The Perceptron

Volker Tresp

Summer 2017

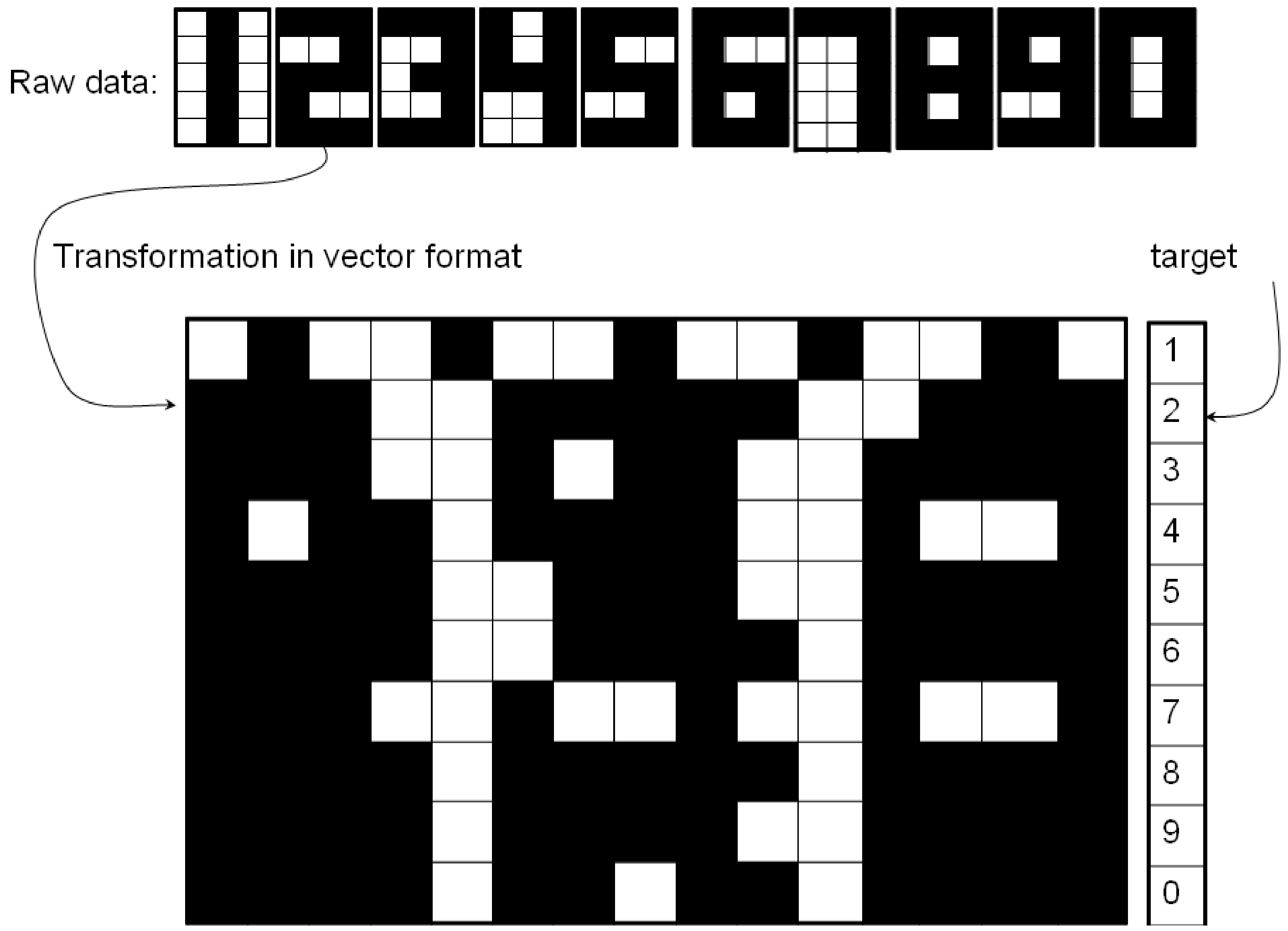
Elements in Learning Tasks

- Collection, cleaning and preprocessing of training data
- Definition of a class of learning models. Often defined by the **free model parameters** in a learning model with a fixed structure (e.g., a Perceptron)
- Selection of a cost function which is a function of the data and the free parameters (e.g., number of misclassifications in the training data as a function of the model parameters)
- Optimizing the cost function via a learning rule to find the best model in the class of learning models under consideration. Typically this means the learning of the optimal parameters in a model with a fixed structure

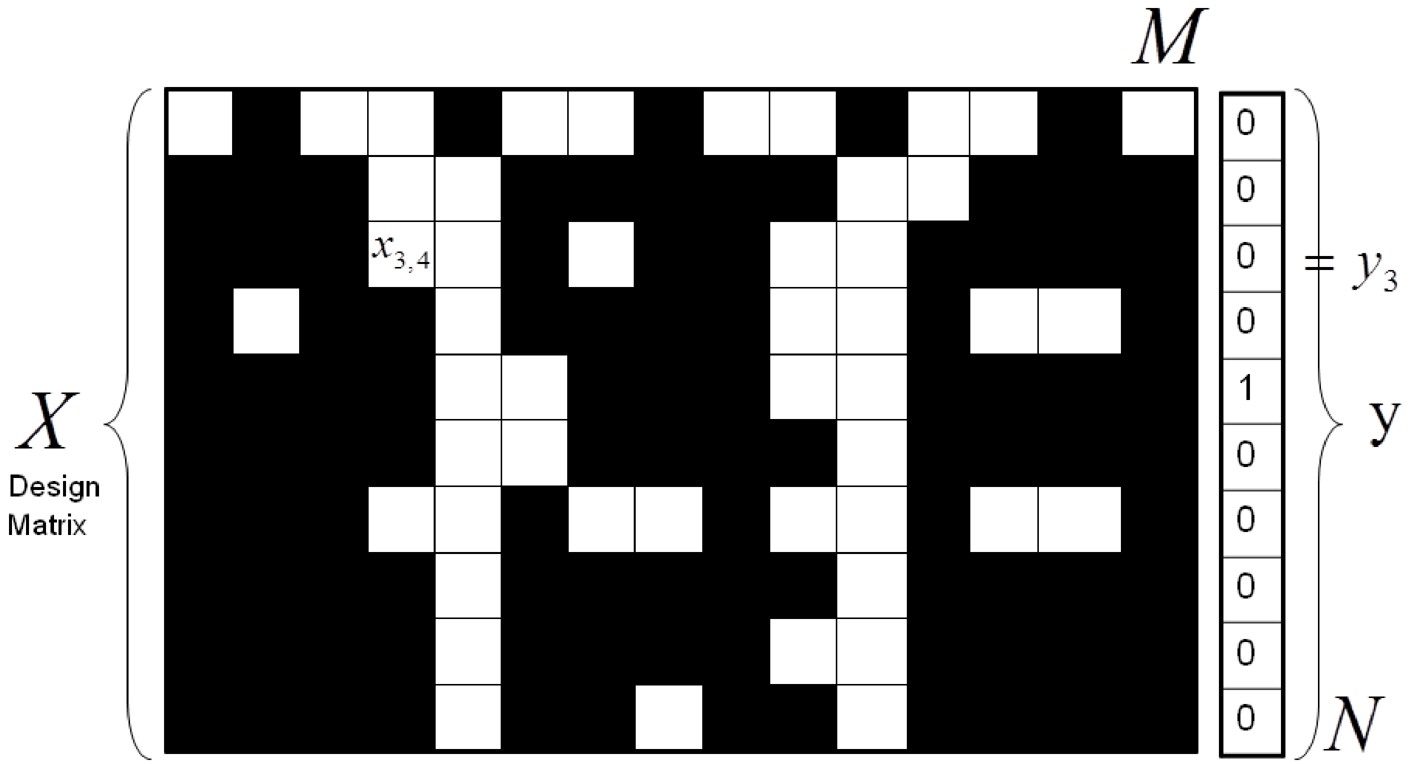
Prototypical Learning Task

- Classification of printed or handwritten digits
- Application: automatic reading of postal codes
- More general: OCR (*optical character recognition*)

Transformation of the Raw Data (2-D) into Pattern Vectors (1-D), which are then the Rows in a Learning Matrix



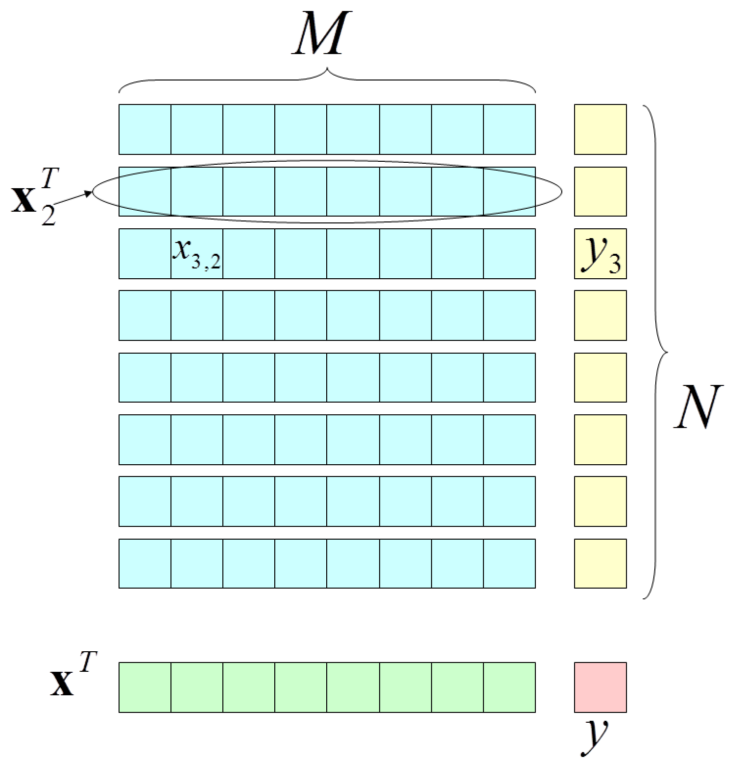
Binary Classification for Digit “5”



