The Perceptron

Volker Tresp

Summer 2021

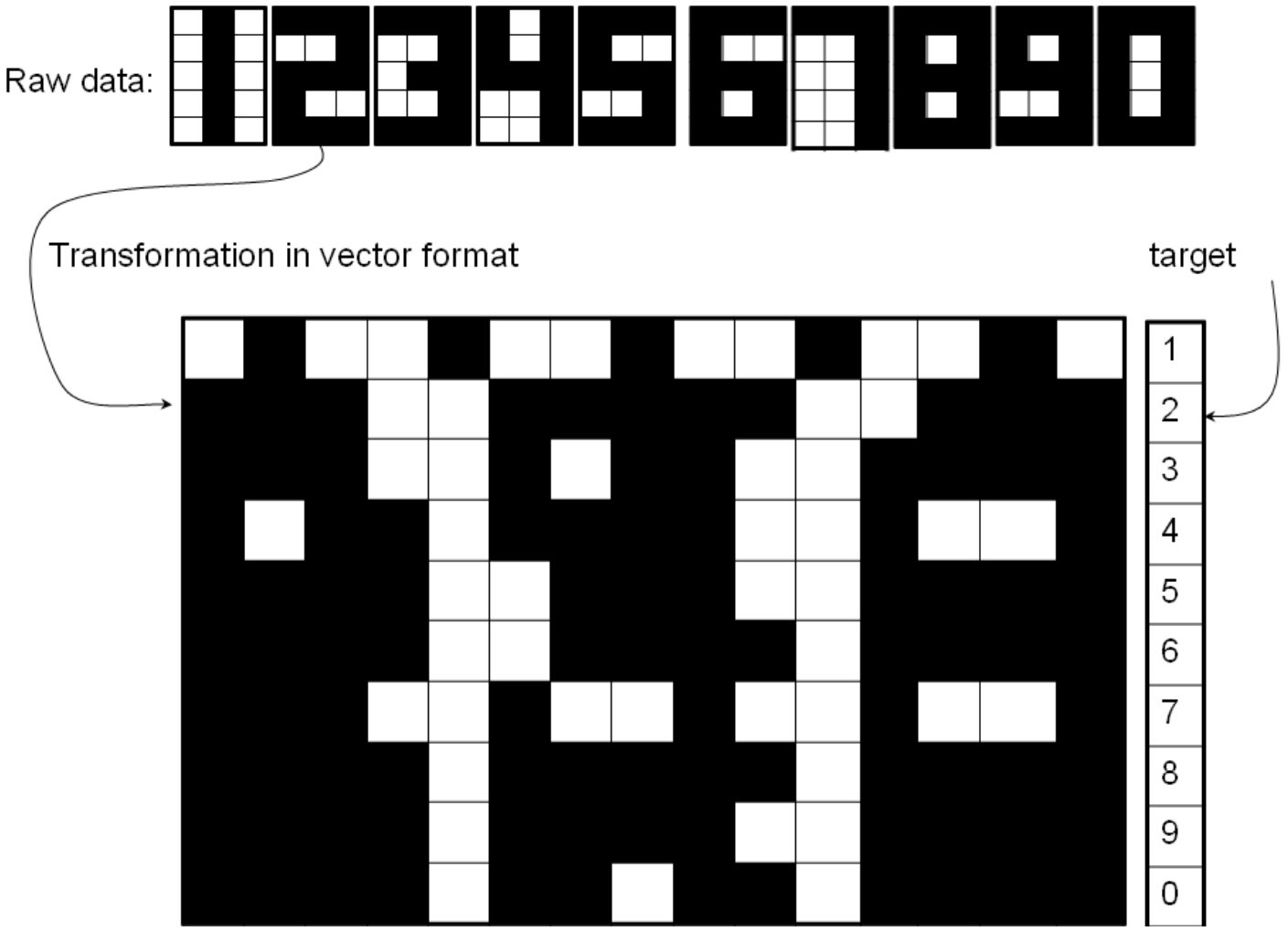
Elements in Learning Tasks

- 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) (**model structure learning**: search about model structure)
- Selection of a **cost function** which is a function of the data and the free parameters (e.g., a score related to the number of misclassifications in the training data as a function of the model parameters); a good model has a low cost
- **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

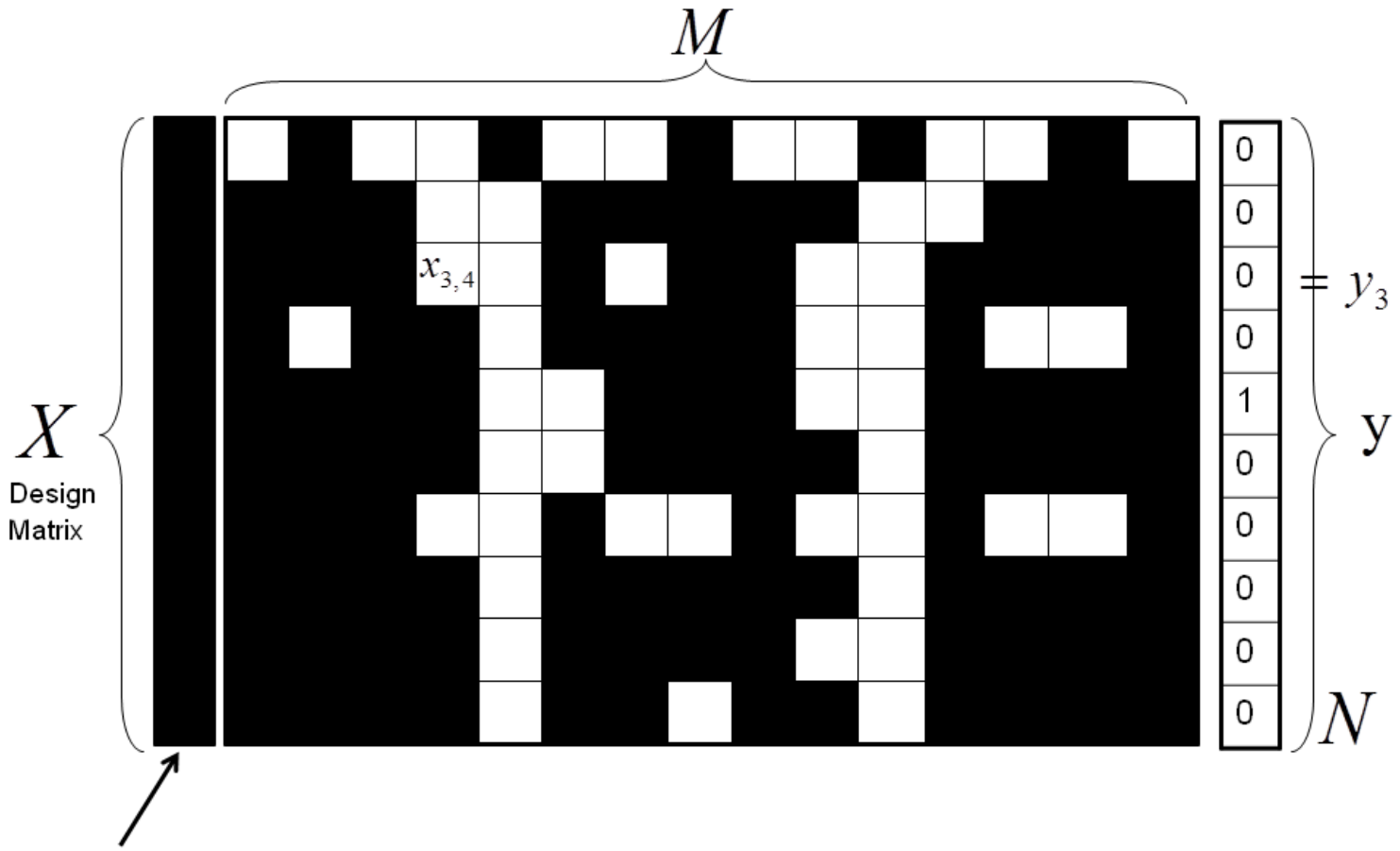
Prototypical Learning Task

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

Reshaping 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

Supervised Machine Learning

- $\mathbf{x} \in \mathbb{R}^M$ is an input, i.e., an element in an M-dimensional vector space;
- $\mathbf{x}_i$ is a specific input; it might represent the features of an entity i (e.g., age, employer)
- Consider a function $f : \mathbb{R}^M \to \mathbb{R}$
- We have $\mathbf{x} \mapsto f(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^M$, and $y \in \mathbb{R}$
- Consider $f(\mathbf{x}_i)$ to be a predicted feature of entity i (e.g., income)

Expected Output

- In general, we are interested in the expected output $y \in \mathbb{R}$, given that we know $\mathbf{x}$

$$f(\mathbf{x}) = \mathbb{E}(y|\mathbf{x})$$

- Example 1: if $\mathbf{x}$ represents a **machine generated** written digit, and y indicates if the digit is a "5" ($y = 1$) or another digit ($y = 0$), then $f(\mathbf{x}) \in \{0, 1\}$, since it is possible to classify machine generated written digits perfectly
- Example 2: if $\mathbf{x}$ represents a **human generated** written digit, and y indicates if the digit is a "5" ($y = 1$) or another digit ($y = 0$), then $f(\mathbf{x}) \in [0, 1]$, since it is impossible to classify human generated written digits perfectly

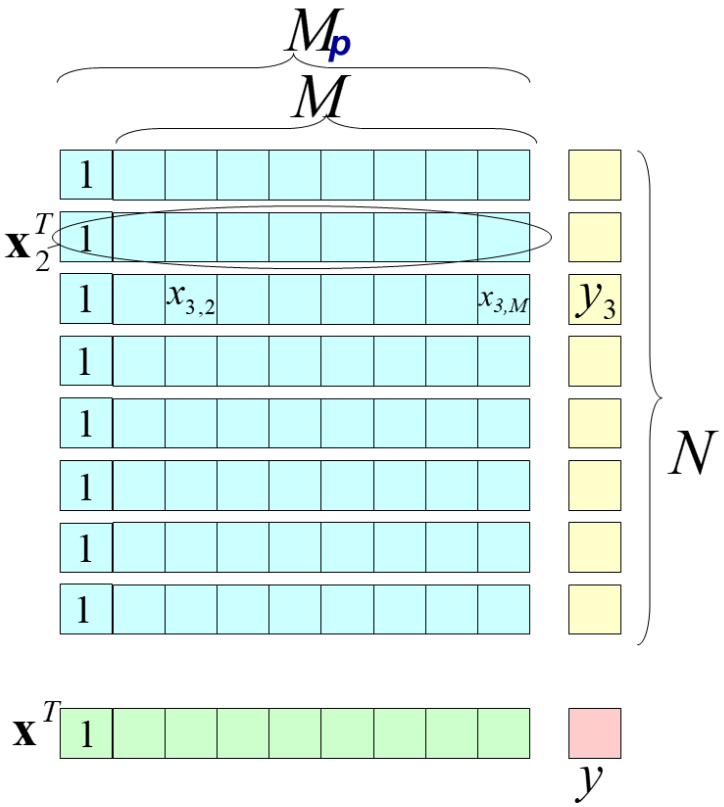
Machine Learning Setting (cont'd)

- The function $f(\mathbf{x})$ is sometimes called the "true function" or the "underlying function" or the "data generating function", or, in engineering, the system
- $f_{\mathbf{w}}(\mathbf{x})$ is a model (function) with parameter vector $\mathbf{w} \in \mathbb{R}^{M_P}$; M_P is the number of model parameters
- The goal of learning is to find a $\hat{\mathbf{w}} \in \mathbb{R}^{M_P}$ such that

$$f_{\hat{\mathbf{w}}}(\mathbf{x}) \approx f(\mathbf{x})$$

- In supervised learning, we "learn" $\hat{\mathbf{w}}$, based on a training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where N is the number of training data points

Data Matrix for Supervised Learning



M	number of inputs (input attributes)
M_p	number of free parameters
N	number of training patterns
$\mathbf{x}_i$	$= (x_{i,0}, \ldots, x_{i,M})^T$ input vector 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 all targets
$\mathbf{d}_i$	$= (x_{i,0}, \ldots, x_{i,M}, y_i)^T$ i-th pattern
D	$= \{\mathbf{d}_1, \ldots, \mathbf{d}_N\}$ (training data)
$\mathbf{x}$	$= (x_0, x_1, \ldots, x_M)^T$, generic (test) input
y	target for $\mathbf{x}$
$\widehat{y} = f_{\mathbf{w}}(\mathbf{x})$	model estimate

