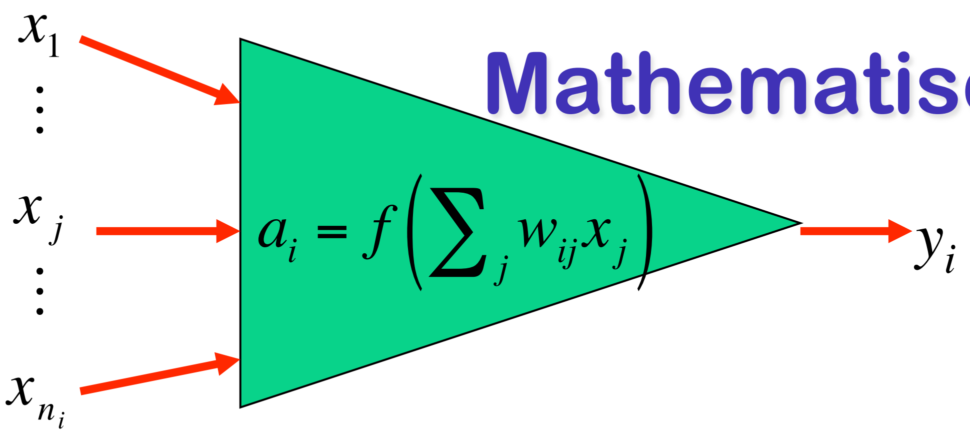
Mathematisches Neuronenmodell



Neuron i

- **Aktivierung** a_i des Neurons i ist gewichtete Summe der Inputs
- **Ausgabe** y_i definiert über Schwellwertfunktion, z.B.

$$f(z) = H_\theta(z) = \begin{cases} 0 & \text{falls} \quad z < \theta \\ 1 & \text{sonst} \end{cases}$$

(alternativ: Sigmoid-Funktion; vgl. Perzeptron mit Schwelle θ, Folie 234)

- Wert y_i ist Eingabe für andere Neuronen
- Aktivierungen und Ausgaben aller Neuronen für Zeit $t+1$ werden synchron aus Werten für t berechnet

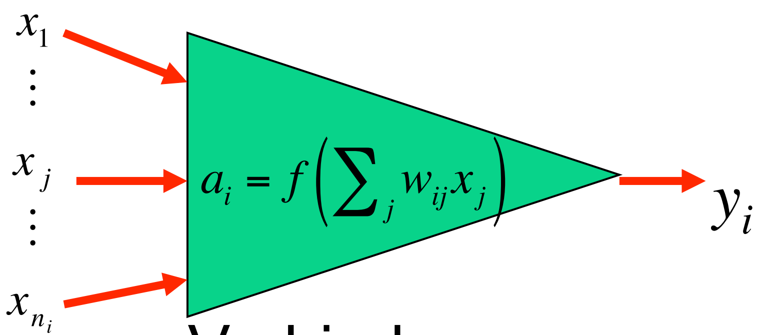
Kritik an der Interpretation des Gebiets NN

Ertel S. 246:

„… anhand von Simulationen eines einfachen mathematischen Modells neuronaler Netze zu erklären, wie z.B. Mustererkennung möglich wird.“

- Das Modell von 1943 ist trivial bis zur Wertlosigkeit! Neuere Erkenntnisse zeigen für biologische Neuronen klar:
  - **Zeitverlauf, (A-)Synchronizität, Signallaufzeiten, Aktivierungsdynamik, … spielen entscheidende Rollen bei biologischer Informationsverarbeitung!**
- Dass ein Biologie-Neuronenmodell von 1943 zur Standard-Definition von **algorithmischen** Neuronalen Netzen gedient hat, ist völlig irrelevant für deren Betrachtung
- Von unreflektiertem Bio-Hype abgesehen, sind algorithmische N.N.e wie definiert sinnvoll

Die Hebbsche Lernregel



- Zentralidee in neuronalen Netzen: Modellierung von Lernen
- Wie soll das gehen?: Neue Neuronen, neue Verbindungen, neue *f*-Funktion?
- Hebb, 1949: **Lernen in (biologischen) neuronalen Netzen funktioniert durch Änderung der Gewichte w_{ij}!**

Ertel schreibt:

