Deep Learning Tutorial

Universität Augsburg
Lehrstuhl für Embedded Intelligence for Health Care and Wellbeing

Univ.-Prof. Dr.-Ing. habil. Björn Schuller

Shahin Amiriparian

Ziel des heutigen Tutorials

Implementierung eines einfachen neuronalen Netzes, das wie folgt funktioniert:

- Eingabe: eine 2 dimensionale Matrix.
- Multiplikation der Eingabe mit festen Gewichten
  - Matrixmultiplikation.
- Aktivierungsfunktion: Sigmoid oder tanh().
- Eine Ausgabe zurückgeben.
- Berechnung des Fehlers, indem die Differenz aus der gewünschten Ausgabe der Daten und der vorhergesagten Ausgabe genommen wird.
- Leichte Anpassung der Gewichte basierend auf dem Fehler.
- Das NN für 3000 Iterationen trainieren.

Einführung in Neuronale Netze (NN)

Motivation: Mensch vs. Maschine

Fähigkeiten des Menschen	Fähigkeiten des Rechners
Schnelle Mustererkennung	Schnelle arithmetische Operationen
Analoges Schließen, Fehlertoleranz	Exakte Berechnungen, fehlerfrei
Assoziation, Interpolation	Genauigkeit, Gleichheit
Ähnlichkeiten kontextabhängig erkennen	Gleichheit kontextunabhängig schnell prüfen
Flexibilität, Leistungsreserven	Konstante Leistung auch bei Dauerbelastung
Parallele Informationsverarbeitung	Sequentielle Informationsverarbeitung (von Neumann)

Einführung in Neuronale Netze (NN)

Motivation: Mensch vs. Maschine

Fähigkeiten des Menschen	Fähigkeiten des Rechners
Einfache Elementaroperationen	Komplexe Elementaroperationen (CISC)
Große Kapazität für ungenaue Muster	Große Speicherkapazitäten möglich
Assoziativer Zugriff auf verteilte Daten	Datenadressierung, lokale Speicherung
Kreativität, Erweitern der Begriffswelt	Schnelles Manipulieren einer festen Begriffswelt
Phantasie, Einführung hypothetischer Welten	
Lernen, Konditionieren	Programmieren

Einführung in Neuronale Netze (NN)

Was ist ein neuronales Netz?

- Simulation der Funktionsweise des **Gehirns**
- Nachbildung des biologischen Nervensystems
  → Ermöglicht **Lernprozess**
- Aufgebaut aus einer großen Anzahl **hoch-vernetzter** Verarbeitungselemente: **Neurone**
- Lernen anhand von **Beispielen**

Einführung in Neuronale Netze (NN)

Vorteile neuronaler Netze

- **Lernfähigkeit** anhand von Beispielen
- **Selbstorganisation**
- Hohe **Parallelität** bei Informations-verarbeitung
- Hohe **Fehlertoleranz**
  → Robustheit im Umgang mit verrauschten Daten

Einführung in Neuronale Netze (NN)

Vorteile neuronaler Netze

- **Verteilte** Wissensrepräsentation
  → Zerstörung eines Neurons führt zu relativ kleinem Wissensausfall
- Generalisierungs- bzw. **Assoziationsfähigkeit**
  → Voraussagen für unbekannte Beispiele

Einführung in Neuronale Netze (NN)

Vorteile neuronaler Netze

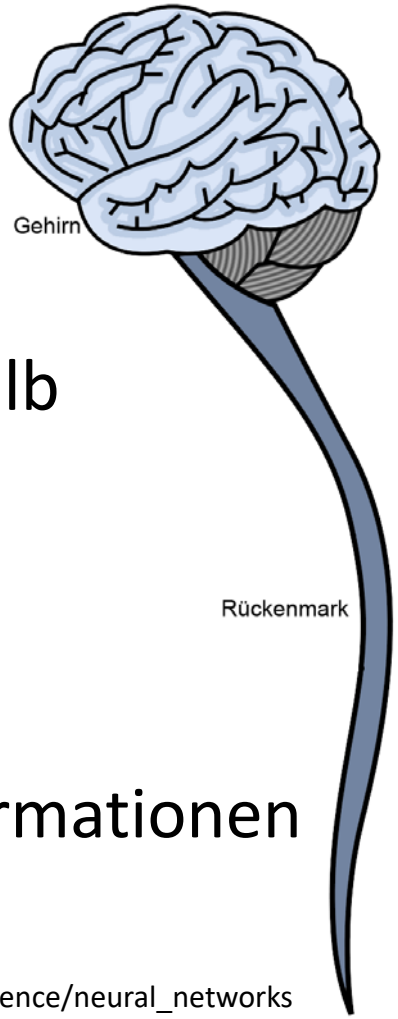
- Selbstorganisation
  → Erzeugen eigener Darstellung erlernter Informationen
- Nichtlinearität
  → Verarbeitung von beliebigen Funktionen
- Ausgaben geben Rückschlüsse auf Sicherheit der Zuordnung

Biologische Neuronale Netze

Biologisches Vorbild

Nervensystem von Wirbeltieren:

- **Peripheres Nervensystem:**
  - Verzweigtes und dichtes Netz außerhalb des Gehirns bzw. Rückenmarks
- **Zentrales Nervensystem:**
  - Gehirn und Rückenmark
  - Speicherung und Verwaltung von Informationen
  - Koordination motorischer Leistungen



Quelle: http://www.dkriesel.com/science/neural_networks

Biologisches Vorbild

Menschliches Gehirn

Informationen sind nicht in den einzelnen Neuronen gespeichert, sondern werden durch den gesamten Zustand des Netzes mit allen Verbindungen und Bindungsstärken repräsentiert.

Biologisches Vorbild

Menschliches Gehirn

	Gehirn	Rechner
Anzahl Recheneinheiten	$\approx 10^{11}$	$\approx 10^{9}$
Art der Recheneinheiten	Neurone	Transistoren
Art der Berechnung	massiv parallel	i.d.R. seriell
Datenspeicherung	assoziativ	adressbasiert
Schaltzeit	$\approx 10^{-3} s$	$\approx 10^{-9} s$
Theoretische Schaltvorgänge	$\approx 10^{13} \frac{1}{s}$	$\approx 10^{18} \frac{1}{s}$
Tatsächliche Schaltvorgänge	$\approx 10^{12} \frac{1}{s}$	$\approx 10^{10} \frac{1}{s}$

Biologisches Vorbild

Neurone

- Informationsverarbeitende Nervenzellen
- Schalter mit Informationseingang und -ausgang
- Etwa 10^{11} Neurone, 20 Typen
  → Kontinuierliche Umstrukturierung
- Jedes Neuron hat ca. $10^3 - 10^4$ Synapsen

Biologisches Vorbild

Aufbau von Nervenzellen

- Synapsen:
  - Kontaktstellen für Übertragung der Erregung
  - Können unterschiedlich starke Impulse auslösen
- Dendriten: Eingänge für elektrische Signale
- Axon: Übertragung von Signalen an andere Neurone mittels baumartig aufgefächerter Synapsen (bis zu $1m$ lang)

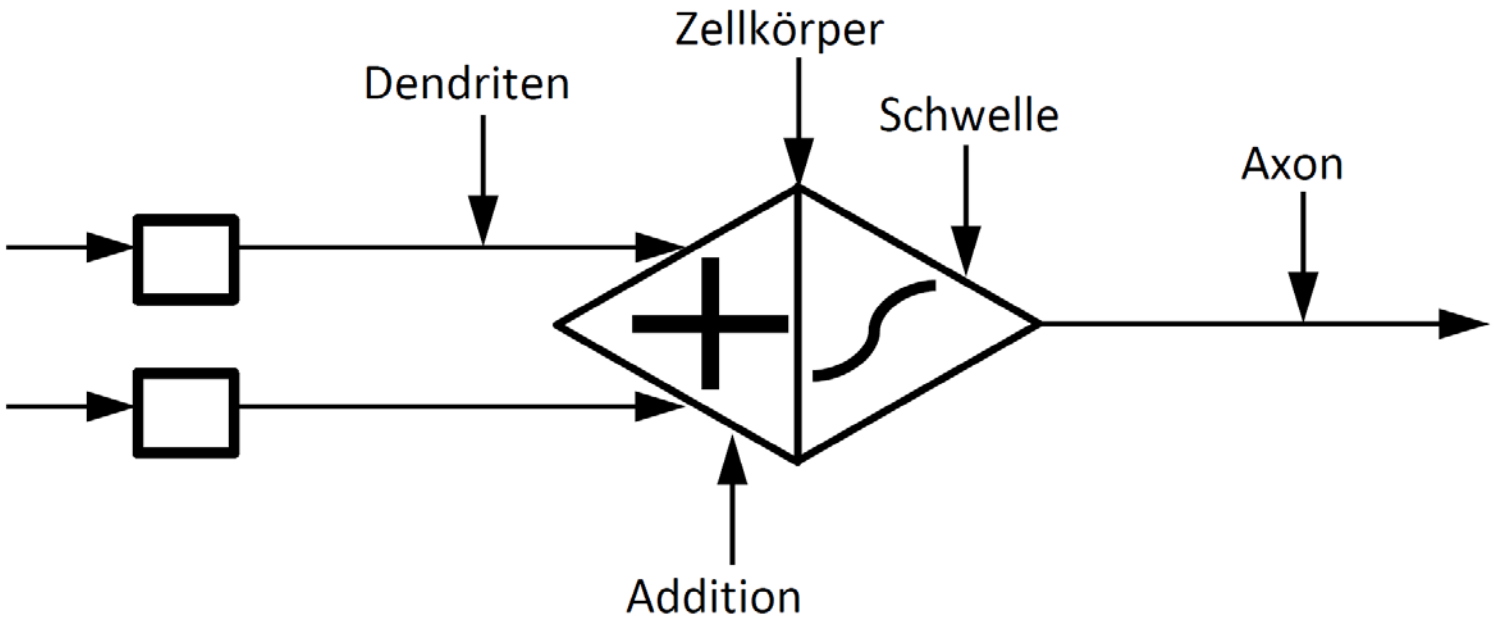
Biologisches Vorbild

Aufbau von Nervenzellen

- Zellkörper:
  - Summiert eingehende Impulse
  - Wird eine bestimmte Reizschwelle überschritten, entsteht im Zellkörper ein Signal
    → Elektrischer Spannungsstoß
  - Ca. $0.25\ mm$ Durchmesser
- Zellkern

Biologisches Vorbild

Neuronenmodell



Biologisches Vorbild

Elektrochemische Vorgänge:

- Neurone weisen gegenüber ihrer Umwelt eine elektrische Ladungsdifferenz auf
- Membranpotential entsteht durch mehrere Arten von Ionen, die innerhalb und außerhalb des Neurons unterschiedlich hoch konzentriert sind (Konzentrationsgradienten)
- Diffusion → Gleichmäßige Ionenverteilung

Biologisches Vorbild

Elektrochemische Vorgänge:

- Neurone erhalten ein elektrisches Membranpotential aktiv aufrecht
- Membran ist für manche Ionen durchlässig, für andere aber nicht
- Elektrischer Gradient wirkt dem Konzentrationsgradienten entgegen
  → stabiler Zustand

Biologisches Vorbild

Elektrochemische Vorgänge:

- Eine „Pumpe“ (das Protein ATP) bewegt aktiv Ionen entgegen der Richtung in die sie sich aufgrund des Konzentrationsgradienten und des elektrischen Potentials eigentlich bewegen würden
  → Natrium-Kalium-Pumpe
- Fließgleichgewicht → Ruhepotential