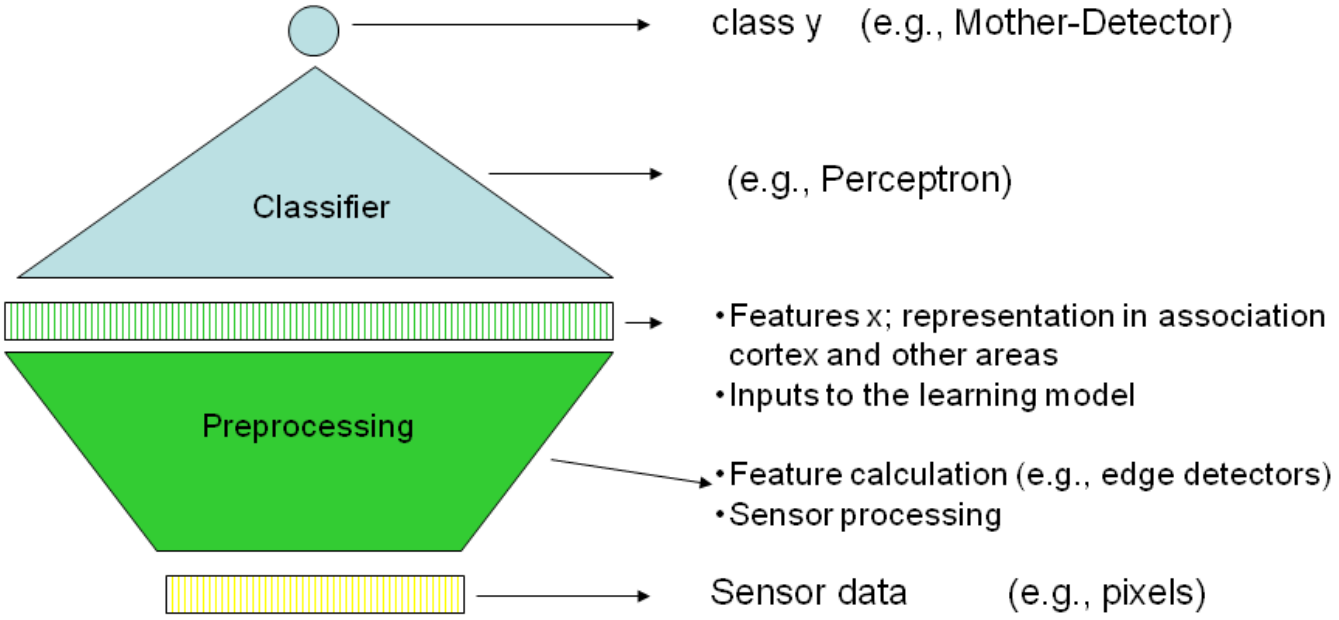
Fine Details on the Notation

- $\mathbf{x}$ 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 $\mathbf{x}_i$
- $x_{i,0} = x_0 = 1$
- $\mathbf{y}$ is the vector of all targets
- Also note that $\mathbf{x}_i$ is a **column** vector but it appears as a **row** in X
- For linear models, $M_p = M + 1$, i.e., the number of free parameters is the number of inputs plus 1

Model

- $f_{\mathbf{w}}(\mathbf{x})$ is a model function with parameters $\mathbf{w}$
- $\mathbf{w}_{opt}$ are the optimal model parameters, based on the training data (according to some cost function; might not be unique)
- $f(\mathbf{x})$ is the "true" but unknown function that (hypothetically) generated the data
- Sometimes (but not in general) the true function can be represented by one to the model functions. Then $\mathbf{w}_{true}$ are the parameters of that model (might not be unique)
- In classical statistics, one is often interested in the distance between $\mathbf{w}_{opt}$ and $\mathbf{w}_{true}$
- In machine learning, one is often interested in the distance between $f_{\mathbf{w}_{opt}}$ and $f_{\mathbf{w}_{true}}$

A Biologically Motivated Model



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

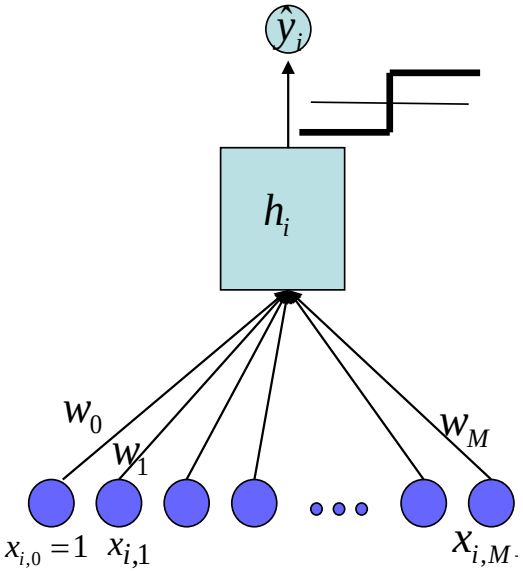
Supervised Learning

- In supervised learning one assumes that in training both inputs X and outputs $\mathbf{y}$ are available; in testing, the model only knows $\mathbf{x}$ and the goal is to predict y
- 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 by 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 Perceptron calculates first a sum of weighted inputs

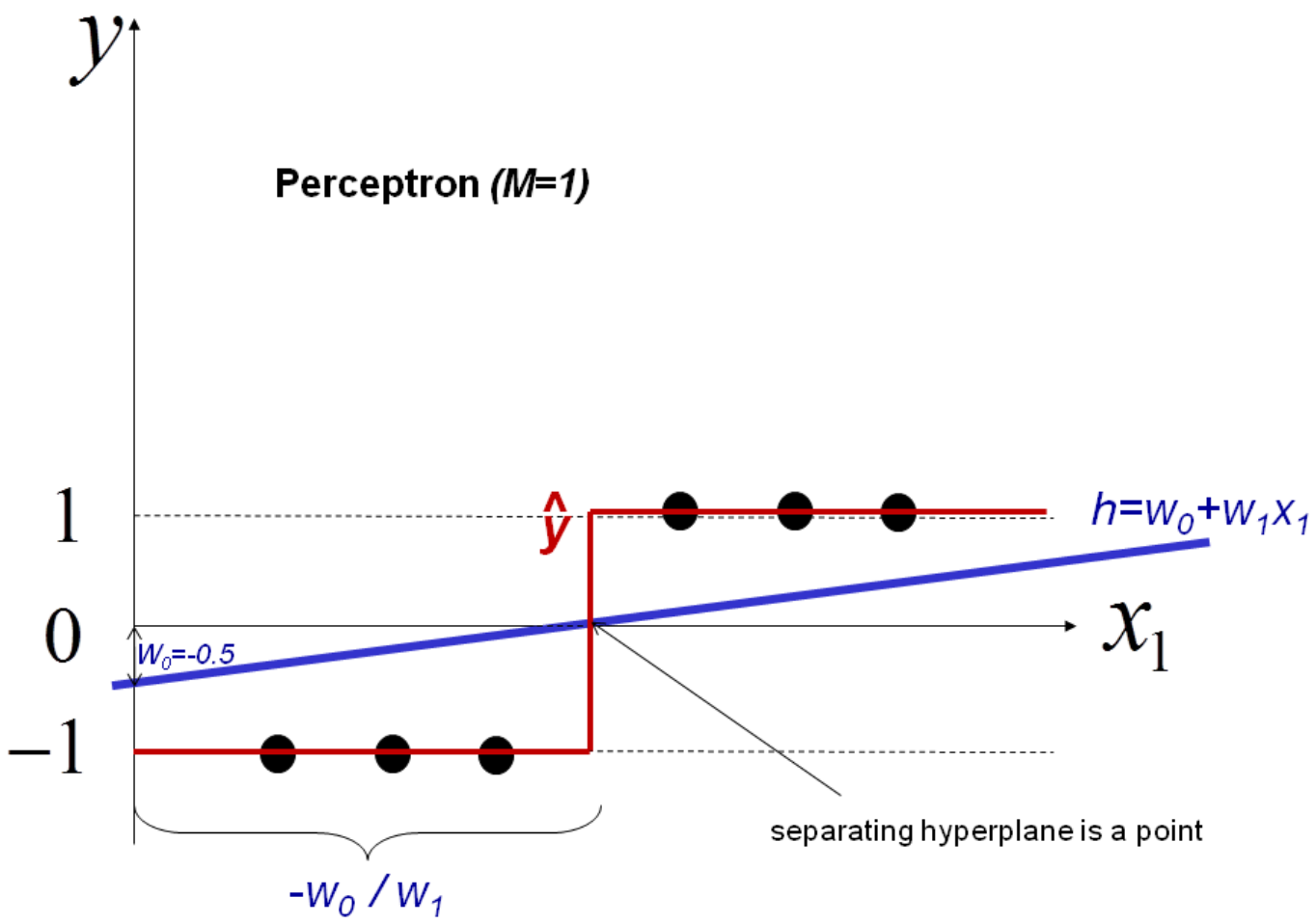
$$h(\mathbf{x}) = \sum_{j=0}^{M} w_j x_j$$

(Note: $x_0 = 1$ is a constant input, such that w_0 can be though of as a bias); in biology this would be the membrane potential; in computer science it is called **net input** "'Net"; some authors call this the **activation**

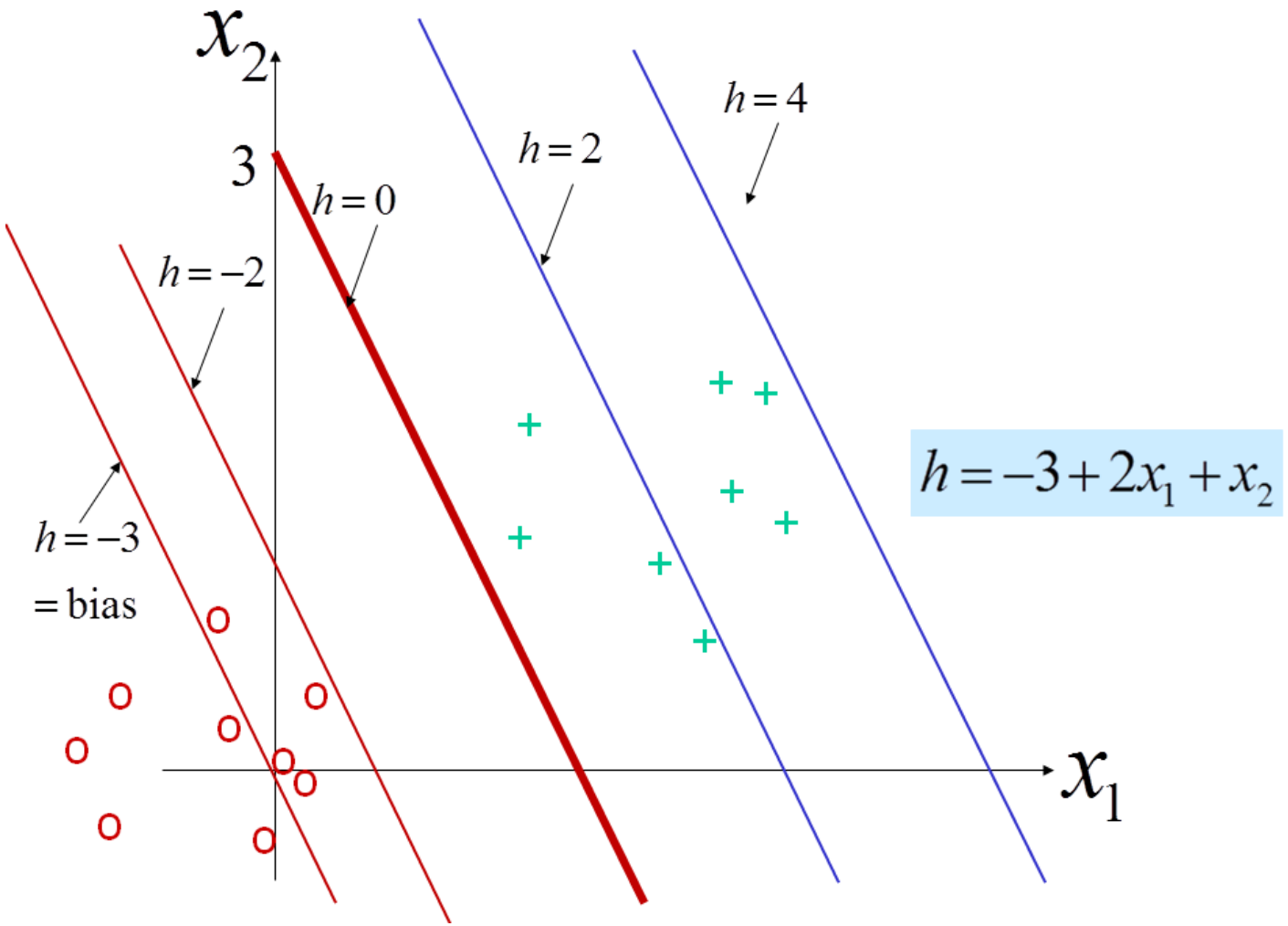
- With **activation function** $\mathsf{sign}(\cdot)$, the binary classification (some authors call this the **activation**) $y \in \{1, -1\}$ is calculated as

