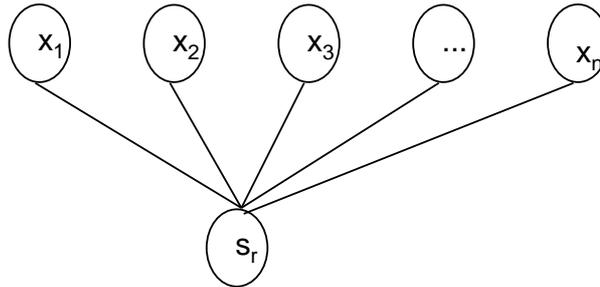


Perceptron

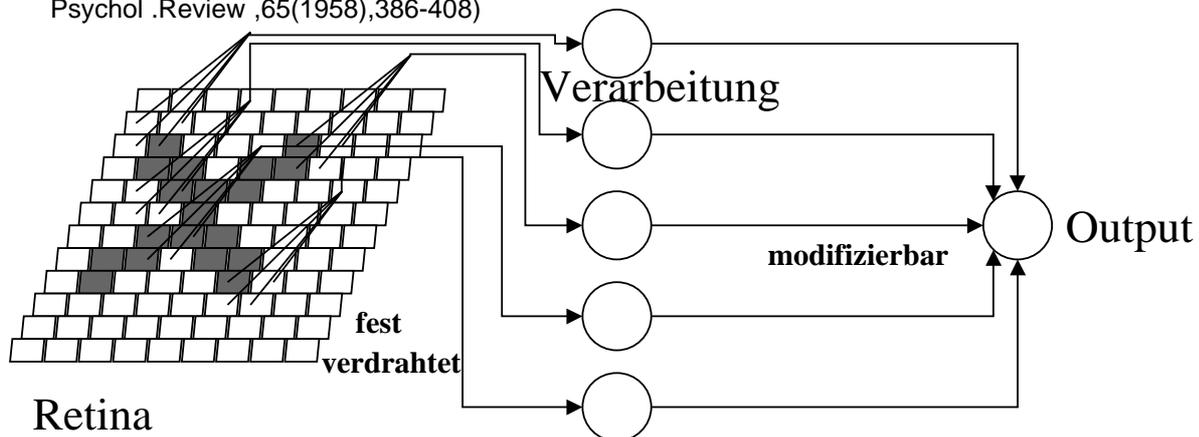
1-Schicht-Struktur, Δ -Regel



historisches Schema

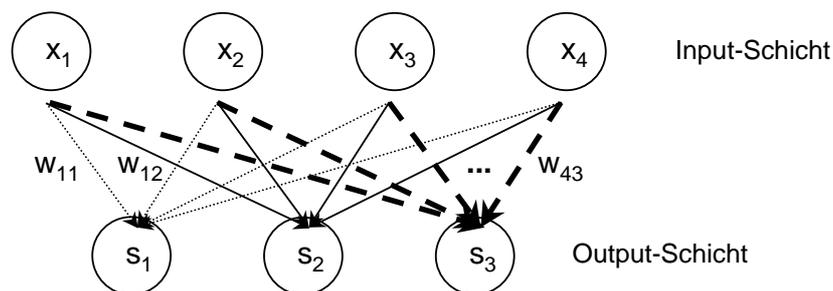
Das von Rosenblatt 1958 entwickelte Perceptron hatte 3 Schichten (Retina - Verarbeitung - Output), aber nur die Verbindungen zwischen Verarbeitung und Output konnten trainiert werden.

(F. Rosenblatt: The Perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Review, 65(1958), 386-408)



- Die Verbindungen zwischen Retina und Verarbeitung stellen eine feste Vorverarbeitung der Eingabe in die Retina auf ein vom Perceptron verarbeitbares Format dar.
- Im Originalmodell werden weder diese Verbindungen noch die Neuronen der Verarbeitungsschicht spezifiziert, lediglich wurden die Ausgaben der Verarbeitungsneuronen auf die Werte 0 oder 1 festgelegt.
- Der eigentliche Lern- und Erkennungsprozess findet also nur zwischen Verarbeitung und Output statt, deshalb werden in moderneren Behandlungen dieses Netztyps nur diese zwei Schichten betrachtet und die Vorverarbeitung ignoriert.

