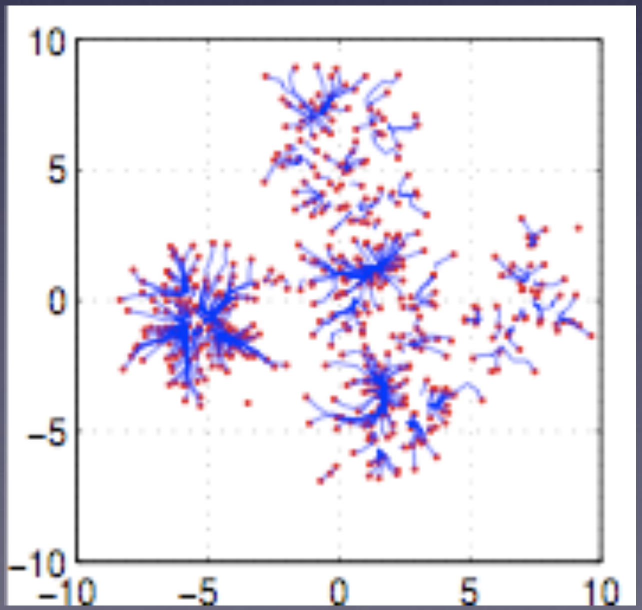
2. Move to position of local average

$$\mathbf{m}_n \leftarrow \frac{1}{|\mathcal{N}(\mathbf{m}_n)|} \sum_{\mathbf{x}_k \in \mathcal{N}(\mathbf{m}_n)} \mathbf{x}_k$$

3. Repeat step 2 until convergence.

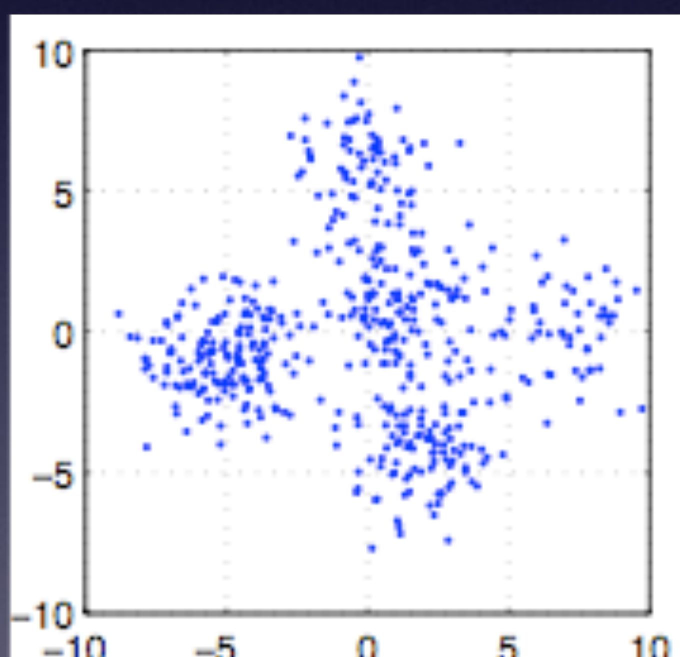



$\Rightarrow$

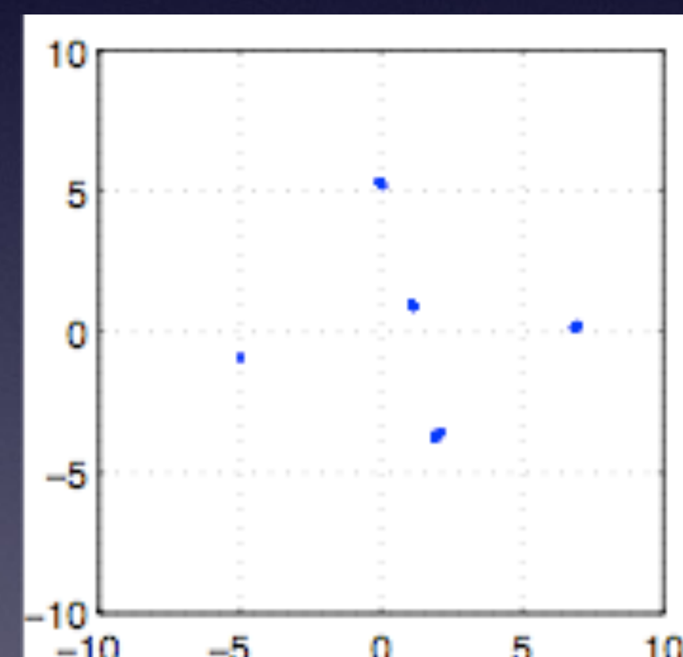


# Mean-shift clustering

- After convergence of the mean-shift filter, all points within a certain distance (e.g. *h*) are said to constitute one cluster.




$\Rightarrow$




- Value of KDE is the confidence in the corresponding hypothesis on pose and object type.

# Support Vector Machines

- Idea by V.N. Vapnik in the 1960s. Many improvements since. The description here is based on [T. Hastie et al. ”The Elements of Statistical Learning”, 2008].
- Binary classification example:
  for a feature vector **x**, we seek a function:

