Chapter 8

Neural networks

Aproaches to designing a classifier

There are in principal two different kinds of approaches to designing a classifier:

1. Statistical parametric modelling of class distributions, then MAP
2. Solving a map problem by function approximation (non-linear regression) of the a-posteriori-probabilities

$$\min E\left\{ \left\| \mathbf{f}(\mathbf{x}) - \mathbf{y} \right\|^2 \right\}$$

$\mathcal{X}$ – *feature space*

$\mathcal{Y}$ – *decision space*

Zu 1.) The approach for designing a classifier described so far is based on approximating *class-specific distribution densities*

$$p(\mathbf{x}|\omega_i)P(\omega_i)$$

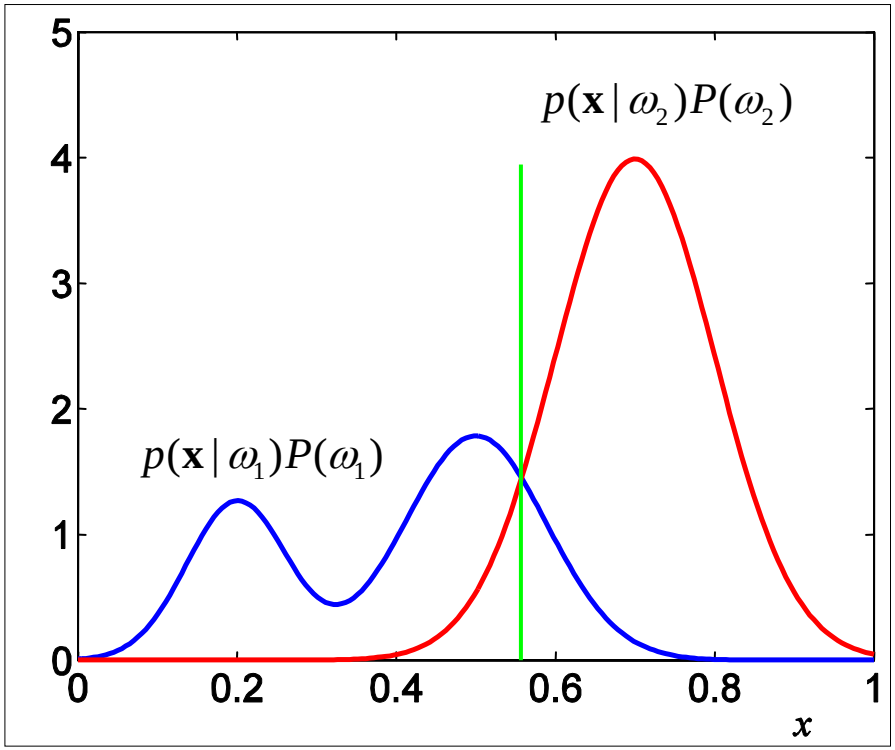
parametrically by static models (by estimating the parameters, e.g. Gaussian distribution) and on deciding using a maximum selection. Learning means: Improving of the parameter-fitting.

Zu 2.) There is a second approach, which is based on evaluation of the a-posteriori-probablity density

$$P(\omega_i|\mathbf{x})$$

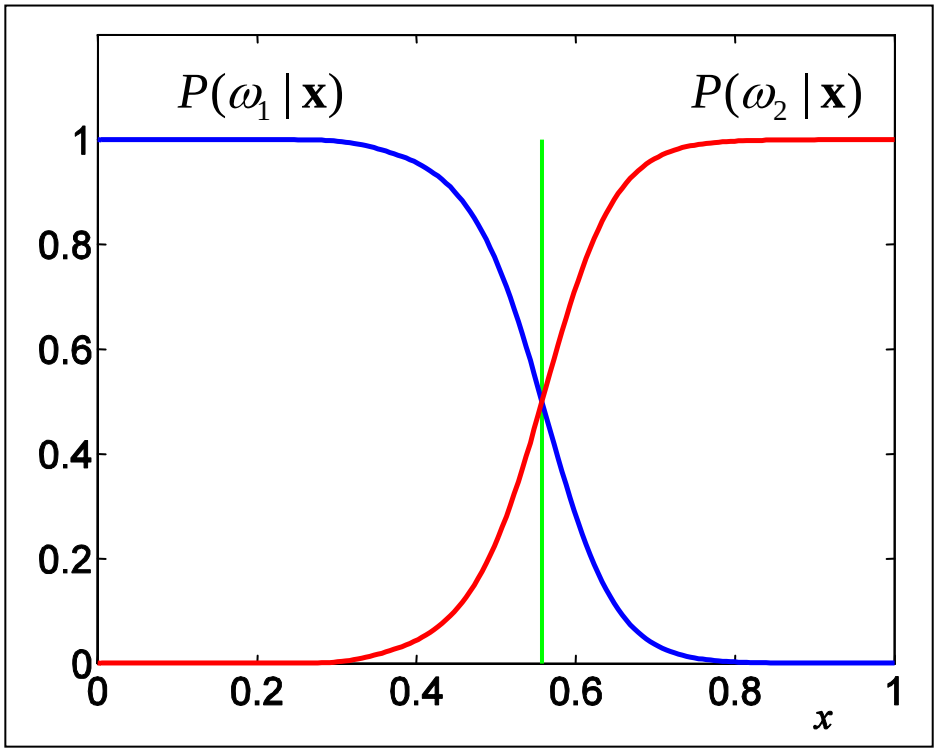
and can be described by a problem of *function approximation*.

The two approaches to classifier design



Modelling of class
distribution densities

$$P(\omega_i\,|\,\mathbf{x}) \sim p(\mathbf{x}\,|\,\omega_i)P(\omega_i)$$



Direct modelling of
a-posteriori-distribution densities

$$P(\omega_i\,|\,\mathbf{x})$$

Equivalence of both approaches

General case: matlab-MAP.bat

This function approximation can be performed e.g. by a *non-linear regression with polynoms* or also using an *artificial neural network* (NN). This lecture aims to cover the foundations for both approaches.

Basically, the search for the best approximation function is a *variation problem*, which can be reduced to a *parametric optimization problem* by choosing base functions. Learning in this context also means: parameter-fitting.

The equivalence of both approaches result from the Bayes-theorem:

$$p(\omega_i \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_i) P(\omega_i)}{\cancel{p(\mathbf{x})}}$$

$p(\mathbf{x})$ — independent on ω

The equivalence results from the denomiator being independent on ω.

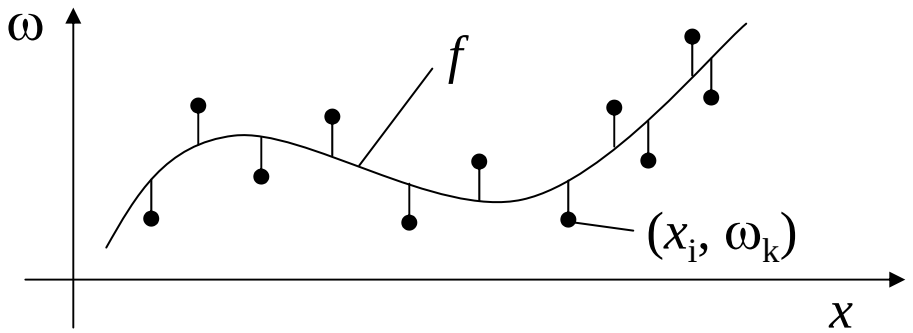
In the following the transfer to a function approximation problem will be established.

With known a-posteriori-p. $P(\omega|\mathbf{x})$ for every continuous x a ω could be assigned to at the best (functional assignment f:$\mathbf{x} \rightarrow \omega$). Given are only *samples* and sought is a function f, that fits the individual experiments at the best and therefore implements a map:

$$\mathbf{f}: \underset{\mathbf{x}}{\text{feature space}} \rightarrow \underset{\omega}{\text{equivalence space}}$$

This task can be solved using variation calculation. Choosing the minimal error square as quality factor, it is about minimizing:

$$J = \min_{\mathbf{f}(\mathbf{x})} E\{\|\mathbf{f}(\mathbf{x}) - \mathbf{y}\|^2\}$$



In this process target vectors $\{\mathbf{y}_i\}$ in the *decision space* $\mathcal{Y}$ by simple map of scalar labels $\{\omega_i\}$