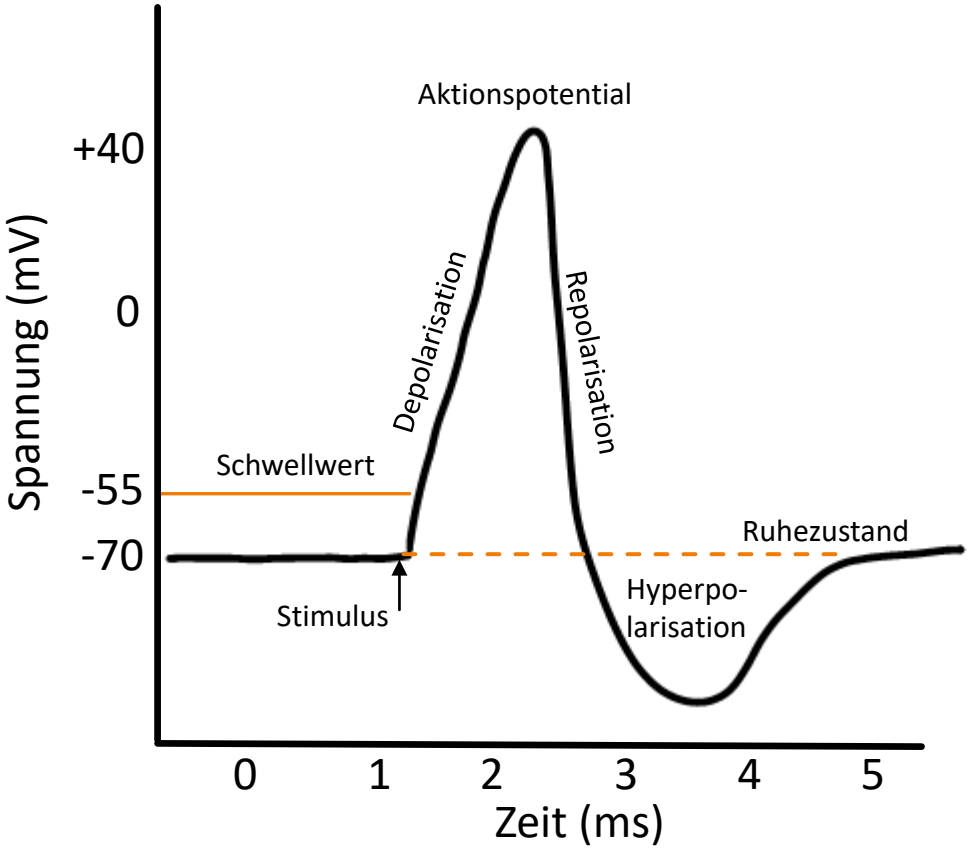
Biologisches Vorbild

Elektrochemische Vorgänge:

- **Steuerbare Kanäle** werden **geöffnet**, wenn eingehende Reize **Schwellwert** überschreiten
- Schwellwertpotential liegt bei ca. -55 mV
- Auslösen eines elektrischen Signals (Aktionspotential)

Biologisches Vorbild

Auslösen eines Aktionspotentials:



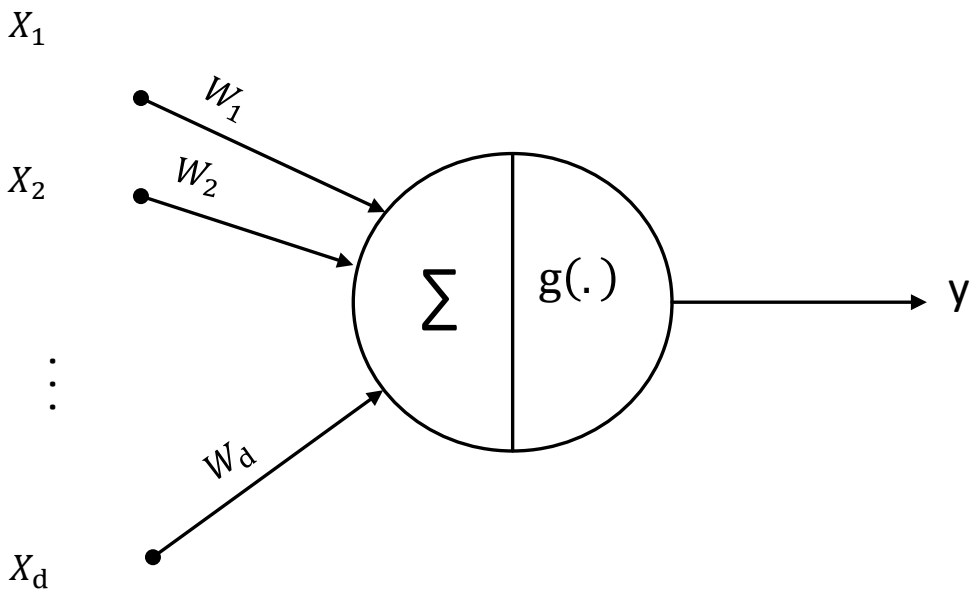
Modellierung von Neuronen

Motivation

- Simulation der Funktionsweise des Gehirns: kognitives *Lernen* statt starres Programm
- Technische Approximation durch extreme Vereinfachung
- Adaptieren besonderer Eigenschaften biologischer Neurone
- Baustein künstlicher neuronaler Netze

Perzeptron

Modell eines Perzeptrons



Perzeptron

Mathematische Definition:

- x_i sind Komponenten des Eingangsvektors $\underline{x} \in \mathbb{R}^d \triangleq$ Dendriten
- $w_i \in \mathbb{R}$ sind Gewichte der Verbindungen, wobei w_i den Anteil von x_i an der Ausgabe quantifiziert $\triangleq$ Synapsen
- $\sum$ berechnet gewichtete Summe $\sum_{i=1}^{d} w_i x_i$ der Eingänge $\triangleq$ Zellkörper

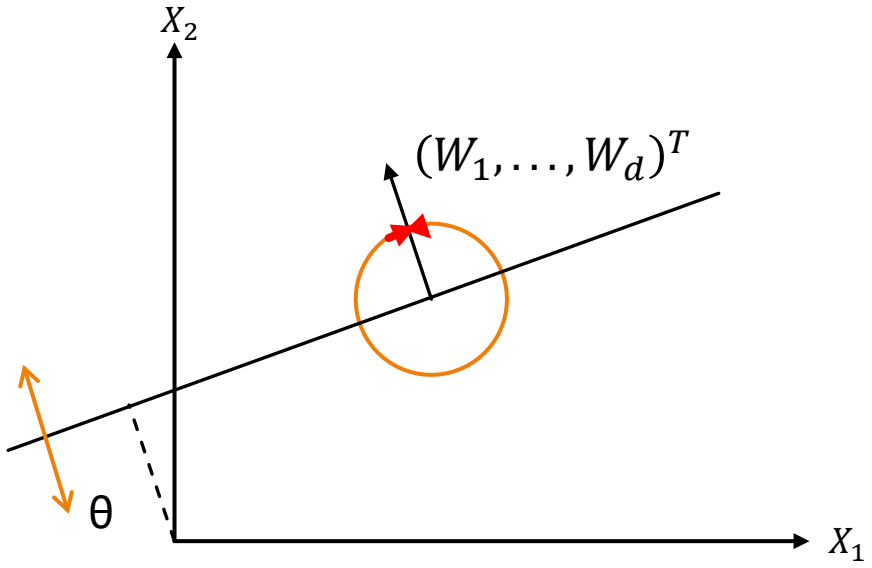
Perzeptron

Mathematische Definition:

- $g(\cdot)$ ist Aktivierungsfunktion, beschränkt
  Ausgabewert ≙ Zellkörper
- y ist Ausgabe ≙ Axon
- Schwellwert: θ

Perzeptron

- Perzeptron definiert eine Hyperebene als Entscheidungsfläche
- Hyperebene steht senkrecht zu $\underline{\mathrm{w}}$
- $\mathrm{w}_0 = \theta$ beschreibt Abstand zum Ursprung

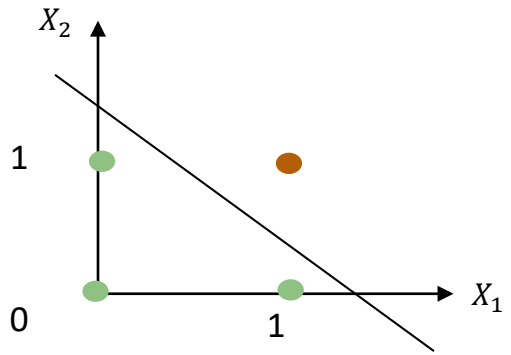


Perzeptron

Beispiele für Entscheidungsflächen

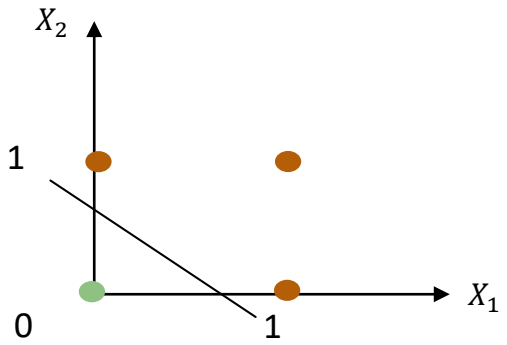
AND:

X_1	X_2	AND
0	0	0
0	1	0
1	0	0
1	1	1



OR:

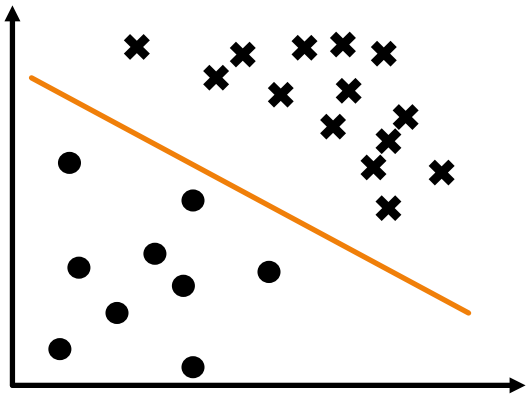
X_1	X_2	OR
0	0	0
0	1	1
1	0	1
1	1	1



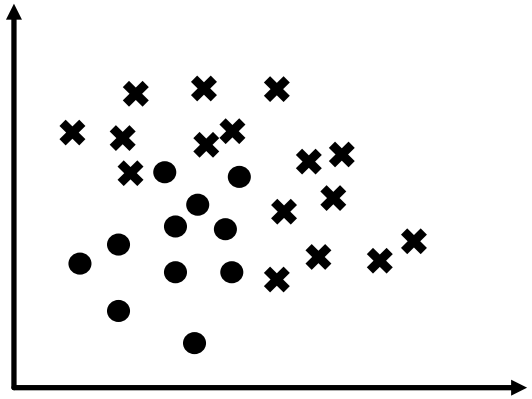
Perzeptron

Lineare Separierbarkeit

Seien $\times$ Vektoren aus M_1 und $\bullet$ Vektoren aus M_2



linear separierbar

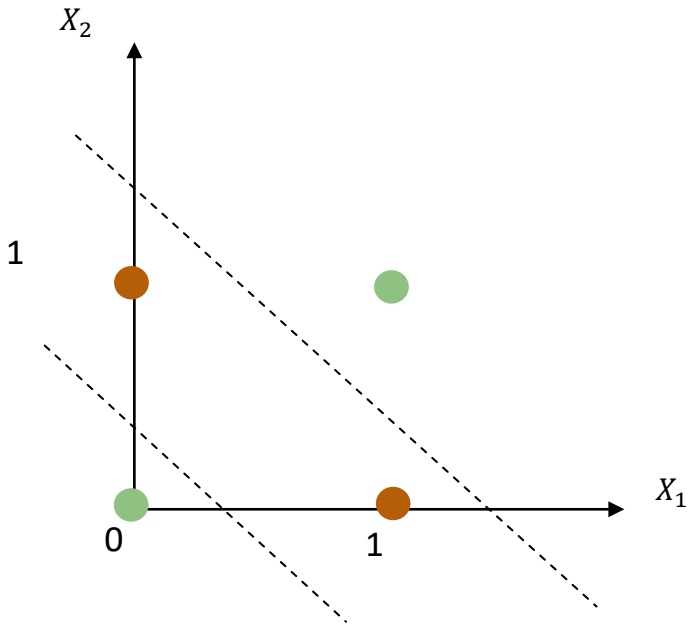


nicht linear separierbar

Perzeptron

XOR-Problem

X_1	X_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



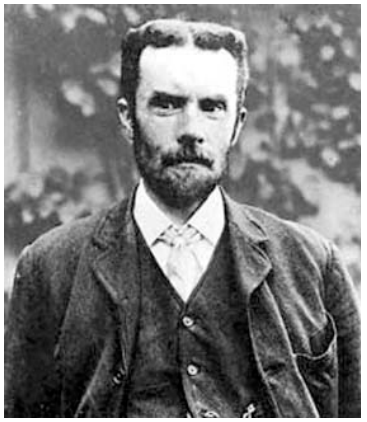
Perzeptron

Perzeptron-Lernalgorithmus:

- Wenn $\sum_{i=1}^{d} w_i x_i \geq \theta$, also $\sum_{i=1}^{d} w_i x_i - \theta \geq 0$ dann soll $y = 1$ gelten, sonst soll $y = 0$ sein
- Erweitern Perzeptron um einen Eingang mit $w_0 = \theta$ und $x_0 = -1$
- Binäre Aktivierungsfunktion: Heaviside-Funktion für Ausgabe

Perzeptron

Heaviside-Funktion:



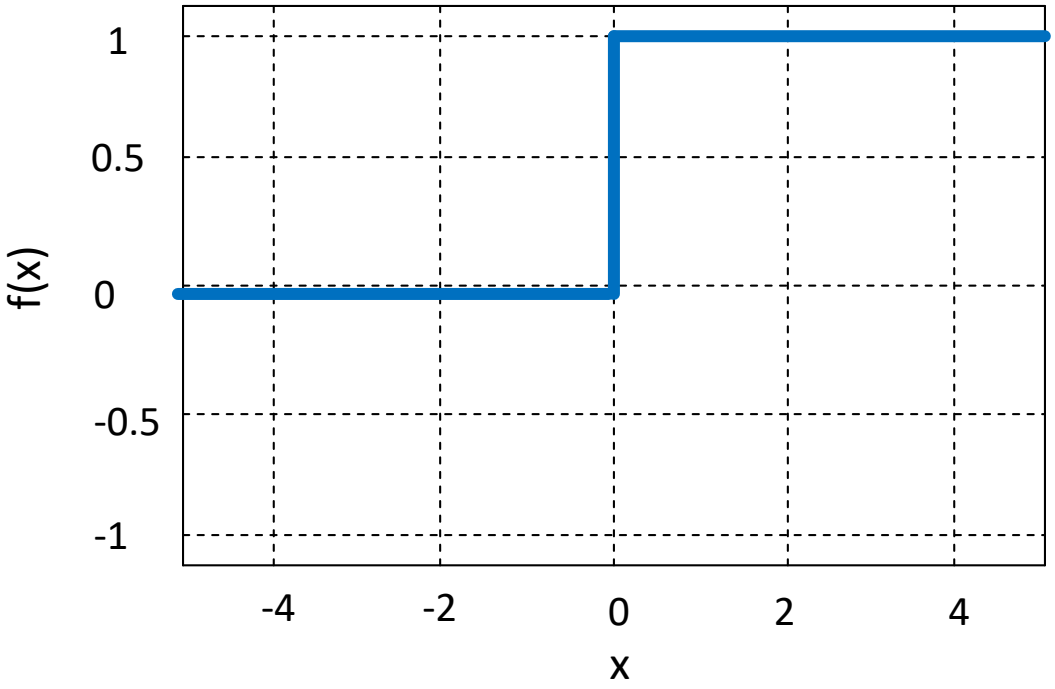
Oliver Heaviside
1850-1925

$$y(\underline{x}) = \text{heaviside}\left(\sum_{i=0}^{d} w_i\, x_i\right) = \begin{cases} 1, \text{wenn} \sum_{i=0}^{d} w_i\, x_i \geq 0 \\ 0, \qquad \text{sonst} \end{cases}$$

Quelle: http://www.oliverheaviside.com/

Perzeptron

Heaviside-Funktion:



Darstellung der binären Schwellwertfunktion

Perzeptron

Perzeptron-Lernalgorithmus

Initialisiere w_i
$t = 0$
wiederhole
$t = t + 1$
wähle zufällig $(x, c(x)) \in \tau$
error = $c(x) - \text{heaviside}(w^T x)$
for $i = 0$ **to** d **do**
$\Delta w_i = \eta * \text{error} * x_i$
$w_i = w_i + \Delta w_i$
solange (Konvergenz ODER $t > t_{max}$)

t … Zeitschritt
$\eta > 0$ … Lernrate
τ … Trainingsmenge

Perzeptron

Lernalgorithmus

- Gewichtsanpassung von η, $\mathrm{x_i}$ und error abhängig
- Lernrate η bestimmt Größe des Lernschritts z.B. $\eta = 0.1$
- $\mathrm{error} = \mathrm{c}(\underline{\mathrm{x}}) - \mathrm{heaviside}(\underline{\mathrm{w}}^T \underline{\mathrm{x}})$

Perzeptron

Lernalgorithmus

- Abweichung der korrekten Klasse von der durch Perzeptron berechneten Klasse
- Neuron feuert fälschlicherweise
  → Alle w_i verringern
- Neuron feuert nicht, obwohl es feuern soll
  → Alle w_i vergrößern

Perzeptron

Konvergenztheorem (Rosenblatt 1962)

Seien M_1, M_2 *zwei nicht-leere, endliche Mengen von Vektoren* $\underline{x} = (-1, x_1, \ldots, x_d)^T$, *wobei* M_1 *und* M_2 *linear separierbar sind.*
Sei $\mathcal{T}$ *die Menge der Beispiele der Form* $(\underline{x}, 0)$ *für* $\underline{x} \in M_1$ *oder* $(\underline{x}, 1)$ *für* $\underline{x} \in M_2$ *und sei* η *genügend klein. Dann gilt …*

Perzeptron

Konvergenztheorem (Rosenblatt 1962)

Dann gilt:

Werden die Vektoren aus $\mathcal{T}$ dem Perzeptron-Lernalgorithmus präsentiert, so konvergiert der Gewichtsvektor $\underline{w}$ des Perzeptrons innerhalb endlich vieler Iterationen so, dass alle Beispiele aus $\mathcal{T}$ korrekt klassifiziert werden.

Perzeptron

Einschränkungen des Perzeptron-Lernalgorithmus:

- Konvergenz bei nicht linear separierbaren Beispielen ist nicht garantiert
- Geeignet nur für *binäre* Aktivierungsfunktion

→ Gradientenabstiegsverfahren

Perzeptron

Lernfehler

- Für Perzeptron ohne Schwellfunktion gilt:

$$y(x) = \sum_{i=0}^{d} w_i x_i$$

- Trainingsfehler von $\underline{w}$ in Abhängigkeit von $\mathcal{T}$:

$$Err(\underline{w}) = \frac{1}{2} \sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} \left(c(\underline{x}) - y(\underline{x})\right)^2$$

Gradientenabstiegsverfahren

Delta-Regel

- Gradientenbasiertes Verfahren zur Fehlerminimierung
- Konvergenz mit genügend kleinem η
- Vorteile:
  → Geeignet für nicht-binäre Aktivierungsfunktionen
  → Schnelleres Lernen bei großer Entfernung zum Lernziel

Gradientenabstiegsverfahren

Gradient des Lernfehlers:

$$\nabla \mathrm{Err}(\underline{\mathrm{w}}) = \left(\frac{\partial \mathrm{Err}}{\partial \mathrm{w}_0}, \ldots, \frac{\partial \mathrm{Err}}{\partial \mathrm{w}_\mathrm{d}} \right)$$

$-\nabla \mathrm{Err}(\underline{\mathrm{w}})$: Richtung des steilsten Abstiegs

Gradientenabstiegsverfahren

Verschieben $\underline{w}$ in Richtung des steilsten Abstiegs

$$\underline{w} = \underline{w} - \eta \cdot \nabla \mathrm{Err}(\underline{w})$$

Komponentenweise:

$$w_i = w_i - \eta \cdot \frac{\partial \mathrm{Err}}{\partial w_i}$$

Gradientenabstiegsverfahren

Herleitung der Gewichtsanpassung:

$$\frac{\partial \mathrm{Err}}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} \left(c(\underline{x}) - y(\underline{x})\right)^2$$

$$= \frac{1}{2}\sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} \frac{\partial}{\partial w_i}\left(c(\underline{x}) - y(\underline{x})\right)^2$$

$$= \frac{1}{2}\sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} 2\left(c(\underline{x}) - y(\underline{x})\right)\frac{\partial}{\partial w_i}\left(c(\underline{x}) - y(\underline{x})\right)$$

$$= \sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} \left(c(\underline{x}) - y(\underline{x})\right)\frac{\partial}{\partial w_i}\left(c(\underline{x}) - \underline{w}^T \cdot \underline{x}\right)$$

Gradientenabstiegsverfahren

Ergebnis der Herleitung:

$$\frac{\partial \mathrm{Err}}{\partial w_i} = \sum_{(\underline{x},\, c(\underline{x}))\in\mathcal{T}} \Big(c(\underline{x}) - y(\underline{x})\Big)(-x_i)$$

Gradientenabstiegsverfahren

Mengenbasiertes Verfahren:

- Fehler wird erst über alle Trainingsbeispiele aufsummiert
- w wird *einmal* am Ende der Lerniteration angepasst
- Längere Berechnungszeit pro Gewichtsanpassung!

Gradientenabstiegsverfahren

Inkrementeller Gradientenabstieg:

- Anpassung w_i in jedem Schritt um Δw_i (Stochastischer Gradientenabstieg)
- Vorhandensein mehrerer lokaler Minima
  → Kann ein Hängenbleiben eher verhindern als das mengenbasierte Verfahren

Gradientenabstiegsverfahren

Vergleich der Lernalgorithmen:

Perzeptron-Lernalgorithmus	Gradientenabstiegsverfahren
Gewichtsanpassung hängt vom Fehler in der Ausgabe und Aktivierungsfunktion ab	Fehler hängt *direkt* von der Abweichung der Linearkombination zur Ausgabe ab
Konvergiert nach endlich vielen Schritten gegen eine perfekte Hyperebene, wenn Trainingsbeispiele linear separierbar sind	Konvergiert asymptotisch gegen eine Hyperebene, die den Lernfehler minimiert
	Keine Garantie der Konvergenz in endlich vielen Schritten, aber Trainingsbeispiele *müssen nicht* linear separierbar sein

Zusammenfassung

Modellierung von Neuronen

- McCulloch-Pitts-Zelle (MCP)
- Perzeptron als einfaches Neuronenmodell
- Konvergenztheorem (Rosenblatt, 1959): Perzeptron-Lernalgorithmus konvergiert in endlicher Zeit für linear separierbare Beispiele
- Gradientenabstiegsverfahren

Modellierung Neuronaler Netze

Motivation

- Einzelne Perzeptronen können nur lineare Entscheidungsflächen lernen
  → Kombination mehrerer Perzeptronen

- Wir benötigen **nichtlineare** Neuronen, da ein Netz aus linearen Neuronen nur lineare Funktionen modellieren kann

Künstliche Neuronale Netze (KNN)

Zweiteilung der Forschung:

1. KNN Modelle zum **besseren Verständnis des menschlichen Verhaltens und der Wahrnehmung** sowie der Funktionsweise des menschlichen Gehirns
2. KNN zur **Lösung konkreter Probleme** aus verschiedenen Bereichen, wie z.B. Wirtschafts-wissenschaften, Statistik, Technik

Künstliche Neuronale Netze (KNN)

Allgemeine Beschreibung:

- Gerichteter, bewerteter Graph
- Neuronenverbände sind in (hierarchischen) Schichten angeordnet
- Verknüpfungen zwischen einzelnen Neurnonenschichten
- Codierung von Informationen wichtig

Künstliche Neuronale Netze (KNN)

Regelbasierte Systeme vs. KNNs

Regelbasierte Systeme	KNNs
Algorithmischer Ansatz, festgelegte Instruktionen	Verkettung nichtlinearer reellwertiger Funktionen, lernen durch ausgewählte Beispiele
↪ Probleme lösen durch Expertenwissen	↪ Probleme lösen durch numerische Optimierung
Problemspezifische Programmierung	Programmierung auf struktureller Ebene