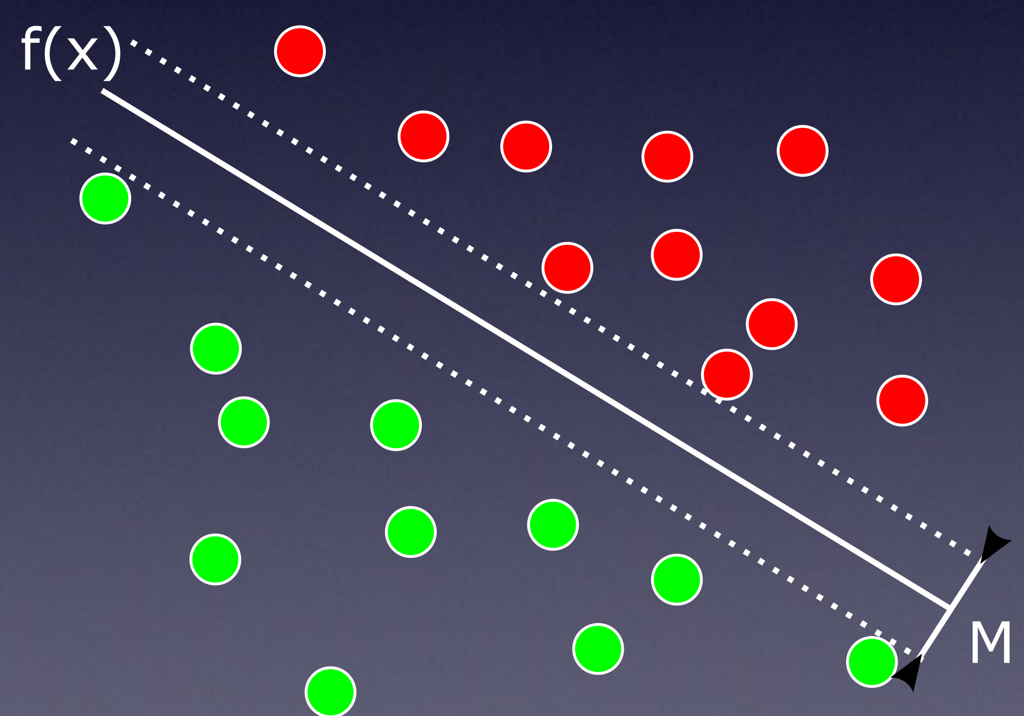
$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$$

- if f(x)>0 class 1, class 2 otherwise

# Support Vector Machines

- The optimization problem: $\max_{\mathbf{w}, w_0, ||\mathbf{w}||=1} M$

$$\text{subject to } y_i(\mathbf{x}_i^T \mathbf{w} + w_0) \geq M, i \in [1, N]$$




- $y_i$ is output class membership {-1,1}

# Support Vector Machines

- For the classification function:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$$

- Solution has the form:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

- where only $\alpha_i$ at the margin boundary are non-zero. The vectors $\mathbf{x}_i$ at the margin are called **support vectors**, as they define **w**.

# Support Vector Machines

- For the classification function:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$$

- Solution has the form:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

- where only $\alpha_i$ at the margin boundary are non-zero. The vectors $\mathbf{x}_i$ at the margin are called **support vectors**, as they define **w**.

- Additional slack variables for all data points are needed to handle cases where all samples cannot be classified correctly.

# Support Vector Machines

- Instead of the linear version:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$$

- A kernel version is typically used:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \langle h(\mathbf{x}), h(\mathbf{x}_i) \rangle + w_0$$

- h(x) is some mapping into a high dimensional space.
- Scalar product can be replaced by a kernel function.

# Support Vector Machines

- Instead of the linear version:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w} + w_0$$

- A kernel version is typically used:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + w_0$$

- This leads to non-linear decision regions, defined by the support vectors.

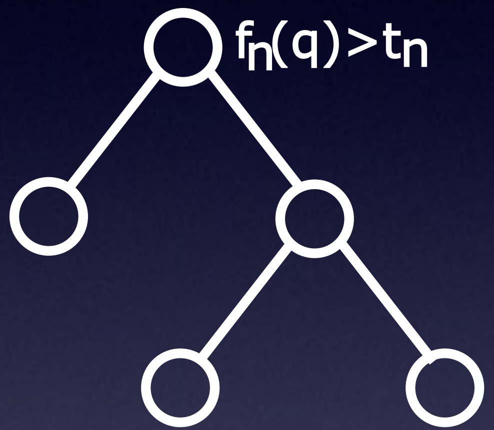
# Support Vector Machines

- SVMs are usually used in a one-vs-all fashion, i.e. one SVM is trained per class.

- Many variants and extensions, e.g. for **regression**, and so called ***latent SVMs*** as used by Felzenzwalb et al. in their DPM system [CVPR08] See lecture 5.

# Classification and Regression Trees

- CART are trees with very simple tests in each node

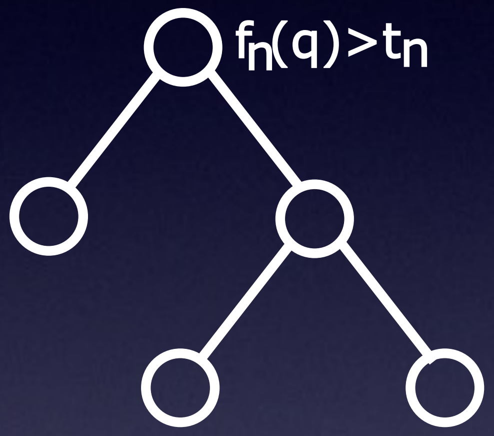



- Tests are data-adaptive. The outcome of previous tests determine which new test to make.

- Leaf nodes store responses instead of samples. E.g. class membership probabilities, or a regression model.