- So sieht ein Perceptron für unsere Untersuchungen wie folgt aus:



- Dabei sieht man sofort, daß man für jedes Output-Neuron eine eigene von den anderen Netzteilen unabhängige Netzteil hat (hierdurch unterschiedliche Verbindungen dargestellt).
 - →
 -
 -
- Es genügt also, das Perceptron mit nur einem Output-Neuron zu untersuchen.

das Perceptron

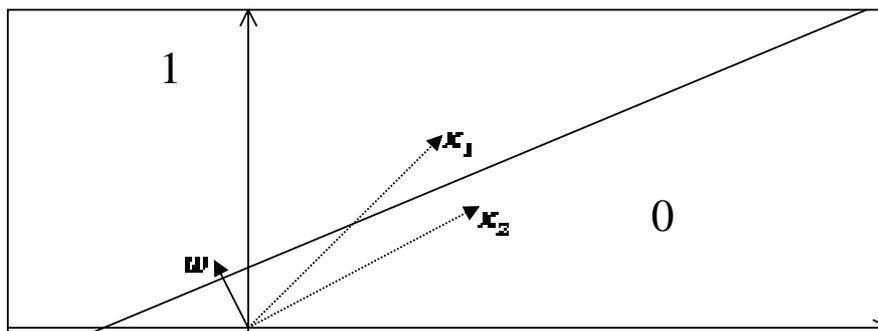
- Ein Perceptron verfügt also über eine Input-Schicht, über die die Input-Vektoren $\mathbf{x}=(x_1, \dots, x_n)$ eingegeben werden.
- Es hat n Verbindungen zum Output-Neuron mit dem Gewichtsvektor $\mathbf{w}=(w_1, \dots, w_n)$.
- Das Output-Neuron hat den Schwellwert s , errechnet seine Aktivität $a = \mathbf{x}\mathbf{w} = x_1w_1 + \dots + x_nw_n$ nach der Skalarprodukt-Regel und verwendet die Sprungfunktion $(x < s) ? 0 : 1$ als Output-Funktion.
- Der Output errechnet sich also aus der Formel:

$$o = 0 \quad \text{falls } \mathbf{x}\mathbf{w} = x_1w_1 + \dots + x_nw_n < s$$

$$o = 1 \quad \text{sonst;}$$
 es entscheidet also das Vorzeichen des Skalarprodukts $\mathbf{y}\mathbf{v} = x_1w_1 + \dots + x_nw_n - s$ der erweiterten Vektoren $\mathbf{y}=(1, x_1, \dots, x_n)$ und $\mathbf{v}=(s, w_1, \dots, w_n)$ über den Output.

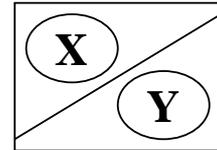
Geometrische Interpretation

- Die Gleichung $\mathbf{w}\mathbf{x} = s$ beschreibt eine Hyperebene im n -dimensionalen Raum, auf der der Vektor \mathbf{w} senkrecht steht.
- Der Inputraum wird dadurch in zwei Hälften geteilt, auf der eine Hälfte ($\mathbf{w}\mathbf{x} < s$) der Wert 0 ausgegeben wird, auf der anderen Hälfte der Wert 1.



Trainings-Problem

- Das zu lernende Problem besteht aus zwei Mengen X, Y von erweiterten Input-Vektoren die auf X mit 1 und auf Y mit 0 bewertet werden sollen.
- Gesucht ist also ein (erweiterter) Gewichtsvektor \mathbf{w} mit $\mathbf{x}\mathbf{w} > 0$ und $\mathbf{y}\mathbf{w} < 0$ (d.h. $-\mathbf{y}\mathbf{w} > 0$) für alle $\mathbf{x} \in X$ und $\mathbf{y} \in Y$. (Beachte, daß durch eine kleine Veränderung des Schwellwertes der Fall $\mathbf{x}\mathbf{w} = 0$ eliminiert werden kann.)
- In der geometrischen Interpretation bedeutet das, wir suchen eine Hyperebene, die die Mengen X und Y voneinander trennt.
- Als weitere Vereinfachung können wir $Z = X \cup Y$ betrachten, für das nun noch die Bedingung $\mathbf{z}\mathbf{w} > 0$ erfüllt werden muß.