⇒ Wir benutzen fortan den Begriff NN

Grundlagen NN

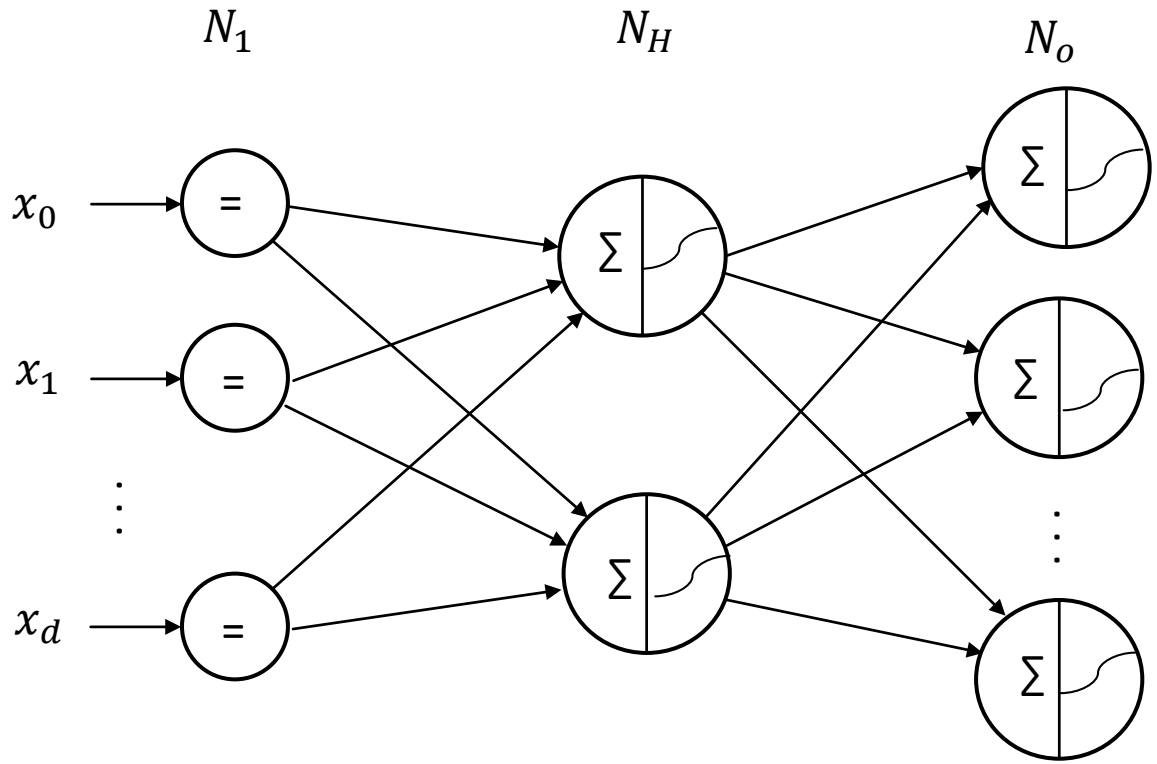
Grundbaustein:
Neurone ≙ Units, Einheiten, Knoten

Unterscheidung:

- Input-Neurone: empfangen externe Signale
- Hidden-Neurone: Zwischenschicht(en)
  → Interne Repräsentation der Außenwelt
- Output-Neurone: geben Signale nach Außen weiter

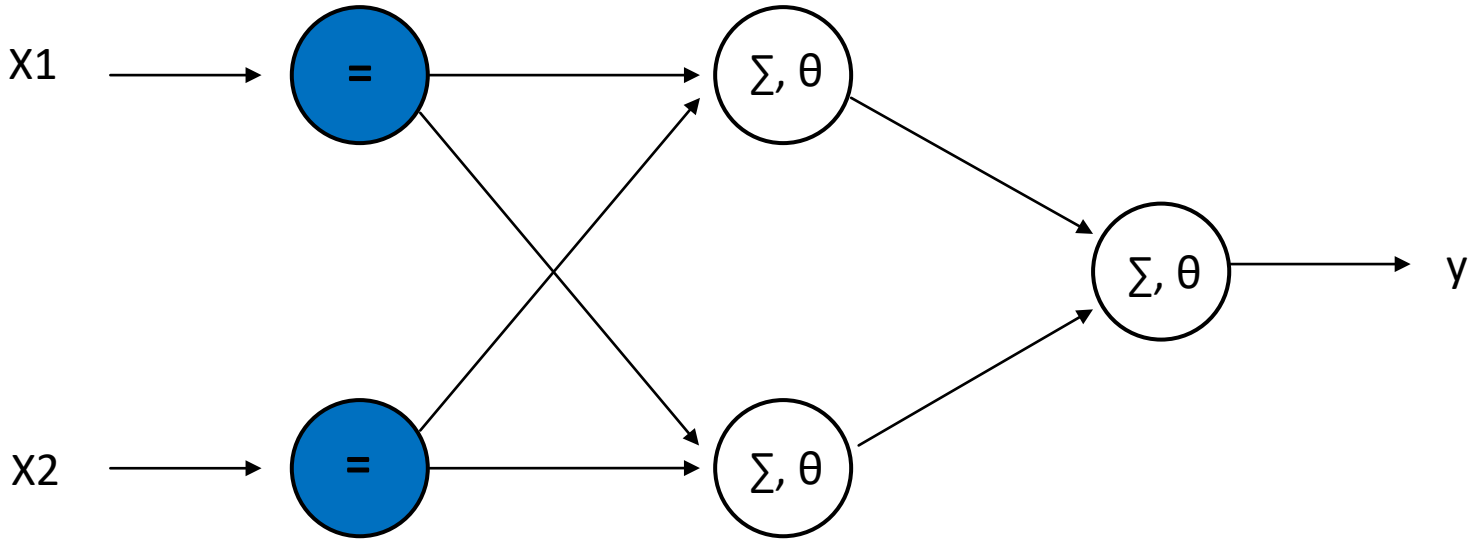
Grundlagen NN

Verschiedene Ebenen in NN:



Grundlagen NN

Beispiel XOR: $x_1 \text{ xor } x_2 \Leftrightarrow (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$



- Hidden-Neurone: erkennen jeweils ein AND
- Output-Neuron: erkennt das OR

Grundlagen NN

Festlegung der Größe des NN

- Dimension des Merkmalsraumes bestimmt Anzahl der Input-Neurone
- Wahl der Ausgabe bestimmt Anzahl der Output-Neurone
  → Bei Klassifizierung meist ein Output-Neuron pro Klasse

Grundlagen NN

Prinzip *trial and error*

- Beginnend mit großem Netz
  → Neurone und Verbindungen entfernen bis Leistung absinkt
- Beginnend mit kleinem Netz
  → Neurone und Verbindungen hinzufügen bis Leistung ausreicht

Grundlagen NN

Annahme:

Einheiten können einen empfangenen Input anders weiterleiten als ihr eigener Zustand ausdrückt

Beispiel:
Ist ein sendendes Neuron im Zustand 0.5, dann kann das Signal, das es aussendet, z.B. die Stärke 0.7 haben

Grundlagen NN

Mathematische Definition NN:

Ein neuronales Netz ist ein sortiertes Tripel (N, V, w) mit zwei Mengen N, V sowie einer Funktion w, wobei N die Menge der Neurone bezeichnet und V eine Menge $\{(i,j)|i,j \in \mathbb{N}\}$ ist, deren Elemente Verbindungen von Neuron i zu Neuron j heißen. Die Funktion $w\colon V \to \mathbb{R}$ definiert die Gewichte, wobei $w\big((i,j)\big)$, das Gewicht der Verbindung von Neuron i zu Neuron j, kurz mit $w_{i,j}$ bezeichnet wird. Sie ist je nach Auffassung entweder undefiniert oder 0 für Verbindungen, welche in dem Netz nicht existieren.

Grundlagen NN

Notationen:

$\underline{x}$: Eingabevektor und

$\underline{y}$: Ausgabevektor eines neuronalen Netzes

c: Korrekte Ausgabe

Ω: Ausgabeneurone

$\mathcal{O}$: Menge der Ausgabeneurone

$\mathcal{I}$: Menge der Eingabeneurone

Grundlagen NN

Notationen:

i: Eingabe sowie

o: Ausgabe eines Neurons

a: Aktivierungszustand

$p \in \mathcal{T}$: Trainingsbeispiel

E_p: Fehlervektor aus Differenz $(c - \underline{y})$

Grundlagen NN

Notationen:

o_i: Ausgabewerte der Neurone i mit $i \in \mathcal{I} = \{i_1, i_2, \dots, i_n\}$, von denen eine Verbindung zu j existiert

a_j: Aktivierungszustand des empfangenden Neurons j

$w_{i,j}$: „Gewichte“ der Verbindungen zwischen Neuronen i und j ≙ Stärke der Interaktion

$\mathcal{W}$: Gewichtsmatrix

Grundlagen NN

Propagierungsfunktion (Inputfunktion) f_{prop}:

- Verwandelt vektorielle Eingaben zur skalaren Netzeingabe

$$net_j = f_{prop}\left(o_{i_1}, \ldots, o_{i_n}, w_{i_1,j}, \ldots, w_{i_n,j}\right)$$

- Lineare Propagierungsfunktion:

$$net_j = \sum_{i \in I} (o_i \cdot w_{i,j})$$

Grundlagen NN

Aktivierungsfunktion (Transferfunktion) f_{act}:

- Berechnet abhängig von Schwellwert und Netzeingabe, wie stark ein Neuron aktiviert ist
- Global für alle oder zumindest eine Menge von Neuronen
- Schwellwerte unterscheiden sich von Neuron zu Neuron und ändern sich durch Lernvorgang

Grundlagen NN

Allgemeine Funktion:

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \theta_j)$$

→ Zustand eines Neurons zu einem neuen Zeitpunkt t ergibt sich aus:

a) Nettoinput zum Zeitpunkt t
b) Aktivierungszustand zum Zeitpunkt $t-1$
c) Schwellwert θ_j

Grundlagen NN

Beispiele für Aktivierungsfunktionen:

- Lineare Aktivierungsfunktion
- Binäre Schwellwertfunktion
- Sigmoidfunktion:
  - Logistische Funktion (Fermifunktion)
  - *Tangens Hyperbolicus*

Grundlagen NN

Lineare Aktivierungsfunktion:

$$a_j = \sum o_i \cdot w_{i,j}$$

Aktivierungsfunktion ≙ andere Darstellung der Propagierungsfunktion / des Nettoinputs

$$a_j = net_j$$

Grundlagen NN

Binäre Schwellwertfunktion:

- Heaviside-Funktion: logischer Schalter
  → Zu niedriger Netzinput (z.B. Rauschen) wird nicht als Signal weitergeleitet
- Schwellwert: θ_j

$$a_j = \begin{cases} net_j & \text{wenn } net_j > \theta_j \\ 0 & \text{sonst} \end{cases}$$

Grundlagen NN

Biasneuron (On-neuron):

- Zugriff auf Aktivierungsfunktion zwecks Training des Schwellwertes θ_j kompliziert
- Einrechnung θ_j wird von Aktivierungsfunktion in Propagierungsfunktion verschoben, indem θ_j von der Netzeingabe subtrahiert wird
  → θ_j als Gewicht einer Verbindung von einem immer 1 ausgebenden Neuron

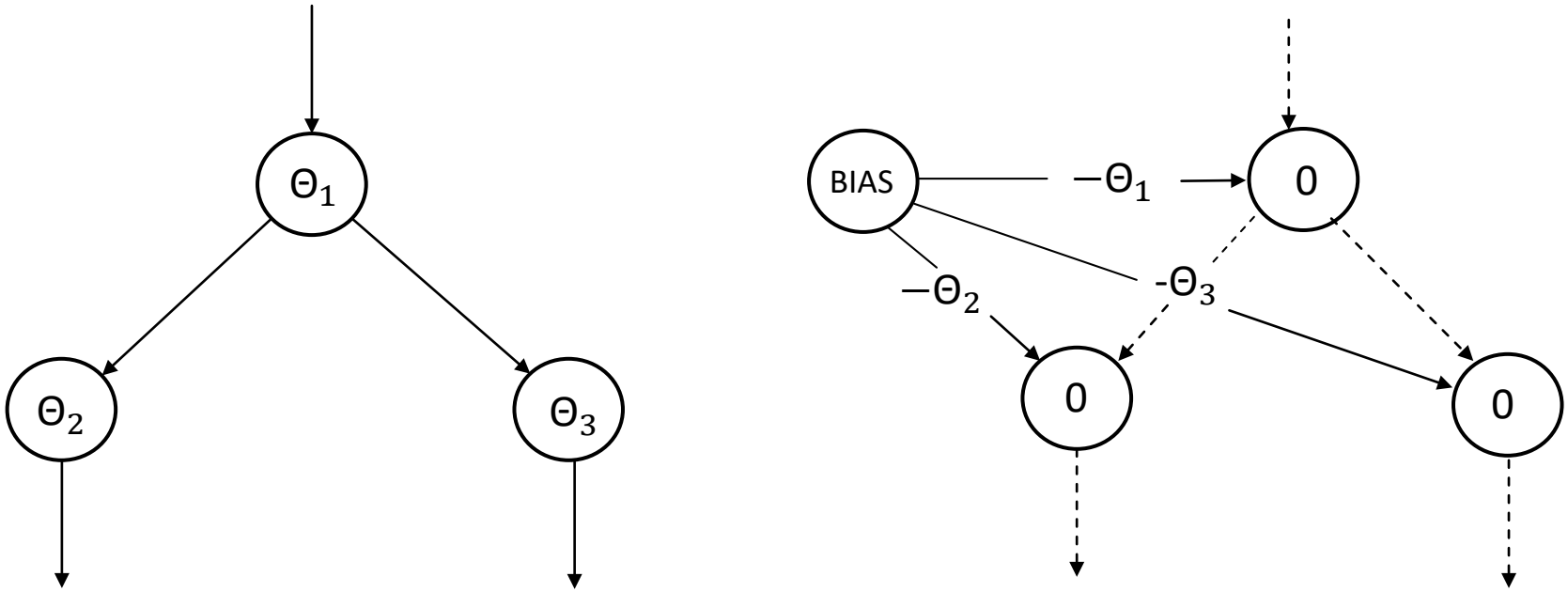
Grundlagen NN

Biasneuron:

- Direktes Trainieren mit Verbindungsgewichten → weniger Lernaufwand
- Einfache Implementierung des Netzes
- Nachteil: unübersichtlich bei großer Anzahl von Neuronen

Grundlagen NN

Beispiel Biasneuron:



Zwei äquivalente Neuronale Netze, links eins ohne, rechts eins mit Biasneuron.

Grundlagen NN

Sigmoidfunktion:

→ Kontinuierlich und nichtlinear

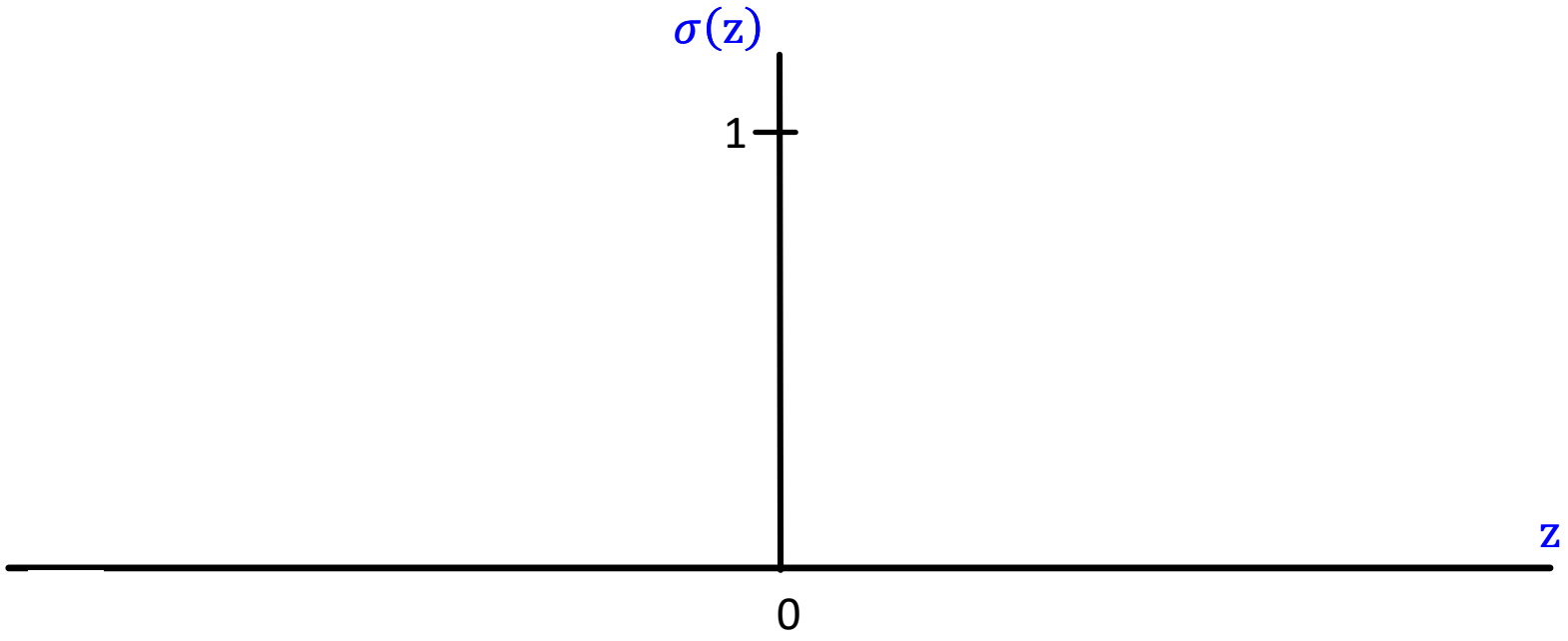
→ Simulation von *kognitiven Prozessen*

Logistische Funktion:

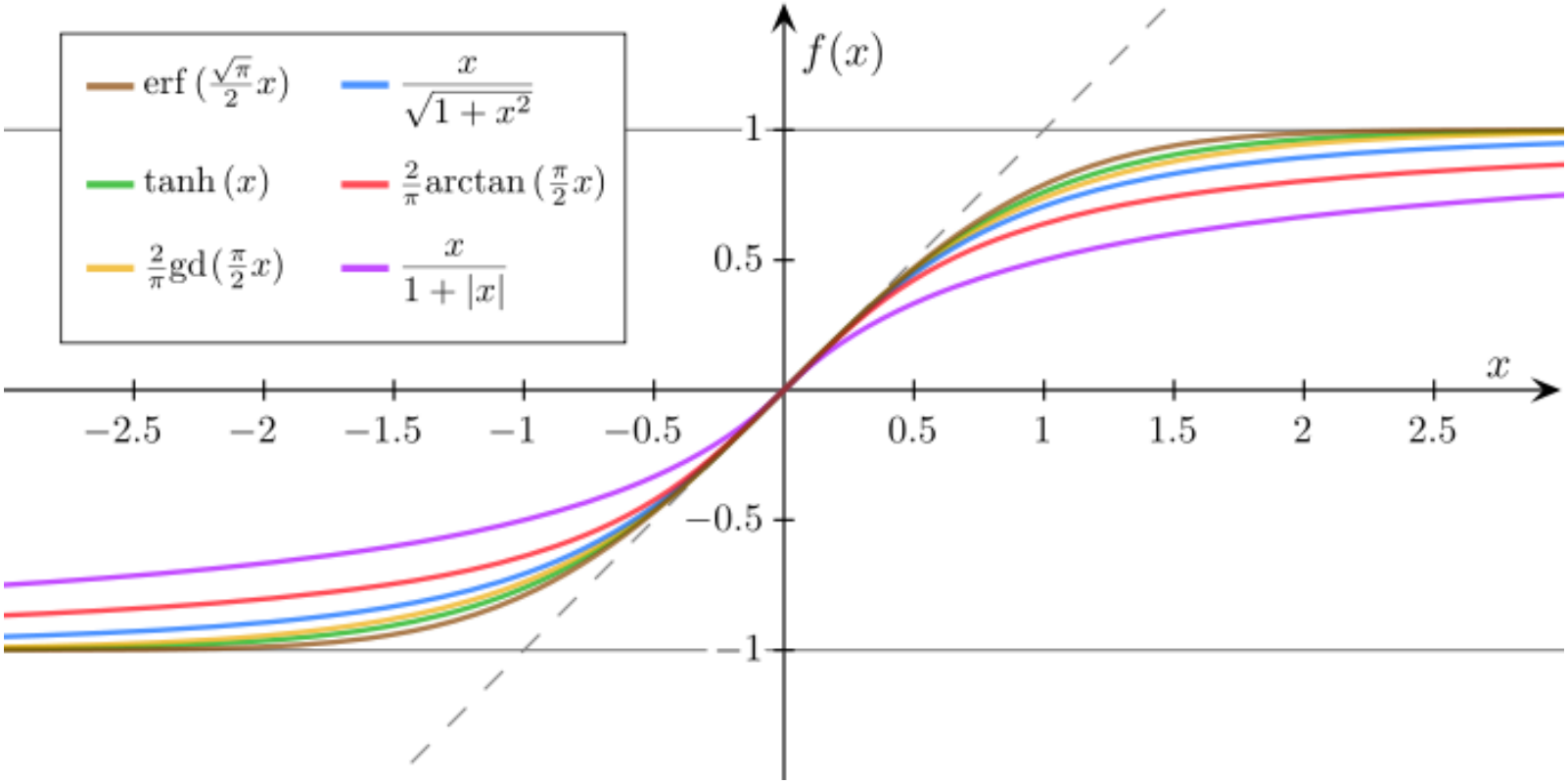
$$\sigma(\mathrm{z}) = \frac{1}{1+e^{-z}} \text{ mit } \frac{d\sigma(\mathrm{z})}{dz} = \sigma(\mathrm{z}) \cdot (1 - \sigma(\mathrm{z}))$$

Grundlagen NN

Logistische Funktion:



Grundlagen NN



Vergleich einiger geeigneter Funktionen. Hier sind sie so normiert, dass ihre Grenzwerte -1 bzw. 1 sind und die Steigungen in 0 gleich 1 sind

Quelle: Georg-Johann

Grundlagen NN

Sigmoidfunktion – Vorteile:

- Begrenzung des Aktivitätslevels
  → Höhere biologische Plausibilität (vgl. begrenzte Intensität des Aktionspotentials biologischer Neurone)
  → Keine Fehlerwerte durch unerwünschte Aktivität im Netz (bedingt durch rekurrente Verbindungen)
- Differenzierbarkeit: Voraussetzung für Rückpropagierung (mehr dazu später)

Grundlagen NN

Ausgabefunktion (Output-funktion) f_{out}

- f_{out} kann genutzt werden um Aktivierung nochmals zu verarbeiten

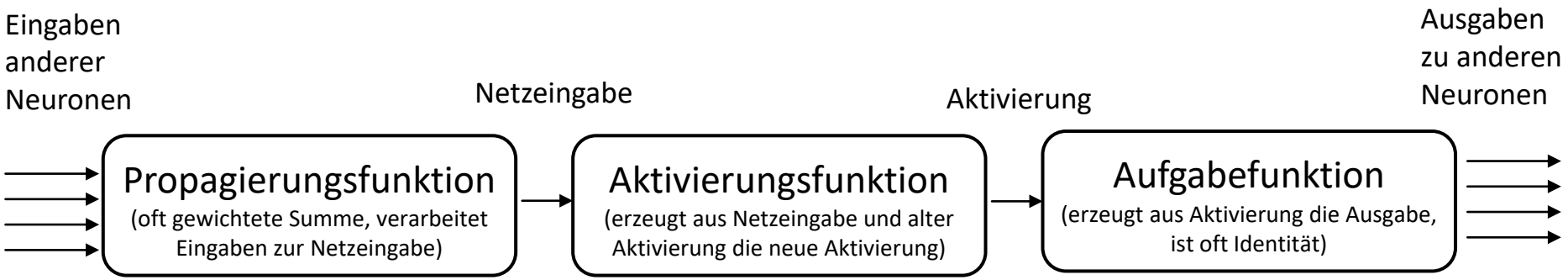
$$f_{out}(a_j) = o_j$$

- Der Einfachheit halber wird die Aktivierungsfunktion gleich der **Identitätsfunktion** gesetzt:

$$o_j = a_j$$

Grundlagen NN

Datenverarbeitung eines Neurons



Eigenschaften NN

Parallelverarbeitung

Berechnungen werden gleichzeitig durchgeführt, zur Zeit nur begrenzt am PC möglich

Verteilte Speicherung

Wissen wird abgespeichert in vielen verschiedenen Gewichten (im Gegensatz zu lokaler Speicherung wie z.B. bei einer CD, Festplatte)

Eigenschaften NN

Reihenfolge der Neuronenaktivierungen wichtig!