# Classification and Regression Trees

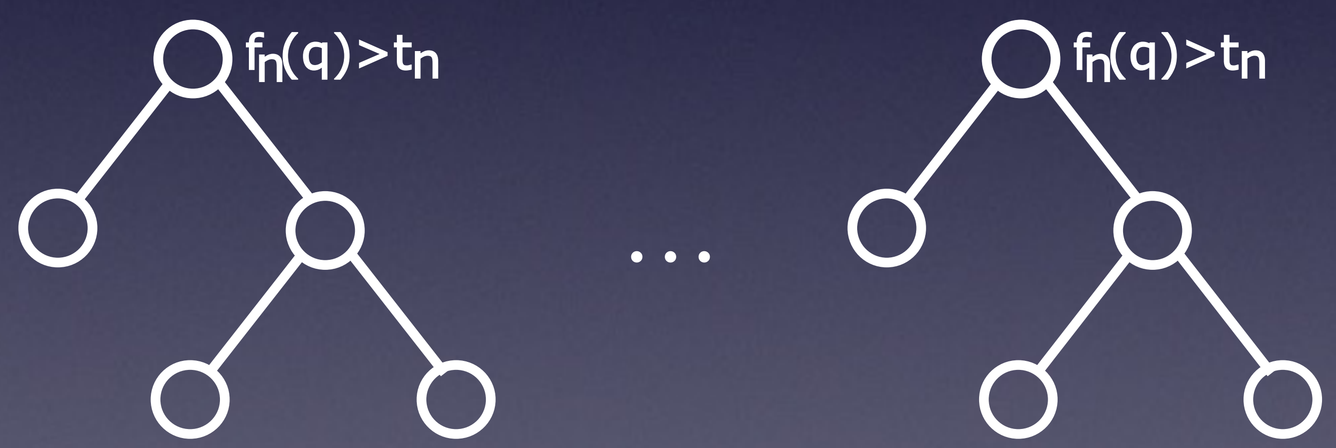
- CART are trees with very simple tests in each node




- Given a tree, the outputs at each leaf node are computed from training samples that end up in this node.
- E.g. a class membership histogram, or a linear regression function fitted from data.

# Random Forests

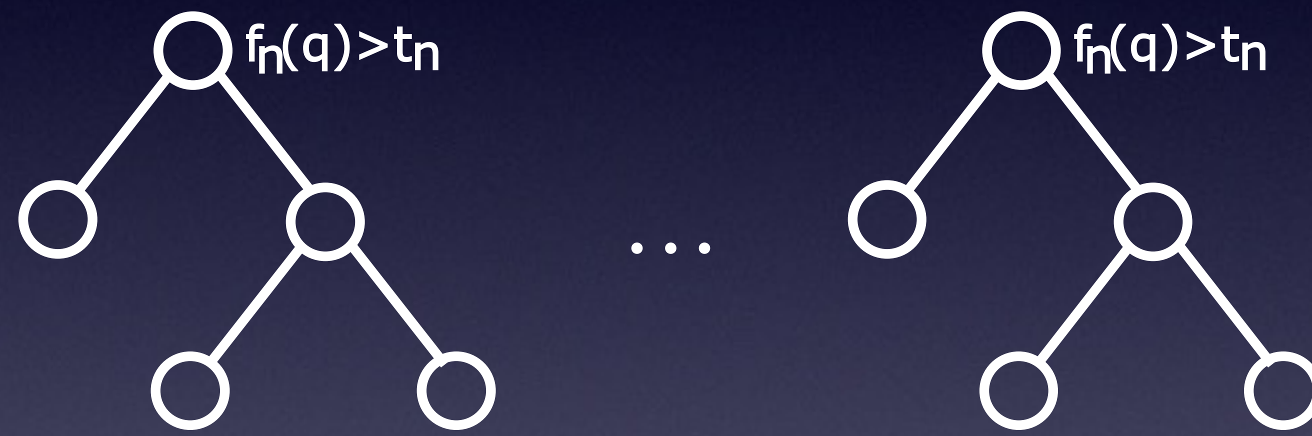
- Invented by Leo Breiman. [L.Breiman ML'2001]. Good Tutorial [Criminisi et al. FTCGV'2011]
- Conceptually similar to the randomized kD-trees in LE6, but actually invented earlier.




- Ensemble of CARTs. All tree outputs are combined to one statement by e.g. histogram averaging.

# Random Forests

- It is well known that fusion of several classifier outputs can improve performance.




- Especially if the classifiers make errors that are uncorrelated.
- Random forests exploit this by creating many relatively poor classifiers, that have uncorrelated errors.

# Random Forests

- Uncorrelated classifier errors in RF are obtained in two ways:
  - **Bagging**. Each tree gets its own training set by drawing random training samples (with replacement).
  - **Randomized node optimization**. Just like in multiple randomized kD-trees (LE6).

# Random Forests

- Randomized node optimization.
  - This is typically involves drawing random split functions and evaluating these.
  - Evaluation is done with respect to the output class histogram entropy H(**v**).

$$H(\mathbf{v}) = \sum_k v_k \log v_k$$

# Random Forests

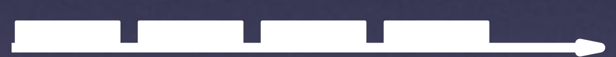
- Evaluation is done with respect to the output class histogram entropy H(**v**).

$$H(\mathbf{v}) = -\sum_k v_k \log v_k$$

# Random Forests

- Evaluation is done with respect to the output class histogram entropy H(**v**).