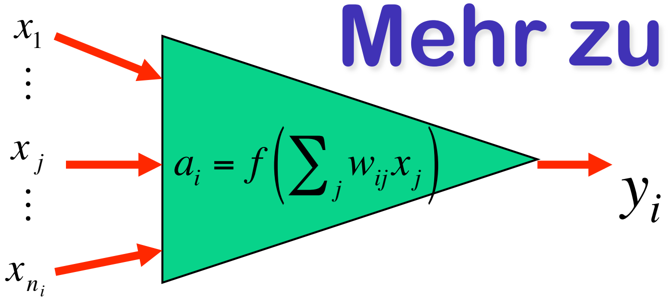
Wenn es eine Verbindung w_{ij} von Neuron j und Neuron i gibt und wiederholt Signale von Neuron j zu Neuron i geschickt werden, was dazu führt, dass die beiden Neuronen gleichzeitig aktiv sind, dann wird das Gewicht w_{ij} verstärkt. Eine mögliche Formel für die Gewichtsänderung Δw_{ij} ist

$$\Delta w_{ij} = \eta x_i x_j$$

mit der Konstante η (Lernrate), welche die Größe der einzelnen Lernschritte bestimmt.

Ertel meint:
sei a_i Aktivierung v. Neuron i

$$\Delta w_{ij} = \eta \cdot a_i x_j$$



Mehr zum Hebbschen Lernen

$$\Delta w_{ij} = \eta \cdot a_i x_j$$

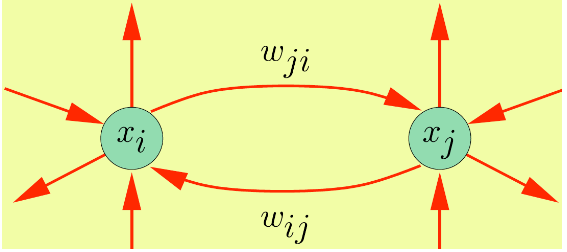
- Lernrate vgl. Perzeptron, Folie 236
- Für positive a_i, x_j würde w_{ij} monoton steigen; daher baue „Verfallsrate“ (*decay rate*) in Gewichtsaktualisierung ein: In jedem Zeittakt multipliziere alle Gewichte mit z.B. $0{,}99$
- Vergleiche <u>nicht-Hebbsche</u> Perzeptron-Lernregel (Folie 236): Für Klassifikation (überwachtes Verfahren!) ist Änderung ± der w_{ij} plausibel, Δw_{ij} kann also auch negativ sein!
- Für unterschiedliche Netztypen und Anwendungen bei algorithmischen NN gibt es unterschiedliche Varianten des Hebbschen Lernens!

Klassen von Neuronalen Netzen

Je nach Struktur/Topologie der Netzknoten und Art der Verbindungen entstehen unterschiedliche Klassen von NN. Sie haben unterschiedliche Eigenschaften und gebrauchen unterschiedliche Lernalgorithmen. (i.d.R. überwachtes Lernen)

Zum Beispiel

- **Perzeptron** (s. Folie 234 ff): Neuronen in Schichten (input, output, ggf. *hidden layer*), Verbindungen bilden gerichteten azyklischen Graphen (von input- nach output-Schicht ↪**feedforward-Netz**), Perzeptron-Lernregel
- **Rekurrente Netze** (RNN): Zyklen im Verbindungsgraph sind erlaubt; ggf. identifiziere input- und output-Schichten mit beliebigem RNN „dazwischen“ (**Reservoire Computing**). Verbreitete Klasse von RNN: **Hopfield-Netze**



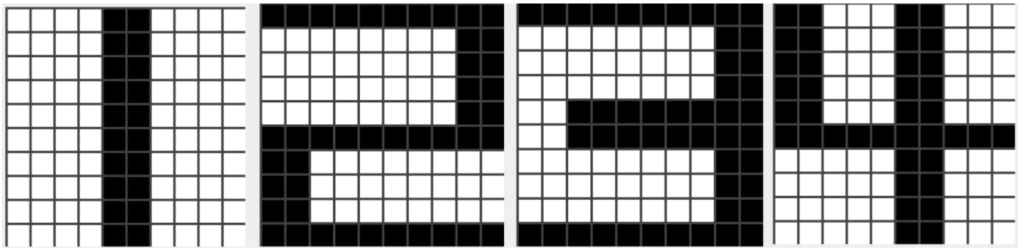
Hopfield-Netze

Bemerkung: Ertel setzt Aktivierung = Ausgabe, also a_i=y_i und nennt beides x_i. Bei HN ist das okay.