die Delta-Regel

- Das Perzeptron wird nach der Delta-Regel output-orientiert trainiert.
- Beim Training wird dem Netzwerk Input aus einer Trainingsdatei präsentiert, für die die gewünschten (soll-) Outputs bekannt sind.
- Die aktuellen Ist-Outputs des Netzes werden mit den Soll-Outputs verglichen und im Falle einer Diskrepanz die Gewichte und der Schwellwert nach folgender Formel angepaßt:
- Delta-Regel: $(w_0 = -s, x_0 = 1)$

$$w_{i,\text{neu}} := w_{i,\text{alt}} + x_i \cdot (\text{Output}_{\text{soll}} - \text{Output}_{\text{ist}})$$
 (der Faktor x_i sorgt dafür, daß eine Angleichung nur erfolgt, wenn die Verbindung w_i tatsächlich zum Output beiträgt.)

Der Perceptron Lernalgorithmus kann nun wie folgt interpretiert werden

- Auf Z soll das Perceptron mit 1 antworten, also $z \cdot v > 0$ für ein geeignetes v gelten. Wir beginnen mit einem beliebigen v .
- Ist $z \in Z$ ein Element, das noch nicht korrekt interpretiert wird ($z \cdot v \leq 0$), dann erzeugt die Delta-Regel ein neues $v_{\text{neu}} := v_{\text{alt}} + z$ und beim nächsten Versuch mit dem selben z ist $z \cdot v_{\text{neu}} = z \cdot v_{\text{alt}} + z \cdot z > z \cdot v_{\text{alt}}$.
- Das Verfahren beginnt also mit einem beliebigen erweiterten Gewichtsvektor v_0 und addiert (subtrahiert) sukzessive erweiterte Inputs z , bei denen der Output noch nicht korrekt ist.

Der Perceptron Konvergenz Satz

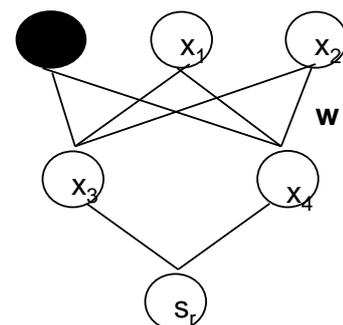
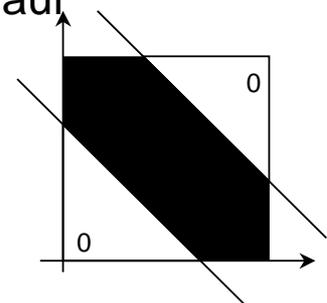
- **SATZ:**
Wenn das Perceptron eine Klasseneinteilung überhaupt lernen kann, dann lernt es diese mit der Delta-Regel endlich vielen Schritten.
 - Problem:
Wenn der Lernerfolg bei einem Perceptron ausbleibt, kann nicht direkt erkannt werden, ob
 1. das Perceptron die Klassifikation prinzipiell nicht lernen kann, oder
 2. der Lernalgorithmus mit seinen endlich vielen Schritten noch nicht fertig geworden ist, denn der Satz als reiner Existenzsatz gibt keinen Anhaltspunkt über eine obere Schranke für die Anzahl von Lernschritten.

Konvergenzbeweis

- Nach Voraussetzung gibt es ein w mit $wz > 0$ für alle $z \in Z$ und wir beginnen den Algorithmus mit $v_0 = 0$.
- Sei a das Minimum aller $wz > 0$ und M das Maximum aller z^2 .
- $v_i = z_1 + \dots + z_i$, also ist $wv_i = wz_1 + \dots + wz_i \geq i \cdot a$.
- Wegen $(wv_i)^2 \leq w^2 v_i^2$ folgt $v_i^2 \geq i^2 \cdot a^2 / w^2$
- Andererseits ist für alle $k \leq i$ ist stets $v_{k-1} z_k < 0$ also $v_k^2 = (v_{k-1} + z_k)^2 = v_{k-1}^2 + 2 \cdot v_{k-1} z_k + z_k^2 < v_{k-1}^2 + z_k^2$ also ist $v_i^2 < z_1^2 + \dots + z_i^2 < i \cdot M$
- Es folgt $i \cdot M > i^2 \cdot a^2 / w^2$ und damit $i < M \cdot w^2 / a^2$
Ohne Kenntnis von w kennen wir weder w^2 noch a , also können wir die obere Schranke von i nicht aus dem Beweis bestimmen!