target - 1: pattern belongs to class 5

target - 0: pattern does not belong to class 5

Data Matrix for Supervised Learning

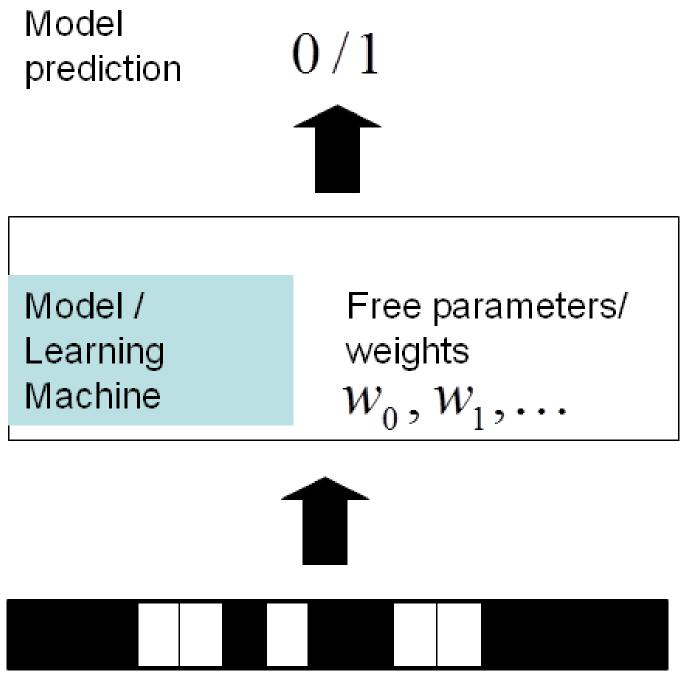


M	number of inputs (input attributes)
N	number of training patterns
$\mathbf{x}_i$	$= (x_{i,0}, \ldots, x_{i,M-1})^T$ input for the i-th pattern
$x_{i,j}$	j-th component of $\mathbf{x}_i$
$\mathbf{X}$	$= (\mathbf{x}_1, \ldots, \mathbf{x}_N)^T$ design matrix
y_i	target for the i-th pattern
$\mathbf{y}$	$= (y_1, \ldots, y_N)^T$ vector of targets
$\widehat{y}_i$	model prediction for $\mathbf{x}_i$
$\mathbf{d}_i$	$= (x_{i,0}, \ldots, x_{i,M-1}, y_i)^T$ i-th pattern
D	$= \{\mathbf{d}_1, \ldots, \mathbf{d}_N\}$ training data
$\mathbf{x}$	$= (x_1, x_2, \ldots, x_{M-1})^T$, generic (test) input
y	unknown target for $\mathbf{x}$
$\widehat{y}$	model estimate
$f_{\mathbf{w}}(\mathbf{x})$	a model function with parameters $\mathbf{w}$
$f(\mathbf{x})$	the true function

Fine Details on the Notation

- $\mathbf{x}$ is a generic input and x_j is its j-th component. y is a generic output
- $\mathbf{x}_i$ is the i-th data point in the training data set and $\mathbf{x}_{i,j}$ is its j-th component. y_i is the target associated with $\mathbf{x}_i$
- $\mathbf{y}$ is the vector of all targets
- Also note that $\mathbf{x}_i$ is a column vector but it appears as a row in X

Model



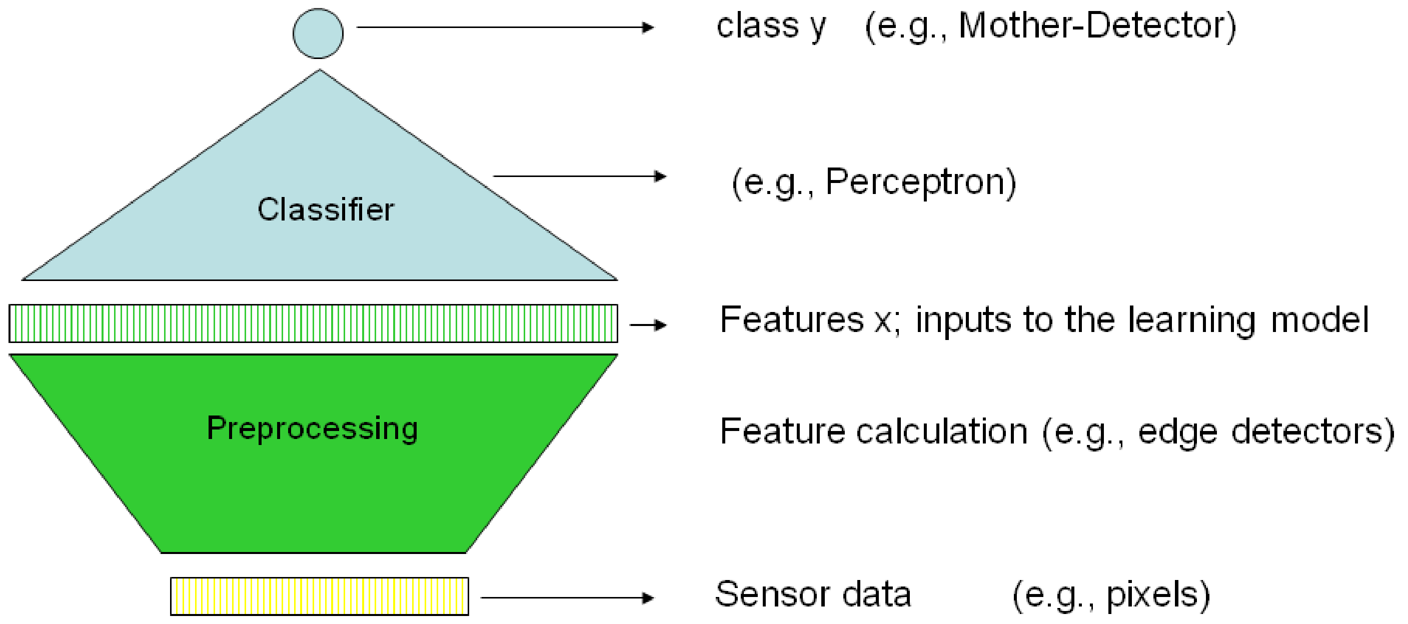
Adaptation of

$w_0, w_1, \ldots$

for good performance on training data D

Real goal: good performance on test data (x, y)

A Biologically Motivated Model


class y (e.g., Mother-Detector)
Classifier
(e.g., Perceptron)
Features x; inputs to the learning model
Preprocessing
Feature calculation (e.g., edge detectors)
Sensor data (e.g., pixels)

Examples of Input-Output Problems (Supervised Learning Problems)

- A biological system needs to make a decision, based on available sensor information
- An OCR system classifies a hand written digit
- A forecasting system predicts tomorrow's energy consumption

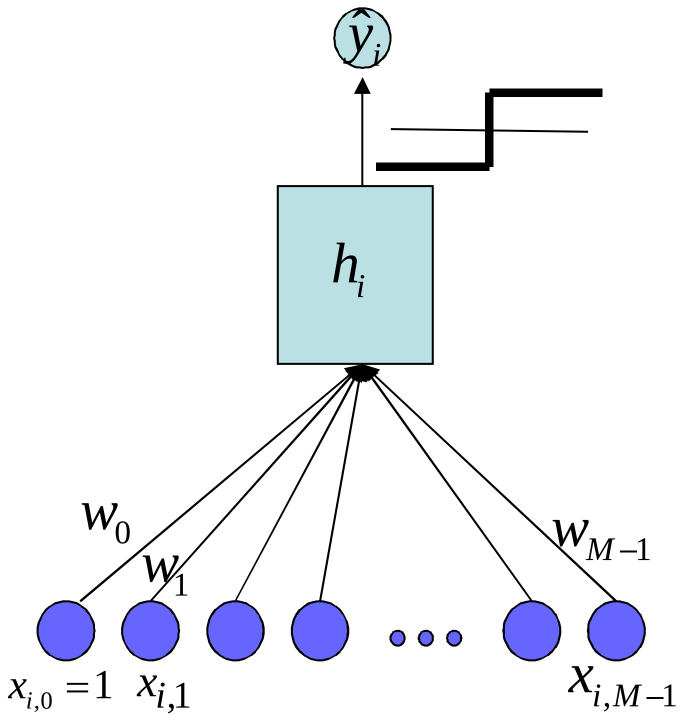
Supervised Learning

- In supervised learning one assumes that in training both inputs and outputs are available
- For example, an input pattern might reflect the attributes of an object and the target is the class membership of that object
- The goal is the correct classification for new patterns (e.g., new objects)
- Linear classifier: one of the simplest but surprisingly powerful classifiers
- A linear classifier is particularly suitable, when the number of inputs M is large; if M is not large, one can transform the input data into a high-dimensional space (preprocessing), where a linear classifier might be able to solve the problem; this idea is central to a large portion of the lecture (basis functions, neural networks, kernel models)
- A linear classifier can be realized through a Perceptron, a single formalized neuron!

The Perceptron: A Learning Machine

- The Perceptron was the first serious learning machine
- The Perceptron learning algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt

The Perceptron: Input-Output



- The activation function of the Perceptron is a sum of weighted inputs

$$h = \sum_{j=0}^{M-1} w_j x_j$$

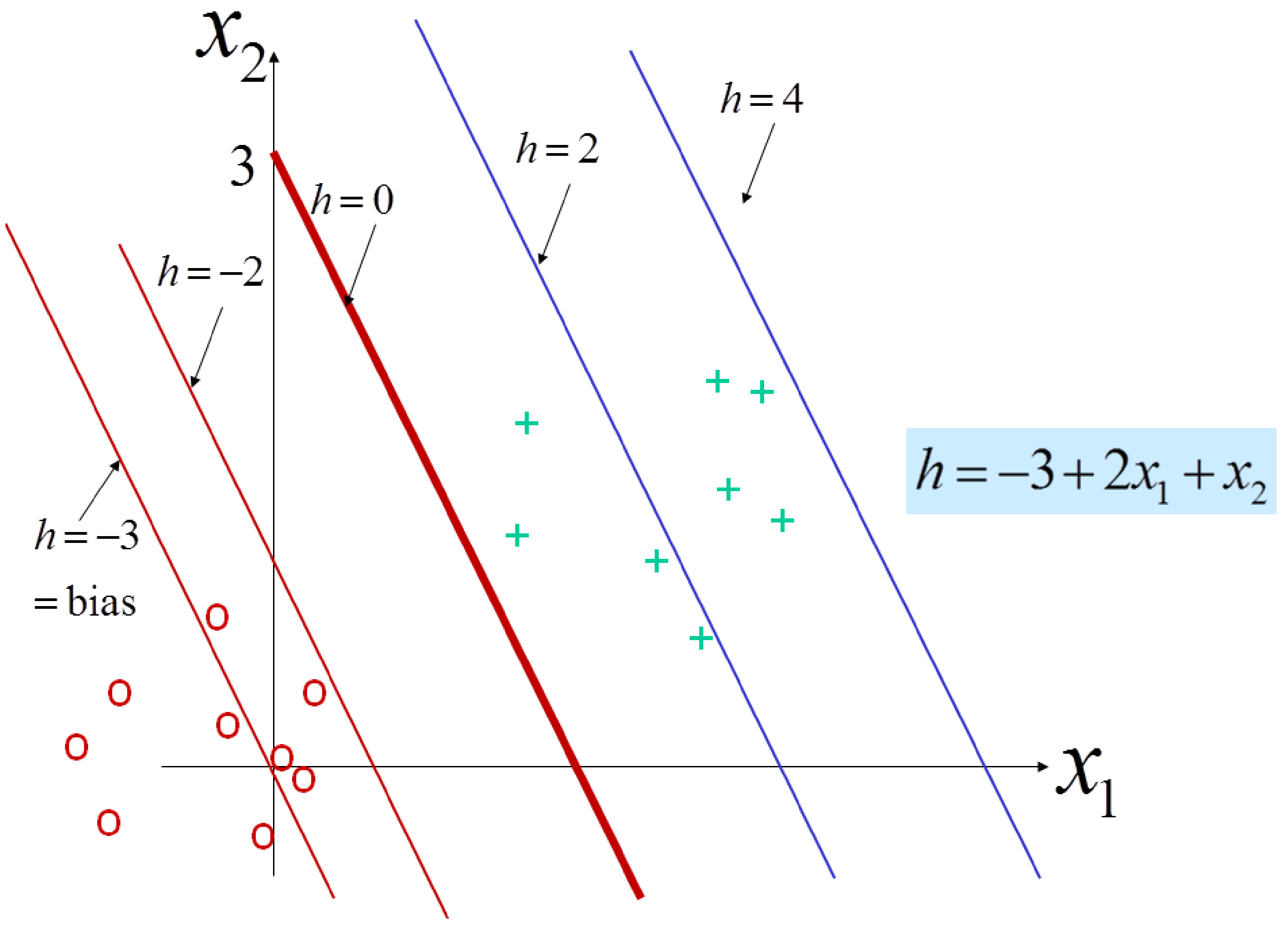
(Note: $x_0 = 1$ is a constant input, such that w_0 can be though of as a bias)

- The binary classification $y \in \{1, -1\}$ is calculated as

$$\hat{y} = \text{sign}(h)$$

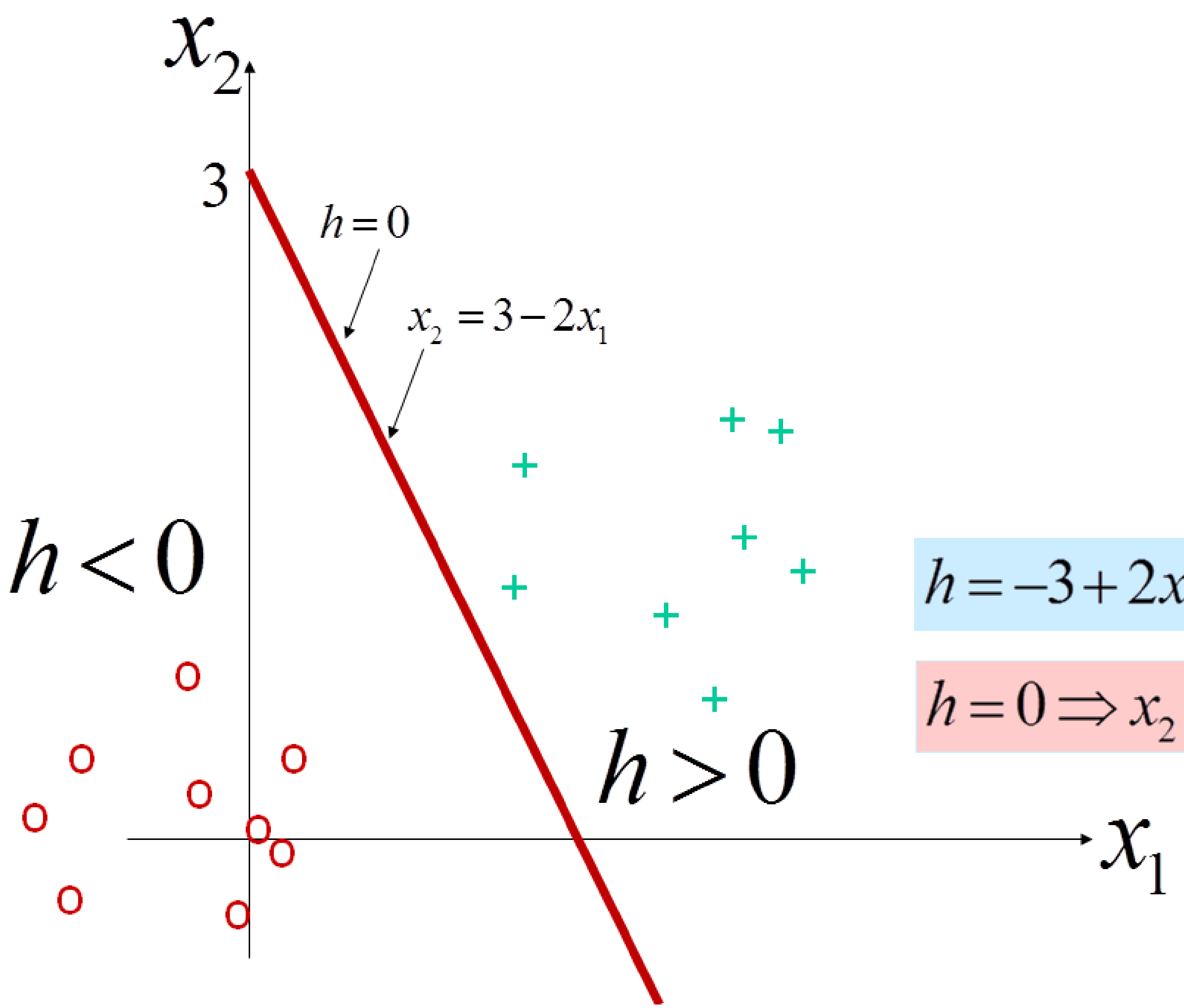
- The linear classification boundary (*separating hyperplane*) is defined by $h(\mathbf{x}) = 0$

The Activation Function h ($M = 3$)



$$h = -3 + 2x_1 + x_2$$

The Decision Boundary ($M = 3$)



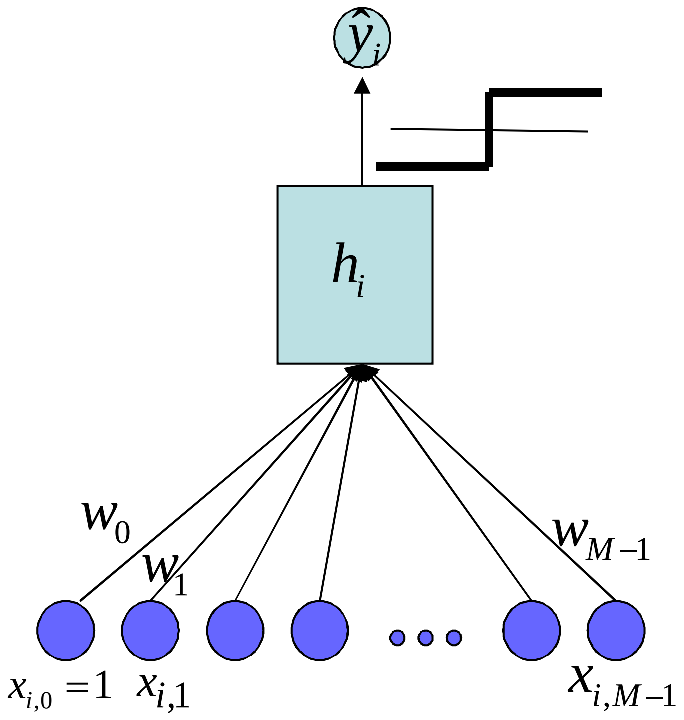
$h = -3 + 2x_1 + x_2$

$h = 0 \Rightarrow x_2 = 3 - 2x_1$

Linearities

- Not to get confused:
- 1: $h(\mathbf{x})$ is a linear function. In the simplest case: $h = w_0 + w_1 x$
- 2: $h(\mathbf{x}) = 0$ defines a linear hyperplane. It looks like a linear function when $M = 3$: $w_0 + w_1 x_1 + w_2 x_2$, thus $x_2 = 1/w_2(-w_0 - w_1 x_1)$

Perceptron as a Weighted Voting Machine



- The Perceptron is often displayed as a graphical model with one input node for each input variable and with one output node for the target
- The bias w_0 determines the class when all inputs are zero
- Consider only binary inputs with $x_{i,j} \in \{0, 1\}$
- When $x_{i,j} = 1$ the j-th input votes with weight $|w_j|$ for class $\text{sign}(w_j)$
- *Thus, the response of the Perceptron can be thought of as a weighted voting for a class*

Perceptron Learning Rule

- We now need a learning rule to find optimal parameters $w_0, \ldots, w_{M-1}$
- We define a cost function that is dependent on the training data and the parameters
- In the learning process (training), one attempts to find parameters that minimize the cost function

The Perceptron Cost Function

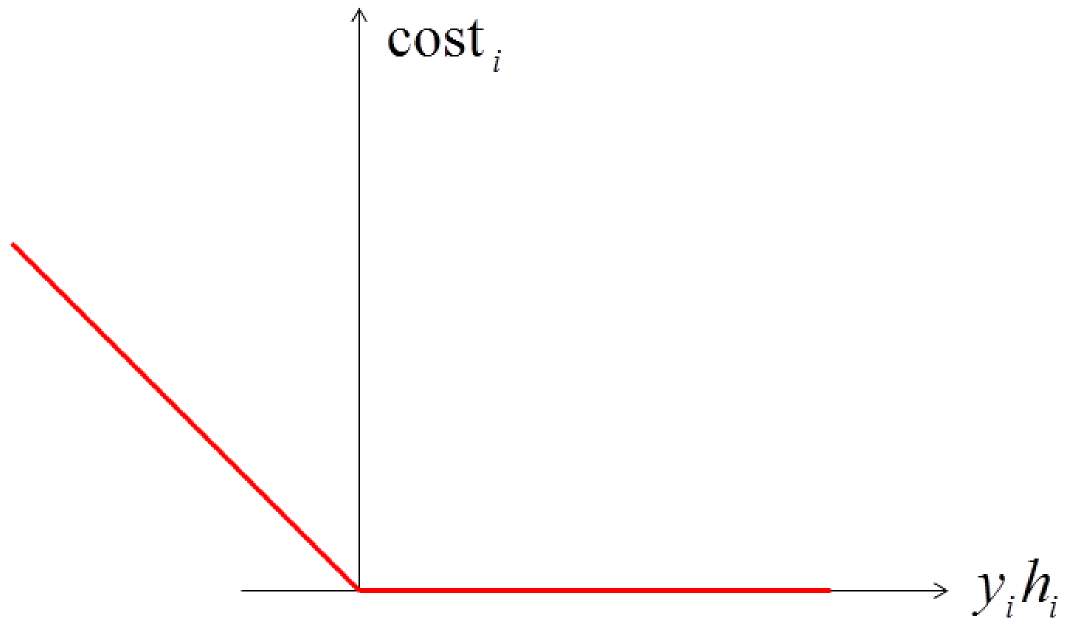
- Goal: correct classification of the N training samples with targets $\{y_1, \ldots, y_N\}$
- The Perceptron cost function is

$$\text{cost} = -\sum_{i \in \mathcal{M}} y_i h_i = \sum_{i=1}^{N} |-y_i h_i|_+$$

where $\mathcal{M} \subseteq \{1, \ldots, N\}$ is the index set of the currently misclassified patterns. $|arg|_+ = \max(arg, 0)$.