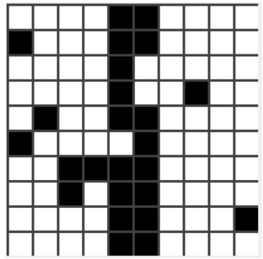
- Rekurrentes Netz; vollständige Vernetzung aller Neuronen untereinander, wobei f.a. i, j gilt: w_{ii}=0 (kein Neuron mit sich selbst rückgekoppelt) und w_{ij}=w_{ji} (Symmetrie der Gewichte) (biologisch unsinnig; stört nicht, da wir hier nicht Biologie machen!)
- Es gibt keine Ein-/Ausgabeschicht – alle Neuronen werden zur E/A verwendet bzw. betrachtet
- binäre Knotenaktivierungen ±1 (Perzeptron: 0/1!)
- Lernen mittels „verallgemeinerter Hebbscher Regel" (s.u.)
- **Trainingsphase**: Für ein HN mit n Neuronen stelle die Gewichte entsprechend N Trainingsmustern mit je n Bit ein **Assoziationsphase**: Aktiviere die Neuronen entsprechend Testmuster aus n Bit; Ergebnis ist das n-Bit-Muster, in welches das HN konvergiert.

Anwendungsbeispiel: Musterassoziation

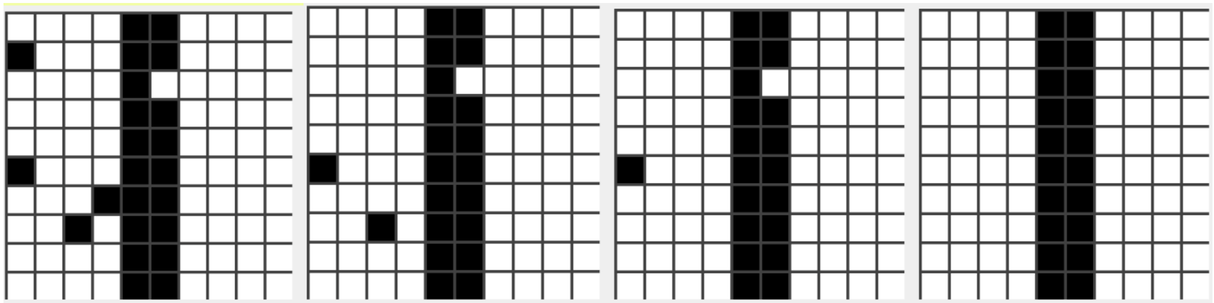
- 10×10 bit Pixelmuster ↪ HN mit $n=100$ Neuronen, vollständig vernetzt
- $N=4$ Trainingsbeispiele



- Assoziation: Lege 10×10-bit-Muster auf die Neuronen, z.B. „1“ mit ca. 10% Rauschen



- Resultat eines HN-Laufs z.B.:



62 Schritte | 144 Schritte | 232 Schritte | 379 Schritte

Hopfield-Netze: Trainingsphase

Aufgabe

- Trainiere N binär kodierte Muster $\boldsymbol{q}^1, \ldots, \boldsymbol{q}^N$, jeweils aus n Pixeln $q_i^j \in \{-1,1\}$
- Finde Gewichtsmatrix symmetrisch, Diagonalelemente $w_{ii}=0$

Lernregel: Verallgemeinerte Hebb-Regel: f.a. $i,j \in \{1,\ldots,n\}, i \neq j$

$$w_{ij} = \frac{1}{n} \sum_{k=1}^{N} q_i^k q_j^k$$

Setze alle $w_{ii}=0$

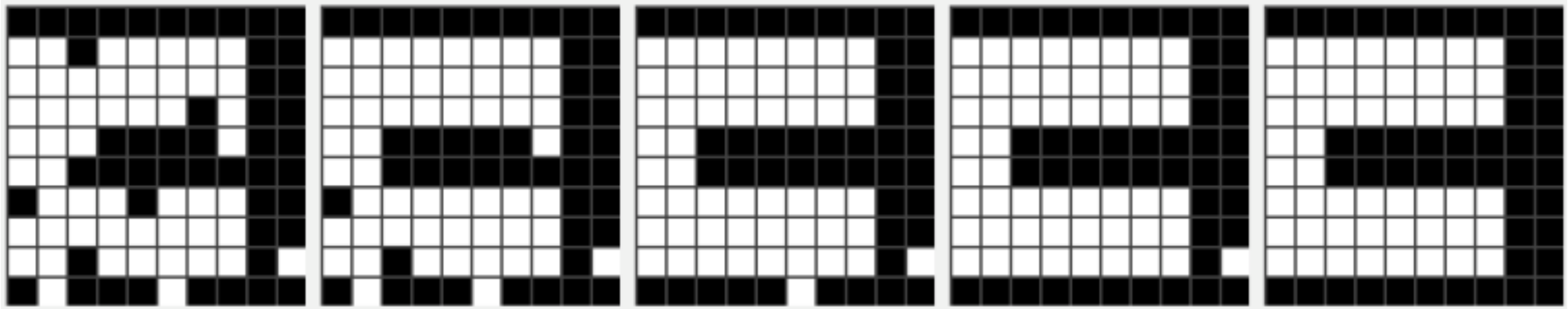
- Jedes Muster k, bei dem die Pixel i,j denselben Wert haben, gibt positiven Beitrag zu w_{ij} (=w_{ji}) sonst negativer Beitrag