$$\hat{y} = \mathsf{sign}(h(\mathbf{x}))$$

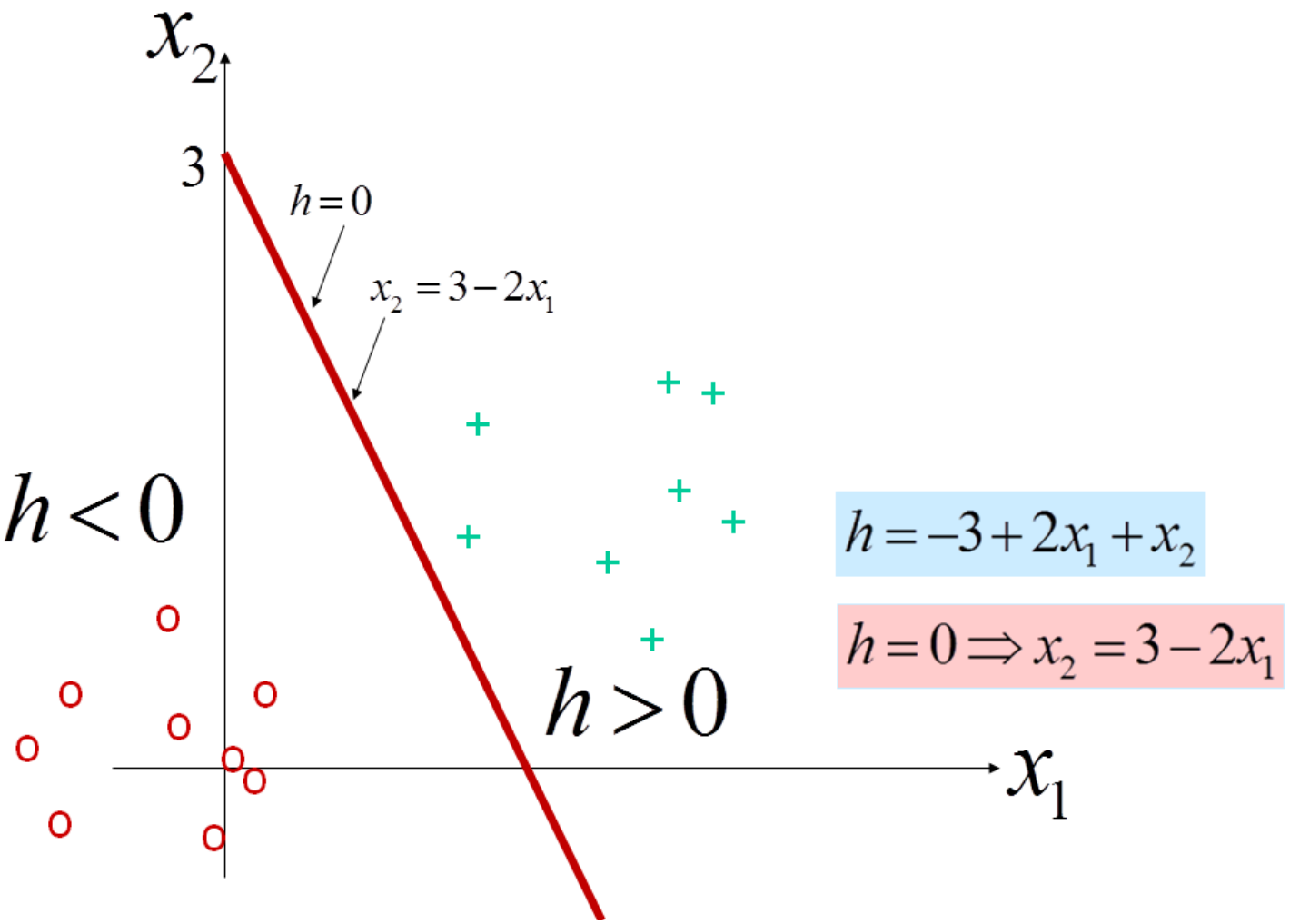
- The linear classification boundary (*separating hyperplane*) is defined by
  $h(\mathbf{x}) = 0$
- Thus here, $f(\mathbf{x}) = \mathsf{sign}(h(\mathbf{x}))$


y
Perceptron (M=1)
1
0
−1
ŷ
h=w0+w1x1
x1
w0=-0.5
separating hyperplane is a point
-w0 / w1

$h\ (M = 2)$



The Decision Boundary / Separating Hyperplane ($M = 2$)



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

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

Linearities

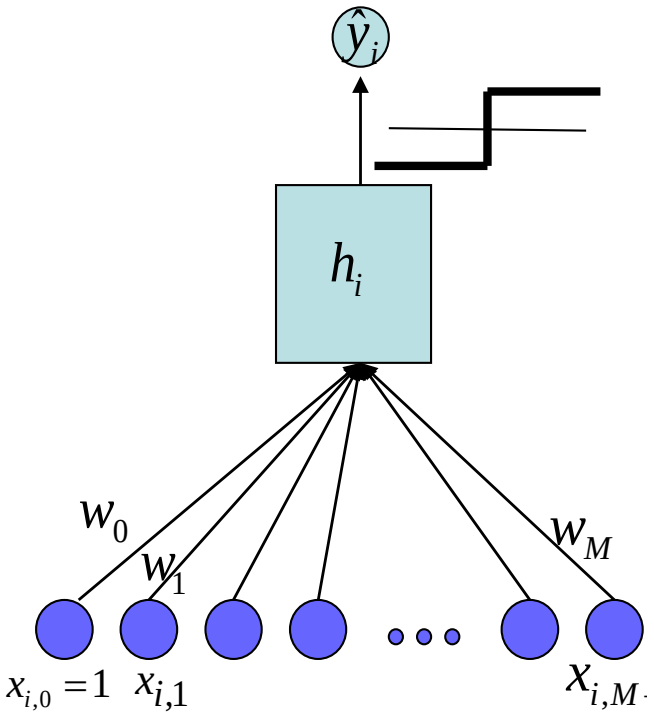
- Not to get confused:
- 1: $h(\mathbf{x})$ is a **linear function**
  - With $M = 1$, $h = w_0 + w_1 x_1$
  - With $M = 2$, $h = w_0 + w_1 x_1 + w_2 x_2$
- 2: $h(\mathbf{x}) = 0$ defines a **linear hyperplane**
  - When $M = 2$, and with

$$h(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2$$

in the separating hyperplane with $h(\mathbf{x}) = 0$, x_2 is a linear function of x_1:

$$x_2 = -\frac{1}{w_2}(w_0 + w_1 x_1)$$

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 $w_0, \ldots, w_M$
- We define a cost function (loss function, objective function) that is dependent on the training data and the parameters. When the training data and the model structure is fixed, it is only a function of the parameters, $\mathsf{cost}(\mathbf{w})$
- In the learning process (training), one attempts to find parameters that minimize the cost function

$$\mathbf{w}_{opt} = \arg\min(\mathsf{cost}(\mathbf{w}))$$

The Perceptron Cost Function

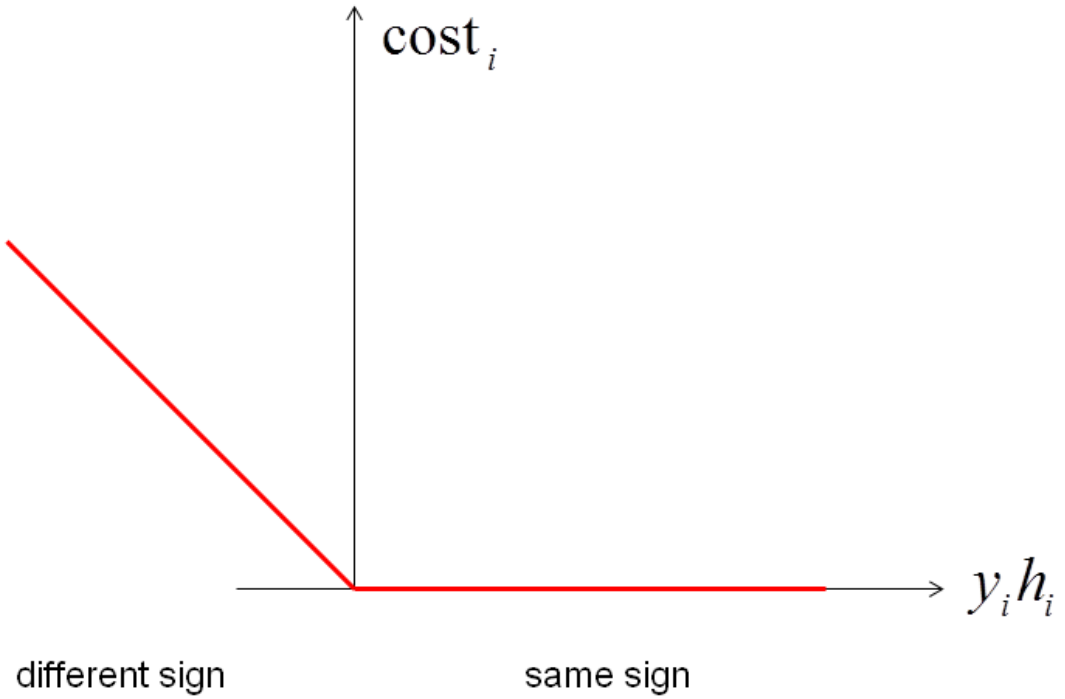
- Goal: correct classification of the N training samples with targets $\{y_1, \ldots, y_N\}$
- The Perceptron cost function is

$$\text{cost} = -\sum_{i \in \mathcal{M}} y_i h(\mathbf{x}_i) = \sum_{i=1}^{N} |-y_i h(\mathbf{x}_i)|_+$$

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

- Here, $h(\mathbf{x}_i) = \sum_{j=0}^{M} w_j x_{i,j}$
- 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 $h(\mathbf{x}_i)$ 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
- With

$$\text{cost} = -\sum_{i \in \mathcal{M}} y_i \left(\sum_{j=0}^{M} w_j x_{i,j} \right)$$

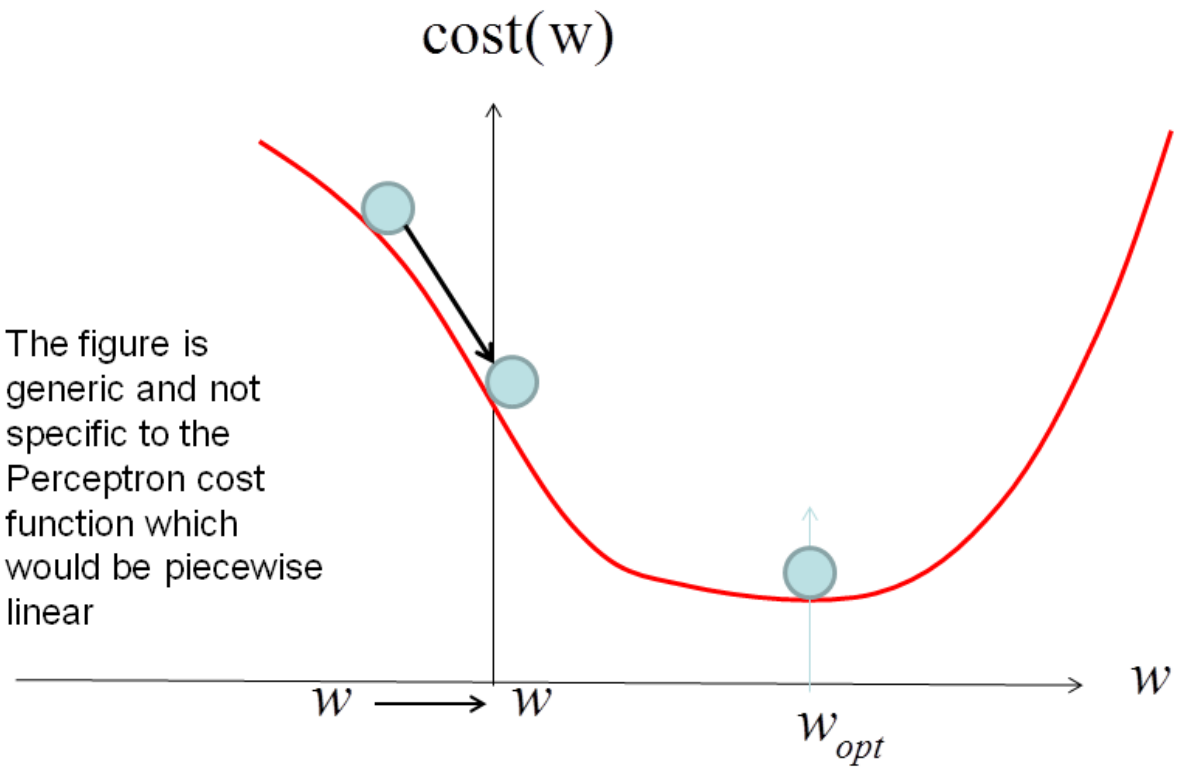
the partial derivative of the weights with respect to the parameters is (Example: w_j)