- Obviously, we get $\text{cost} = 0$ only, when all patterns are correctly classified (then $\mathcal{M} = \emptyset$); otherwise $\text{cost} > 0$, since y_i and h_i have different signs for misclassified patterns

Contribution of one Data Point to the Cost Function



Gradient Descent

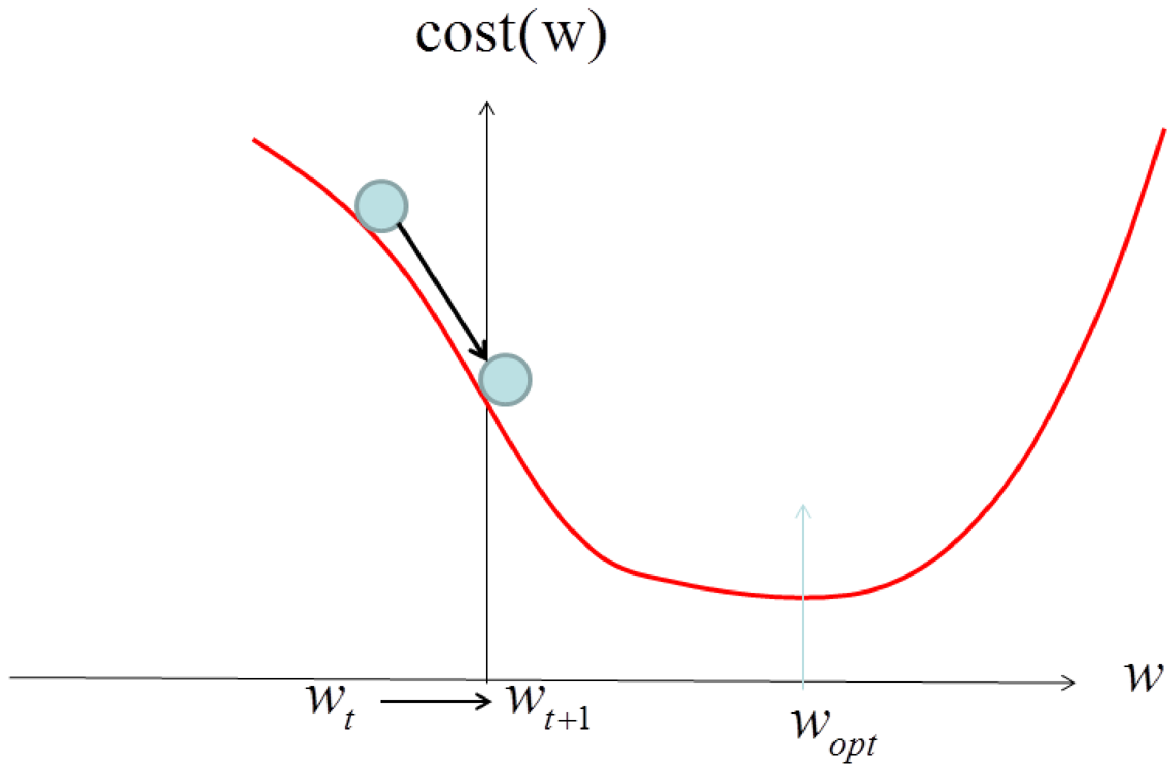
- Initialize parameters (typically small random values)
- In each learning step, change the parameters such that the cost function decreases
- Gradient descent: adapt the parameters in the direction of the negative gradient
- The partial derivative of the weights with respect to the parameters is (Example: w_j)

$$\frac{\partial \text{cost}}{\partial w_j} = - \sum_{i \in \mathcal{M}} y_i x_{i,j}$$

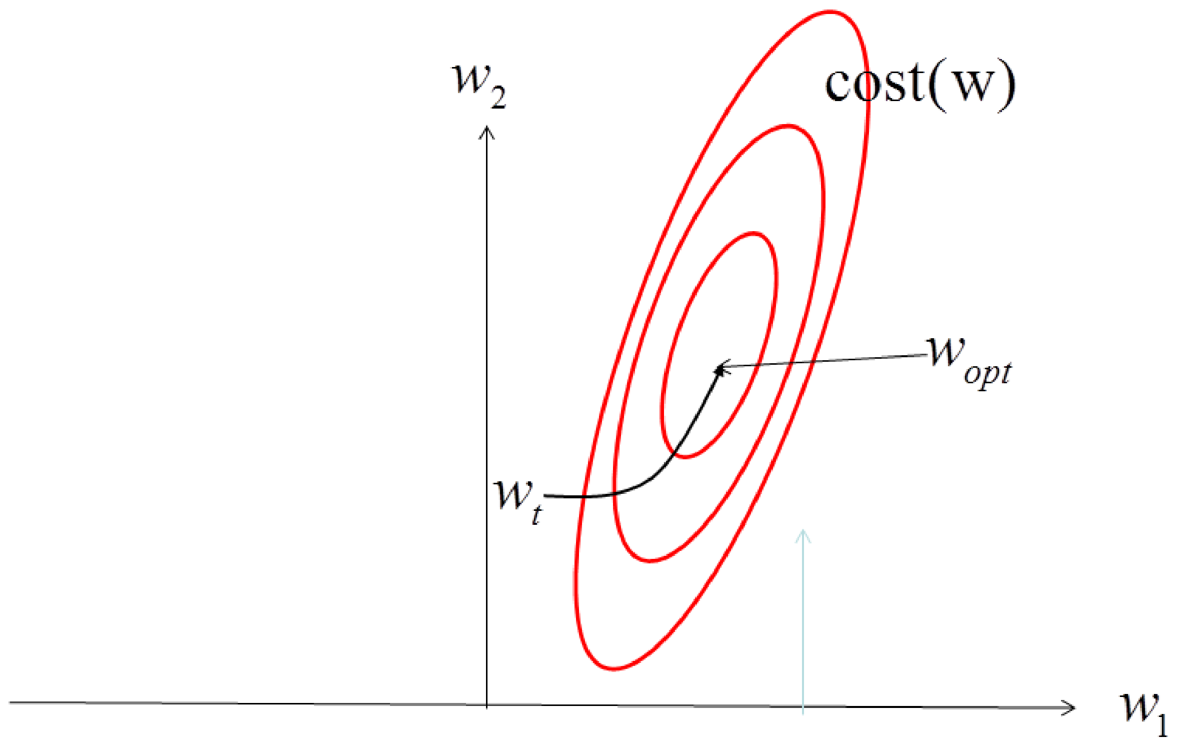
- Thus, a sensible adaptation rule is, $\forall w_j$,

$$w_j \longleftarrow w_j + \eta \sum_{i \in \mathcal{M}} y_i x_{i,j}$$

Gradient Descent with One Parameter



Gradient Descent with Two Parameters



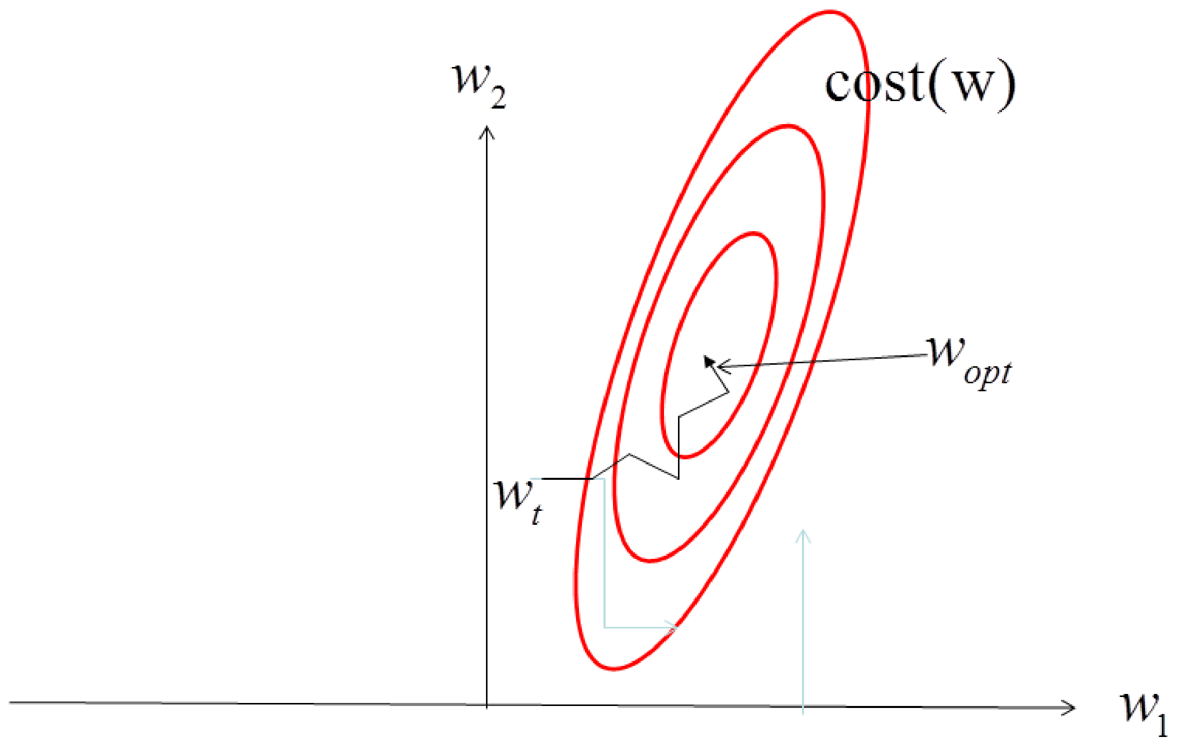
The Perceptron Learning Rule

- In the actual Perceptron learning rule, one presents randomly selected currently misclassified patterns and adapts with only the currently selected pattern. This is biologically more plausible and also leads to faster convergence. Let $\mathbf{x}_t$ and y_t be the training pattern in the t-th step. One adapts $t = 1, 2, \ldots$

$$w_j \longleftarrow w_j + \eta y_t x_{t,j} \quad j = 0, \ldots, M-1$$

- A weight increases, when (postsynaptic) y_t and (presynaptic) $x_{t,j}$ have the same sign; different signs lead to a weight decrease (compare: **Hebb Learning**)
- $\eta > 0$ is the learning rate, typically $0 < \eta << 1$
- Pattern-based learning is also called stochastic gradient descent (SGD)