Hopfield-Netze: „Autoassoziation“

„Anwendung“ eines trainierten Hopfield-Netzes geht so:

- Gegeben Muster $\boldsymbol{q}$
- Initialisiere Aktivierungswerte a_i aller Neuronen mit entsprechendem $\boldsymbol{q}$-Wert, also f.a. $i \in \{1,\ldots,n\}$, $a_i = q_i$
- Repeat
  - Wähle zufällig Neuron i aus
  - Setze $a_i = \begin{cases} -1 & \text{falls } \sum_{j=1}^{n} w_{ij} x_j < 0 \\ 1 & \text{sonst} \end{cases}$
- Until Aktivierung stabil (z.B. unverändert in k Läufen)

Eigenschaften der Autoassoziation

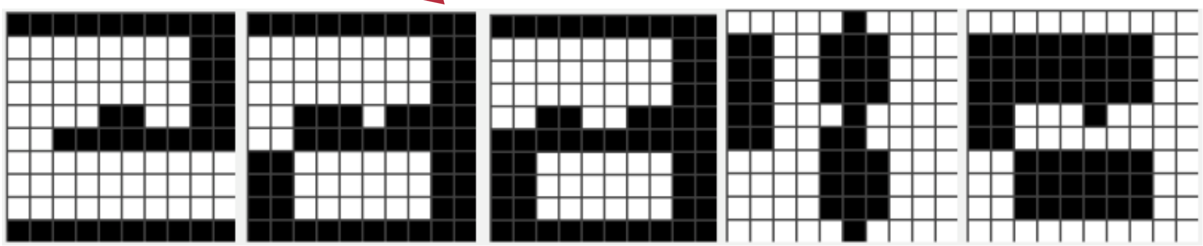


10% Rauschen | 66 Schritte | 112 Schritte | 150 Schritte | 359 Schritte

- „Rekonstruiert"/„assoziiert" trainierte Muster zu möglicherweise verrauschten Eingaben
- Konvergiert garantiert in stabilen Zustand („Energieminimum", Ertel 9.2.2)
- Nicht alle stabilen Zustände entsprechen trainierten Eingaben!

Beispiel (Ertel): Einige weitere stabile Zustände im trainierten 10x10-HN

invers!



Ausblick zu Hopfield-Netzen

- Ertel schreibt (9.2.3, S.255), HN hätten „biologische Plausibilität". Haben sie <u>nicht</u>! ($w_{ij}=w_{ji}$, vollst. Vernetzung,...)
- Als einfache Form rekurrenter NNe sind sie aber mathematisch faszinierend: diskrete, deterministische **dynamische Systeme**
  - interner Zustand (wegen Rückkopplungen nie „leerer input")
  - **Attraktoren** (stabile Zustände, lokale Energieminima) analytisch extrem schwierig zu charakterisieren („**chaotisches System**")
  - s. Forschung/Lehre Prof. Frank Pasemann, CogSci
- Stochastische Verfahren, um lokale Energieminima zu verlassen: **Boltzmann-Maschinen**, Äquivalenz zu Simulated Annealing (s. Folie 155)