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

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

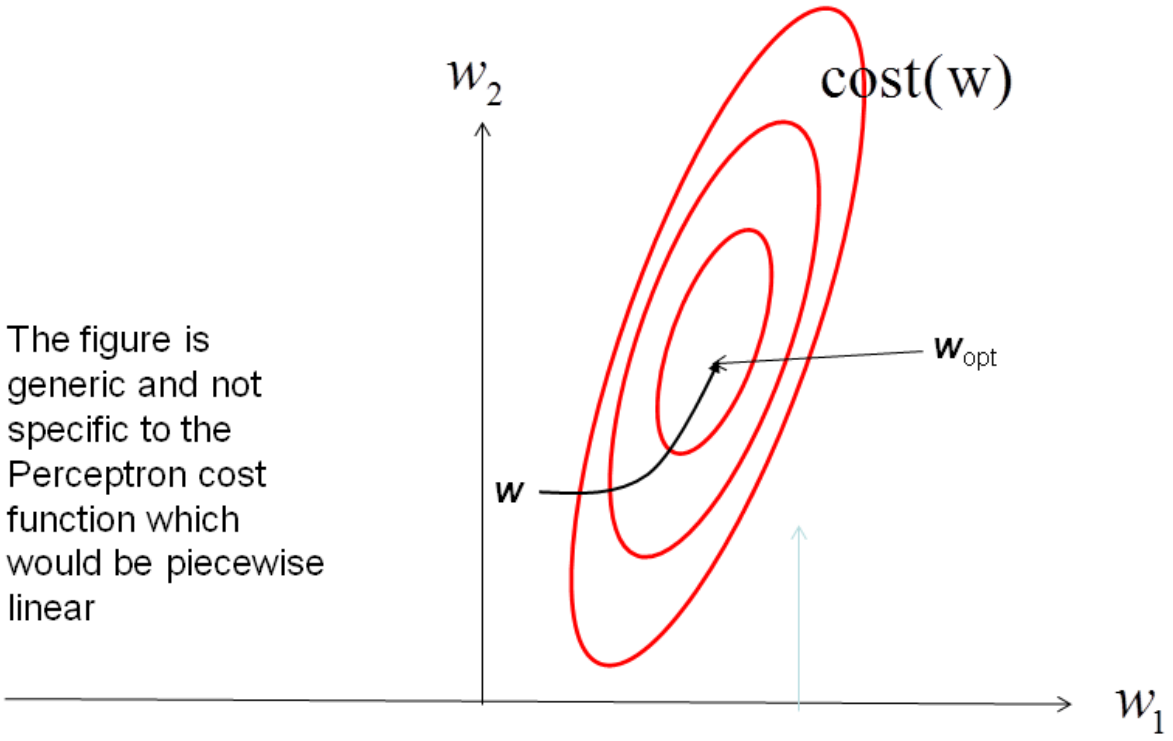
$$w_j \longleftarrow w_j + \eta \sum_{i \in \mathcal{M}} y_i x_{i,j}$$

Gradient Descent with One Parameter



Gradient Descent with Two Parameters

The figure is generic and not specific to the Perceptron cost function which would be piecewise linear



The Perceptron Learning Rule

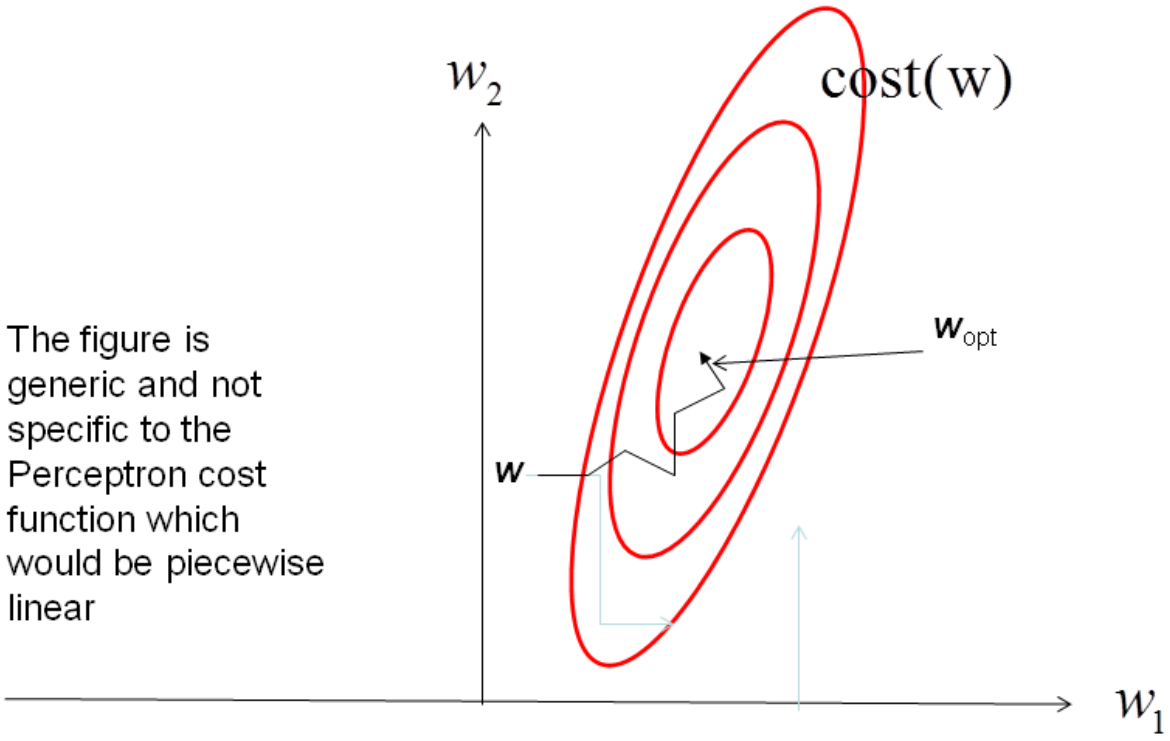
- In the actual Perceptron learning rule, one presents randomly selected currently misclassified patterns and adapts with only the currently selected pattern. This is biologically more plausible and also leads to faster convergence. Let $\mathbf{x}_t$ and y_t be the training pattern in the t-th step.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

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

- In the weight update, 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

The figure is generic and not specific to the Perceptron cost function which would be piecewise linear

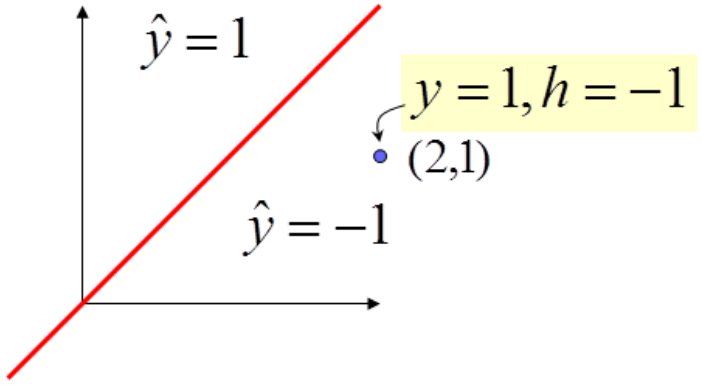


Comments

- 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, $\eta = 0.1$

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



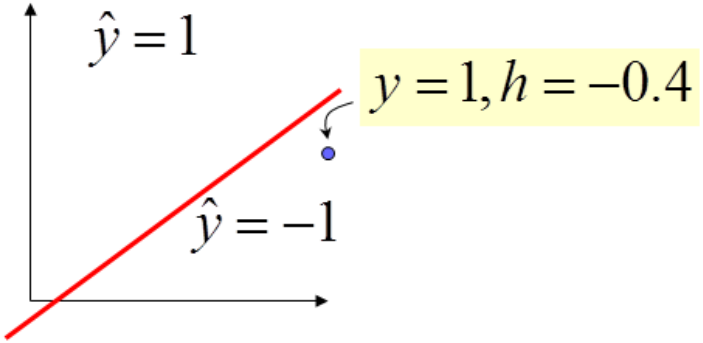
Separation plane prior to adaptation step: $x_2 = x_1$

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

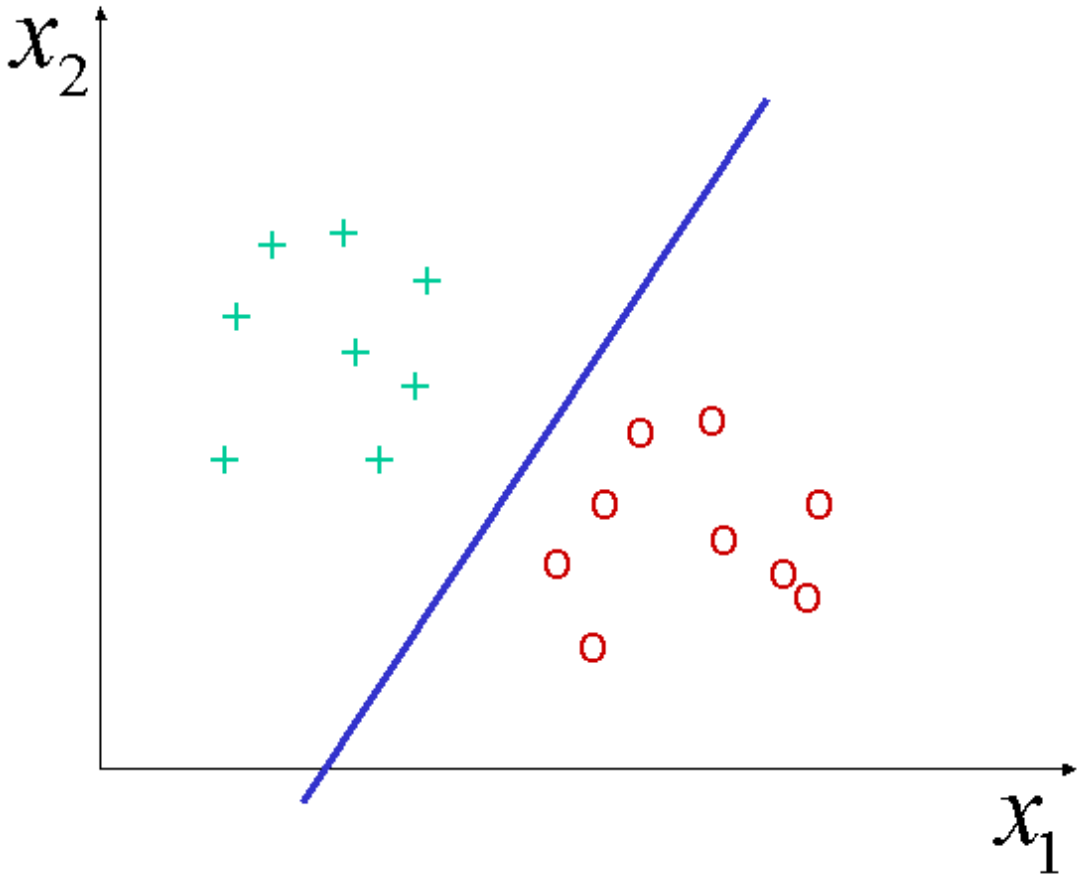
$$h = 0.1 \times 1 - 0.8 \times x_1 + 1.1 \times x_2$$