Synchrone Aktivierung

- Simultane Berechnung von Netzeingaben, Aktivierung und Ausgabe
- Dem biologischen Vorbild am ähnlichsten
- Hardware-Implementierung nur auf bestimmten Parallelrechnern sinnvoll und speziell für Feed-Forward-Netze nicht sinnvoll

Eigenschaften NN

Asynchrone Aktivierung

→Neurone ändern ihre Werte zu verschiedenen Zeitpunkten

Ordnungen:

- Zufällige Ordnung
- Zufällige Permutation
- Topologische Ordnung
- Feste Ordnung

Eigenschaften NN

Asynchrone Aktivierung - Zufällige Permutation:

- Pro Zyklus wird jedes Neuron genau einmal berücksichtigt
- Zufällige Aktivierungsreihenfolge (im Allgemeinen nicht sinnvoll)
- Sehr zeit- bzw. rechenaufwändig (bei jedem Zyklus eine neue Permutation)

Eigenschaften NN

Asynchrone Aktivierung - Zufällige Ordnung:

Ein Neuron i wird zufällig gewählt und dessen net_i, a_i, und o_i aktualisiert. Bei n Neuronen ist ein Zyklus die n-malige Durchführung dieses Schrittes

→ Nicht immer sinnvoll, weil manche Neuronen pro Zyklus mehrfach aktualisiert werden, andere hingegen gar nicht

Eigenschaften NN

Asynchrone Aktivierung - Topologische Ordnung:

- Neuronen werden pro Zyklus in fester Ordnung aktualisiert
- Nur für rückkopplungsfreie Netze, da sonst keine Aktivierungsreihenfolge
- Zeitsparend: für ein Feed-Forward-Netz mit drei Schichten reicht ein Propagierungszyklus

Eigenschaften NN

Asynchrone Aktivierung - Feste Ordnung:

- Feste Aktivierungsreihenfolge für gesamte Laufzeit
- Beliebte Methode bei Implementierung von z.B. Feed-Forward-Netzen
- Nicht immer sinnvoll bei Netzen die ihre Topologie verändern

Eigenschaften NN

Biologische Plausibilität durch:

- Generalisierung / Diskrimination von Reizen
- Toleranz gegenüber
  - internen Schäden → Richtiger Output trotz z.B. Absterbens einzelner Neurone oder Verbindungen
  - externen Fehlern → Mustererkennung auch bei unvollständigem oder fehlerhaftem Input
- Kategorienbildung: Output zentraler Tendenz (Prototyp einer Kategorie)

Eigenschaften NN

- Abruf von Inhalten (engl.: *content addressability*): Abrufen von Informationen, indem dem Netz Inhalte präsentiert werden, die mit dem gesuchten Inhalt in Verbindung stehen
- Hohe Lernfähigkeit:
  Lernen beliebiger Abbildungen (nichtlineare Zusammenhänge oder Interaktionseffekte zwischen mehreren Variablen)
- Große Anzahl an Freiheitsgraden / Parametern

Eigenschaften NN

Probleme

Große Anzahl an Freiheitsgraden:

→ Simulation grundsätzlich jeder menschlichen Verhaltensweise

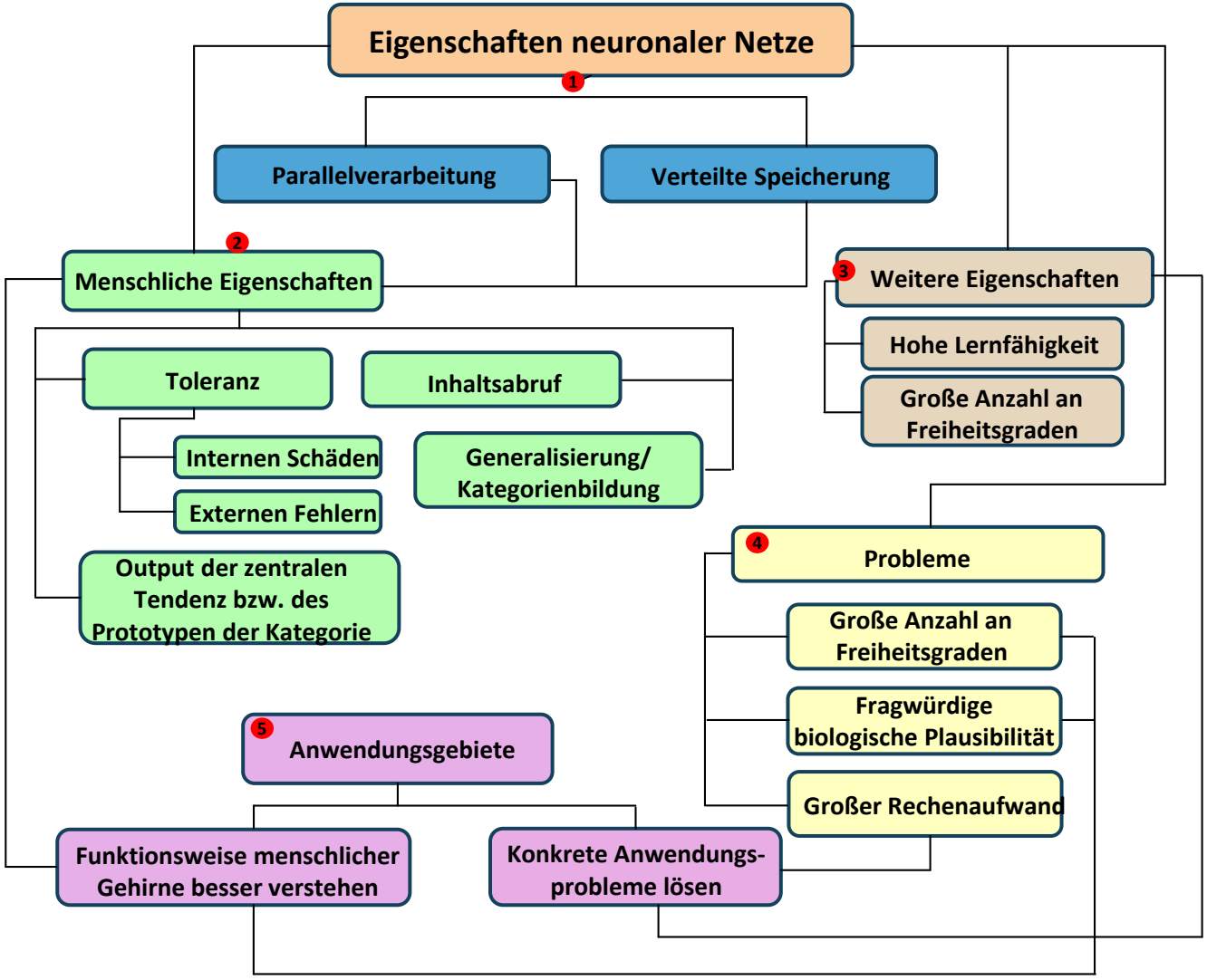
→ Immunisierungsstrategie: NN als Konzept zur Erklärung menschlichen Verhaltens nicht falsifizierbar (Falsifizierbarkeit nach Karl Popper, 1996)

Eigenschaften NN

Probleme

- Widerspruch zu biologischen Grundannahmen, z.B. Rückpropagierung
- Großer Rechenaufwand
  (betrifft nur neuronale Netze, die dazu dienen konkrete Anwendungsprobleme zu lösen)

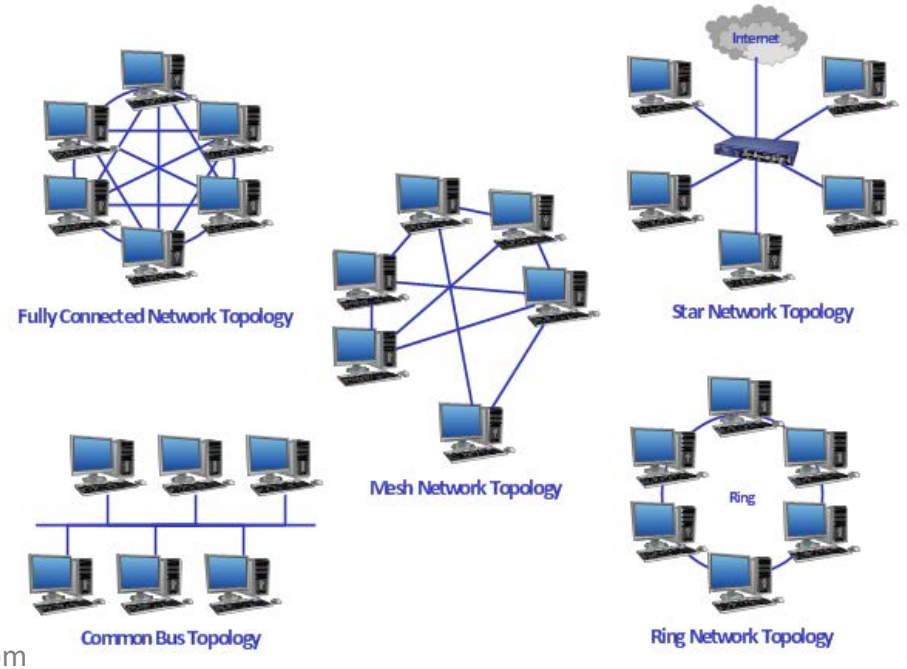
Eigenschaften – Zusammenfassung



Einleitung
Netztopologie

Topologie:

Struktur der Verbindungen zwischen Neuronen (vgl. Topologie von Rechnernetzen)



Quelle: conceptdraw.com

Netztopologie

Einschichtige Netze:

Eingabeschicht ➡ Variable Gewichtsschicht ➡ Ausgabeschicht

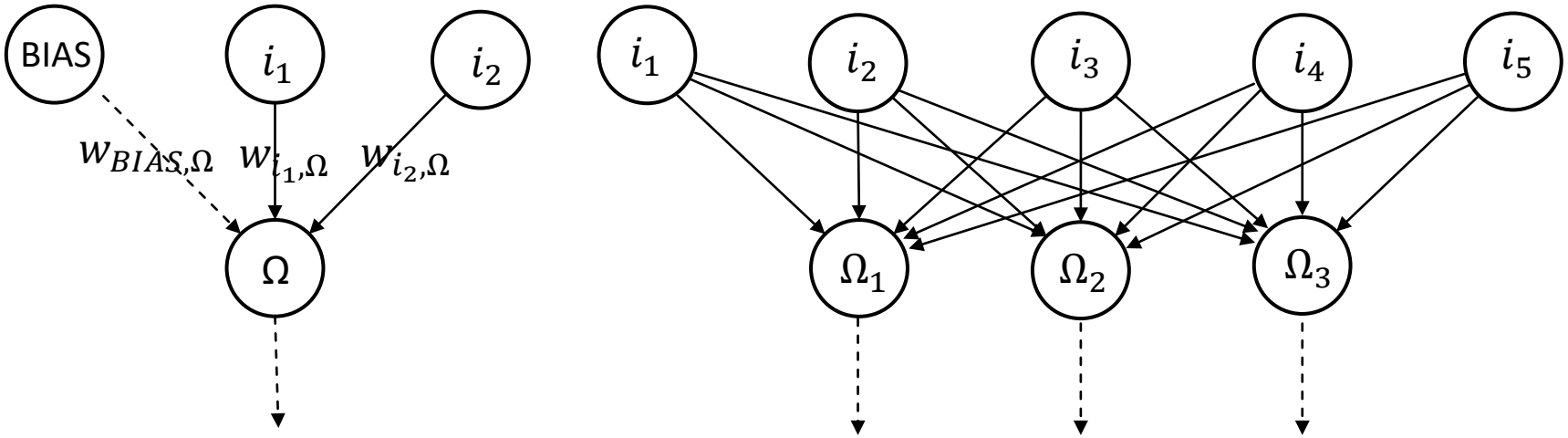
Mehrschichtige Netze:

Mehr als eine Zwischenebene (hidden layers) zwischen Eingabe- und Ausgabeschicht

Definition: ein n-stufiges NN hat n variable Gewichtsschichten und n+1 Schichten Neurone