Was weiter im Ertel zu NN? 1/3

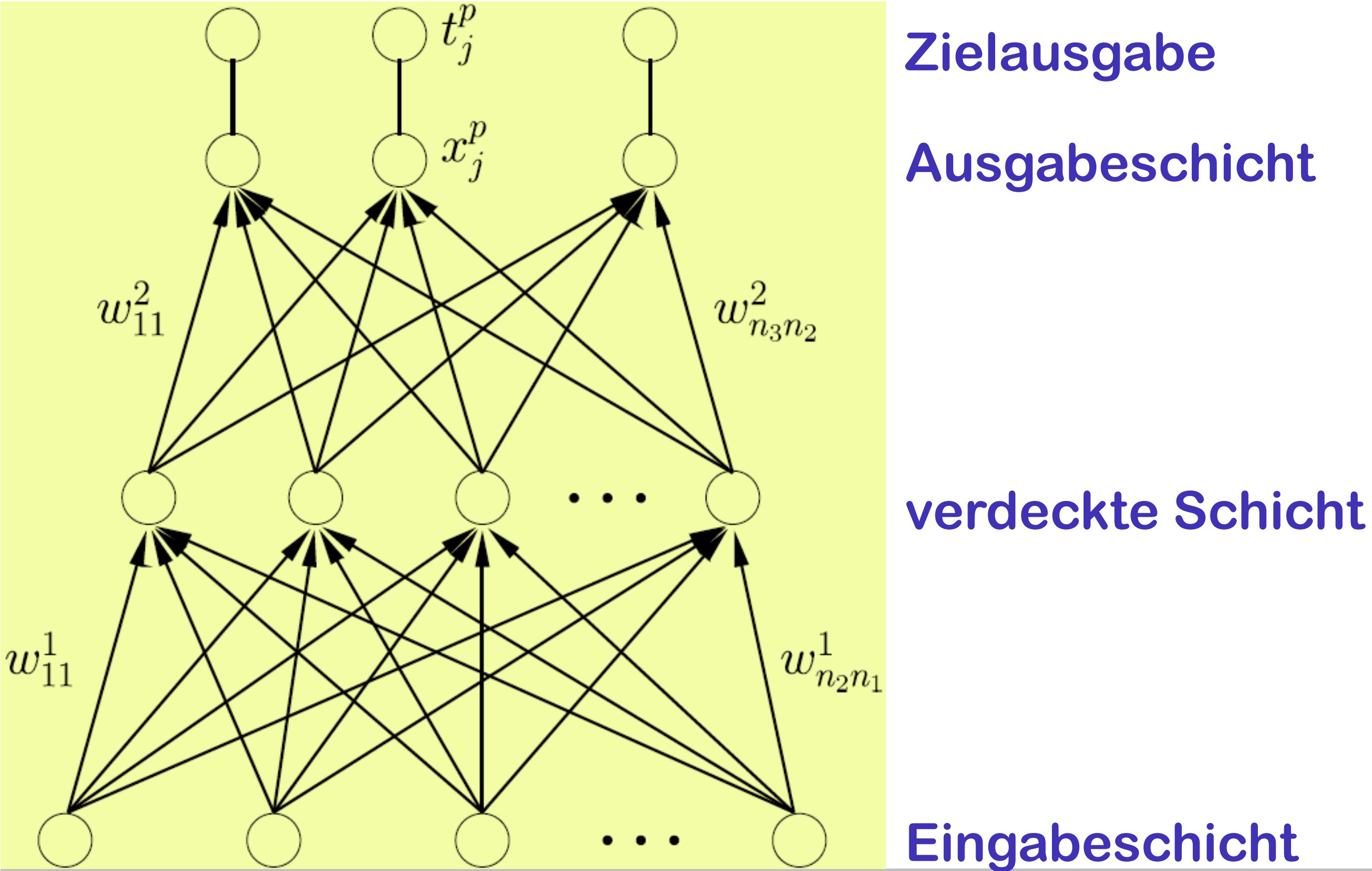
9.4 Lineare Netze mit minimalem Fehler

- Trainingsbeispiele dürfen Fehler enthalten! Also:
  für N Paare von Trainingsvektoren $\{(\mathbf{q}^1, \mathbf{t}^1), \ldots, (\mathbf{q}^N, \mathbf{t}^N)\}$
  ($\mathbf{q}$ input, $\mathbf{t}$ erwarteter output)
  minimiere quadratischen Fehler $\sum_{j=1}^{N} \left(f\left(\mathbf{q}^j\right) - \mathbf{t}^j \right)^2$
- Lösung: Fehlerminimierung durch Nullsetzen seiner partiellen Ableitung nach ∂w_j und Lösen des entstehenden Gleichungssystems
- Verfeinere das Verfahren zur **Delta-Lernregel**:
  Statt batch-Lernen auf allen N Trainingsbeispielen **inkrementelles Lernen** aus Einzelbeispielen: Errechne aus einzelnem Trainingsbeispiel ein Δw_j (Gradientenabstieg)