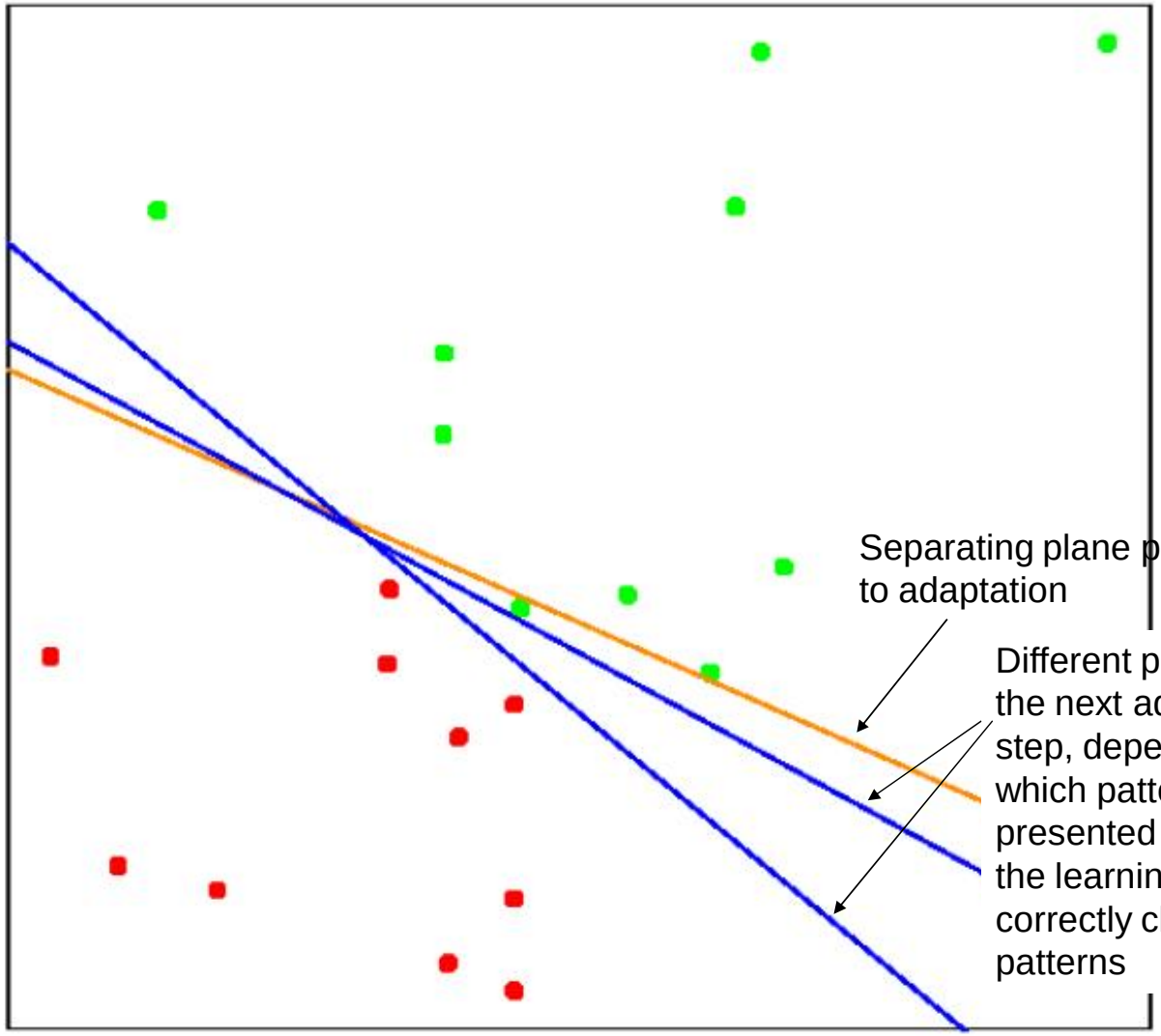
Separating plane after adaptation step:

$$x_2 = -0.09 + 0.72 \times x_1$$

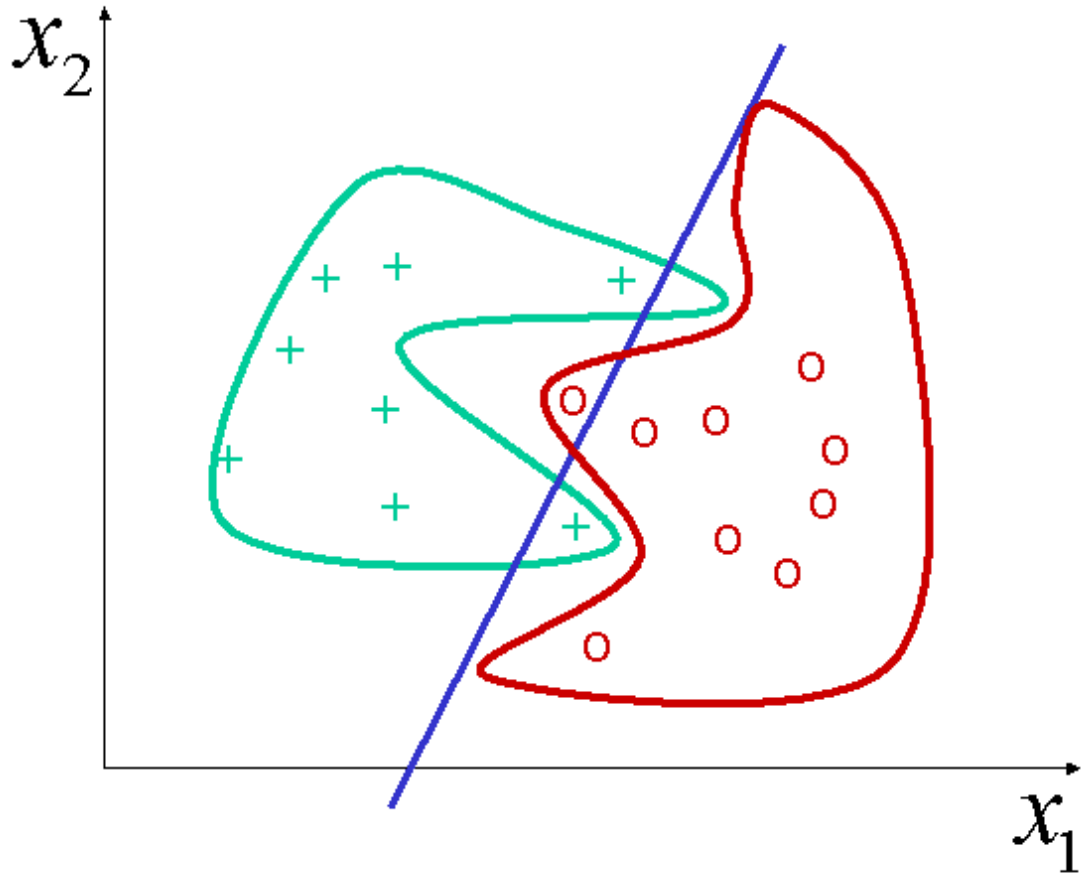
Linearly Separable Classes



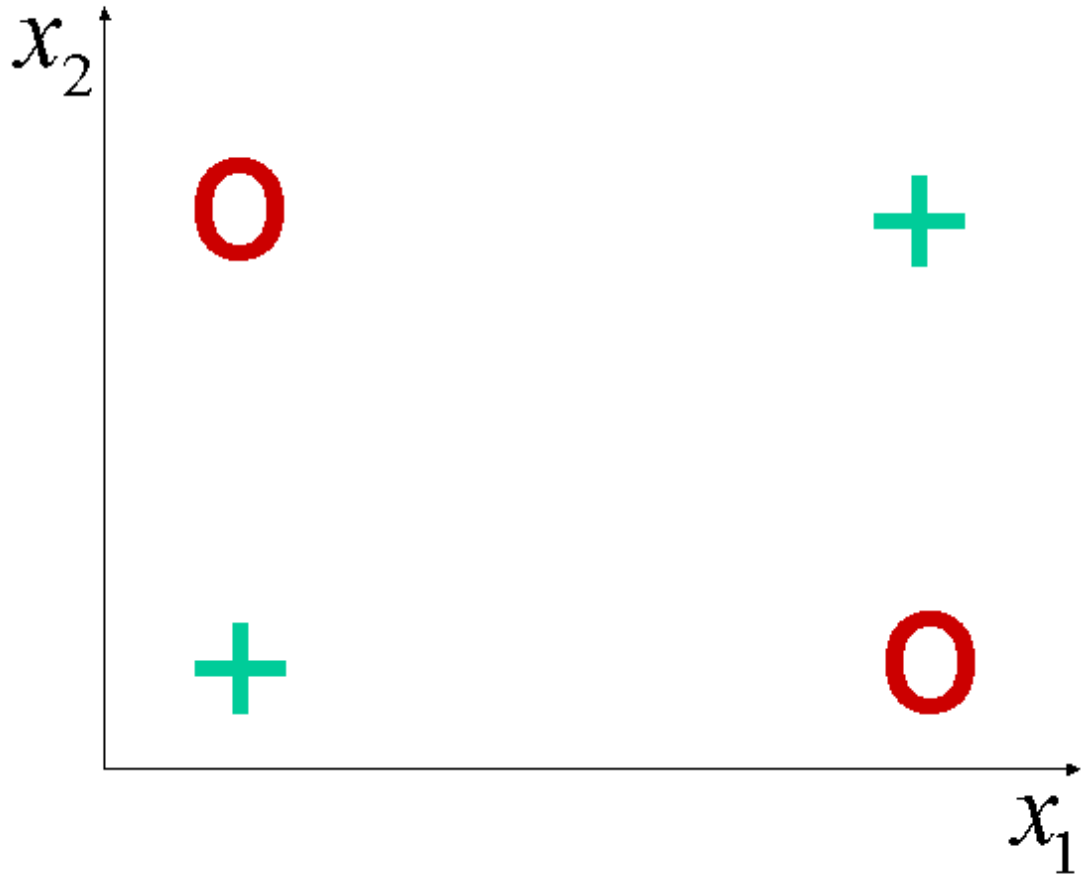
Convergence and Non-uniqueness



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

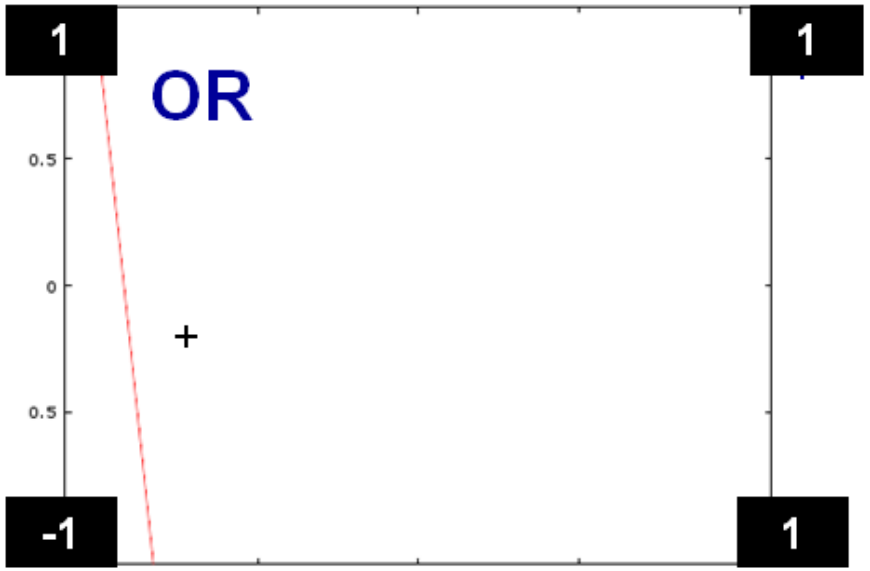


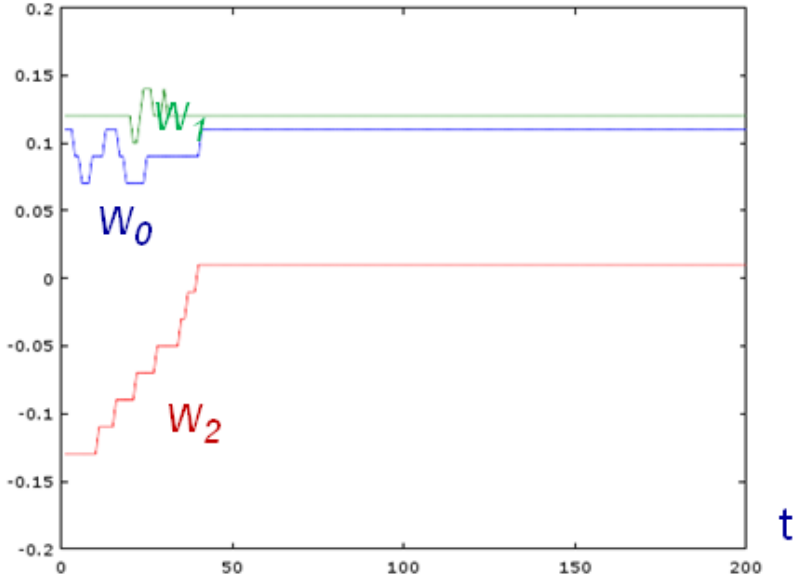
The Classical Example for Linearly Non-Separable Classes: XOR