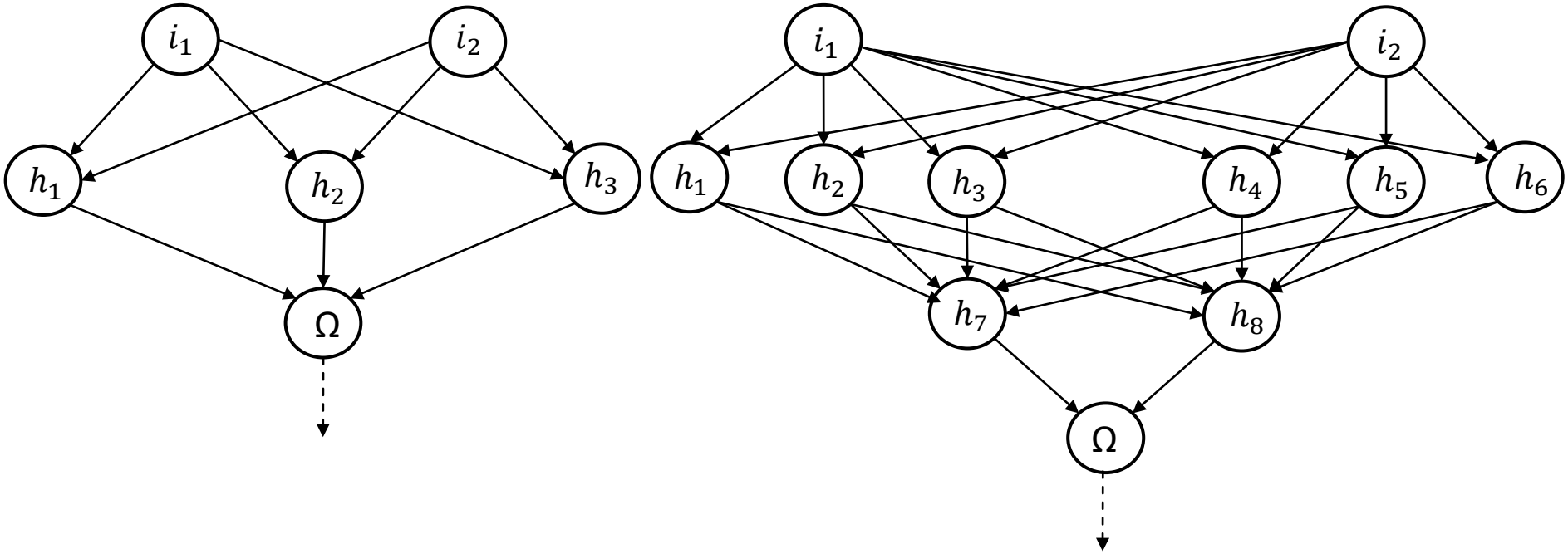
Netztopologie

Beispiel Single-layer perceptron (SLP):



Netztopologie

Beispiel Multi-layer perceptron (MLP):



Netztopologie

Leistungsfähigkeit NN:

- Single-layer perceptron kann Boolesche Funktionen AND, OR, NAND, NOR darstellen
- Zweistufiges Netz: Zusammensetzung mehrerer Geraden zu konvexen Polygonen
  → Approximation von:
  - S*tetigen* Funktionen mit beliebig kleinem Fehler
  - Funktionen mit endlich vielen Unstetigkeitsstellen

Netztopologie

Leistungsfähigkeit NN:

- Dreistufiges Netz: Modellierung beliebiger Mengen mit mehreren Polygonen
- Anzahl der Hidden-Neurone kann exponentiell zur Größe der Eingabe steigen
  → Funktionsabhängig

Netztopologie

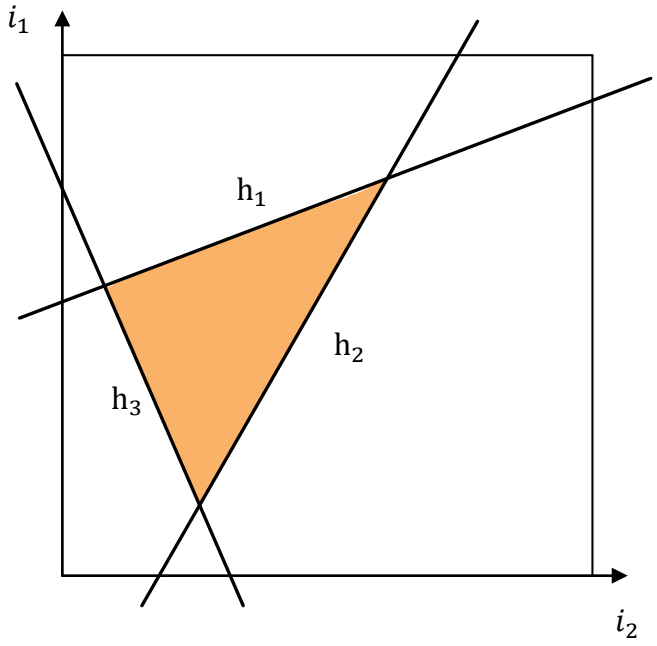
Leistungsfähigkeit NN:

n	Klassifizierbare Menge
1	Hyperebene
2	Konvexes Polygon
3	jede beliebige Menge
4	auch jede beliebige Menge, also zunächst kein weiter Vorteil (siehe Deep Learning)

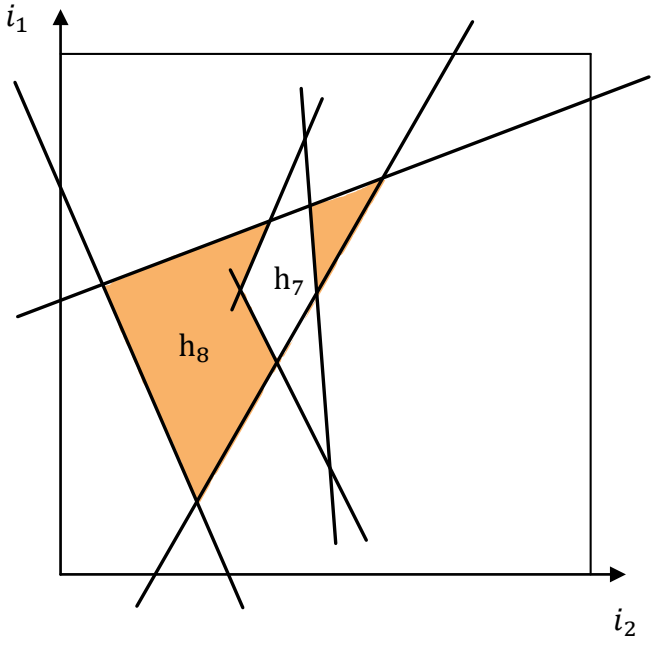
Hier wird dargestellt, mit welchem Perzeptron sich Mengen welcher Art klassifizieren lassen, wobei das *n* die Anzahl der trainierbaren Gewichtsschichten darstellt.

Netztopologie

Leistungsfähigkeit NN:



2-stufiges Netz



3-stufiges Netz

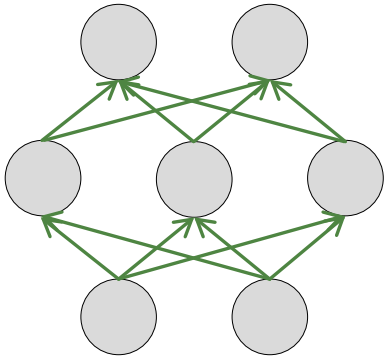
Netztopologien

Vorwärtsgerichtete Netze/ Feed-Forward-Netze:

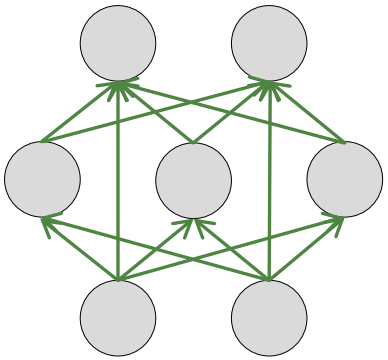
- Bestehend aus Schichten und Verbindungen zur jeweils nächsten Schicht
- Keine Rückkopplungen
- Netztypen:
  - Pattern Associator
  - Kompetitive Netze
  - Kohonennetze bzw. Selforganizing Maps (SOMs)

Netztopologien

Feed-Forward-Netze – Varianten:



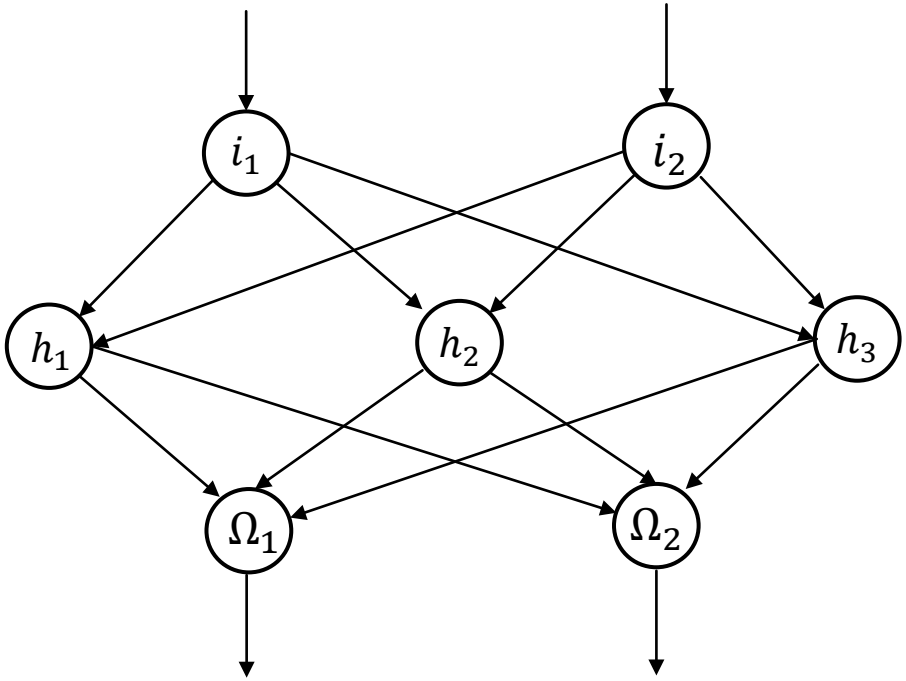
Ebenenweise verbundenes Netz

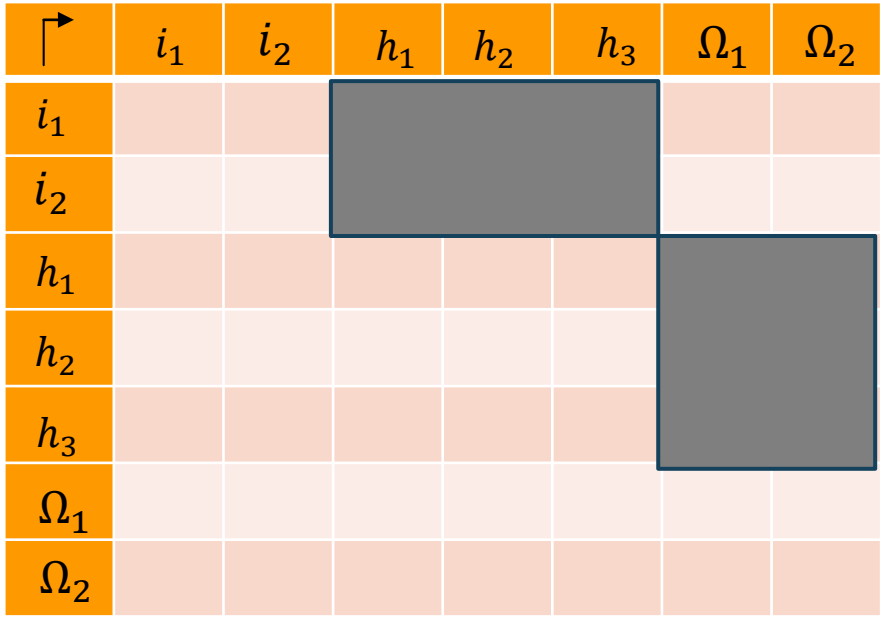


Netz mit Vorwärts-Verbindungen über **mehrere Ebenen** (shortcut-connections)