$$H(\mathbf{v}) = -\sum_k v_k \log v_k$$



H([0.25,0.25,0.25,0.25])=1.39

H([0.10,0.10,0.70,0.10])=0.94

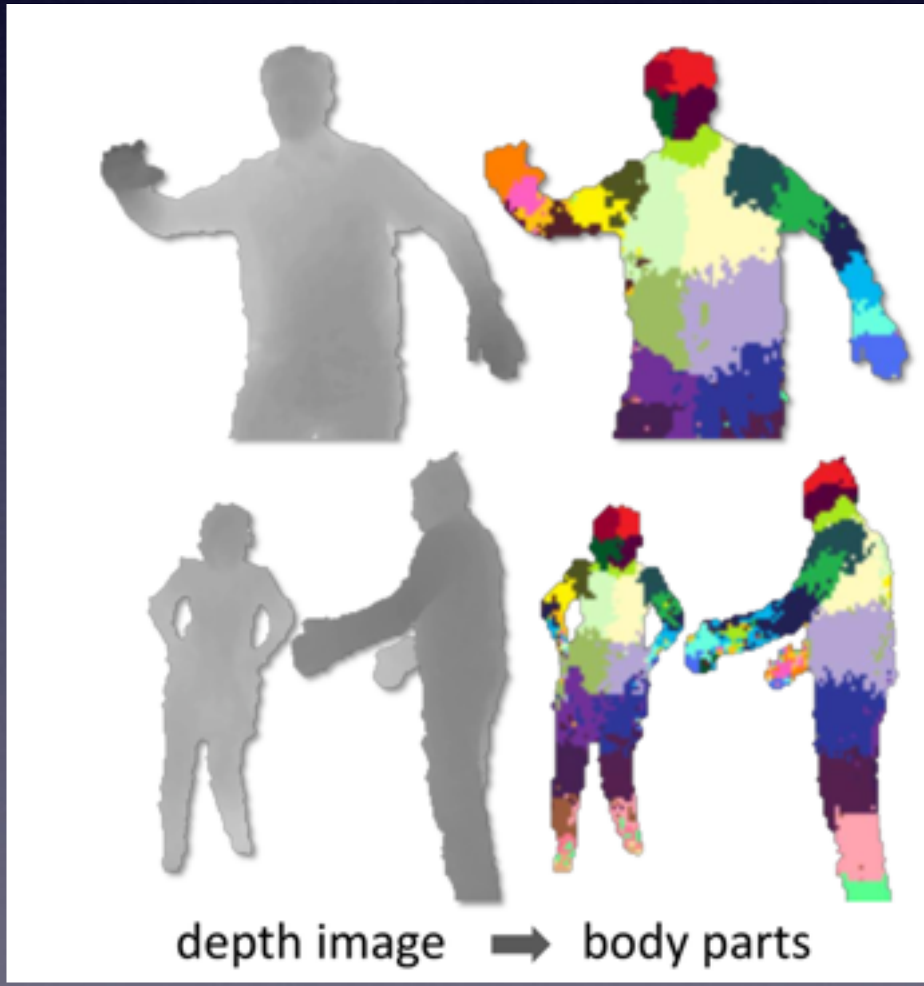
H([0.00,0.00,1.00,0.00])=0.00

- Information gain criterion

$$I = H(\mathcal{S}) - \frac{|\mathcal{S}_L|}{|\mathcal{S}|} H(\mathcal{S}_L) - \frac{|\mathcal{S}_R|}{|\mathcal{S}|} H(\mathcal{S}_R)$$

# Random Forests

- Application example: Pose from Kinect depth maps [Shotton et al. CVPR’11, PAMI’13]
- 31 classes for body parts
- RF applied in each pixel
- Parameters:
  depth 20, 3 trees
- Journal version also tests direct regression to 16 joint positions.



# Boosting

- General principle for ensemble optimization
- **weak learners** are optimized in sequence
- before adding another weak learner, the samples are tested on the current ensemble, and misclassified samples are given higher weight.
- The ensemble now exhibits **complementarity**, not just uncorrelated errors.

# Boosting

- General principle for ensemble optimization
- **weak learners** are optimized in sequence
- before adding another weak learner, the samples are tested on the current ensemble, and misclassified samples are given higher weight.
- The ensemble now exhibits **complementarity**, not just uncorrelated errors.
- Often the weak learner is a linear classifier, but it could also be a classification tree. Such sequential optimisation of trees is reported to have marginally better performance than random forests [K.P. Murphy "Machine Learning", 2012]

# Boosting

- Application example: Face detection [Viola & Jones ICCV'01]

Viola&Jones IJCV'04

www.adorama.com review of Fujifilm finepics F40fd

- Here each weak learner is based on a single Haar-filter response.