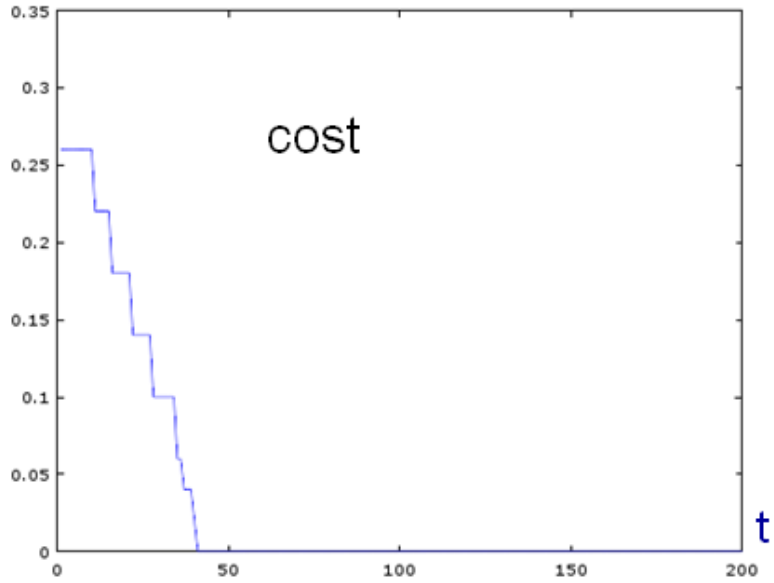


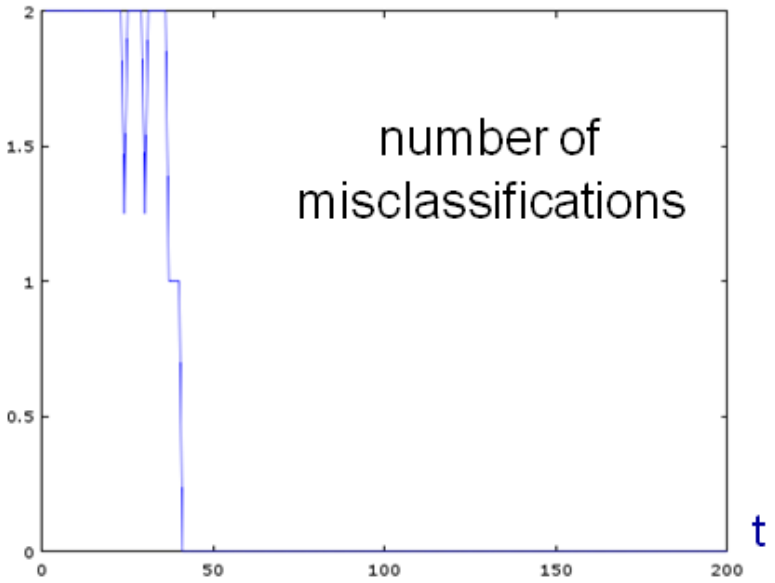
Learning Behavior for Nonseparable Classes

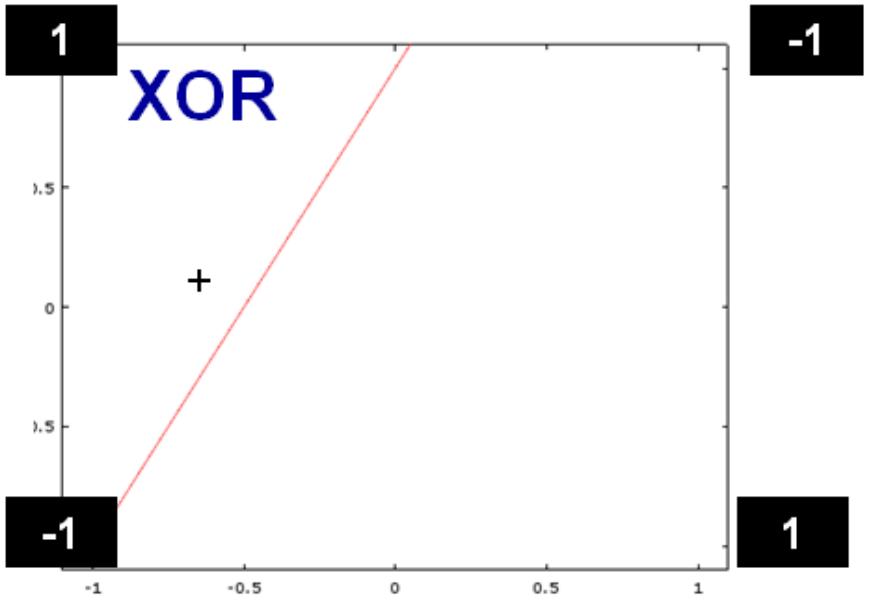
- The cost is minimum if as many data points as possible are correctly classified
- For the misclassified data points, $|h|$ should be small; this is achieved with $\|\mathbf{w}\| \rightarrow 0$ which leads to instable learning behavior
- Next, we show the learning behavior for the linearly separable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

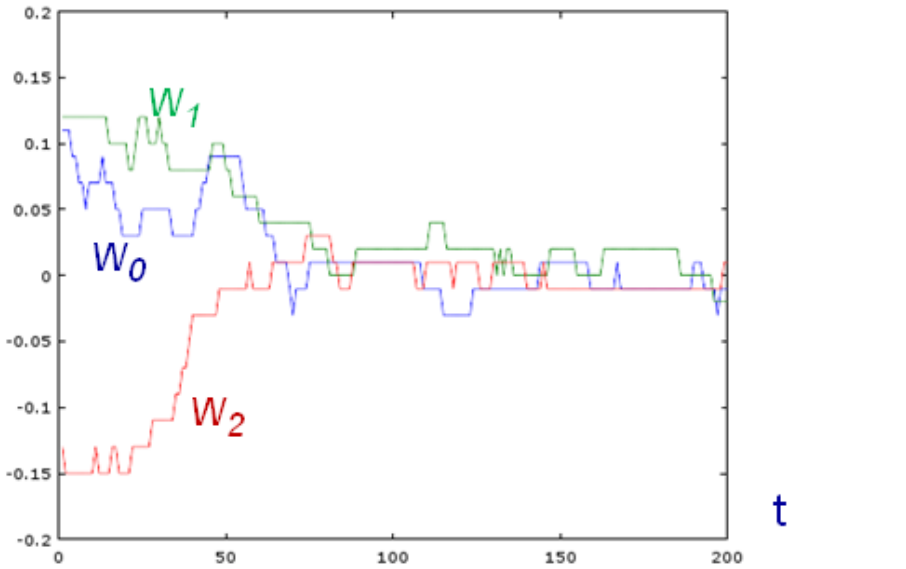

1
1
OR
+
-1
1

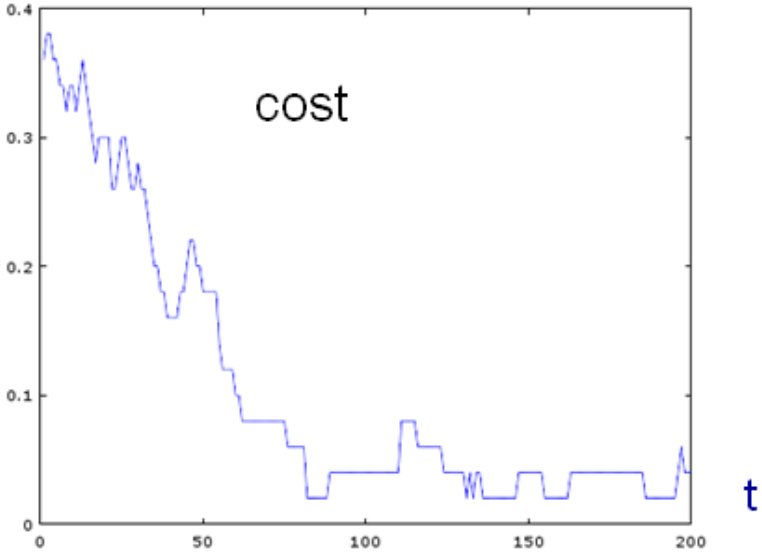

w_1
w_0
w_2
t

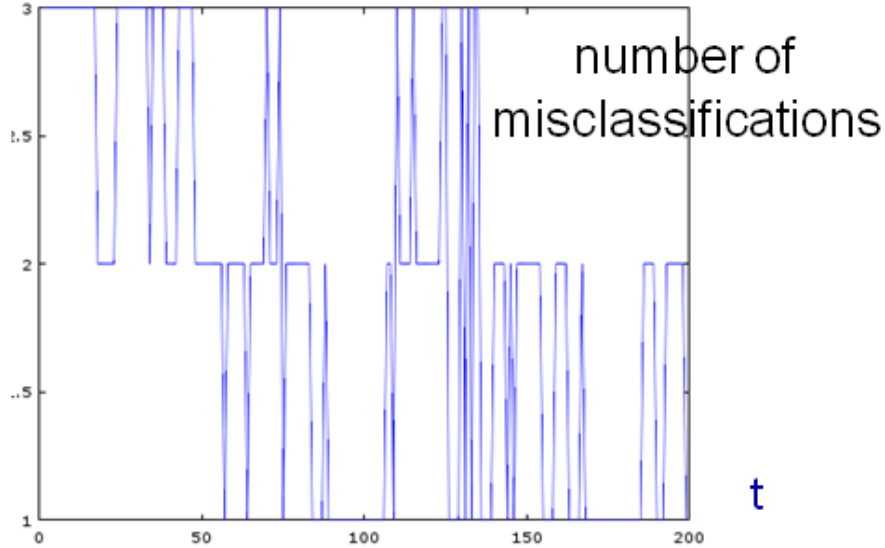

cost
t


number of
misclassifications
t


1
-1
XOR
+
-1
1


w_1
w_0
w_2
t


cost
t


number of misclassifications
t

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
  - Thus, $\mathbf{w}_{opt}$ is not unique!
- Alternative learning rules:
  - Optimal separating hyperplanes (Linear Support Vector Machine)
  - Fisher Linear Discriminant
  - Logistic Regression

More on Geometric Relationships

- In vector notation (introduced in a following lecture), the separating hyperplane is defined as

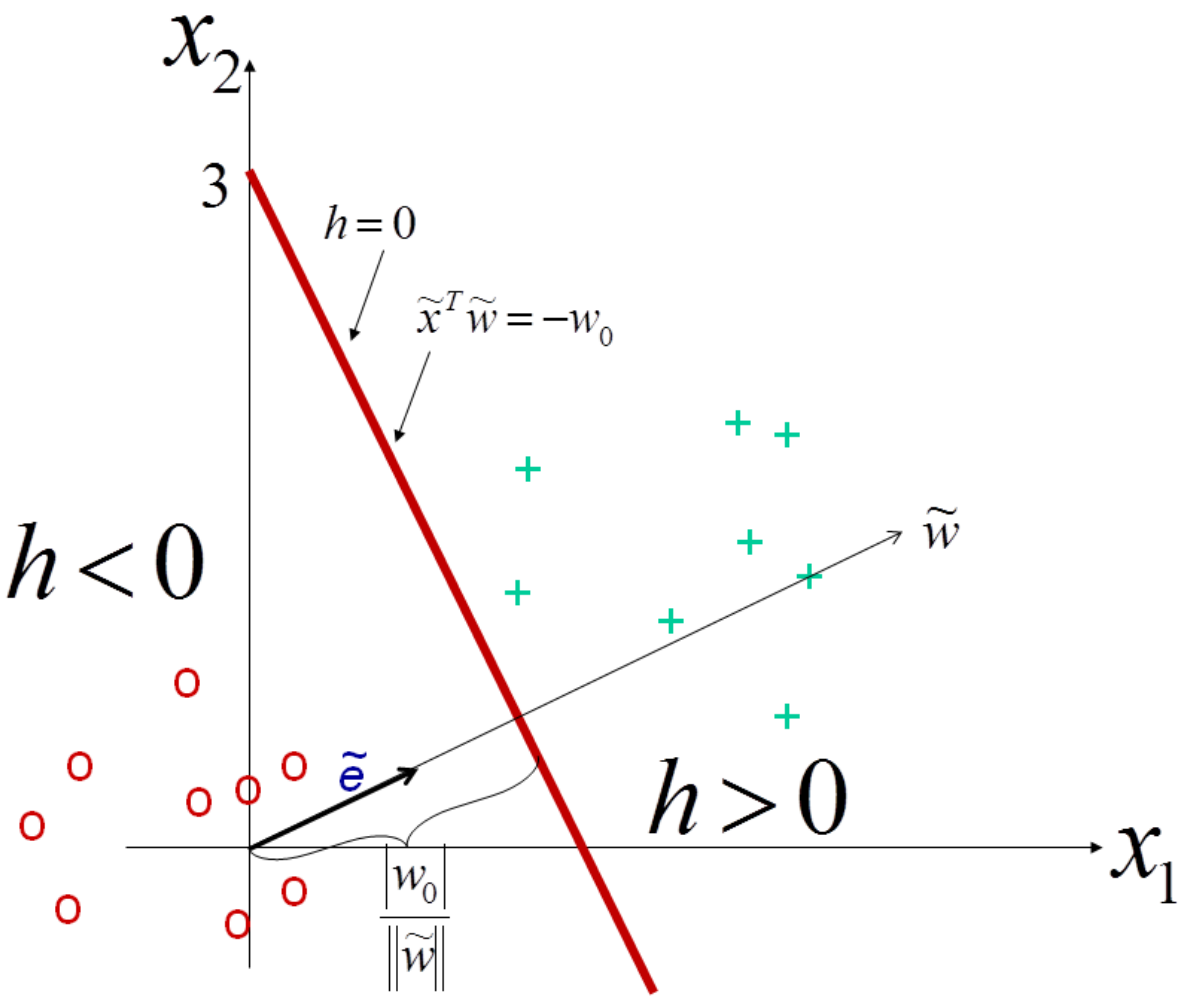
$$\mathbf{x}^T\mathbf{w} = 0 \quad \text{or} \quad \tilde{\mathbf{x}}^T\tilde{\mathbf{w}} = -w_0$$