$$\Omega := \{\omega_1, \omega_2, \ldots, \omega_K\}$$

$$\updownarrow \quad \updownarrow \quad \updownarrow$$

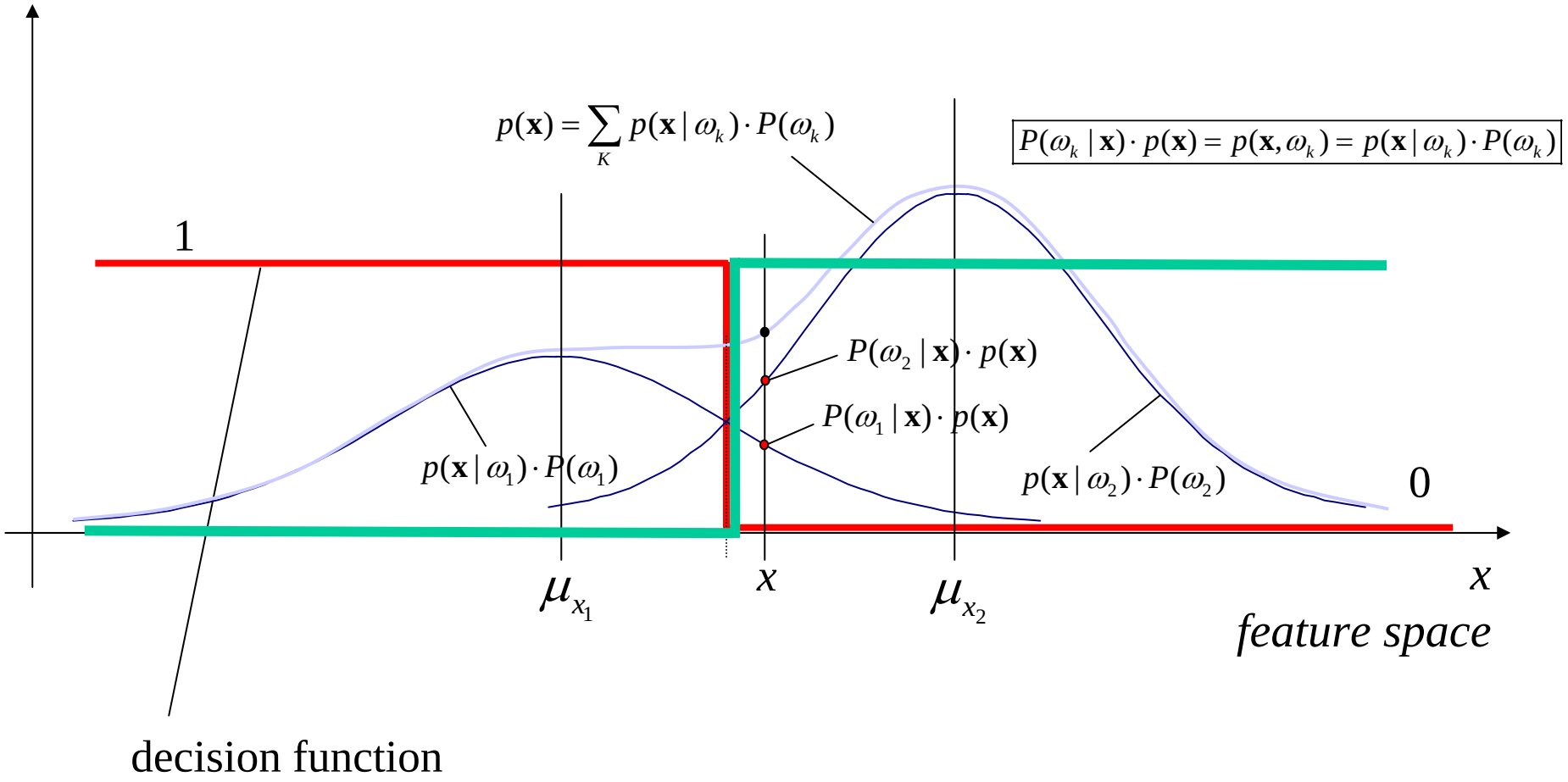
$$\mathcal{Y} := \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K\}$$

with

$$\mathbf{y}_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

i-th unit vector

Two-class-problem with Gaussian distribution density



Regression using *artificial* neural nets

Usage of a multilayer perceptron for function approximation

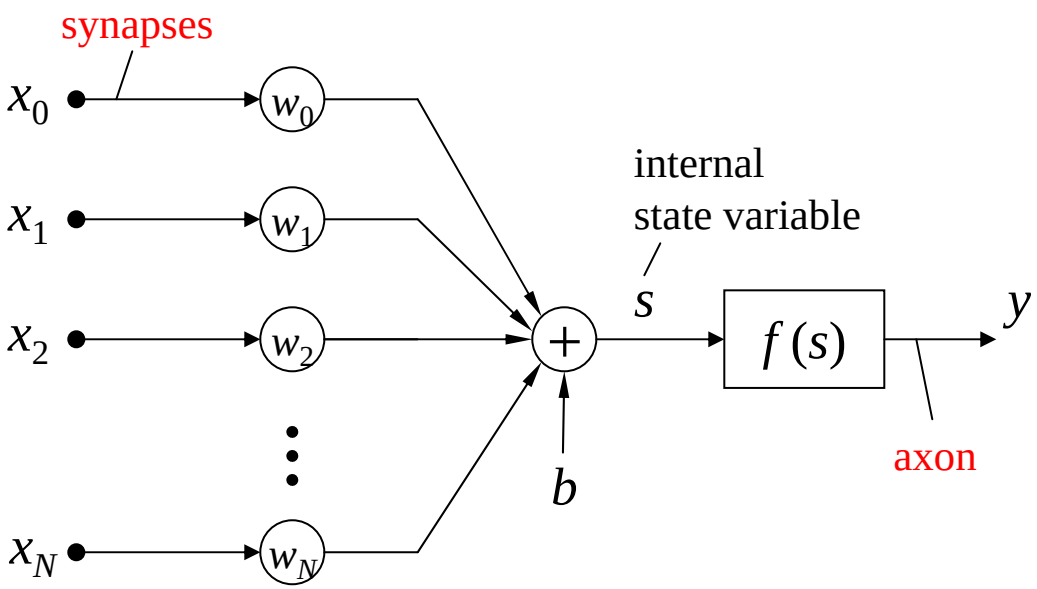
Motivation:

- Imitation of human approach
- Parallel evaluation with NN-computer (hardware)

Human brain: 10^{11} neurons with up to 10^4 links per neuron

Model of a neuron

(McCulloch & Pitts, 1943)

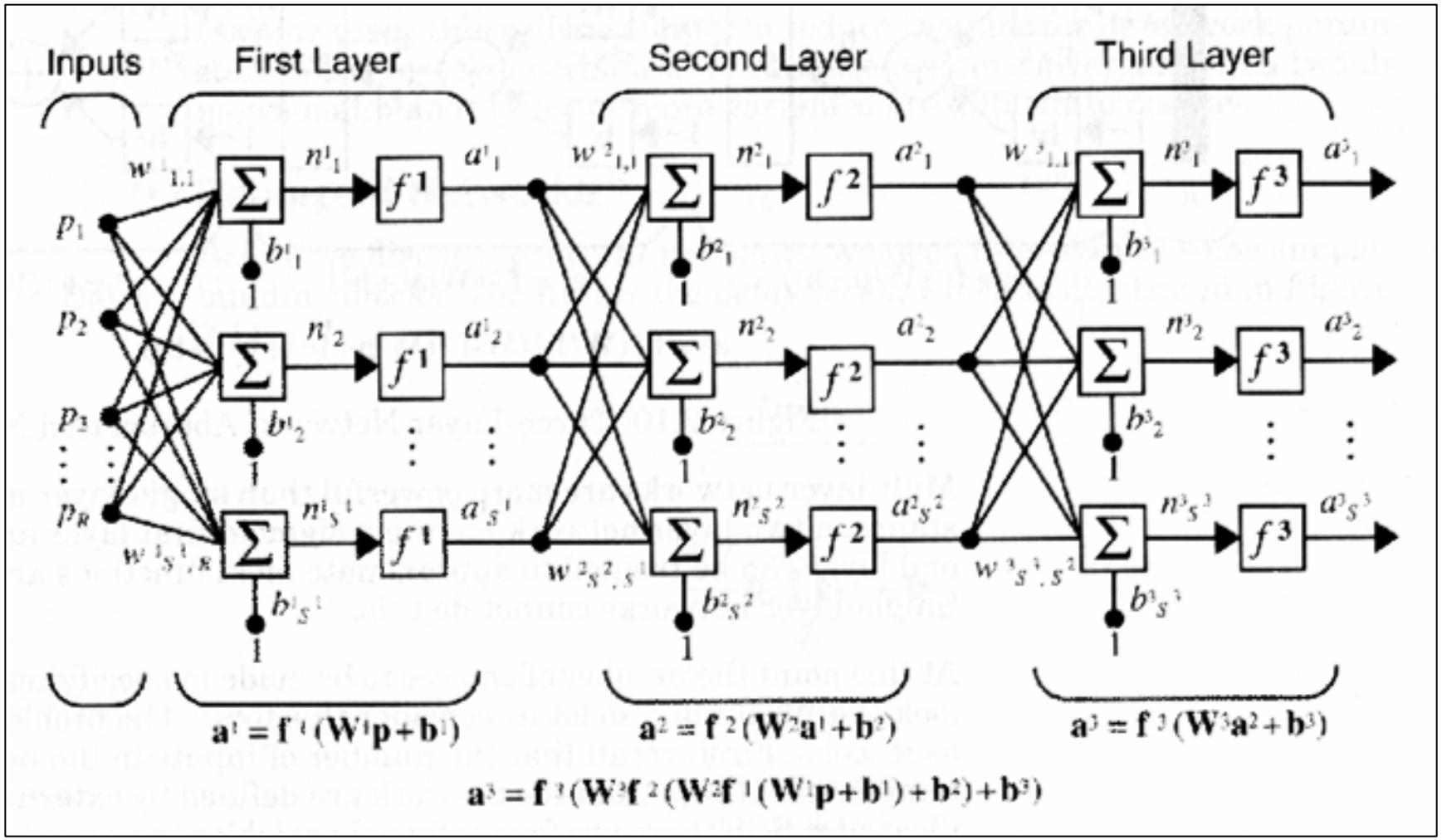


$$y = f(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

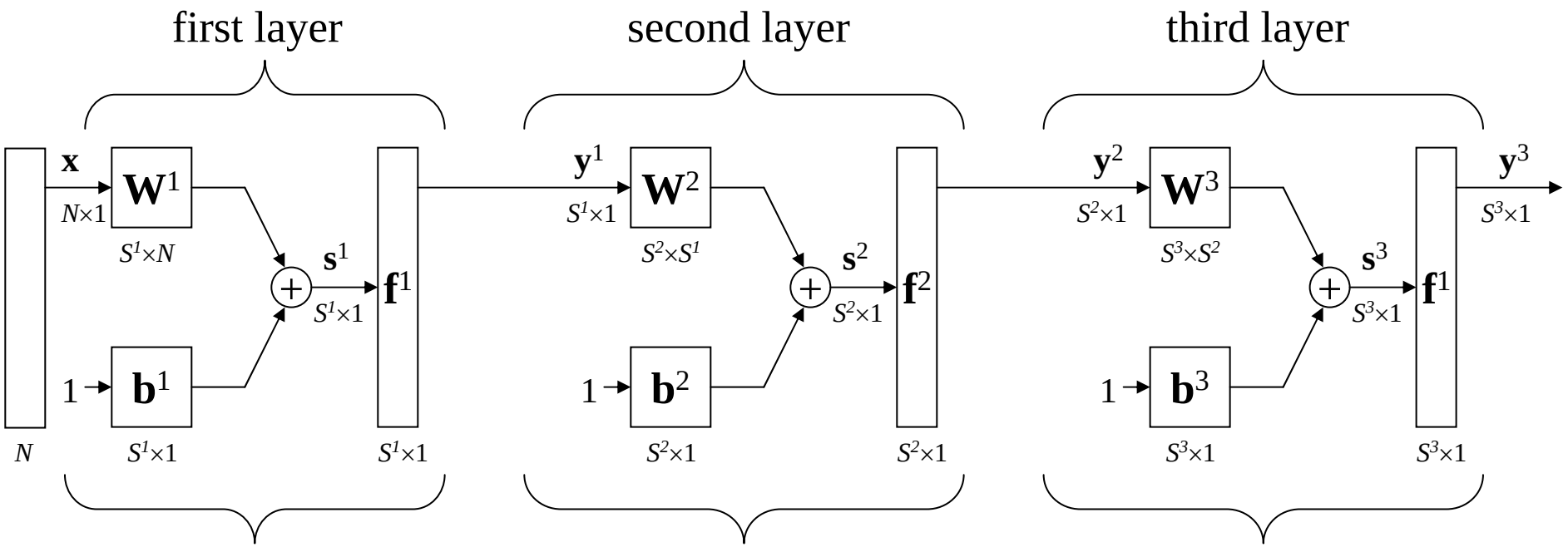
The neural activity is described by a scalar product of the weight vector **w** and the input channels x_i with a subsequent non-linear activation function $f(s)$. Excitations x_i can have a strengthening or repressice effect, which corresponds to positive or negative weight values w_i .