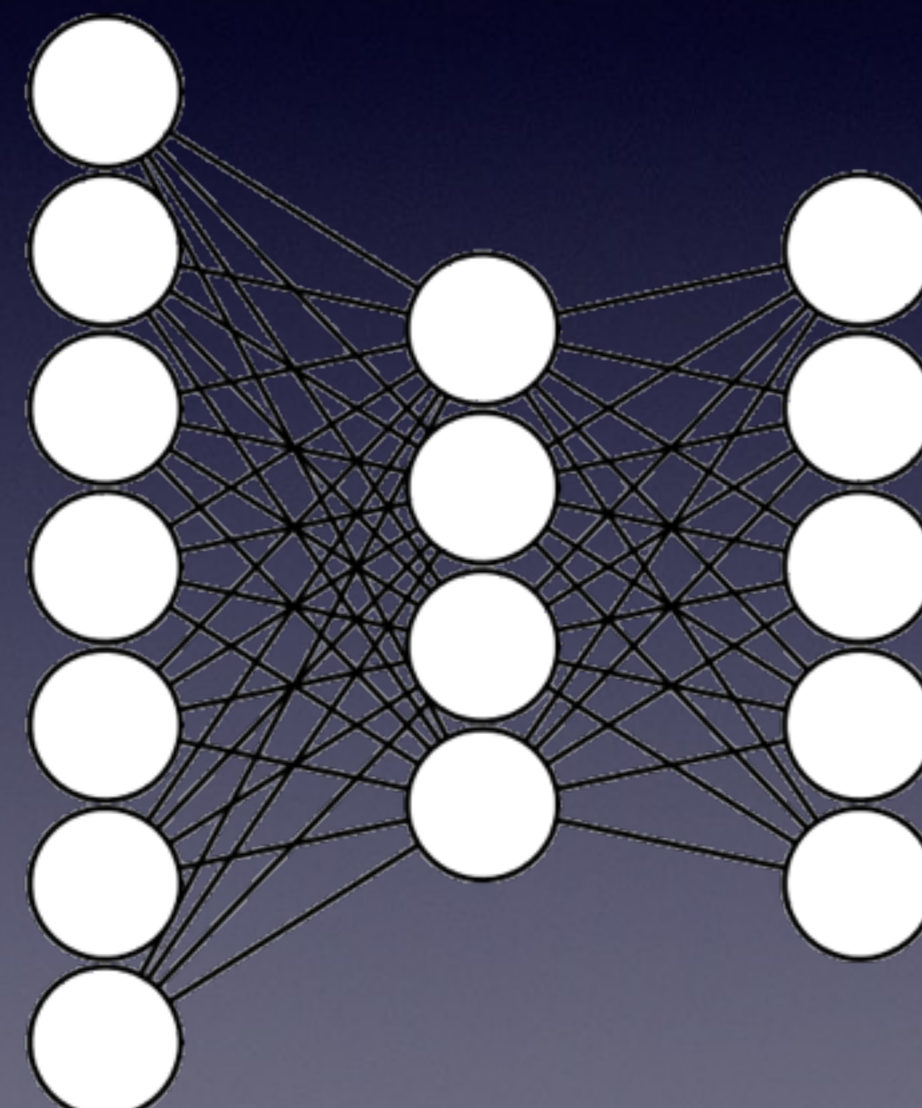
# Deep learning

- This is essentially the Perceptron from the 70s with a bag of tricks added:
  - Better non-linearity
  - Convolutional layers with max-pooling
  - Drop-out

# Deep learning

- Multi-Layer Perceptron (MLP)
- Each node contains a weighted sum of inputs $x_l$, and an activation function f()

$$y_k = f\left(\sum_{l=0}^{L} x_l w_l\right)$$

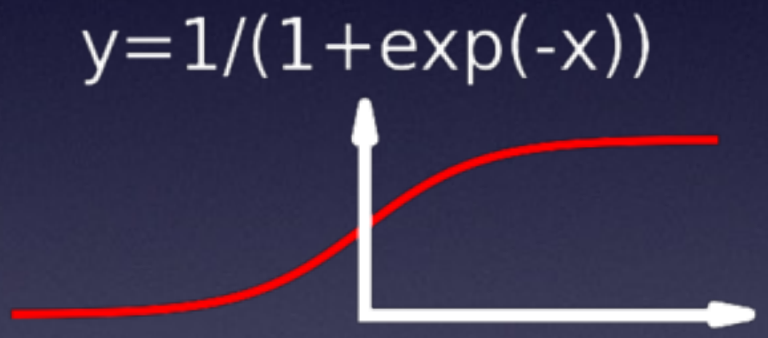


# Deep learning

- Multi-Layer Perceptron (MLP)
- Originated in the 70s
- Training by error back-propagation using the derivative chain rule [thesis by Paul Werbos 1974] [D.E. Rumelhart et al. Nature 1986]
- Many details have changed since to 70s
- Today more training data is available and GPUs can be used for training.

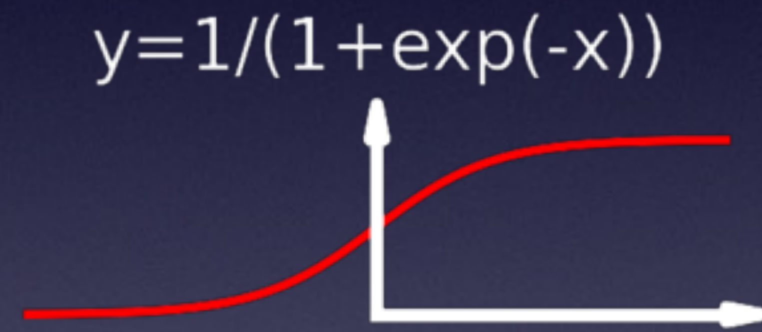
# Deep learning

- The activation function f() must be non-linear otherwise a multi-layer network can be replaced by a single layer one.
- ”classic” logistic function

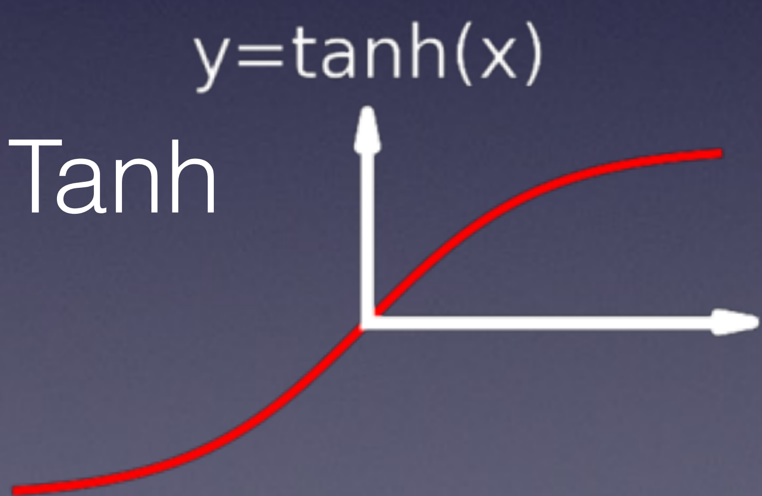


# Deep learning

- The activation function f() must be non-linear otherwise a multi-layer network can be replaced by a single layer one.
- ”classic” logistic function