- Here, $\mathbf{x} = (x_0, x_1, \ldots, x_M)^T$, $\mathbf{w} = (w_0, w_1, \ldots, w_M)^T$
- And, $\tilde{\mathbf{x}} = (x_1, \ldots, x_M)^T$, $\tilde{\mathbf{w}} = (w_1, \ldots, w_M)^T$ contains only the "real" inputs without the first constant dimension
- A unit-length vector $\tilde{\mathbf{e}}$ is orthogonal to the separating hyperplane

$$\tilde{\mathbf{e}} = \frac{1}{\|\tilde{\mathbf{w}}\|}\tilde{\mathbf{w}}$$

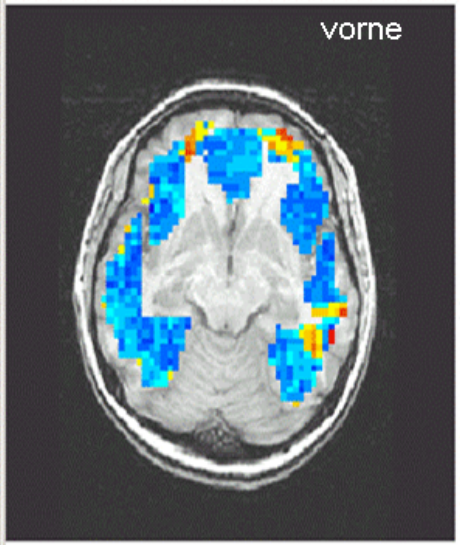
and it "hits" the separating hyperplane at

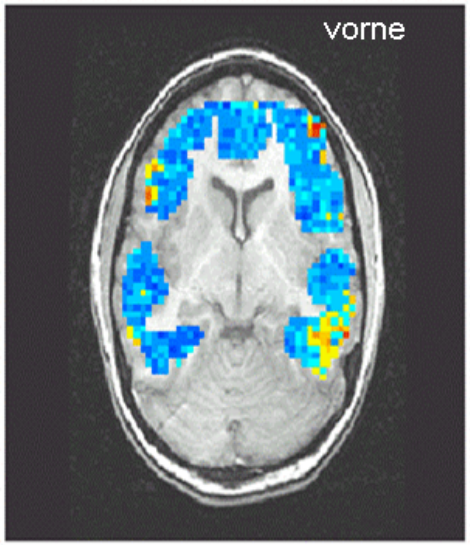
$$-\frac{w_0}{\|\tilde{\mathbf{w}}\|}\tilde{\mathbf{e}}$$

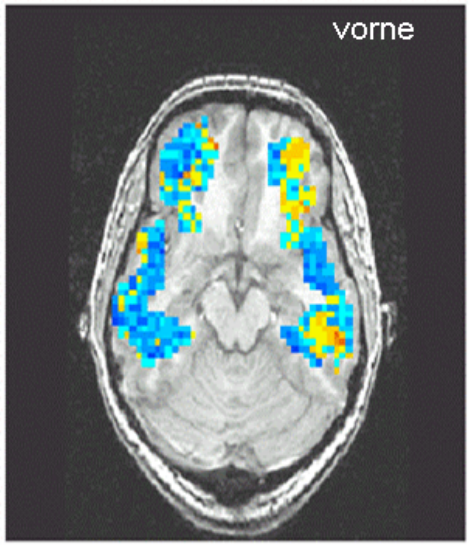

x_2
3
$h = 0$
$\tilde{x}^T \tilde{w} = -w_0$
$h < 0$
$\tilde{w}$
$\tilde{e}$
$h > 0$
x_1
$\frac{|w_0|}{\|\tilde{w}\|}$

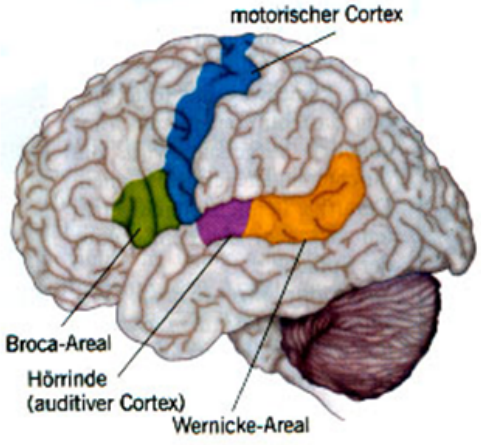
Application for a Linear Classifier; Analysis of fMRI Brain Scans (Tom Mitchel et al., CMU)

- 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

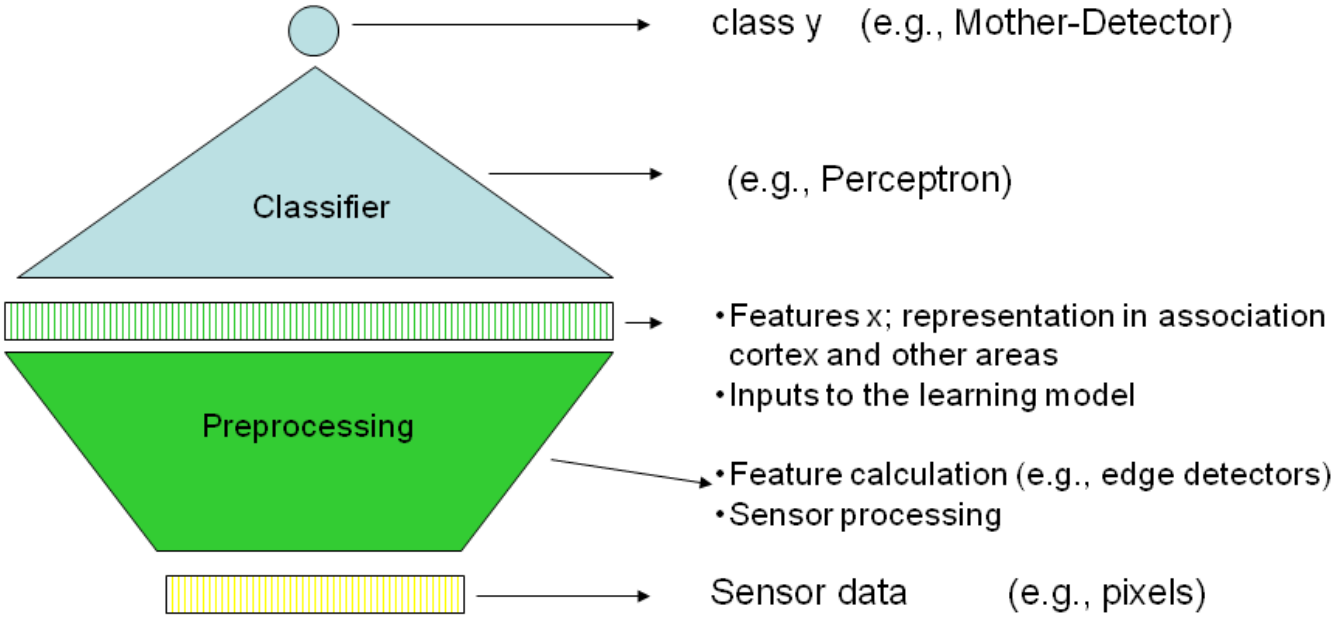

vorne


vorne


vorne


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

Recall: A Biologically Motivated Model



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); advantage: explainability.
- 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, when we communicate some learned insight, it is often communicated by simple probabilistic rule-like statements: "Rich people are often interested in buying a yacht, unless they already own one!"
- Also: computer programs, machines ... follow simple deterministic rules? (This might change in the area of deep learning)

Example: Birds Fly

- Define flying: using its own force, a distance of at least 20m, at least 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!

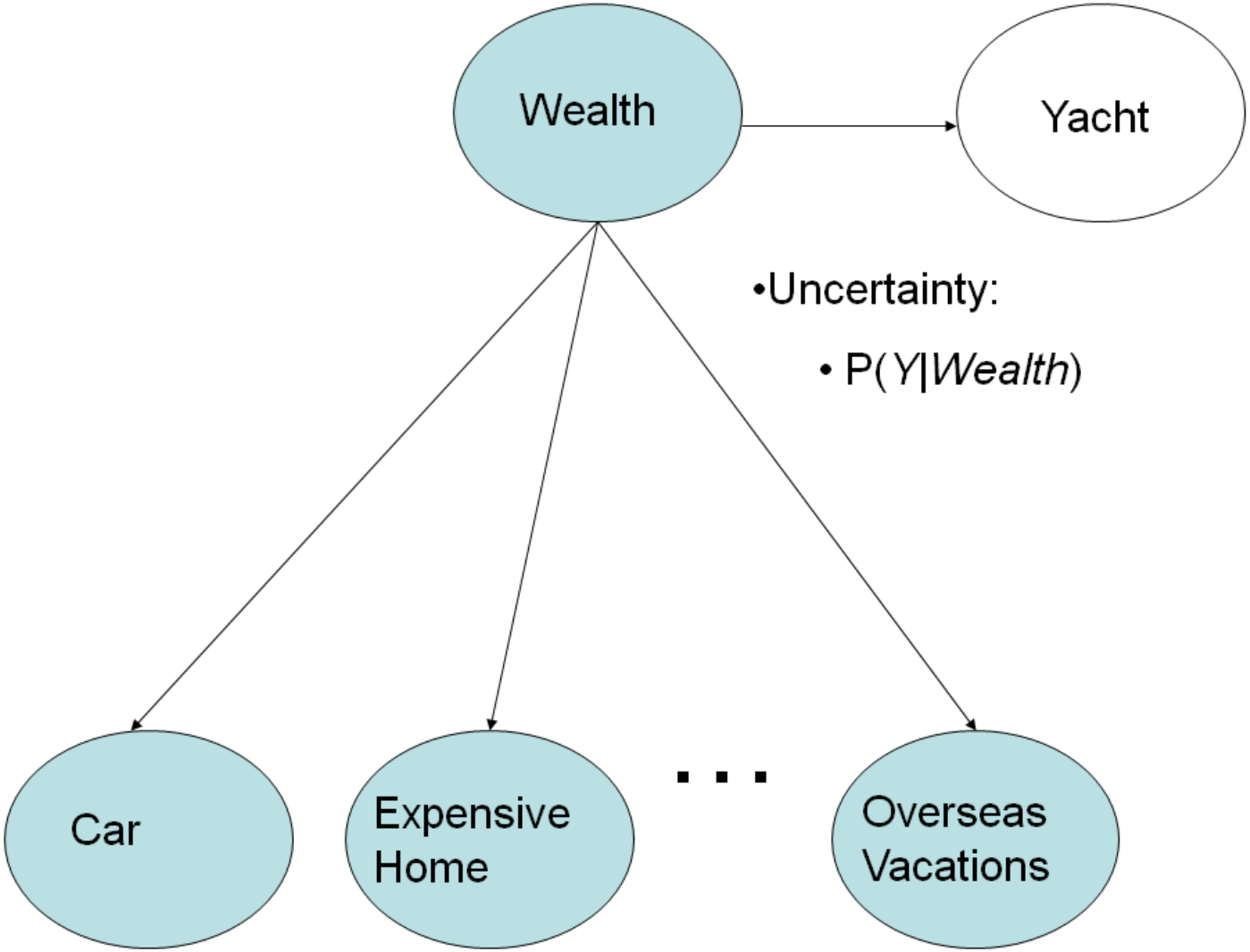
From Rules to Probabilities and High Dimensions