The multilayer perceptron with 3 layers

(Rosenblatt 1958)



Symbolic description of a perceptron with 3 layers



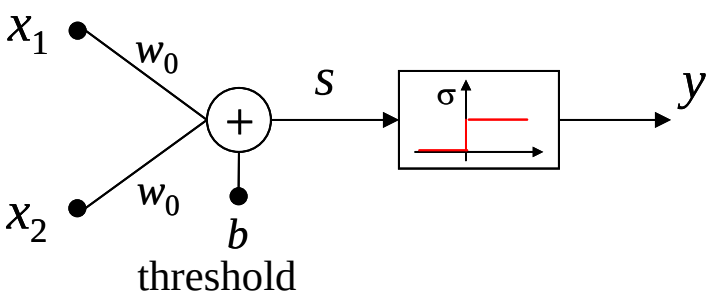
$\mathbf{y}^1=\mathbf{f}^1(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1)$ $\mathbf{y}^2=\mathbf{f}^2(\mathbf{W}^2\mathbf{y}^1+\mathbf{b}^2)$ $\mathbf{y}^3=\mathbf{f}^3(\mathbf{W}^3\mathbf{y}^2+\mathbf{b}^3)$

$$\mathbf{y}^3=\mathbf{f}^3(W^3f^2(W^2f^1(W^1x+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

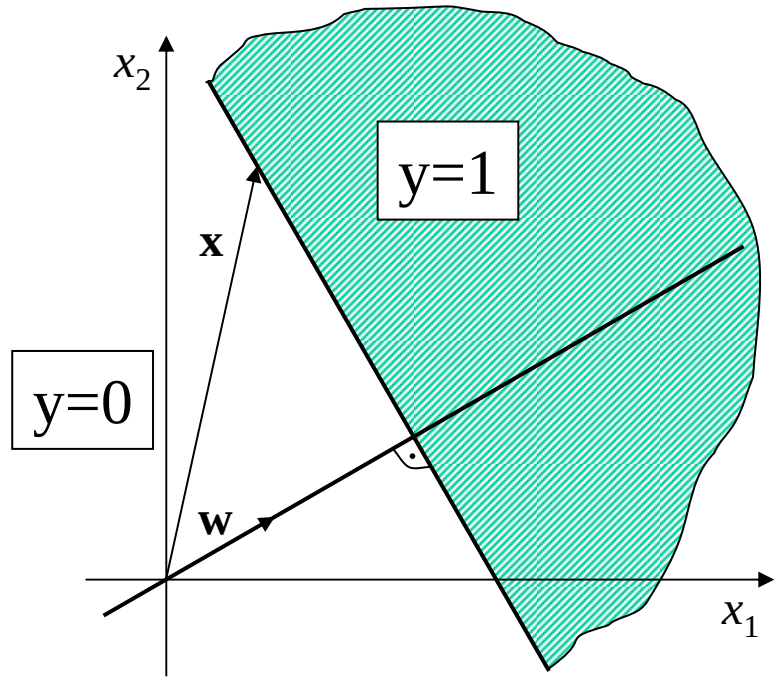
The perceptron with one neuron and two inputs
(Rosenblatt, 1958)

In the beginning of NN-research threshold logic was prefered to use, i.e. the input vector was *two-valued* and as activation function *unit step function* ($f(s)=\sigma(s)=1$ for $s\geq 0$ and $\sigma(s)=0$ für $s<0$).

Since every Boolean function can be written in disjunctive or conjunctive normal form, it is possible to implement *every logic function* with a *two-layered* perceptron.



$$y = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \sigma(g(\mathbf{x}))$$



functionality with real-valued inputs

Functionality of the perceptron

The neuron divides the input space of x with a *hyper plane* (here: *line*) into two halves. The hyper plane is the *geometric location*, on which all vectors lie, whose projections to **w** are constant:

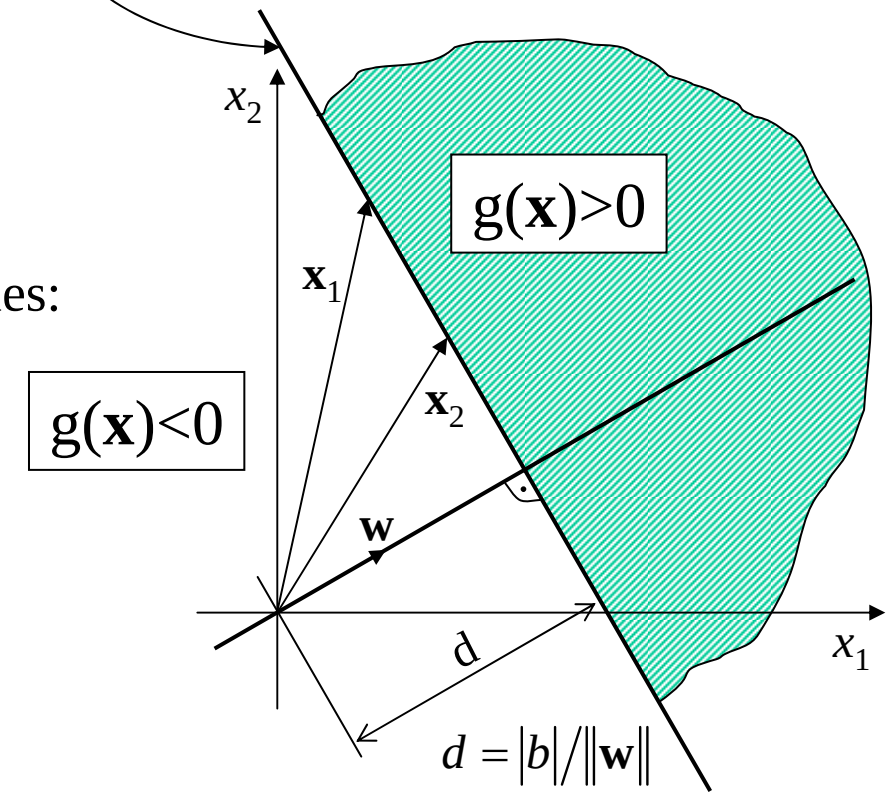
$$\boxed{g(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = 0}$$

orthogonal projection:

$$\frac{\langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|} = \|\mathbf{x}\| \cos\varphi \overset{?}{\gtrless} \frac{|b|}{\|\mathbf{w}\|}$$

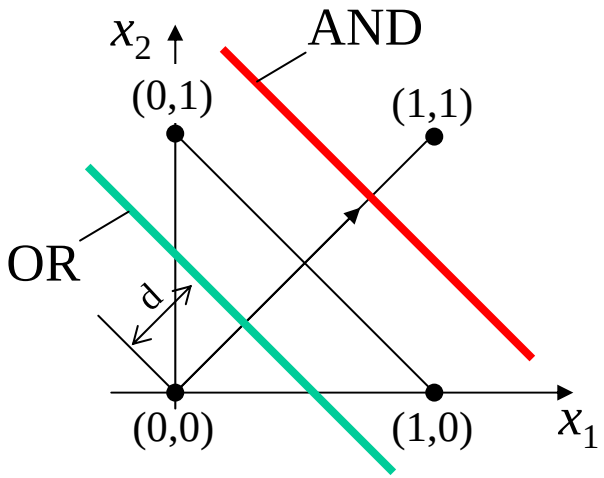
For two points on the separation plane applies:

$$0 = \langle \mathbf{w}, \mathbf{x}_1 \rangle + b = \langle \mathbf{w}, \mathbf{x}_2 \rangle + b$$
$$\Rightarrow \langle \mathbf{w}, (\mathbf{x}_1 - \mathbf{x}_2) \rangle = 0$$
$$\Rightarrow \mathbf{w} \perp (\mathbf{x}_1 - \mathbf{x}_2)$$

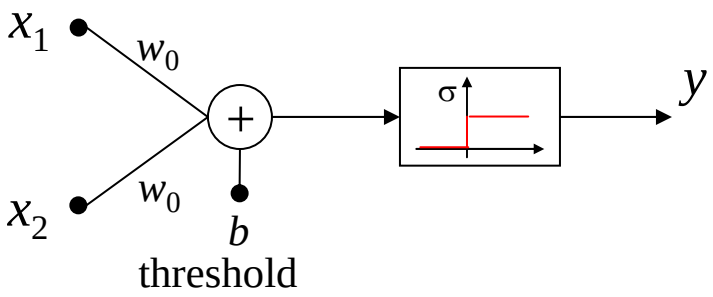


Non-linear classifier design - the XOR problem

It is obvious, that the Boolean functions AND and OR can be solved by a linear classifier and therefore with a one-layered perceptron:



x_1	x_2	AND	class	OR	class
0	0	0	ω_1	0	ω_1
0	1	0	ω_1	1	ω_2
1	0	0	ω_1	1	ω_2
1	1	1	ω_2	1	ω_2



$$\mathbf{w} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$$

AND	$b = -\frac{3}{4}\sqrt{2}$
OR	$b = -\frac{1}{4}\sqrt{2}$

or simplified:

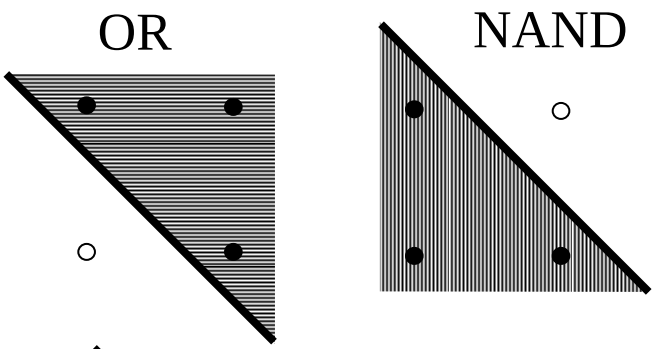
AND: $g(\mathbf{x}) = x_1 + x_2 - \frac{3}{2} = 0$

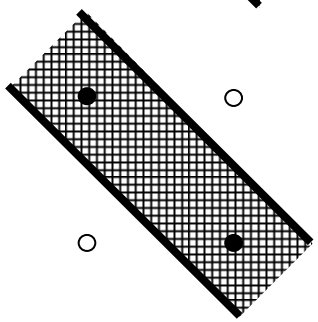
OR: $g(\mathbf{x}) = x_1 + x_2 - \frac{1}{2} = 0$

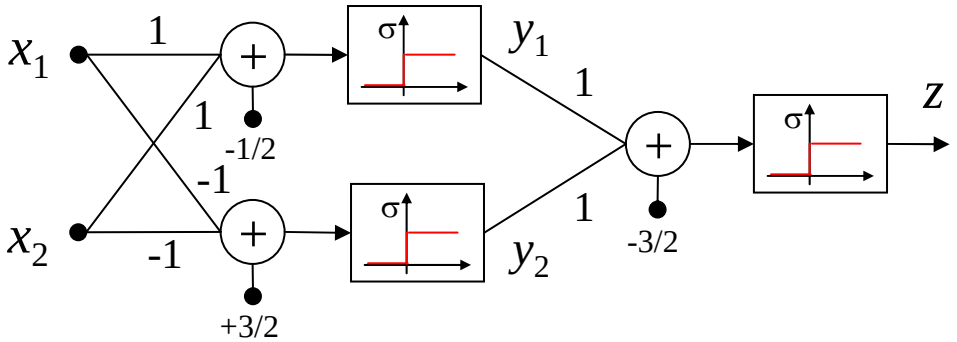
Solving the XOR problem with a two-layered perceptron

The exclusive-OR problem contrary to AND and OR can *not* be solved by a linear classifier.

A solution can be achieved by combination of two separating lines. This leads to a two-layered perceptron!

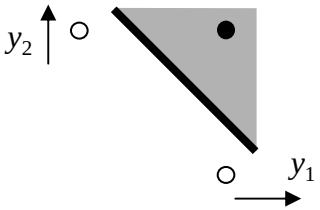






		first layer		second layer	
x_1	x_2	y_1	y_2	XOR	class
0	0	0	1	0	ω_1
0	1	1	1	1	ω_2
1	0	1	1	1	ω_2
1	1	1	0	0	ω_1

$$z = (x_1 \vee x_2) \wedge \overline{(x_1 \wedge x_2)} = y_1 \wedge \overline{y_2} \qquad \text{XOR}$$



$$y_1 = \sigma(x_1 + x_2 - \tfrac{1}{2}) \qquad \text{OR}$$

$$y_2 = \sigma(-x_1 - x_2 + \tfrac{3}{2}) \qquad \text{NAND}$$

$$z = \sigma(y_1 + y_2 - \tfrac{3}{2}) \qquad \text{AND}$$

After mapping the first layer a linearly separable problem results!!

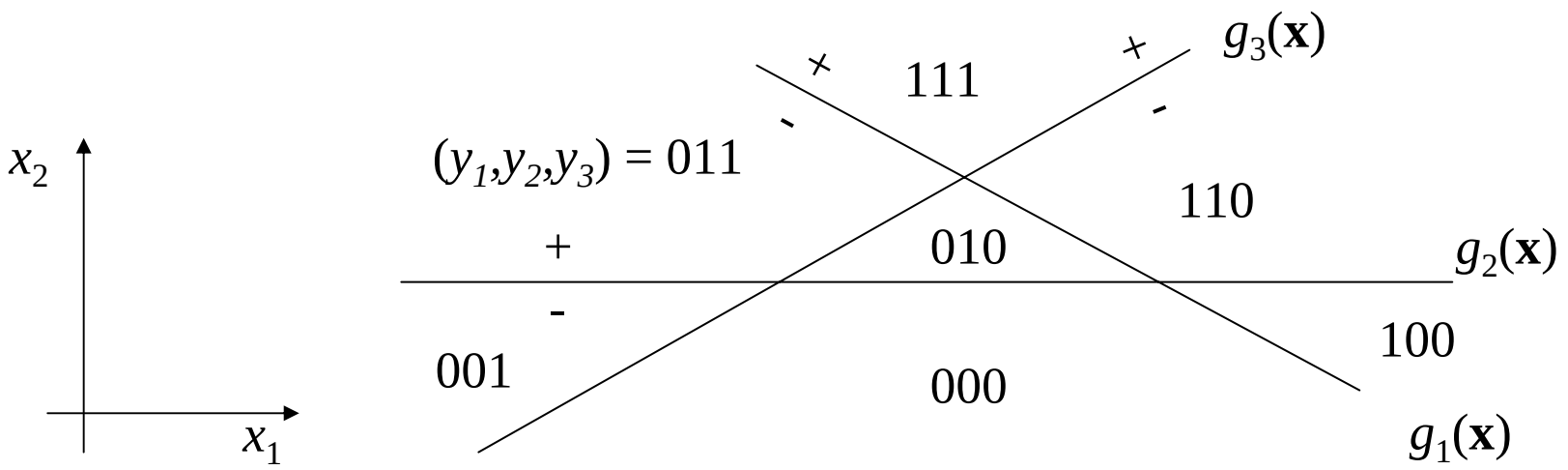
Functionality of the first layer: Mapping the real-valued input space to the corners of a hyper cube

In the first layer of a perceptron a neuron reacts with 0 or 1, dependung on in which half of the half spaces created by the hyper plane the input vector lies.

Each further neuron creates an additional separation plane. The first layer of a perceptron therefore maps the real-valued input space to the corners of a hyper cube.

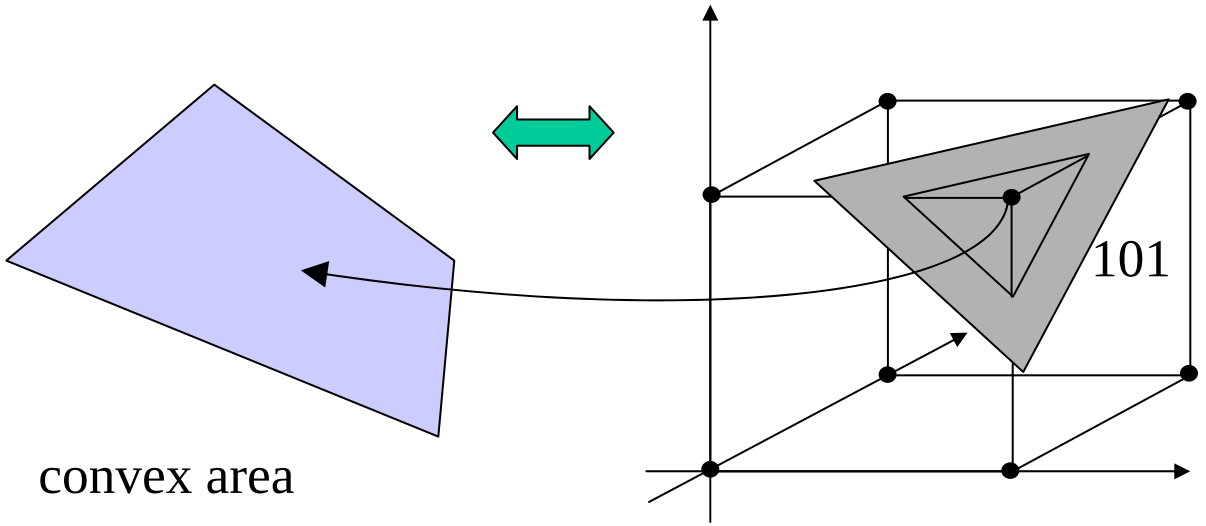
The intersecting separating planes form linearly limited (convex) ***areas***, so-called ***polyhedron***. Each area corresponds to a corner of the hyper cube.

The following diagram shows 3 neurons and a two-dimensional input space **x**:



Functionality of the second layer: Cutting a corner off the hyper cube

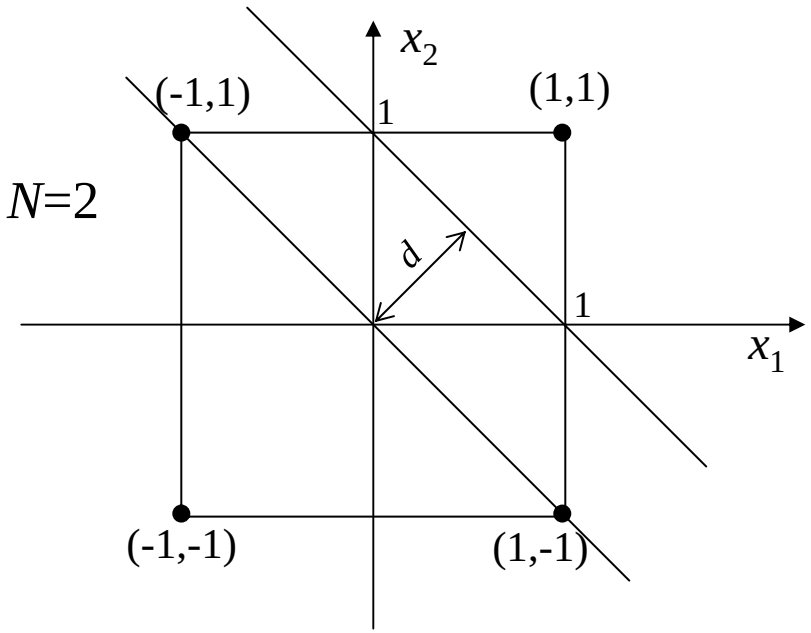
In the second layer a unique feature for a convex area can be created by cutting a corner off the hyper cube with a new separation plane:



Optimal separation plane for cutting a corner off a hyper cube

If the signum function is chosen as the non-linear function in the neuron, the resulting hyper cube lies centered within the origin. The optimal separation plane lies in the middle of a corner and a plane, which is spanned by the next neighbours.