Stochastic Gradient Descent

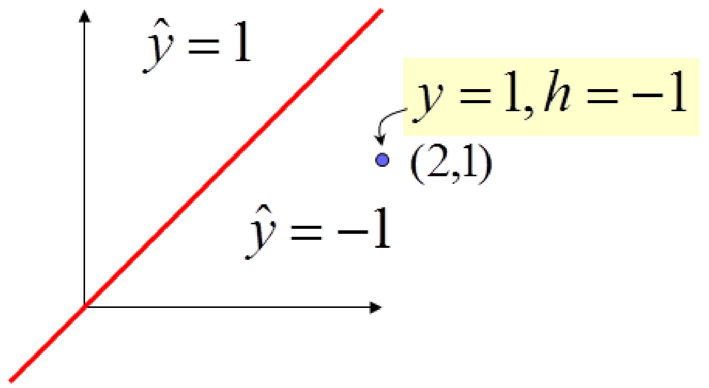


Comments

- Convergence proof: with sufficiently small learning rate η and when the problem is linearly separable, the algorithm converges and terminates after a finite number of steps
- If classes are not linearly separable and with finite η there is no convergence

Example: Perceptron Learning Rule, $\eta = 0.1$

$h = 0 \times 1 - 1 \times x_1 + 1 \times x_2$



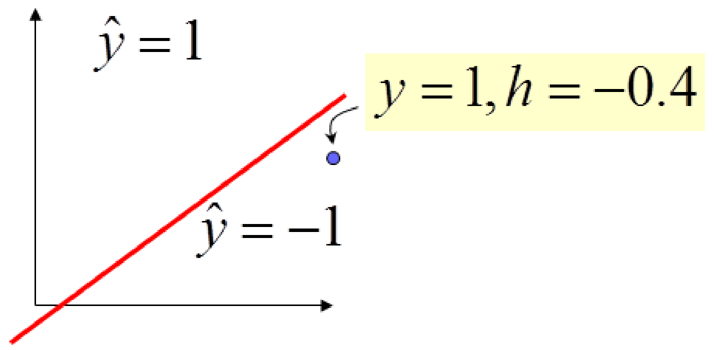
Separation plane prior to adaptation step: $x_2 = x_1$

Adaptation:

$$w_0 \leftarrow 0 + 0.1 \times (1 \times 1) = 0.1$$
$$w_1 \leftarrow -1 + 0.1 \times (1 \times 2) = -0.8$$
$$w_2 \leftarrow 1 + 0.1 \times (1 \times 1) = 1.1$$

"Hebb": all parameters increase

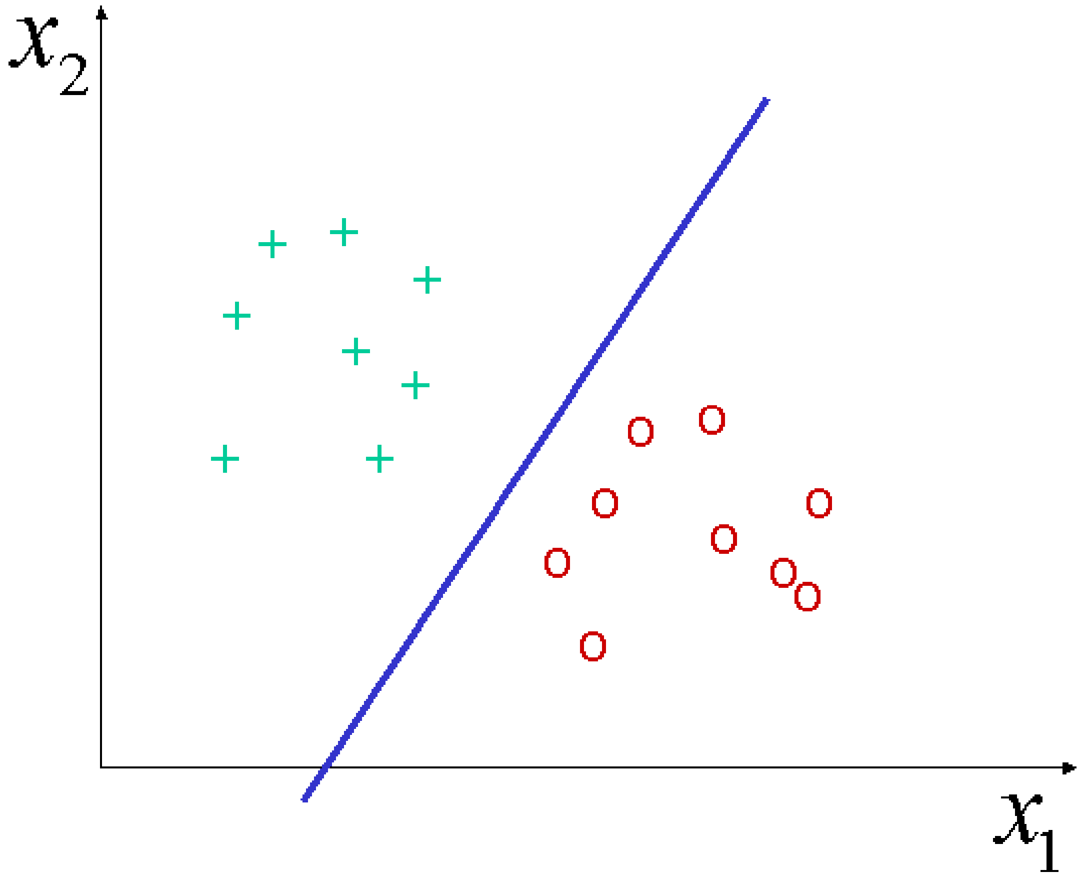
$h = 0.1 \times 1 - 0.8 \times x_1 + 1.1 \times x_2$



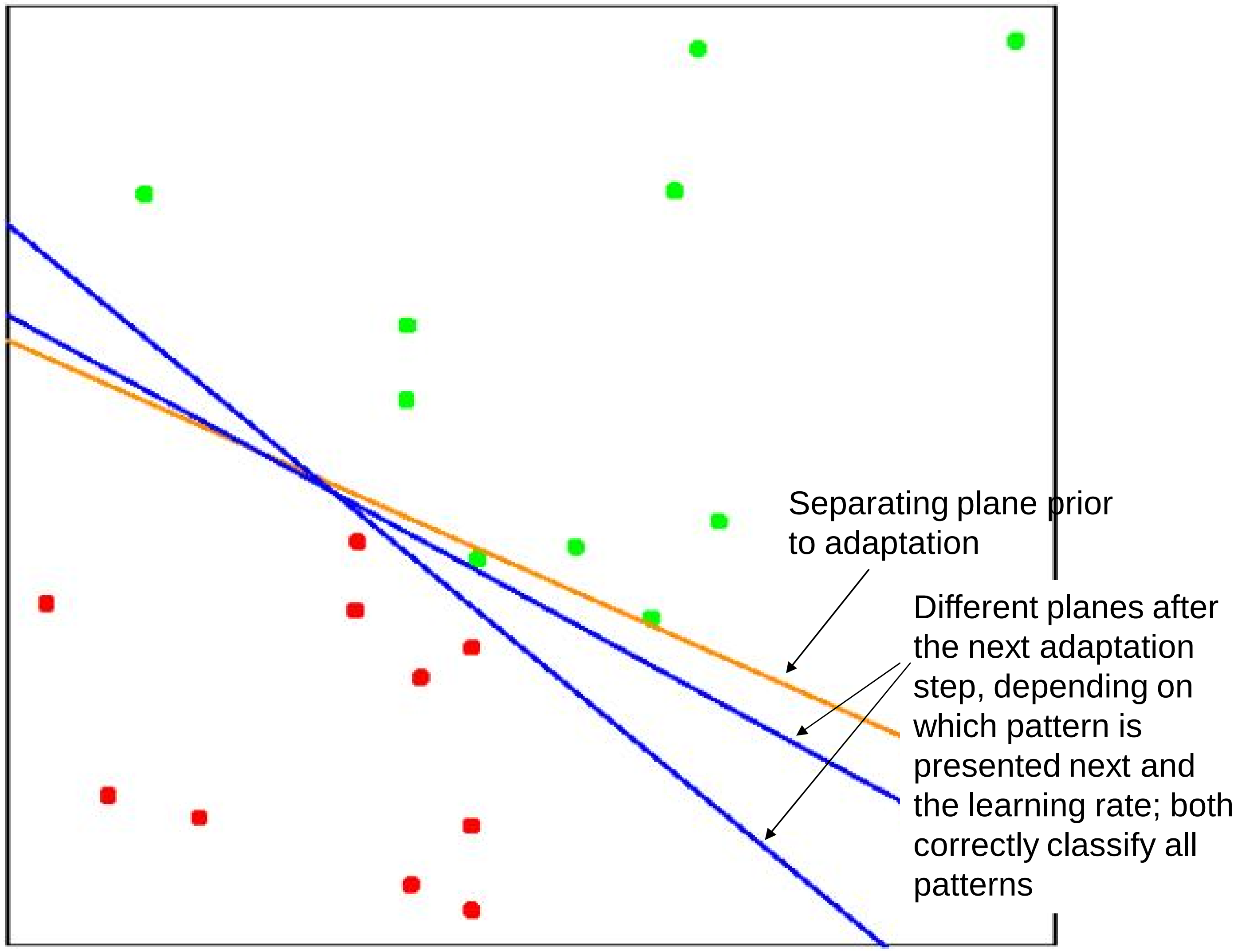
Separating plane after adaptation step:

$x_2 = -0.09 + 0.72 \times x_1$

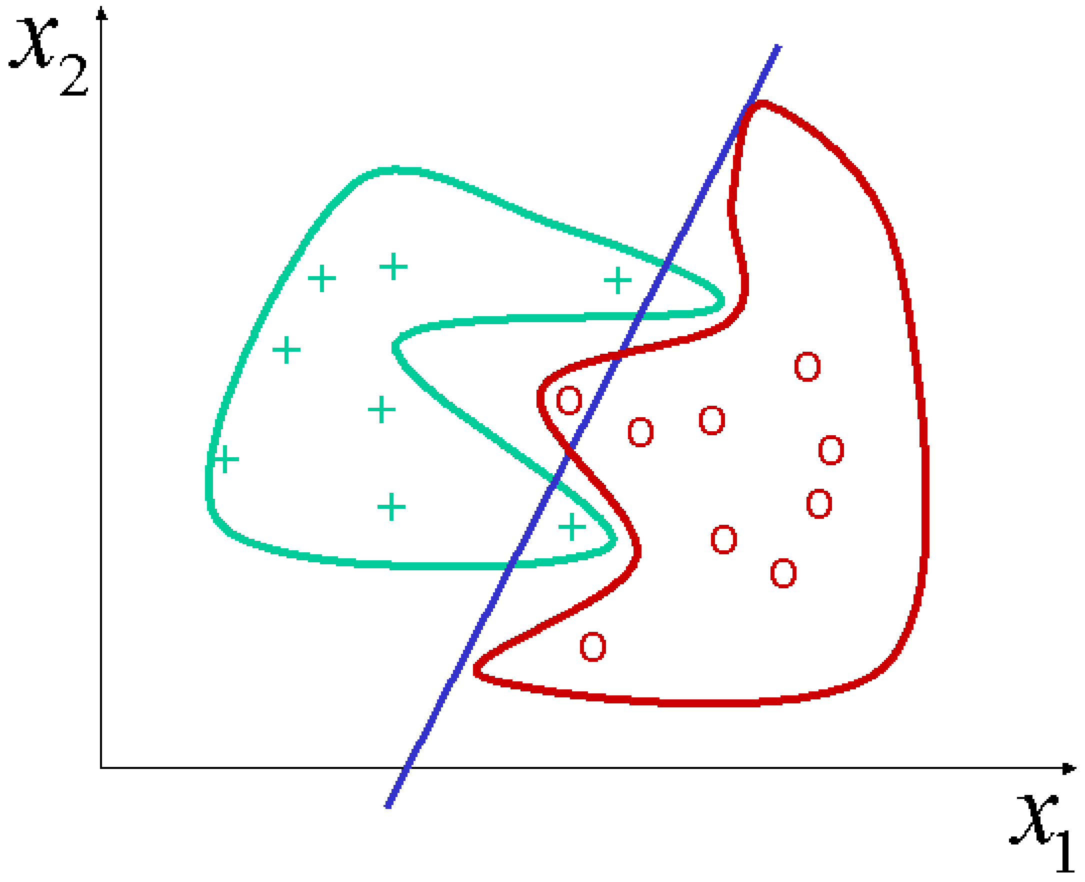
Linearly Separable Classes



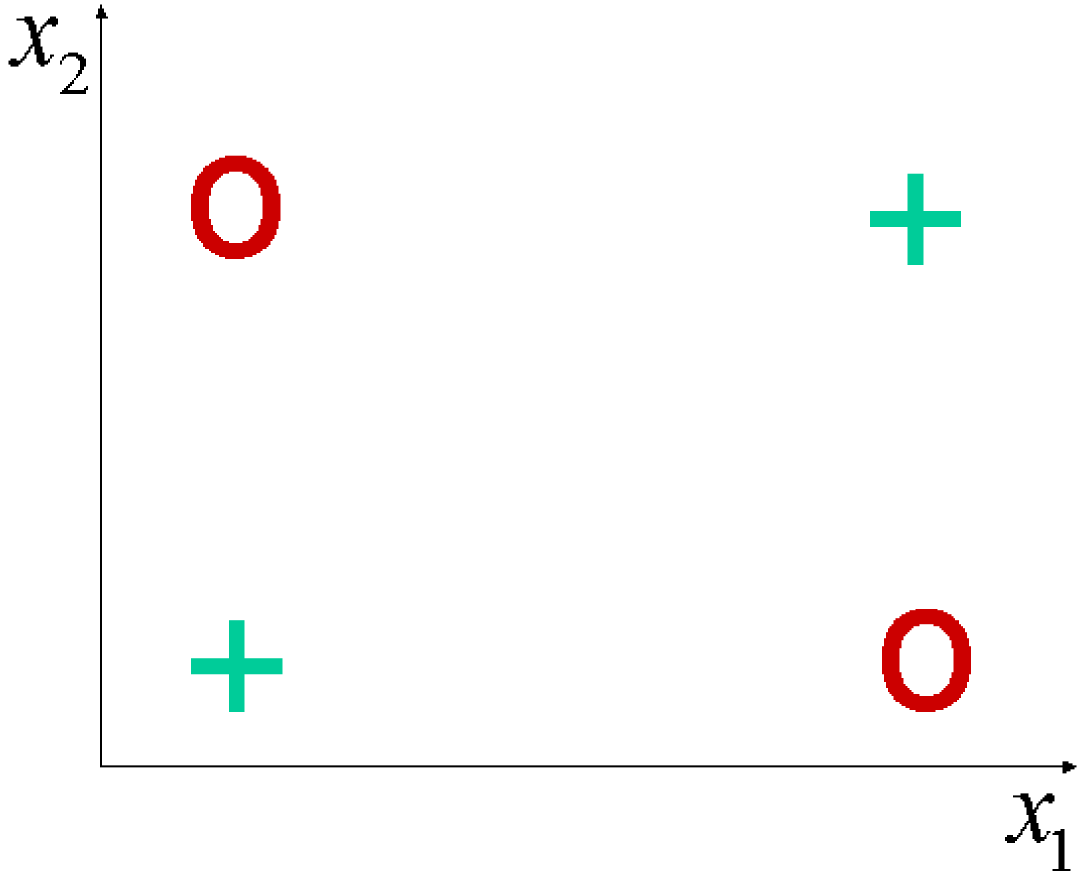
Convergence and Non-uniqueness


Separating plane prior to adaptation
Different planes after the next adaptation step, depending on which pattern is presented next and the learning rate; both correctly classify all patterns

Two Classes with Data Distributions that Cannot be Separated with a Linear Classifier



The Classical Example for Linearly Non-Separable Classes: XOR



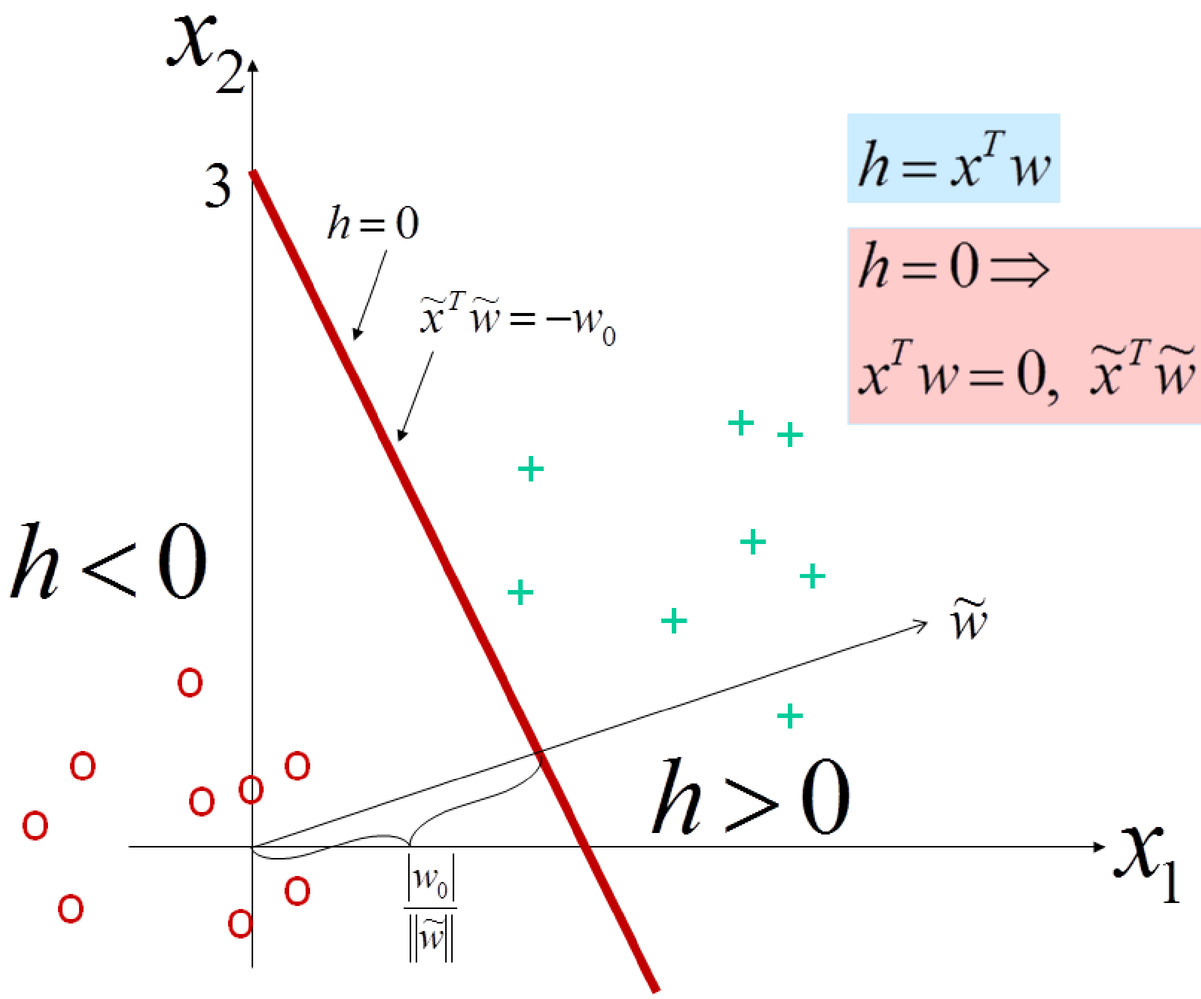
Learning Behavior for Nonseparable Classes