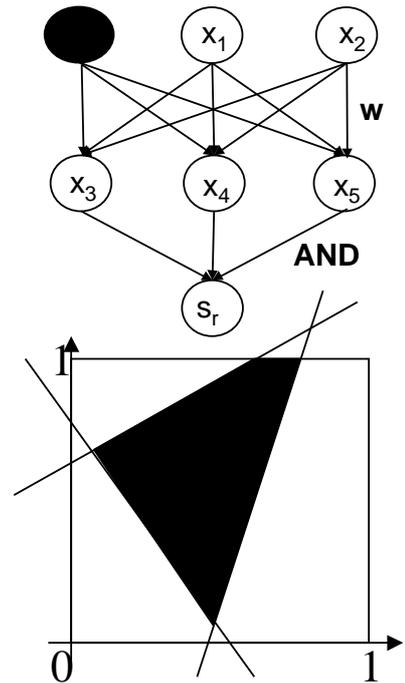
Das XOR-Problem

- Die boolesche Funktion XOR stellt ein nicht linear separierbares Problem dar, weil sich die beiden Klassen auf Diagonalen gegenüberliegen.
- Mit einer Zwischenschicht lassen sich aber zwei trennende Hyperebenen definieren, zwischen denen die eine der Klassen lokalisiert werden kann.
- komplexere Klassifikationen lassen sich durch Einfügen von Zwischenschichten realisieren.
- Problem: Wie trainiert man mehrere Schichten?



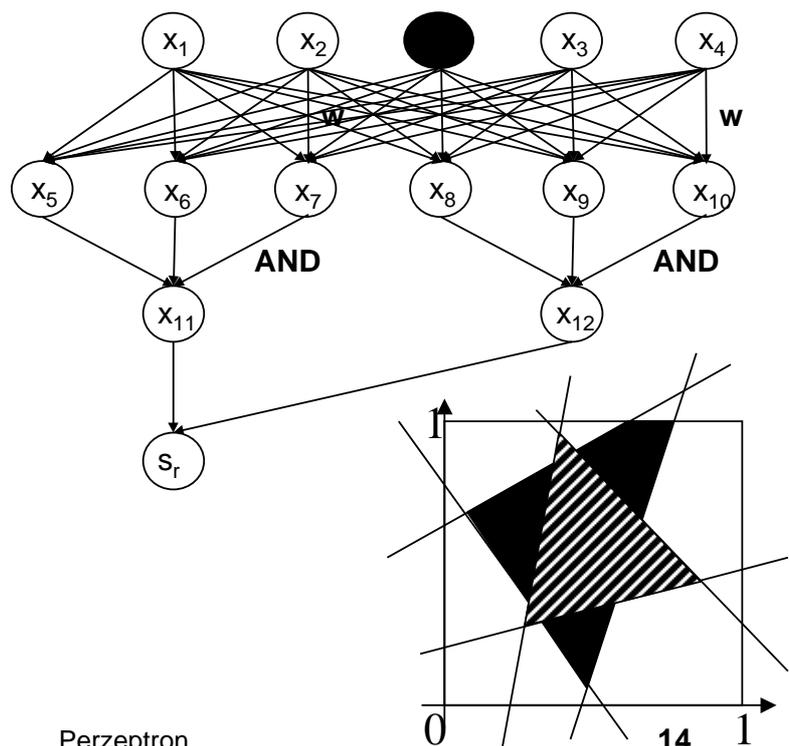
zweistufiges Perceptron

- Wenn man ein Perceptron mit n Output-Neuronen in ein neues Output-Neuron mit AND-Verbindungen (Gewicht 1, Schwellwert $n-0,5$) anhängt, erhält man ein konvexes n -fläch (den Durchschnitt von n Halbräumen) als akzeptierten (Output 1) Bereich.