Without loss of generality we can consider the corner along the first space diagonal **u**. The following applies:



$$\mathbf{u}^T = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix} \quad \text{with:} \quad \dim(\mathbf{x}) = N$$

$$\Rightarrow \mathbf{e}_u = \mathbf{u} / \|\mathbf{u}\| = \mathbf{u} / \sqrt{N}$$

Actually placing the separation plane with distance $d=N^{1/2}-\varepsilon$ with sufficiently small distance ε will do. Problem: an exact statement about ε depending on N cannot be made.

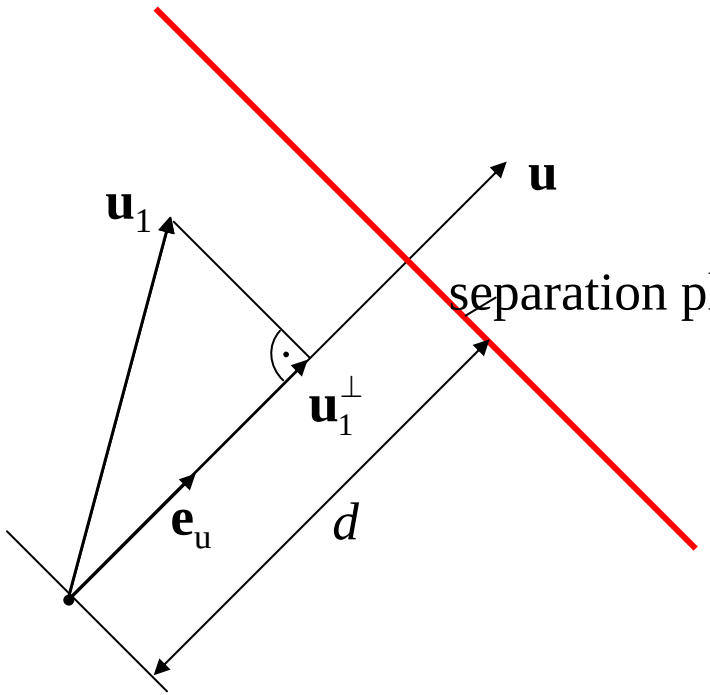
The separation plane lies exactly between **u** and the orthogonal projection of one of the neighbouring corners $\mathbf{u}_1$ to **u**:

$$\mathbf{u}_1^T = [\underbrace{1 \quad 1 \quad \ldots \quad 1}_{N-1} \quad -1]$$

$$\mathbf{u}_1^{\perp} = \underbrace{\langle \mathbf{u}_1, \mathbf{u} \rangle}_{N-2} \, 1/\sqrt{N} \, \mathbf{e}_u^T = \frac{N-2}{N}\mathbf{u}$$

and therefore the rootpoint of the separation plane:

$$\frac{\mathbf{u}_1^{\perp} + \mathbf{u}}{2} = \tfrac{1}{2}\left(\frac{N-2}{N} + 1\right)\mathbf{u} = \left(1 - \frac{1}{N}\right)\mathbf{u} = \frac{N-1}{N}\mathbf{u} = \underbrace{\frac{N-1}{\sqrt{N}}}_{d}\mathbf{e}_u$$



z. Bsp. $N = 3 \Rightarrow 2/3\,\mathbf{u}$

$N = 4 \Rightarrow 3/4\mathbf{u}$

$N = 100 \Rightarrow 0,99\mathbf{u}$

A **u** neighbouring corner of the cube produces in the scalar product $\langle \mathbf{u}_1, \mathbf{u} \rangle$ exactly (N-2) !

The equation of the separation plane results as:

$$\langle \mathbf{x}, \mathbf{e}_u \rangle = d = \frac{N-1}{\sqrt{N}}$$

$$\Rightarrow \langle \mathbf{x}, \mathbf{u} \rangle \frac{1}{\sqrt{N}} = \frac{N-1}{\sqrt{N}}$$

$$\Rightarrow \boxed{\langle \mathbf{x}, \mathbf{u} \rangle = (N-1)}$$

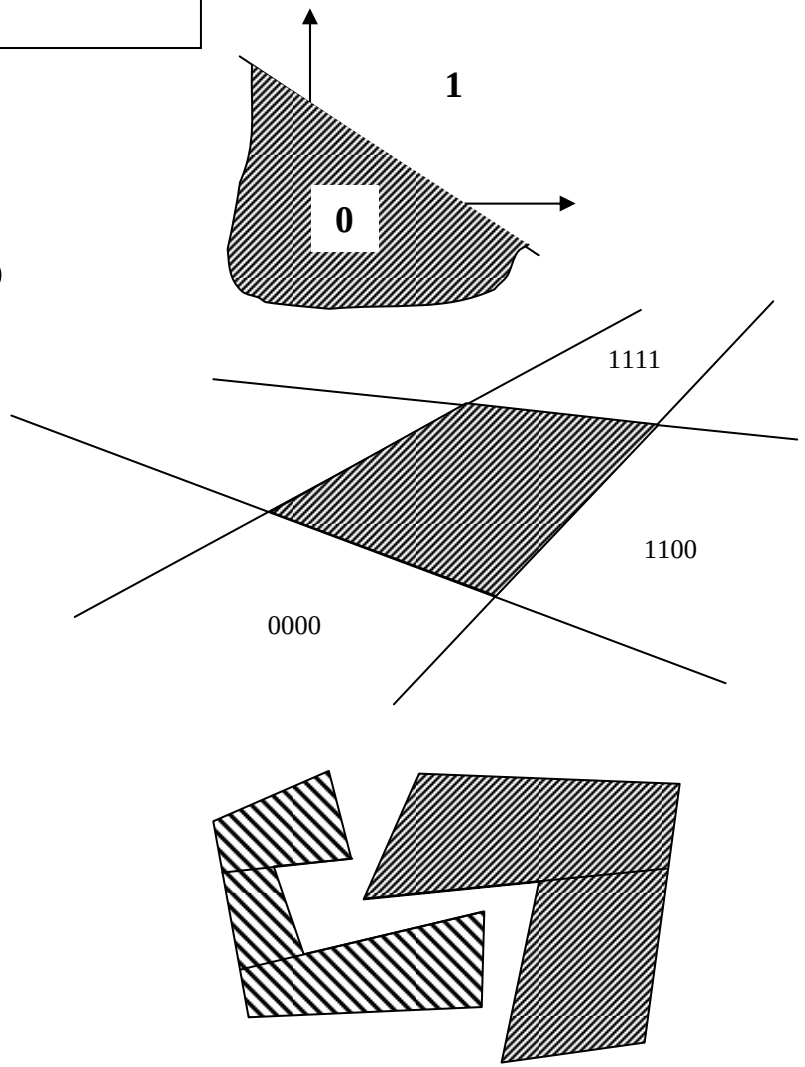
The 3-layer perceptron with threshold function

With a 3-L-S-MLPC arbitrary linealy limited cluster can be classified in feature spaces!!

<u>1st layer:</u> intersecting hyper planes form ***convex polyhedron .*** These are mapped to the ***corners of a hyper cube***.

<u>2nd layer:</u> By cutting corners off the hyper cube ***convexe polyhedrons*** are selected by intersection of half spaces (AND).

<u>3rd layer:</u> ***arbitrary linearly limited areas*** evolve from union of convex areas (OR).

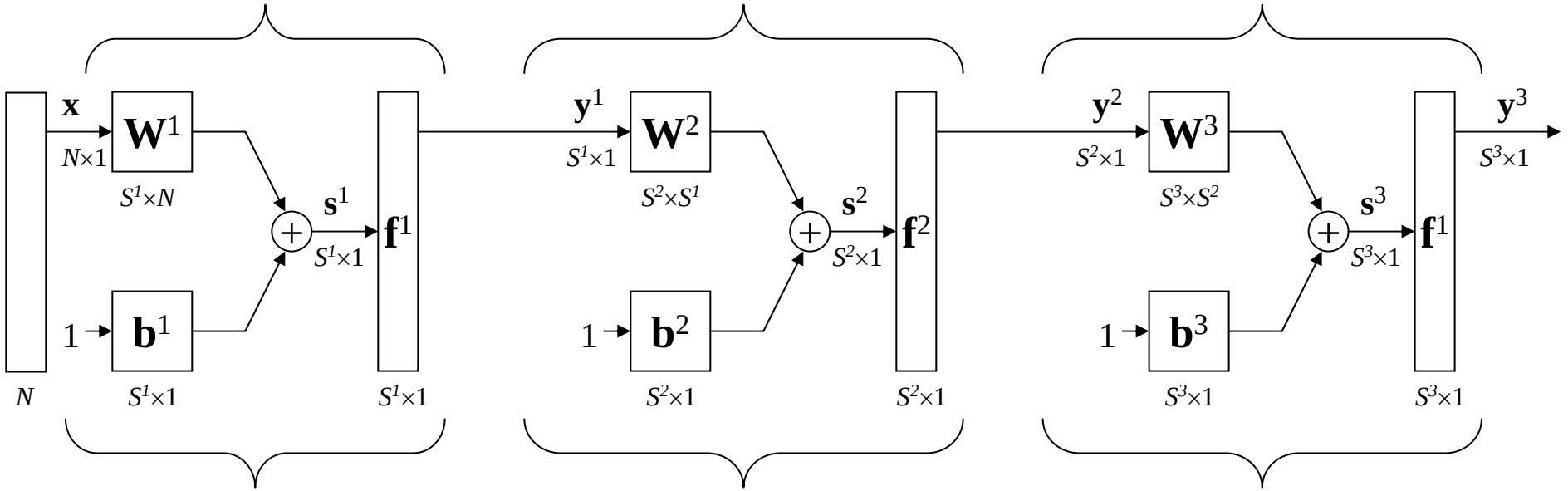


Perceptron with 3 layers for implementing a classifier for arbitrary linearly limited areas (polyhedrons)

first layer:
linearly limited areas
are mapped to
hypercubes

second layer:
intersections of half spaces
(polyhedrons)
are assigned to Boolean
vectors (AND)

third layer:
union of
convex areas (OR)