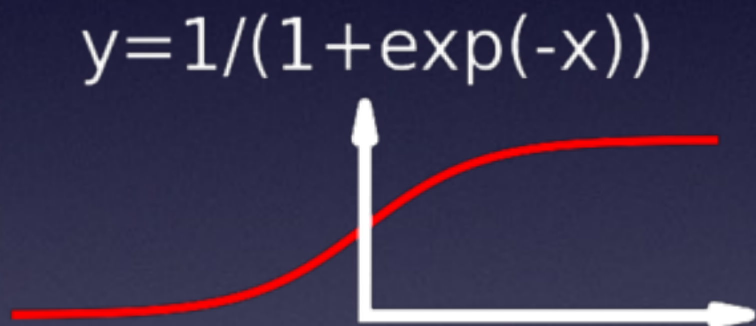

- Tanh



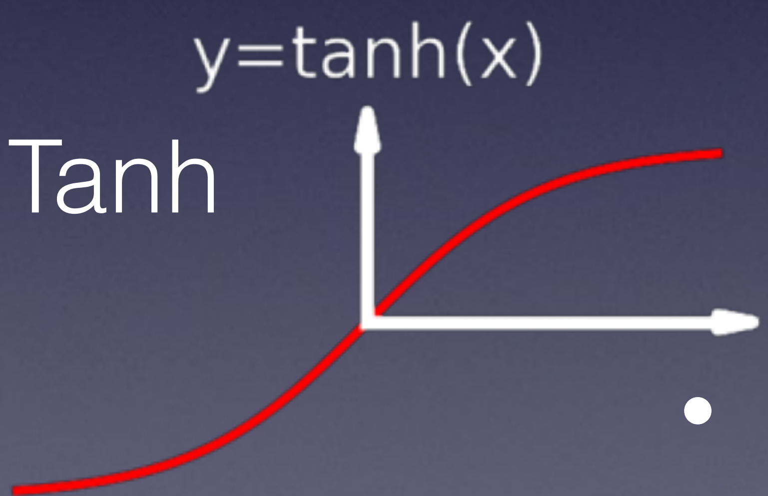
# Deep learning

- The activation function f() must be non-linear otherwise a multi-layer network can be replaced by a single layer one.

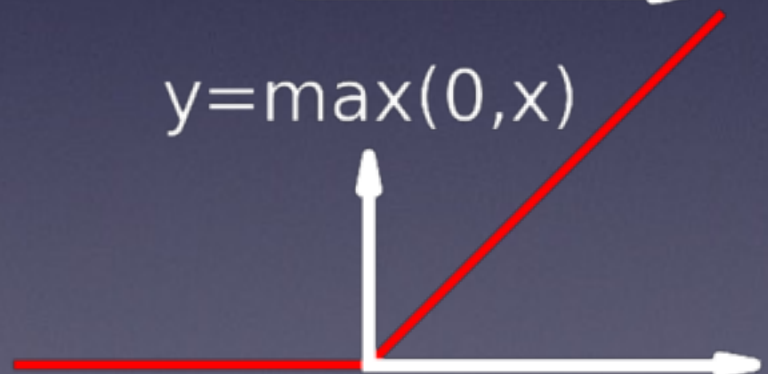
- ”classic” logistic function




- Tanh

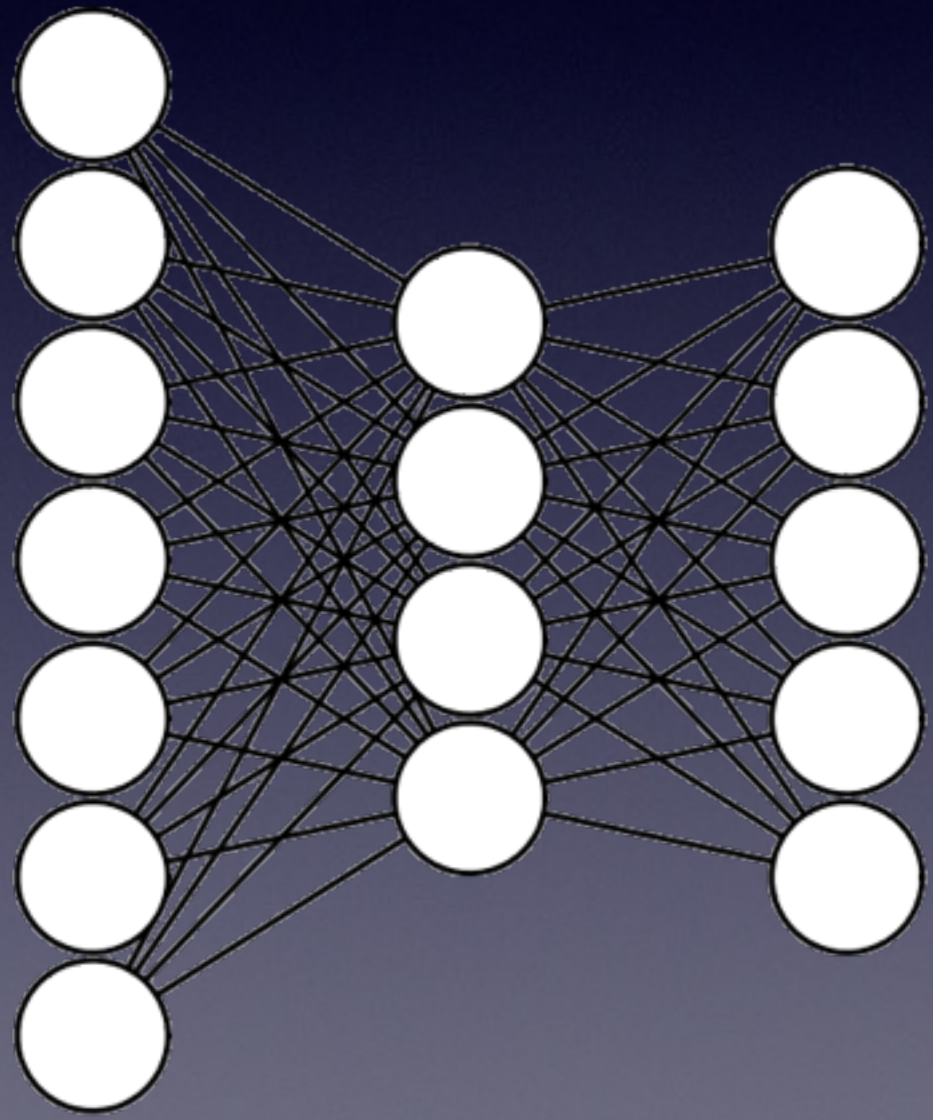



- ReLU
(today’s paper)