Was weiter im Ertel zu NN? 2/3

9.5 Backpropagation-Netze, Backprop-Lernregel

Backprop-Lernregel ist Variante der Delta-Regel für Backprop-Netze

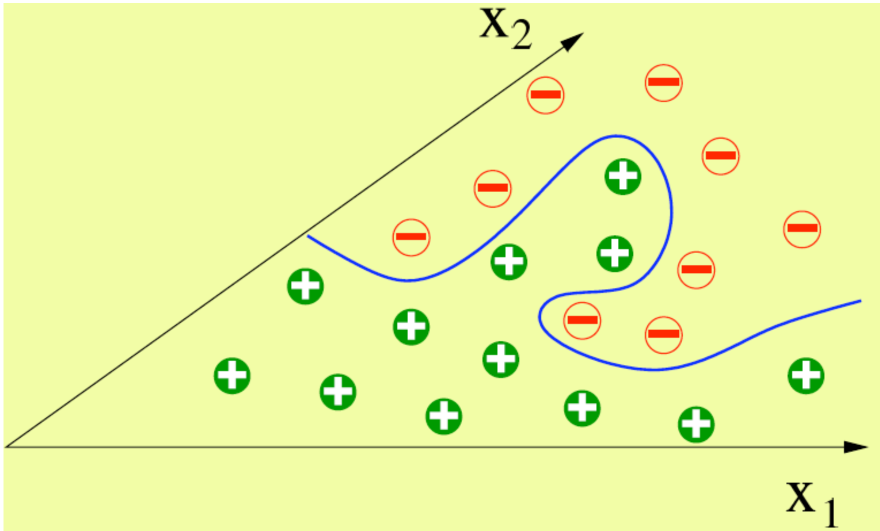


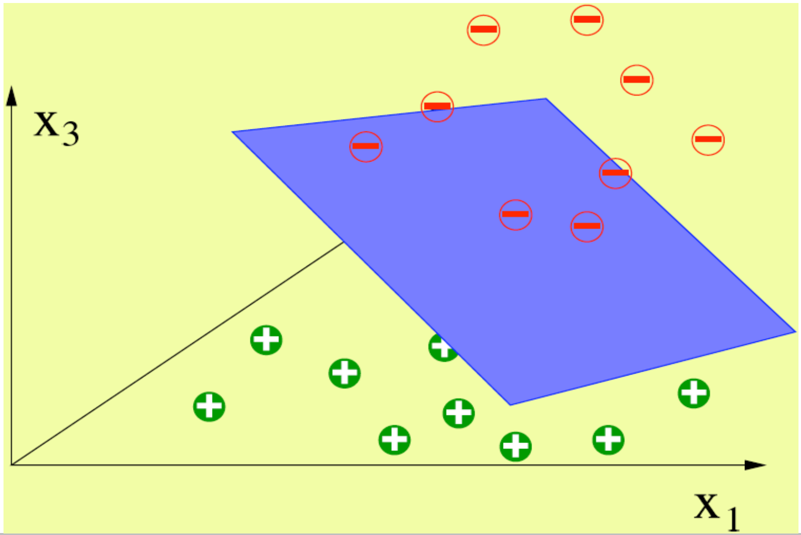
„Zweiwege-Propagieren“ vorwärts & rückwärts ist nötig, um die Differenz festzustellen zwischen Aktivierung auf Basis aktueller Gewichte und gefordertem Output gemäß aktuellem Trainingsvektor

Was weiter im Ertel zu NN? 3/3

9.6 Support-Vektor-Maschinen (SVM) (s. Folie 243-245)

- Für **nicht linear separierbare** Trainingsdaten finde spezielle nichtlineare Transformation (**Kernel**) in höherdimensionalen Parameterraum, in dem sie linear separierbar sind
- Finde dort (optimale) lineare Separierung (s. Perzeptron)
- Nicht wirklich ein NN-Verfahren; die Mathe ist ähnlich





5.4 Reinforcement-Lernen

- **Ziel**: Lerne auf Grund von Reward-Signalen (überwacht? unüberwacht?) aus der Umgebung optimales Handeln in sequenziellen Entscheidungsproblemen unter Unsicherheit (MDPs, POMDPs)
- **Weg**: Lerne durch „Ausprobieren" „Modelle" der Umgebung, und der Wirkungen eigener Handlungen in der Umgebung
- **Beispiel**: Erlernen der Koordination der Gelenksteuerungen einer Laufmaschine