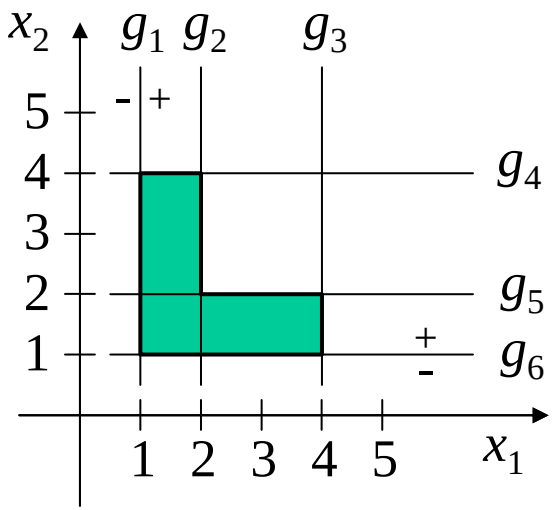
$\mathbf{y}^1=\mathbf{f}^1(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1)$ $\mathbf{y}^2=\mathbf{f}^2(\mathbf{W}^2\mathbf{y}^1+\mathbf{b}^2)$ $\mathbf{y}^3=\mathbf{f}^3(\mathbf{W}^3\mathbf{y}^2+\mathbf{b}^3)$

$$\mathbf{y}^3=\mathbf{f}^3(W^3f^2(W^2f^1(W^1x+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Example: Selecting a non-convex area with a 3-layered perceptron with $f(s)=\text{sign}(s)$

$$\mathbf{y}^3=\mathbf{f}^3(W^3f^2(W^2f^1(W^1x+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

task:



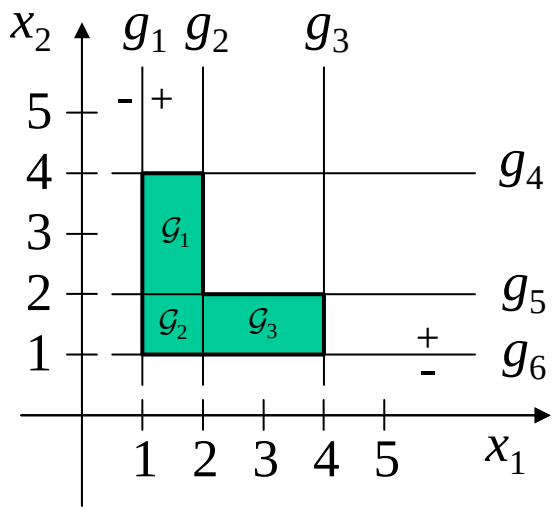
Designing the first layer: definition of hyper planes

$$\mathbf{W}^1=\begin{bmatrix} w_{1,1}=1 & w_{1,2}=0 \\ w_{2,1}=1 & w_{2,2}=0 \\ w_{3,1}=1 & w_{3,2}=0 \\ w_{4,1}=0 & w_{4,2}=1 \\ w_{5,1}=0 & w_{5,2}=1 \\ w_{6,1}=0 & w_{6,2}=1 \end{bmatrix} \begin{matrix} \leftarrow g_1 \\ \leftarrow g_2 \\ \leftarrow g_3 \\ \leftarrow g_4 \\ \leftarrow g_5 \\ \leftarrow g_6 \end{matrix} \qquad \mathbf{b}^1=\begin{bmatrix} -1 \\ -2 \\ -4 \\ -4 \\ -2 \\ -1 \end{bmatrix}$$

e.g.: $g_1: \ \langle \mathbf{w}^1, \mathbf{x} \rangle + b^1 = 0$ with: $\mathbf{w}^1=\begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad b^1=-1$

Designing the second layer:

task:



Designing the second layer:
Cutting corners off the hyper cube

Marking the areas:

	g_1	g_2	g_3	g_4	g_5	g_6
$\mathcal{G}_1$	+	−	−	−	+	+
$\mathcal{G}_2$	+	−	−	−	−	+
$\mathcal{G}_3$	+	+	−	−	−	+

$$\mathbf{W}^2 = \begin{matrix} g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \end{matrix}$$
$$\begin{bmatrix} 1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 \end{bmatrix} \begin{matrix} \leftarrow \mathcal{G}_1 \\ \leftarrow \mathcal{G}_2 \\ \leftarrow \mathcal{G}_3 \end{matrix} \quad \mathbf{b}^2 = \begin{bmatrix} -5 \\ -5 \\ -5 \end{bmatrix}$$

The row vectors of $\mathbf{W}^2$ point to the corners to be cut off; the threshold has value N-1

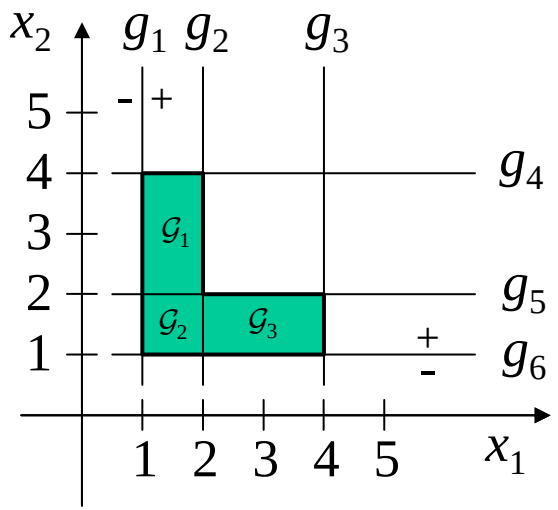
Designing the third layer:

Within the third layer the union of areas has to be implemented, which is done by a simple OR-conjunction. This means that the corner [–1 –1 –1] has to be cut off, which is done using:

$$\mathbf{W}^3 = \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} \qquad b^3 = -2$$

Simplified design of the second and third layer:

task:



Designing the second layer:
Cutting corners off the hyper cube

Marking the areas: the table contains “don‘t care”-elements (∗). The corresponding neurons can be left unconsidered (cutting the link)

	g_1	g_2	g_3	g_4	g_5	g_6
$\mathcal{F}_1 = \mathcal{G}_1 \cup \mathcal{G}_2$	+	–	*	–	*	+
$\mathcal{F}_2 = \mathcal{G}_2 \cup \mathcal{G}_3$	+	*	–	*	–	+

$$\mathbf{W}^2 = \begin{matrix} g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \end{matrix} \begin{bmatrix} 1 & -1 & 0 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 & -1 & 1 \end{bmatrix} \begin{matrix} \leftarrow \mathcal{F}_1 \\ \leftarrow \mathcal{F}_2 \end{matrix} \quad \mathbf{b}^2 = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

The dimension of the reduced cube is only N=4 (without “don‘t care”-elements!)

The row vectors if $\mathbf{W}^2$ point exactly to the corners to be cut off; the threshold has value N-1 of the reduced cube!

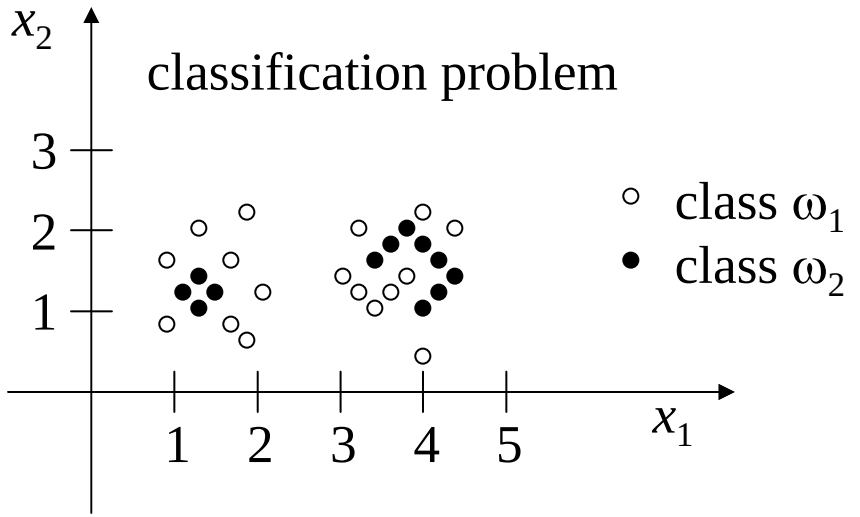
Simplified design of third layer:

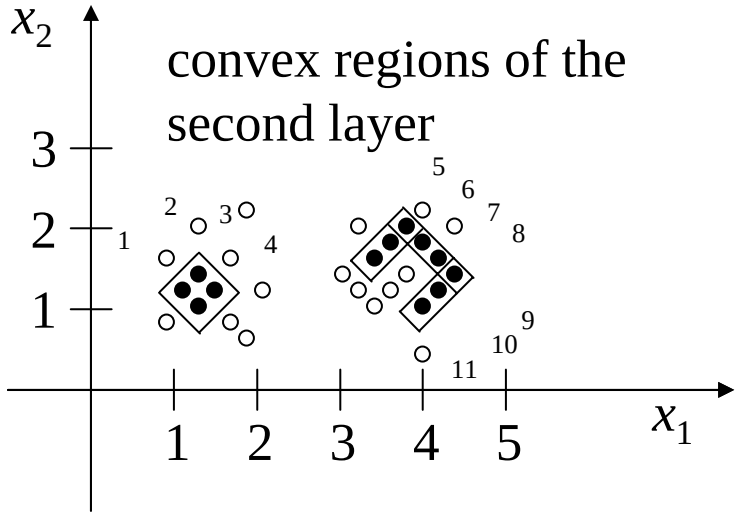
Within the third layer union of areas has to be implemented, which is done by a simple OR-conjunction. This means, that the corner [–1 –1] has to be cut off, which is done using:

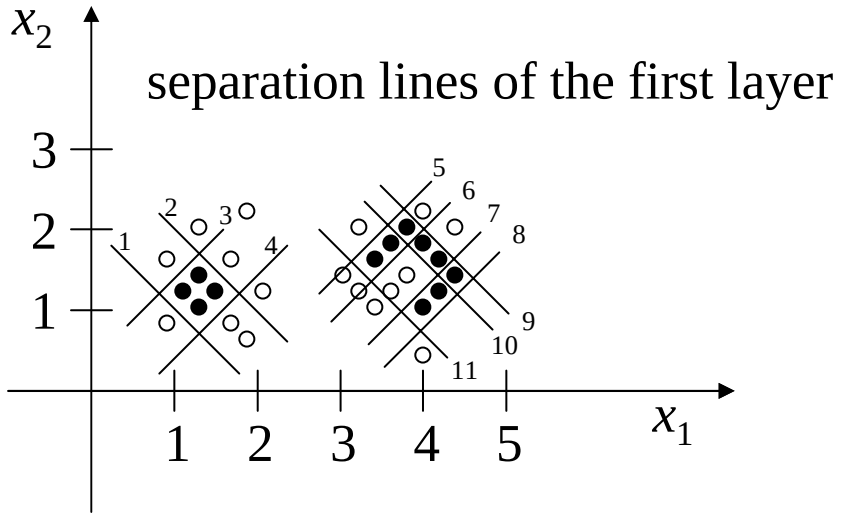
$$\mathbf{W}^3 = \begin{bmatrix} -1 & -1 \end{bmatrix} \qquad b^3 = -1$$

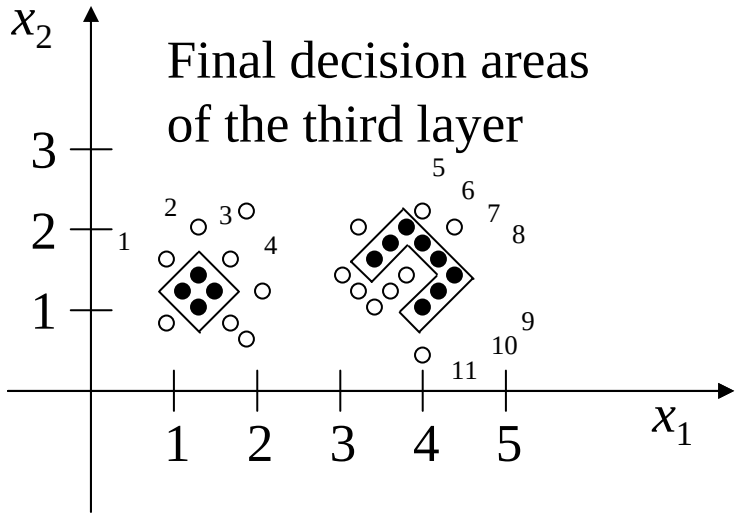
Example for a non linearly separable two-class problem

(M.T. Hagan, H.B. Demuth, M. Beale „Neural Network Design“, PWS Publishing Company, Boston, 1995)









Example for a non linearly separable two-class problem

First layer: 11 lines for separating the areas as function of two input variables are needed!

$$\mathbf{W}^{1T} = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{b}^{1T} = \begin{bmatrix} -2 & 3 & 0.5 & 0.5 & -1.75 & 2.25 & -3.25 & 3.75 & 6.25 & -5.75 & -4.75 \end{bmatrix}$$

Second layer: four convex areas are selected!

$$\mathbf{W}^{2} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad \mathbf{b}^{2} = \begin{bmatrix} -3 \\ -3 \\ -3 \\ -3 \end{bmatrix}$$

Third layer:

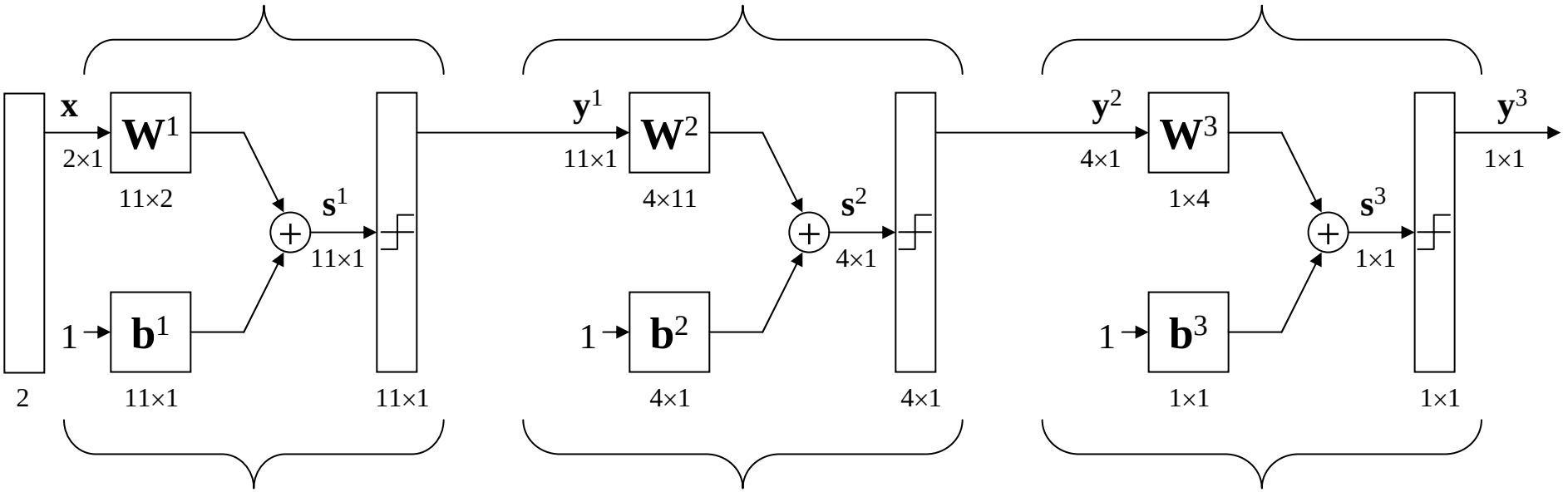
$$\mathbf{W}^{3} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} \quad b^{3} = 3$$

Example for a non linearly separable two-class problem

first layer:
linearly limited areas
are mapped to
hypercubes

second layer:
intersections of half spaces
(polyhedrons)
are assigned to
Boolean vectors (AND)

third layer:
union of
convex areas (OR)



$\mathbf{y}^1=\mathbf{sign}(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1)$ $\mathbf{y}^2=\mathrm{sign}(\mathbf{W}^2\mathbf{y}^1+\mathbf{b}^2)$ $\mathbf{y}^3=\mathrm{sign}(\mathbf{W}^3\mathbf{y}^2+\mathbf{b}^3)$

$$\mathbf{y}^3=\mathrm{sign}(W^3\mathrm{sign}(W^2\mathrm{sign}(W^1x+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

En route to automatic design of a neural network

The approach drafted so far is very descriptive and in the two-dimensional case intuitive, but for higher dimensions it is not applicable. In practice we only have learning samples. An automatic algorithm is needed, that based on the approach so far and on a learning process calculates the optimal weights of the NN autmatically.

If one chooses to employ iterative algorithms for this kind of non-linear optimization, differientiable non-linearities in the neurons are needed (the threshold functions are terminally applicable for this task!)

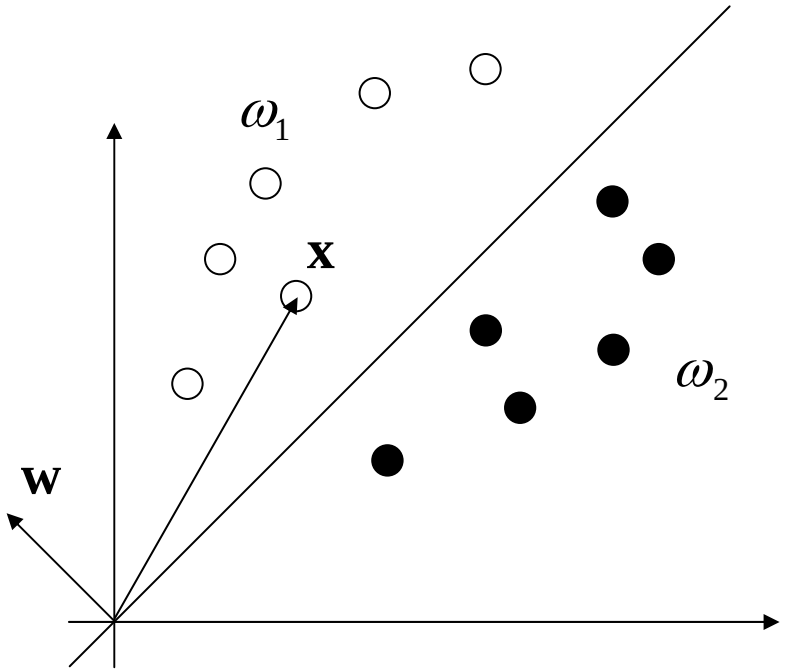
Linearly separable classes – the perceptron algorithm

Target is an algorithm for *automatic* adjustment of the unknown weight matrices $\mathbf{W}^i$ of a perceptron to a classification problem, given a finite number of samples $\{(\mathbf{x}_i, \omega_k)\}$.
In case of linearly separable classes Rosenblatt proposed an iterative algorithm for solving this problem. Supposing that a linear separation plane, defined by the equation $\mathbf{w}^{*T}\mathbf{x}=0$, exists for the two-class problem:

$$\mathbf{w}^{*T}\,\mathbf{x} > 0 \quad \forall \mathbf{x} \in \omega_1$$
$$\mathbf{w}^{*T}\,\mathbf{x} < 0 \quad \forall \mathbf{x} \in \omega_2$$

This includes the case, that the separation plane does not intersect the origin (as it is here), i.e. $\mathbf{w}^{*T}\mathbf{x}+b^*=0$, since this case can by extension of $\mathbf{x}$ be reduced to the equation above:

$$\mathbf{x}'^T = \left[1, \mathbf{x}^T\right] \quad \mathbf{w}'^{*T} = \left[b^*, \mathbf{w}^{*T}\right]$$
$$\Rightarrow \mathbf{w}'^{*T}\mathbf{x}' = \mathbf{w}^{*T}\mathbf{x} + b^*$$



Linearly separable problem

The problem is solved by choosing an applicable objective function and declaration of an optimization algorithm. The perceptron quality measure that is to be minimized is given by (measure for mal-classification):