- The cost is minimum if as many data points as possible are correctly classified
- For the misclassified data points, $|h|$ should be small; this is achieved with $||\mathbf{w}|| \rightarrow 0$ which leads to instable learning behavior

Perceptron as a Pattern Recognition System

- In vector notation (introduced in a following lecture), the geometric relationships can easily be describes
- Let, $\mathbf{x} = (x_0, x_1, \ldots, x_{M-1})^T$, $\mathbf{w} = (w_0, w_1, \ldots, w_{M-1})^T$
- Let, $\tilde{\mathbf{x}} = (x_1, \ldots, x_{M-1})^T$, $\tilde{\mathbf{w}} = (w_1, \ldots, w_{M-1})^T$ (same, but without the first dimension)
- When the length is fixed, an input vector $\tilde{\mathbf{x}}$ in the direction $\tilde{\mathbf{w}}$ has the largest activation
- The distance between the separating hyperplane and the origin is $|w_0|/\|\tilde{\mathbf{w}}\|$

Geometric Relationships



$h = x^T w$

$h = 0 \Rightarrow$

$x^T w = 0,\ \tilde{x}^T \tilde{w} = -w_0$

Comments on the Perceptron

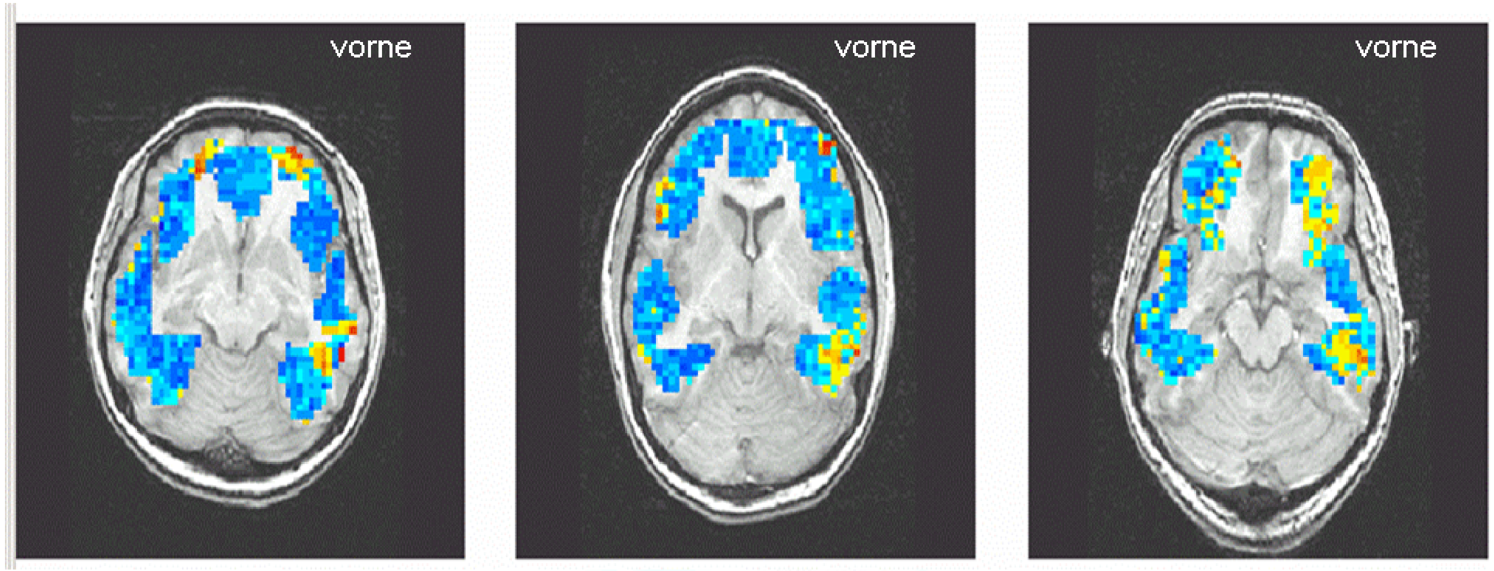
- With separable classes, convergence can be very fast
- A linear classifiers is a very important basic building block: with $M \to \infty$ most problems become linearly separable!
- In some case, the data are already high-dimensional with $M > 10000$ (e.g., number of possible key words in a text)
- In other cases, one first transforms the input data into a high-dimensional (sometimes even infinite) space and applies the linear classifier in that space: basis functions, kernel trick, Neural Networks
- Considering the power of a single formalized neuron: how much computational power might 100 billion neurons possess?
- Are there *grandmother cells* in the brain? Or grandmother areas?

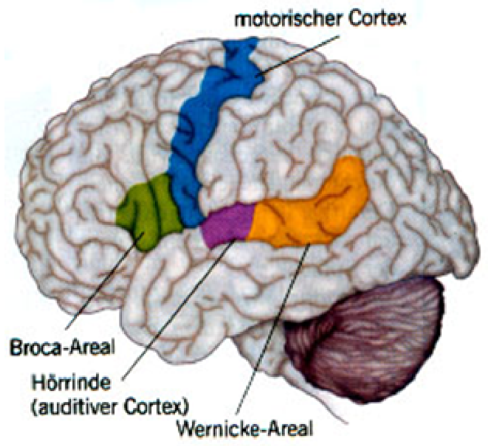
Comments on the Perceptron (cont'd)

- The Perceptron learning rule is not much used any more
  - No convergence, when classes are not separable
  - Classification boundary is not unique, even in the case of separable classes
- Alternative learning rules:
  - Optimal separating hyperplanes (Linear Support Vector Machine)
  - Fisher Linear Discriminant
  - Logistic Regression

Application for a Linear Classifier; Analysis of fMRI Brain Scans (Tom Mitchel et al., CMU)

- Goal: based on the fMRI image slices, determine if someone thinks of tools, buildings, food, or a large set of other semantic concepts
- The trained linear classifier is 90% correct and can, e.g., predict if someone reads about tools or buildings
- The figure shows the voxels, which are most important for the classification task. All three test persons display similar regions


vorne
vorne
vorne


motorischer Cortex
Broca-Areal
Hörrinde
(auditiver Cortex)
Wernicke-Areal

Pattern Recognition Paradigm

- von Neumann: *... the brain uses a peculiar statistical language unlike that employed in the operation of man-made computers...*
- A classification decision is done by considering the complete input pattern, and NOT as a logical decision based on a small number of attributes nor as a complex logical programm
- The linearly weighted sum corresponds more to a voting: each input has either a positive or a negative influence on the classification decision
- Robustness: in high dimensions, a single —possible incorrect— input has little influence

Epilog

Why Pattern Recognition?

- Alternative approach to pattern recognition: learning of simple close-to deterministic rules (naive expectation)
- One of the big mysteries in machine learning is why rule learning is not very successful in predictive systems, although they might be useful to gain a degree of insight in a domain
- Problems: the learned rules are often either trivial, known, or extremely complex and very difficult to interpret
- This is in contrast to the general impression that the world is governed by simple rules
- Also: computer programs, machines ... follow simple deterministic rules? How else could one drive a car?

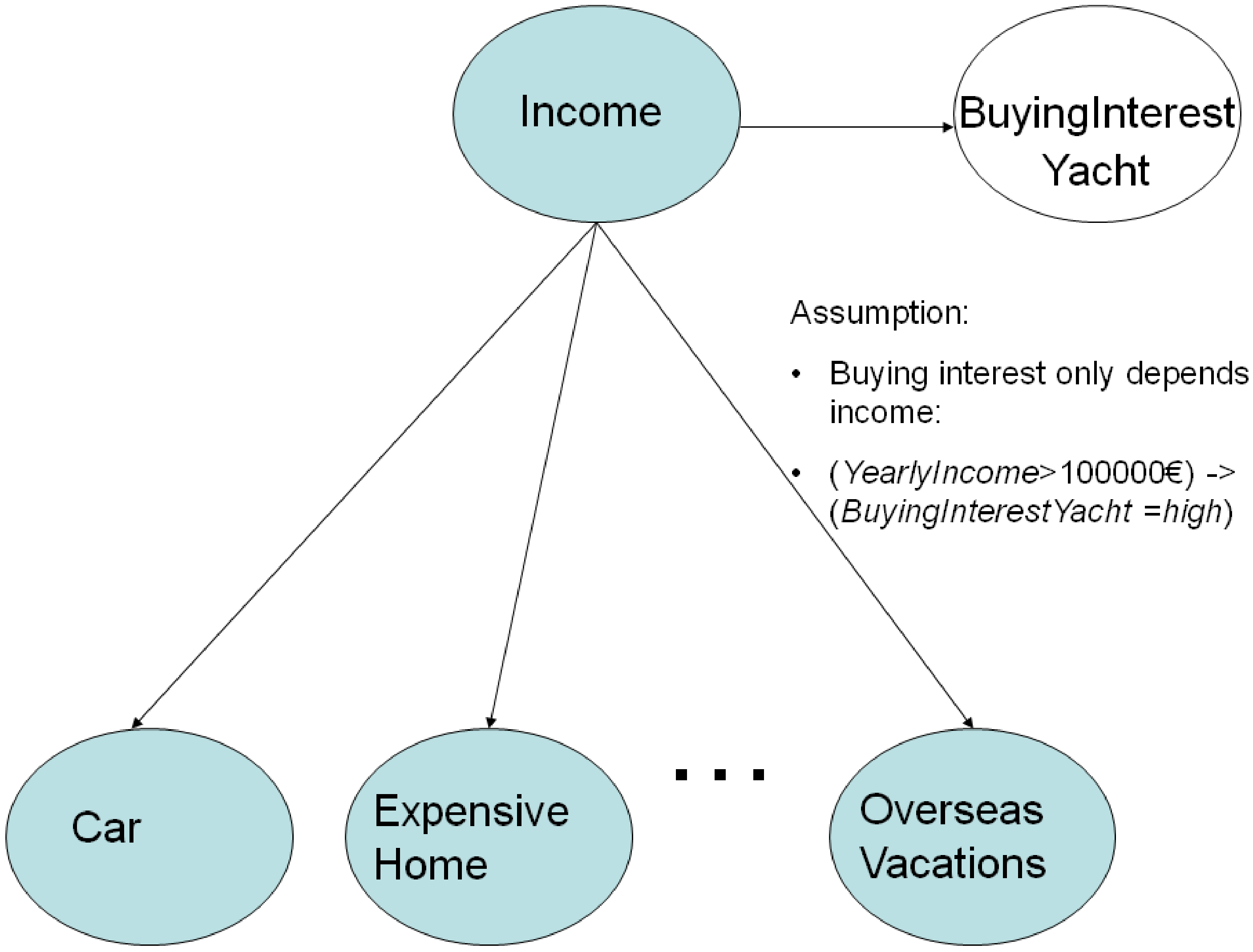
Example: Birds Fly

- Define flying: using its own force, a distance of at least 20m, at leat 1m high, at least once every day in its adult life, ...
- A bird can fly if,
  - it is not a penguin, or ....
  - it is not seriously injured or dead
  - it is not too old
  - the wings have not been clipped
  - it does not have a number of diseases
  - it only lives in a cage
  - it does not carry heavy weights
  - ...