# Deep learning

- In deep learning there are three distinct types of layers:



1. **Convolutional**
   First few layers

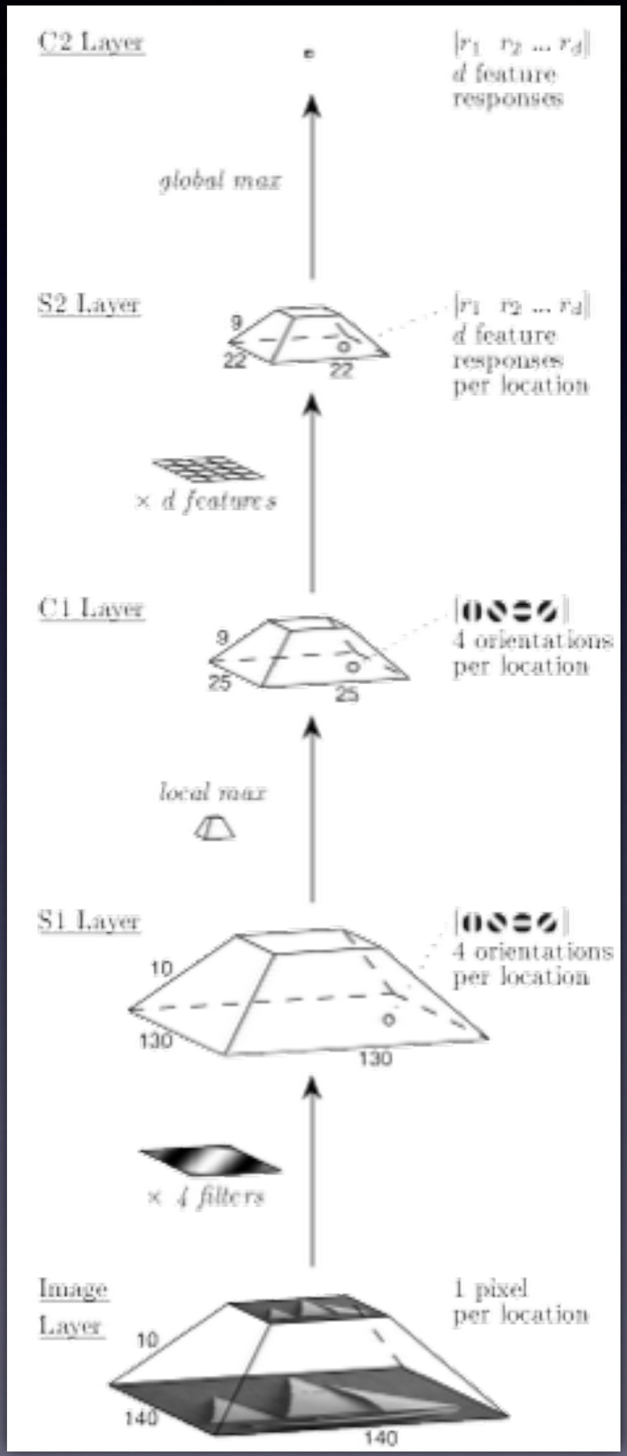
2. **Fully-connected**
   Near output

3. **Output**

# Deep learning

- Convolutional layers
  - Weight sharing: A set of linear filter kernels that are shifted spatially
  - Non-linearity: activation function+normalization.
  - The output map is then reduced by max-pooling
  - Higher layers have more filter types, but smaller spatial resolution (c.f. ”Standard model” in lecture 1)