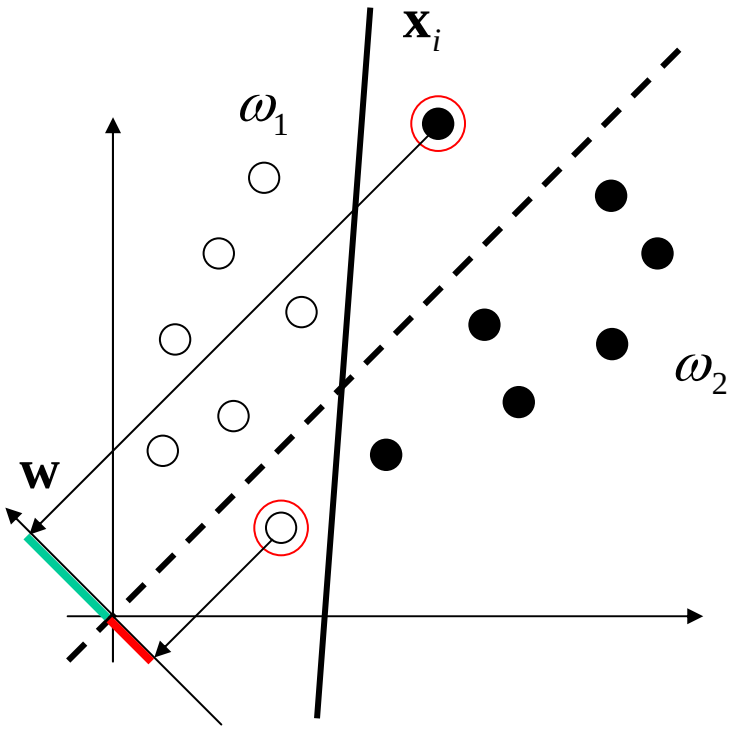
$$J(\mathbf{w}) = \sum_{\mathbf{x}\in\mathbb{Y}} (\delta_x \mathbf{w}^T \mathbf{x}) = \text{—} + \text{—}$$

sum of all absolute values of the projections of the mal-classified vectors to the normal vector of the separation line

mit: $\delta_x = +1$ für $\mathbf{x} \in \omega_1$

und: $\delta_x = -1$ für $\mathbf{x} \in \omega_2$

where $\mathbb{Y}$ contains the set of training vectors, which are *mal*- classified by the given hyper plane.
Obviously the sum above is always positive and becomes zero, if $\mathbb{Y}$ is the empty set, i.e. all elements have been classified *correctly*.
In case of a wrongly classified vector $\mathbf{x} \in \omega_1$ we get with $\mathbf{w}^T\mathbf{x} > 0$ and $\delta_x > 0$ a positive productof the two terms. The same applies for vectors of class ω_2 . The objective function is zero (its minimal value), if all samples have been classified correctly.



The objective function is *continuous and piecewise linear*. If we alter the weight vector little, $J(\mathbf{w})$ changes linearly up to the point where the numer of mal-classified vectors changes. At this point the gradient of J is not defined and not continuous.
The iterative minimization of the cost function is done by an algorithm, which is similiar to gradient descent:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \rho_i \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w_i}}$$

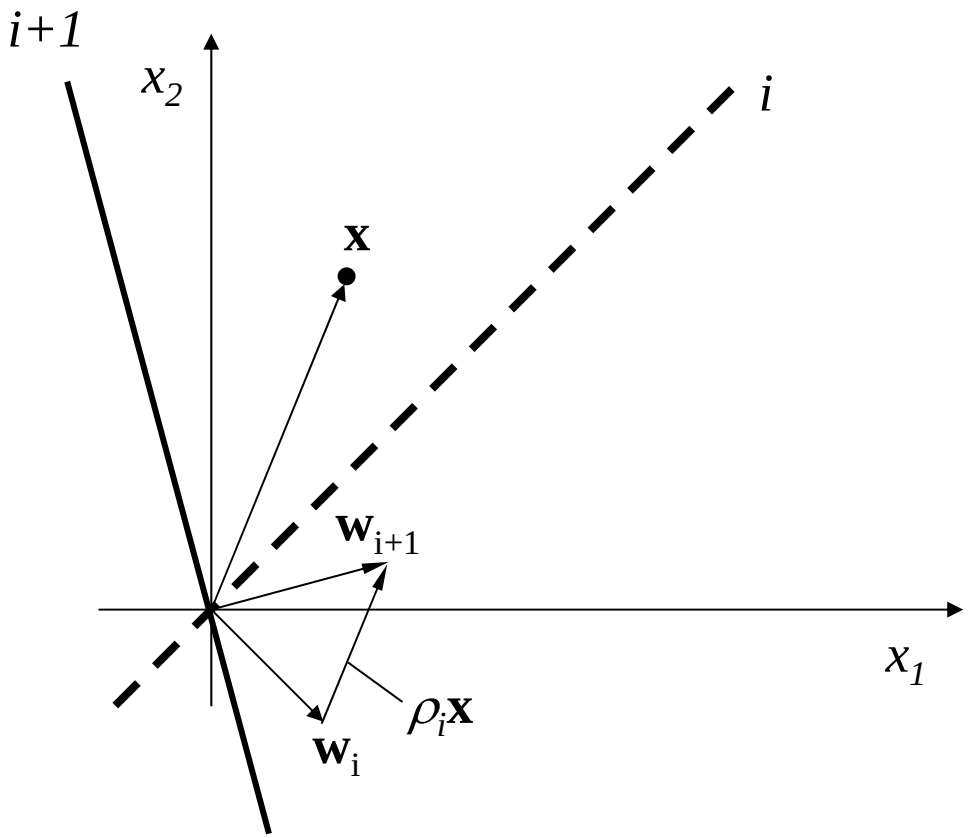
$\mathbf{w}_i$ is the weight vector for the i-th iteration and ρ_i a squence of positive numbers. Points of discontiniuty are to be considered carefully! From the objective function results:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \sum_{\mathbf{x}\in\mathbb{Y}} \delta_x \mathbf{x} \quad \text{due to:} \quad \frac{\partial \langle \mathbf{w}, \mathbf{x} \rangle}{\partial \mathbf{w}} = \mathbf{x}$$

and therefore:

$$\boxed{\mathbf{w}_{i+1} = \mathbf{w}_i - \rho_i \sum_{\mathbf{x}\in\mathbb{Y}} \delta_x \mathbf{x}} \qquad \text{perceptron algorithm}$$

The iteration has to be continued until the algorithm converges.
The following picture illustrates the functionality, supposing, that within iteration step *i* only one **x** is mal-classified
($\mathbf{w}^T\mathbf{x} > 0$) and let ρ_i=0,5. The perceptron algorithm corrects the weight vector *in the direction of* **x**. Therefore the separation line rotates and **x** is correctly classified ($\mathbf{w}^T\mathbf{x} > 0$) .



In the following example we suppose, that we have reached the last but one iteration step and only two feature vectors still are mal-classified:

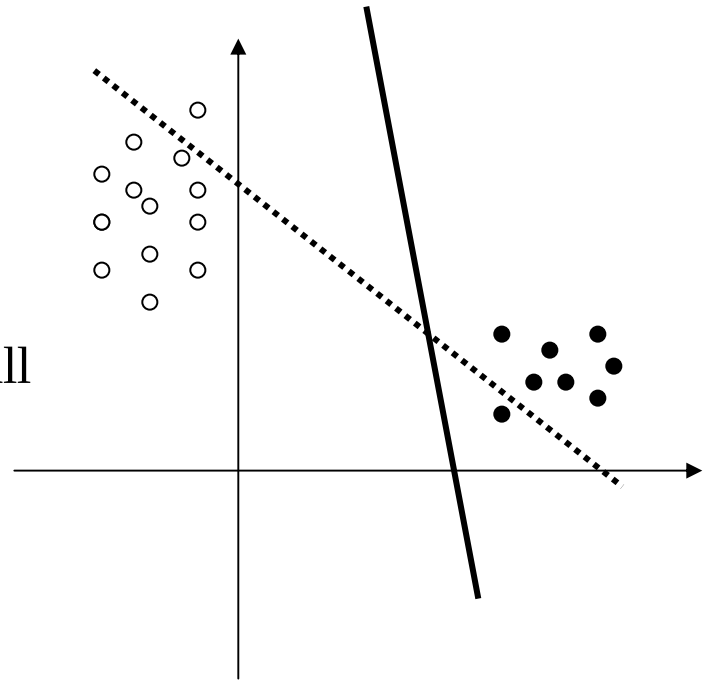
separation line: $x_1 + x_2 - 0,5 = 0 \Rightarrow$ weight vector: $\mathbf{w}^T = \begin{bmatrix} 1 & 1 & -0,5 \end{bmatrix}^T$

mal-classified vectors: $\begin{bmatrix} 0,4 & 0,05 \end{bmatrix}^T$ $\begin{bmatrix} -0,2 & 0,75 \end{bmatrix}^T$

With the perceptron algorithm results the next weight vector with ρ_i=0,7 as:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - 0,7(-1)\begin{bmatrix} 0,4 \\ 0,05 \\ 1 \end{bmatrix} - 0,7(+1)\begin{bmatrix} -0,2 \\ 0,75 \\ 1 \end{bmatrix} = \begin{bmatrix} 1,42 \\ 0,51 \\ -0,5 \end{bmatrix}$$

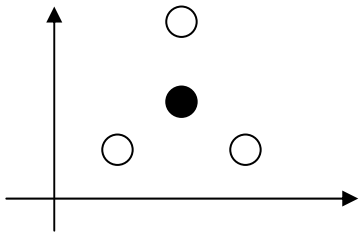
The new separation line: $1,42x_1 + 0,51x_2 - 0,5 = 0$ separates all vectors correctly. The algorithm terminates.



Properties of the perceptron algorithm

- One can prove, that the algorithm converges in a finite number of iterations towards a solution (with properly chosen values of ρ_i, which is always a problem for gradient algorithms), if the condition is fulfilled, that the classes are linearly separable.
- The solution found is not unique though.

- For non linearly separable problems no solution results. Correction in the iteration always has absolute values different from zero and the algorithm therefore cannot terminate.



- Later we will see multi-layer NN, which can solve arbitrary classification problems (even non-linear problems). The l
be used to train neural networks.

Demo from Matlab: NN-Toolbox

-Perceptron Learning rule

-Linearly non-separable vectors