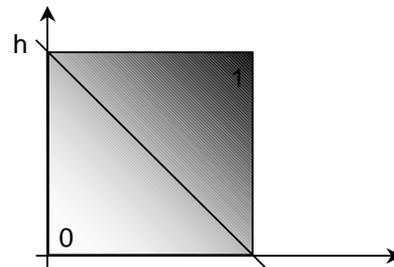
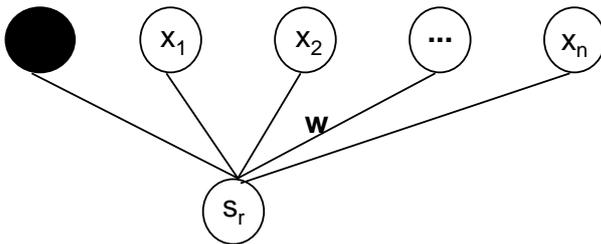
dreistufiges Perceptron

- Miteiner weiteren Output-Stufe durch ANDNOT (Gewicht 1, -1, Schwellwert 0,5) angehängt, kann man sogar nicht zusammenhängende Bereiche als akzeptierten Bereich modellieren. (In dem Beispiel der sichtbare schwarze Bereich)



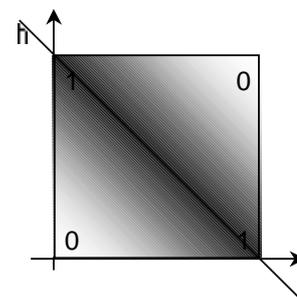
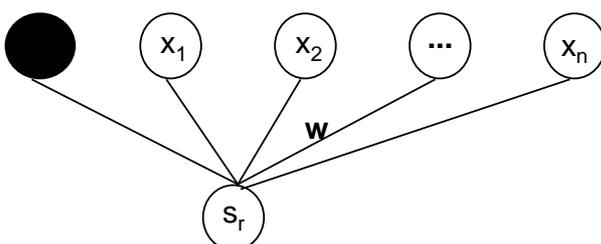
sigmoide Outputfunktion

- Statt zweier Klassen (pro Output-Neuron) entsteht parallel zu einer Hyperebene ein Übergang zwischen den Extrema 0 und 1.
- Die Lage der trennenden Hyperebene wird durch die Richtung des Gewichtsvektors \mathbf{w} bestimmt.
- Die Steilheit des Übergangs wird durch die Länge $|\mathbf{w}|$ des Gewichtsvektors bestimmt



radiale Basis-Outputfunktion

- Wählt man für das Outputneuron eine radiale Basisfunktion als Output, so erhält man eine Outputfläche mit dem Maximum entlang einer Hyperebene.
- Die Lage der Hyperebene wird durch die Richtung des Gewichtsvektors \mathbf{w} bestimmt.
- Die Steilheit des Übergangs zu den minimalen Werten wird durch die Länge $|\mathbf{w}|$ des Gewichtsvektors bestimmt



Einzel-schritt/Gesamt-schritt

Die Delta-Regel kann sowohl im Einzel-schritt- wie auch im Gesamt-schritt-Verfahren angewandt werden

- Einzel-schritt-Verfahren:
 - + Nach jedem Input werden die Gewichte für diesen Input angepasst.
 - Das Lernverfahren kann sich in die Länge ziehen, weil verschiedene Inputs des Trainingssets immer wieder Anpassung desselben Gewichts in entgegengesetzte Richtungen erfordern.
- Gesamt-schritt-Verfahren:
 - + bei jedem Gesamtdurchgang durch alle Trainingsdaten werden die Outputs (IST und SOLL) aufaddiert und gemäß den Summen die Gewichtsveränderung nach dem Durchgang ausgeführt.
 - Beim Aufaddieren können sich verschiedene Fehler gegeneinander aufheben und dadurch den Lernvorgang in die Länge ziehen.

Lernprogramm

- Die Lernregel in Programmstruktur lautet:

```

repeat
reset(Trainingsfile); abbruch := true;
while not EOF(Trainingsfile) do
  begin read(Item, Trainingsfile);
  o=Output(Item.Input);
  for j in OutputNeuronen do
    if (o[j]=Item.SollOutput[j])
      then begin end {tue nichts}
    else begin abbruch:=false;
      if o[j]=0 {SollOutput[j]=1}
      then for k in InputNeuronen do
        w[k, j]:=w[k, j]+Item.Input[k]
      else {o[j]=1 SollOutput[j]=0}
        for k in InputNeuronen do
          w[k, j]:=w[k, j]-Item.Input[k];
        end {else}
      end; {while}
until abbruch;
end {repeat}

```