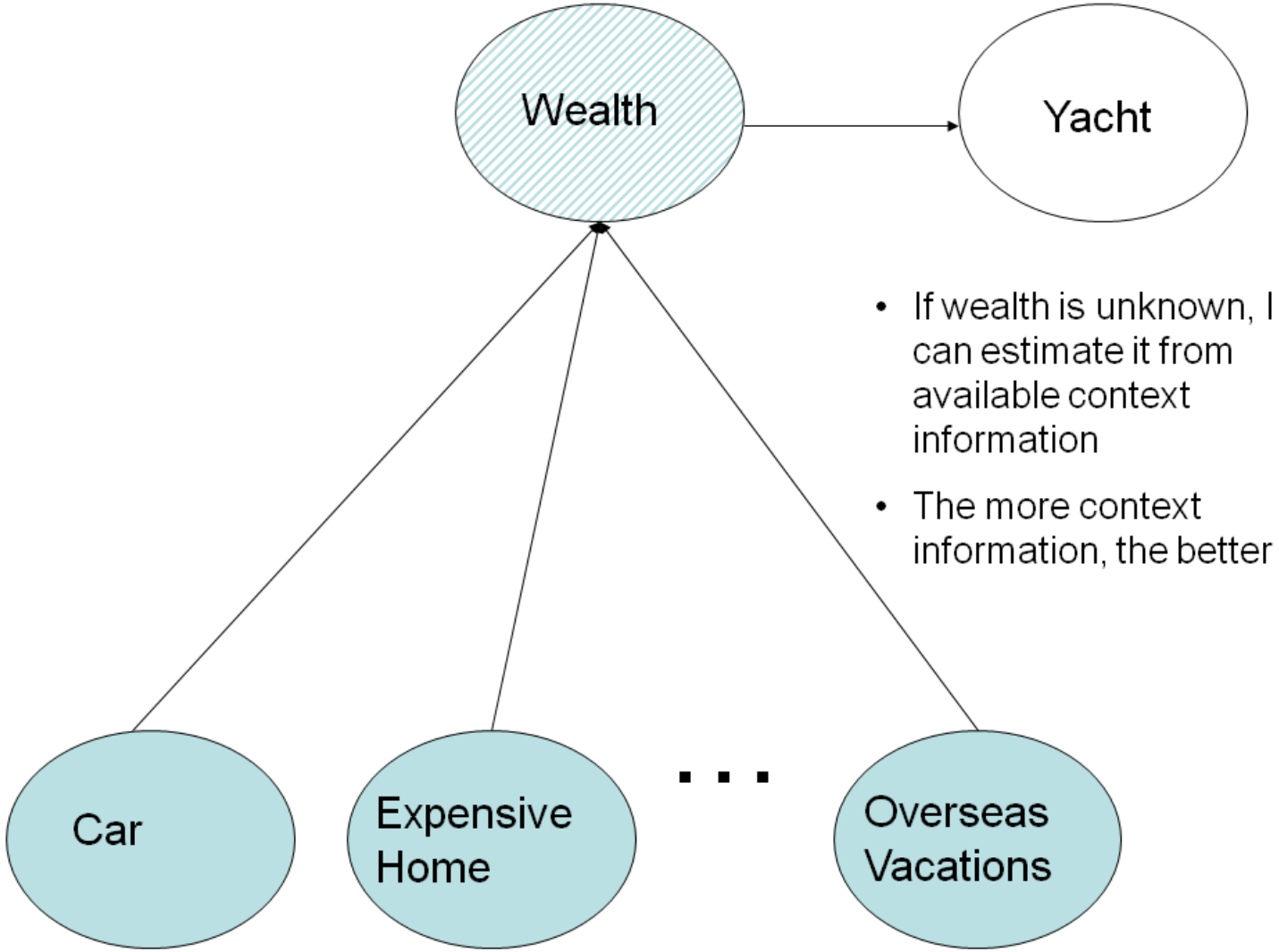
- Consider the statement: "Rich people are often interested in buying a yacht, unless they already own one!"
- One cannot observe the desire to buy a yacht (unless one does interviews) but only the fact (ownership of a yacht)
- We might translate this into a conditional probability $P(Y|\mathit{Wealth})$; right away we need to deal with uncertainty, probabilities
- Now we might not know the wealth of a person, only some secondary attributes that indicate a person's wealth
- Since a larger number of those secondary attributes permits a better prediction of one's wealth, we can either use those to estimate a person's wealth, and thus also ones probability in buying a yacht, or one trains a probabilistic classifier with many inputs, and limited explainability
- "...unless they already own one": a fact which might be checked in some database

A Conversation

- "Who might want to buy my yacht?" "We should talk to Jack, I have the feeling that he might be interested in buying your yacht!"
- "Why?" "He is rich, and does not have a yacht!"
- "How do you know that he is rich?" "Look at his car, his house, ...!"
- Hypothesis: the first evaluation "We should talk to Jack, I have the feeling that he might be interested in buying your yacht!" might be done by some internal, potentially high-dimensional, classifier, more or less unconsciously. When we need to justify our opinion, we refer to some low dimensional explanation
- A posterior explanation and justification of a decision ("In neuropsychology the left brain interpreter refers to the construction of explanations by the left brain in order to make sense of the world by reconciling new information with what was known before. The left brain interpreter attempts to rationalize, reason and generalize new information it receives in order to relate the past to the present", Schacter, Gazzaniga)

Example: Predicting Buying Pattern

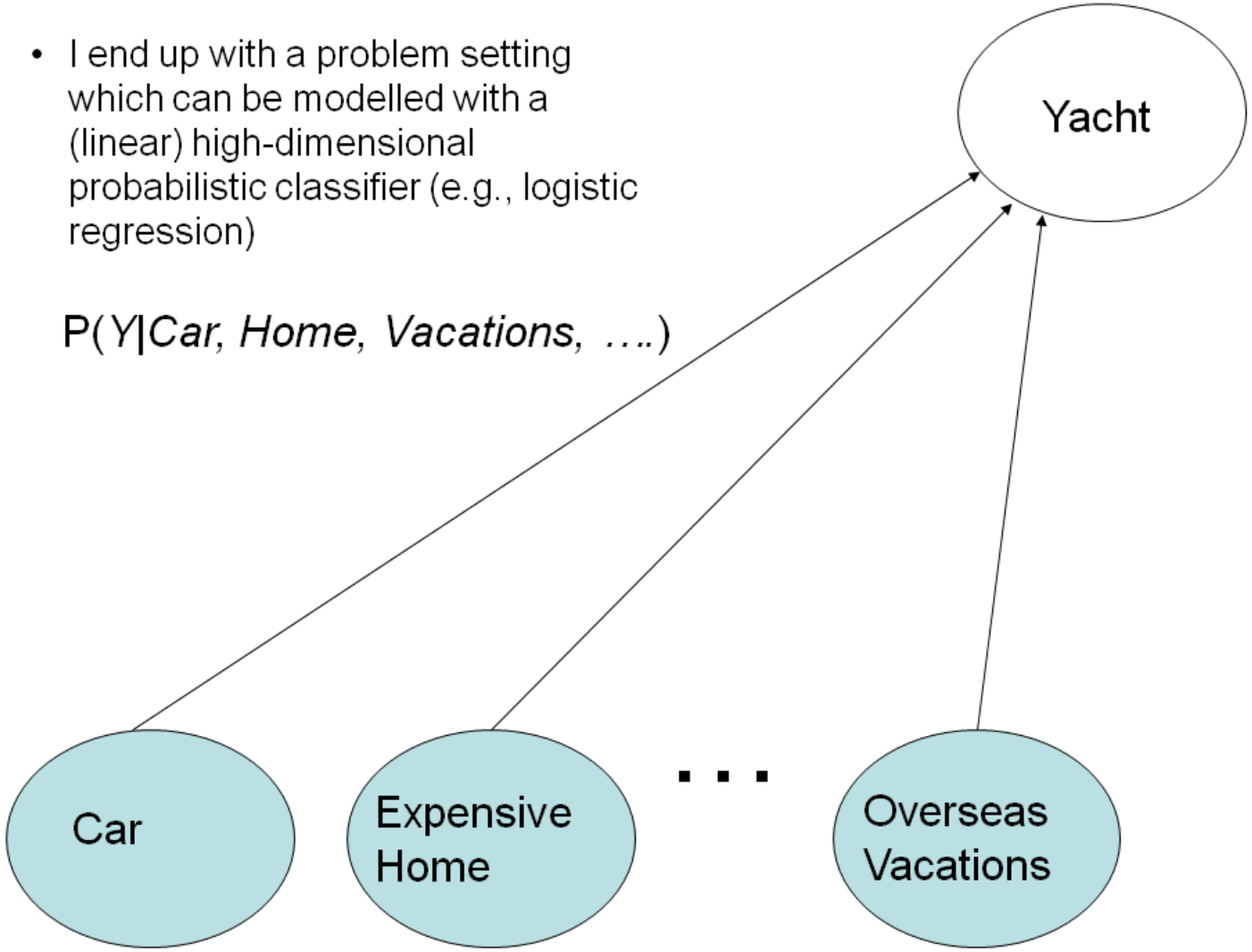




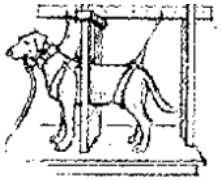
- If wealth is unknown, I can estimate it from available context information
- The more context information, the better

- I end up with a problem setting which can be modelled with a (linear) high-dimensional probabilistic classifier (e.g., logistic regression)

P(*Y*|*Car*, *Home*, *Vacations*, ….)

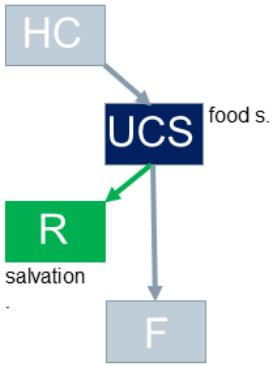


Recall: Pawlow's Dog



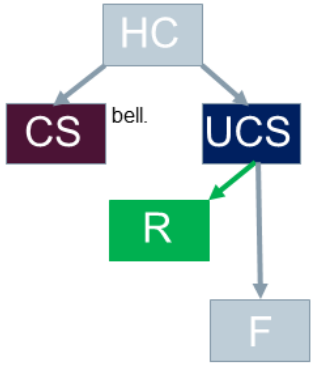
Classical Conditioning: ´Learning to predict important events (Pawlow)

Phase 1:
Unconditional response (UCR)



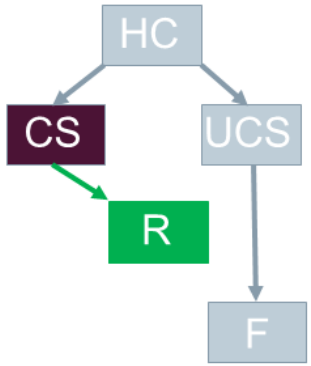
The **unconditional stimulus (UCS, food smell)** produces the **response (R, salvation),** since the dog has learned that after food-smell comes food

Phase 2



A hidden cause (HC, experimenter) might produce both the **UCS** and the **conditional stimulus (CS, bell)** which might come slightly earlier than the food smell

Phase 3:
Conditional response (CR)



The dog leans that after the **CS (bell),** food (F) follows and starts salvation, even without the **UCS (food-smell)**

- As a predictive model this makes absolutely sense but it might not reflect causality or be interpretable

Another example. **UCR**: A stomach virus (UCS) would produce a response of nausea (R). **CR**: Chocolate (CS) which was eaten before the person was sick with a virus now produces a response of nausea

Conclusion

- One reason, why dependencies tend to be high-dimensional and statistical in nature is that the input of interest, here wealth, is not available (latent, hidden); latent variable models attempt to estimate these latent factors (see lecture on PCA)
- 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. But natural laws where only discovered very late in history (Galileo, ...)

- 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!"**

Appendix

- Another form of the update rule
- One adapts $t = 1, 2, \ldots,$ for **any pattern**

$$w_j \longleftarrow w_j + \frac{1}{2}\eta(y_t - \widehat{y}_t)x_{t,j} \quad j = 0, \ldots, M$$

- Note, that $(y_t - \widehat{y}_t) = 0$ for correctly classified patterns
- Formally identical to the ADALINE learning rule