Pattern Recognition

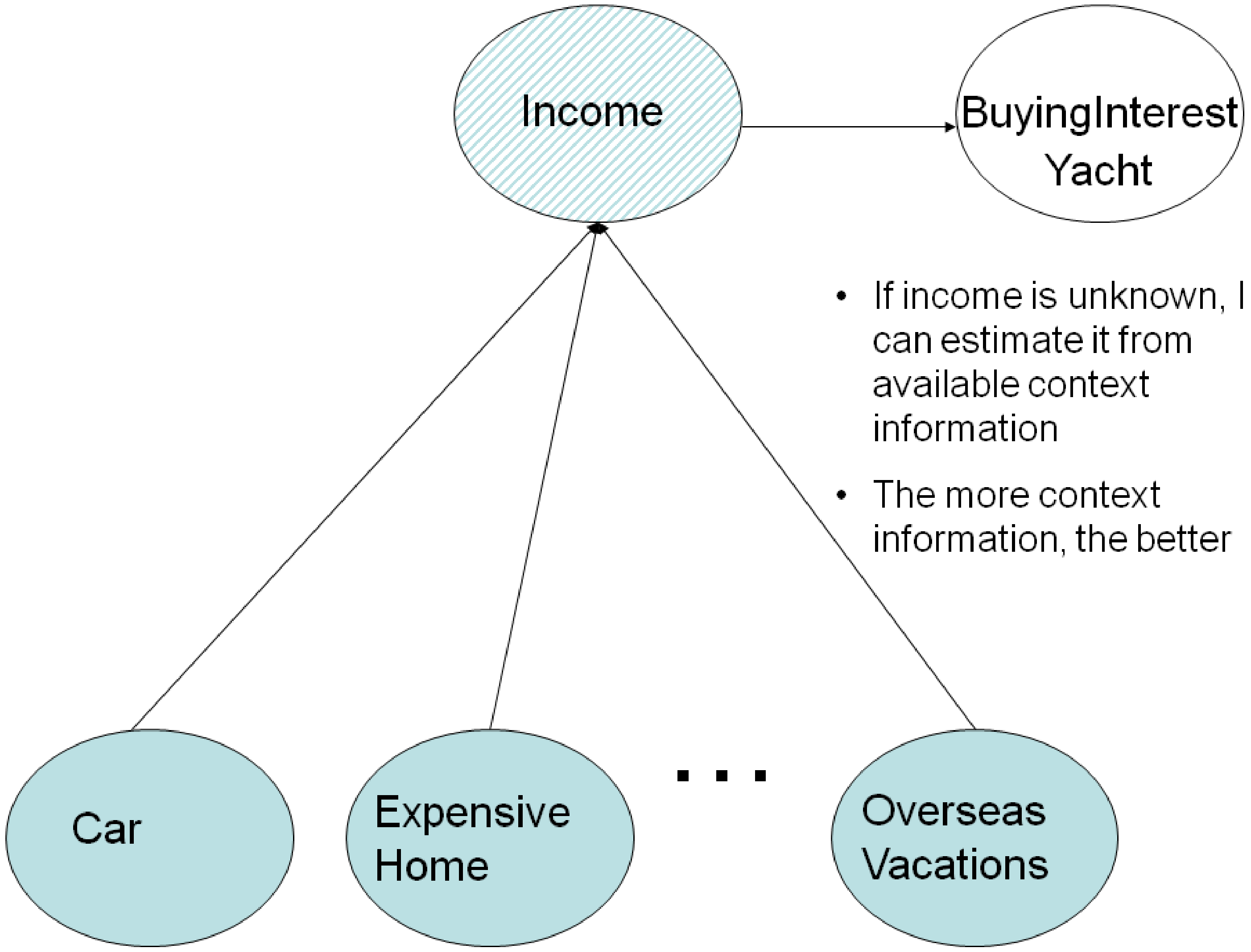
- 90% of all birds fly
- Of all birds which do not belong to a flightless class 94% fly
- ... and which are not domesticated 96% ...
- Basic problem:
  - Complexity of the underlying (deterministic) system
  - Incomplete information
- Thus: success of statistical machine learning!

Example: Predicting Buying Pattern



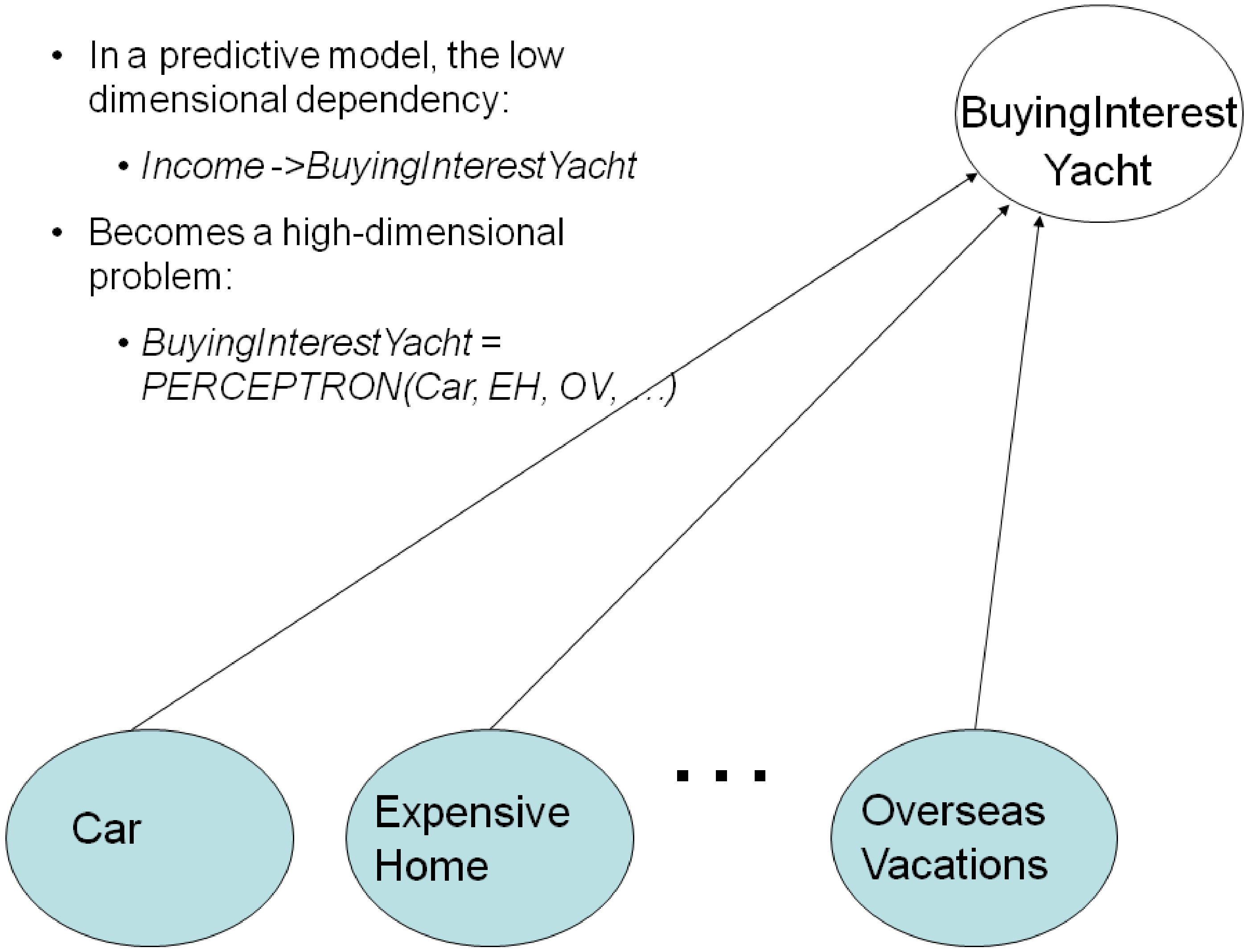
Assumption:

- Buying interest only depends on income:
- (*YearlyIncome*>100000€) -> (*BuyingInterestYacht* =*high*)



- If income is unknown, I can estimate it from available context information
- The more context information, the better

- In a predictive model, the low dimensional dependency:
  - *Income ->BuyingInterestYacht*
- Becomes a high-dimensional problem:
  - *BuyingInterestYacht = PERCEPTRON(Car, EH, OV, …)*



Conclusion

- One reason, why dependencies tend to be high-dimensional and statistical in nature is that the input of interest, here income, is not available (latent, hidden); latent variable models attempt to estimate these latent factors (see lecture on PCA); note, that also the target "buying interest" is often latent (unless, e.g., asked for in a questionnaire)
- Another reason is that many individual factors contribute to a decision: as discussed, the Perceptron can be thought of as a weighted voting machine
- Exceptions (where rule-learning might works as predictive systems):
  - Technical human generated worlds ("Engine A always goes with transmission B")
  - Tax forms (although for a "just" fined-tuned tax system, the rules become complex again); legal system; business rules (again: human generated)
  - Natural laws under idealized conditions; under real conditions (friction, wind, ...) laws become complex again; rules in chemistry; weather forecasting is done with stochastic natural laws, but can also be done via purely statistical methods

- Also language seems to happen more on the deterministic level; although language is often used to justify a decision, and not to explain a decision. In making a decision (vacationing in Italy), many issues are considered; after the decision, the result is often communicated as a simple rule: “We will go to Italy, because it has the best beaches!”