# Visual Object Recognition

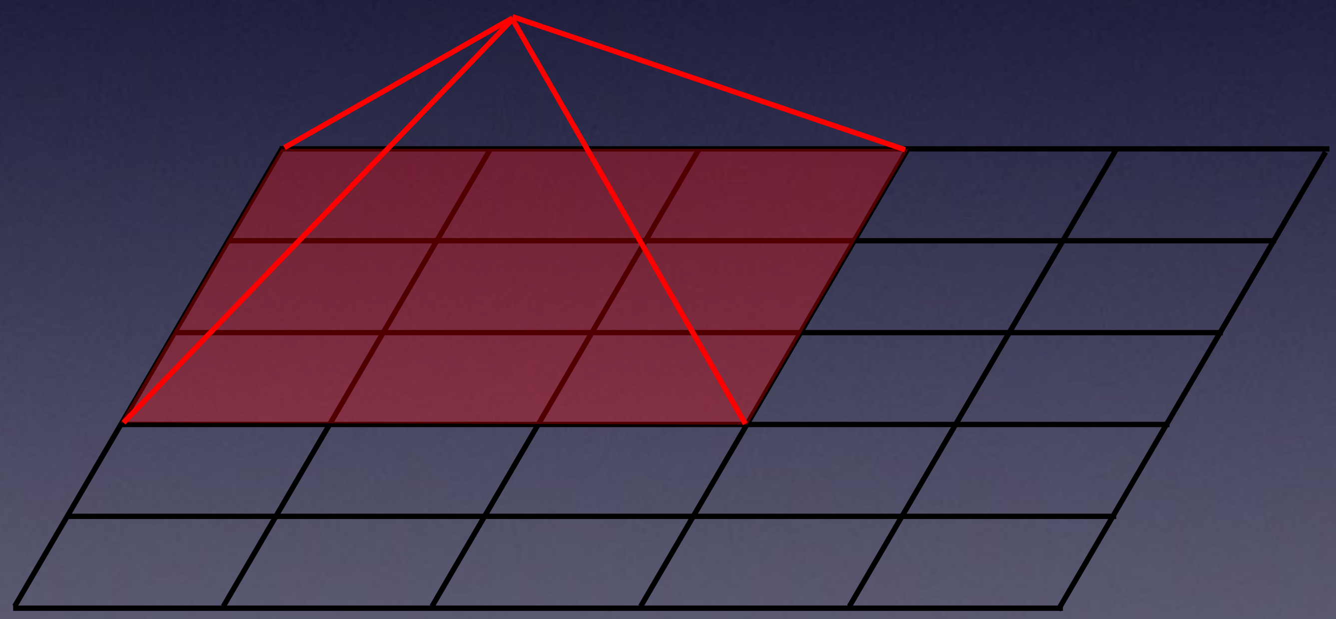



Mutch&Lowe CVPR'06

- The "Standard Model", Riesenhuber&Poggio, Nature Neuroscience vol.2 no.11, 1999
- Alternating template matching and local max operations.
- Decreasing spatial resolution, increasing number of feature types
- Perception only, no motor functions (head&eye movements)

# Deep learning

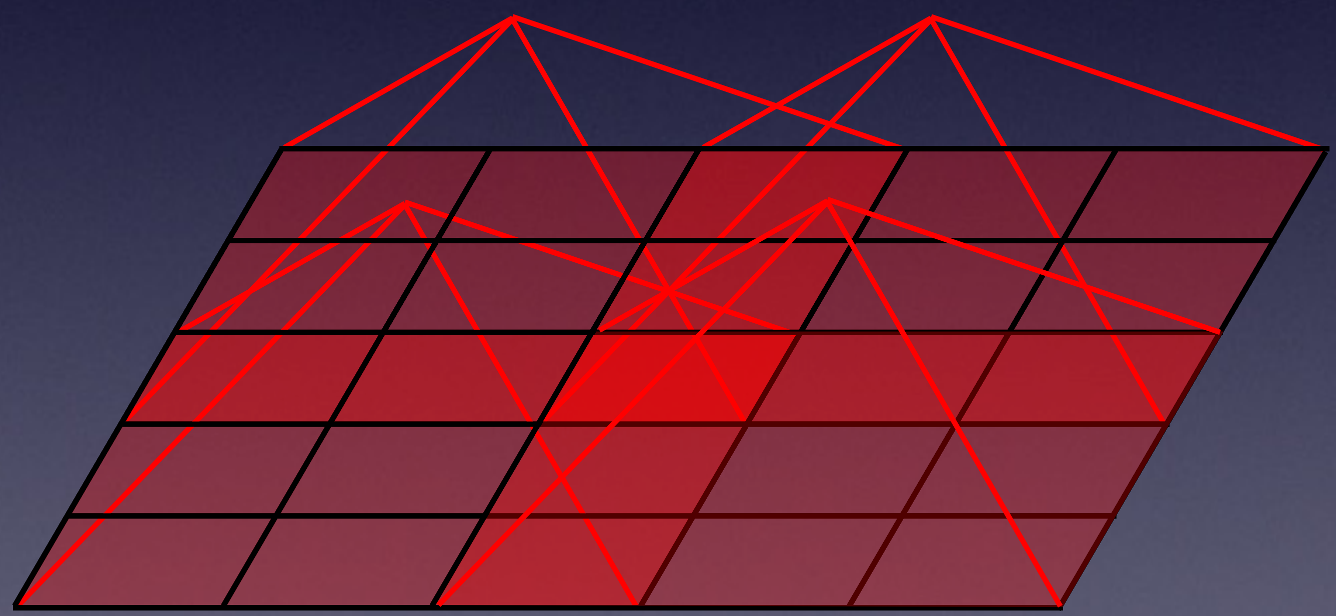
- Max-pooling
  - Max response in 3x3 neighbourhood, stride 2



  - Improve performance in convolutional layers.

# Deep learning

- Max-pooling
  - Max response in 3x3 neighbourhood, stride 2



  - Overlapping pooling as in today's paper.

# Deep learning

- Fully connected layers
  - Layers before output have no spatial shifts
  - During training a technique called dropout is applied here.

# Deep learning

- Drop-out

- During training, randomly set 50% of nodes to zero.
- At test time scale responses by 0.5 to compensate.
- Usually applied at the fully-connected layers to make responses more independent of each other.

# Deep learning

- Output layer
  - For classification the output layer has a softmax logistic normalization

$$\sigma_k = \frac{e^{y_k}}{\sum_{n=1}^{K} e^{y_n}}$$

  - Gives values in range [0,1] that can be interpreted as probabilities.

# Deep learning

- Output layer
  - Another option is to train the network to output the input images again
  - Networks with such an output layer are called **autoencoder networks**
  - last hidden layer can now be used as an image feature (see also DeCAF, lecture 3)

# Deep learning

- Training
  - Basic principle is still back-propagation of errors.
  - Random initialization of weights
  - Stochastic (block) estimation of error gradients
  - Many passes (epochs) through the data.
  - Bag of tricks. See todays paper.

# Discussion

- Questions/comments on today's paper:

  A. Krizhevsky, I. Sutskever, G.E. Hinton, ”ImageNet Classification with Deep Convolutional Neural Networks”, **NIPS 2012**

# Paper for next week

- Paper for next week will be announced over email later…