Netztopologien

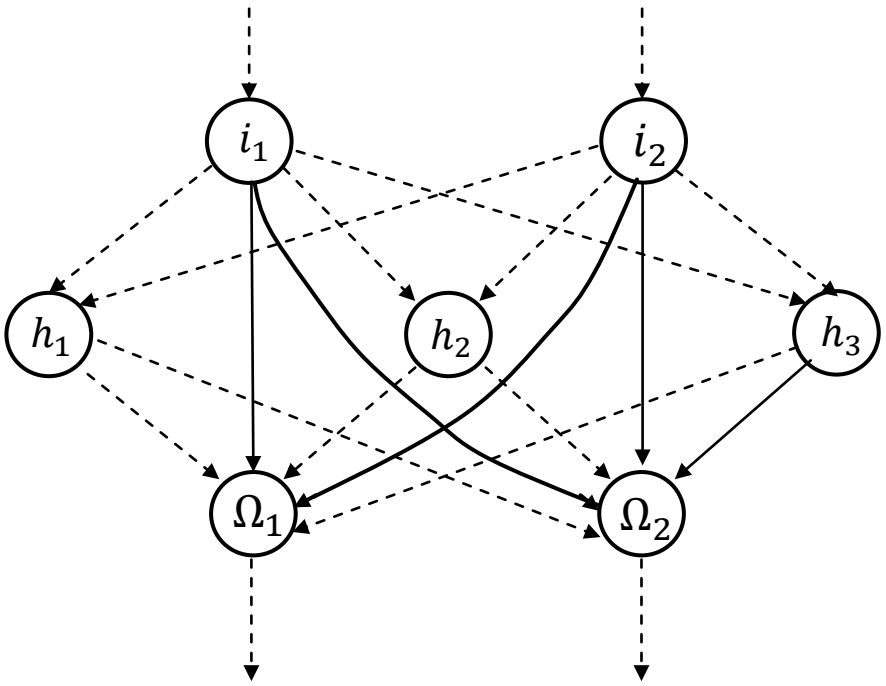
Beispiel Feed-Forward – Netz:

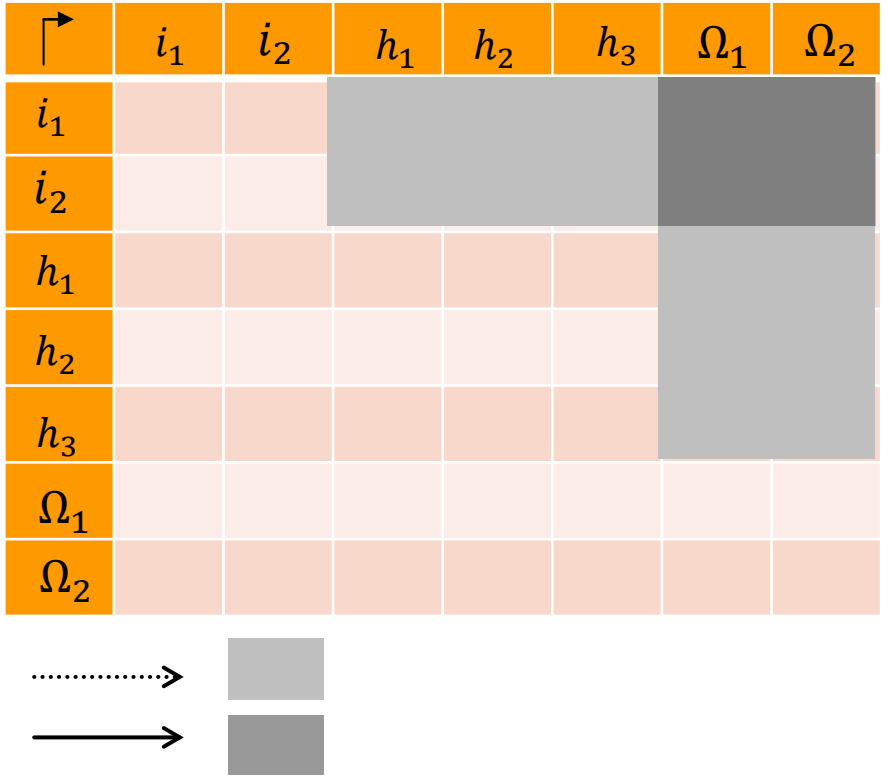




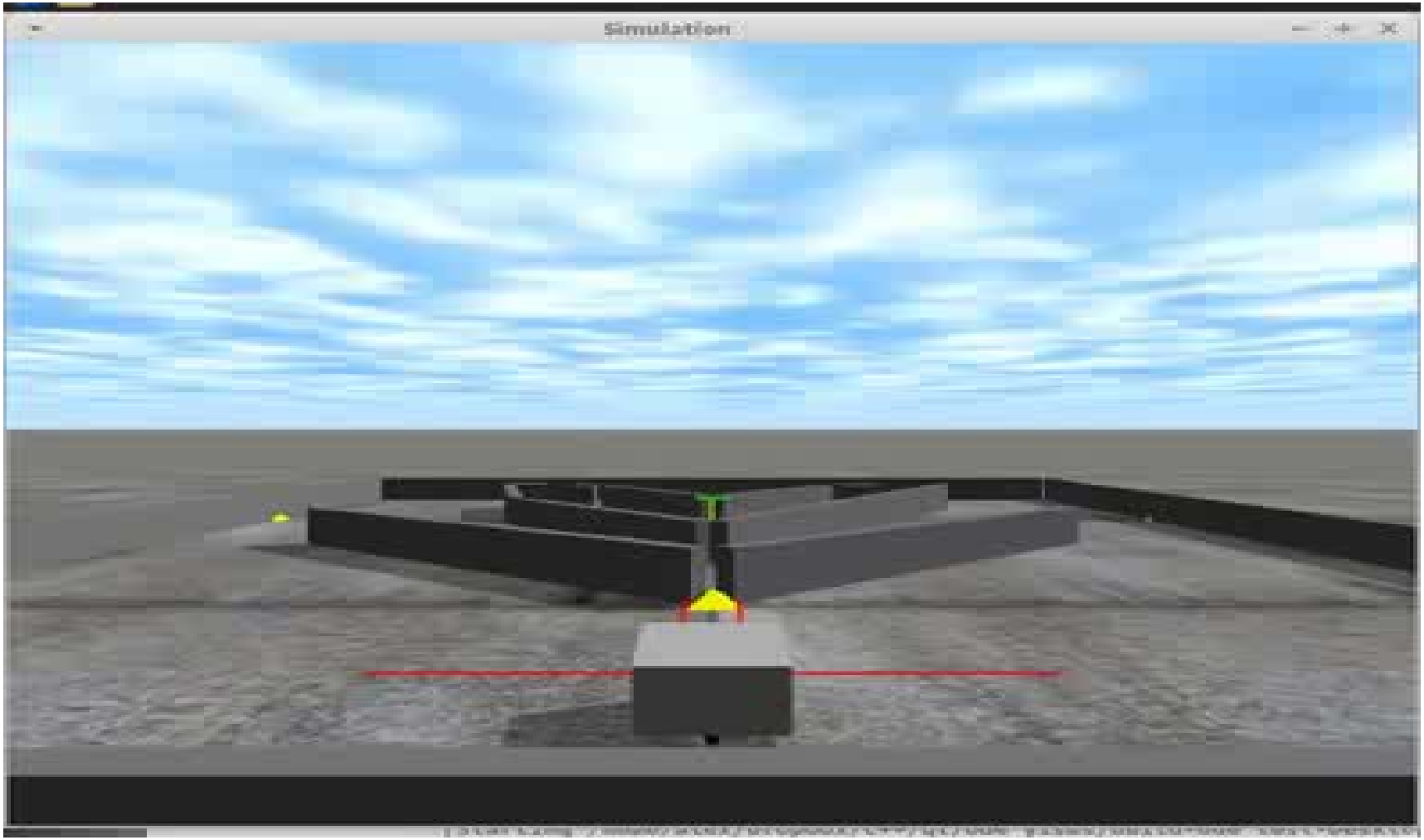
Netztopologien

Beispiel Feed-Forward-Netz mit durchgezogen Shortcut-Connection:





Netztopologien



www.youtube.com/watch?v=zG28vxlVNKA

Netztopologien

Feed-Forward-Netze – Pattern Associator:

- Erkennung bekannter Muster
- Klassische Konditionierung: Assoziationen zwischen verschiedenen Reizpaaren
- Keine Hidden-Units
- Trainingsphase: Hebb‘sche oder Delta-Regel (mehr dazu später)

Netztopologien

Feed-Forward-Netze – Kompetitive Netze:

- Keine Hidden-Units, aber prinzipiell möglich
- Trainingsphase erfolgt in drei Schritten:
  1. Erregung
  2. Wettbewerb
  3. Anpassung der Gewichte
- Unüberwacht → Ohne Vorgabe eines korrekten, externen Output-Reizes

Netztopologien

Kompetitive Netze:

- Gewichtsvektor = alle Gewichte einer bestimmten Output-Unit
- Begrenzung der Größe (Absolutbetrag) aller einzelnen Gewichtsvektoren einer Schicht auf konstanten Wert

Netztopologien

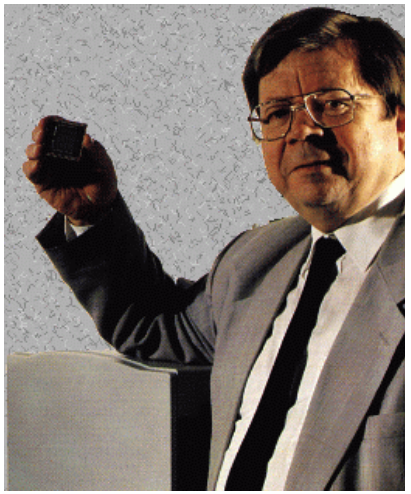
Kompetitive Netze – Anwendungen:

- Filtern von Redundanzen und Alternative zur Faktorenanalyse (z.B. Erzeugung von Output, der weniger korreliert ist als Input)
- Vorgeschaltetes Netz für andere Netztypen
- Mustererkennung (z.B. von Buchstaben)
- Korreliertes Lernen (z.B. bei nicht linear separierbaren Problemen)

Netztopologien

Feed-Forward-Netze – Kohonennetze/ SOMs:

- Erweiterung kompetitiver Netze
- Unüberwachtes bzw. selbst-organisiertes Lernen
  → Zustandsänderungen ≙ Ausgabe
- Frage: *welches* Neuron ist aktiv?
- Biologische Plausibilität



Teuvo Kohonen
(*1934)

Quelle: http://www.neuronalesnetz.de/kohonennetze1.html

Netztopologien

Kohonennetze/ SOMs:

- Clustern von Inputraum: Abbildung eines hochdimensionalen Eingaberaumes auf Bereiche in einem niedrigdimensionalen Gitter → Kartographieren
- Gitter: gibt Topologie der darauf liegenden Neurone an

Netztopologien

Definition (SOM-Neuron):

Ähnlich zu den Neuronen in einem RBF-Netz besitzt ein SOM-Neuron k eine feste Position c_k (Zentrum) im Eingaberaum.

Netztopologien

Definition (Self Organizing Map):

Eine Self Organizing Map ist eine Menge K SOM-Neuronen. Bei Eingabe eines Eingabevektors wird genau dasjenige Neuron $k \in K$ aktiv, welches dem Eingabemuster im Eingaberaum am nächsten liegt. Die Dimension des Eingaberaumes nennen wir N.

Netztopologien

Definition (Topologie):

Die Neurone sind untereinander durch Nachbarschaftsbeziehungen verbunden: **Topologie**
Die Topologie nimmt starken Einfluss auf das Training einer SOM. Sie wird durch die Topologiefunktion $h(i, k, t)$ definiert, wobei i das Gewinnerneuron ist, k das gerade zu adaptierende Neuron und t der Zeitschritt. Wir bezeichnen die Dimension der Topologie mit G.

Netztopologien

SOMs – Topologiefunktion h:

Die Topologiefunktion $h(i, k, t)$ beschreibt die Nachbarschaftsbeziehungen in der Topologie. Sie kann eine beliebige unimodale Funktion sein, die maximal wird, wenn $i = k$ gilt.

Eine Zeitabhängigkeit ist optional, wird aber oft verwendet

Netztopologien

SOMs – Topologiefunktion h:

Gaußfunktion

$$h(i,k,t) = e^{\left(-\frac{\|g_i - g_k\|^2}{2 \cdot \sigma(t)^2}\right)}$$

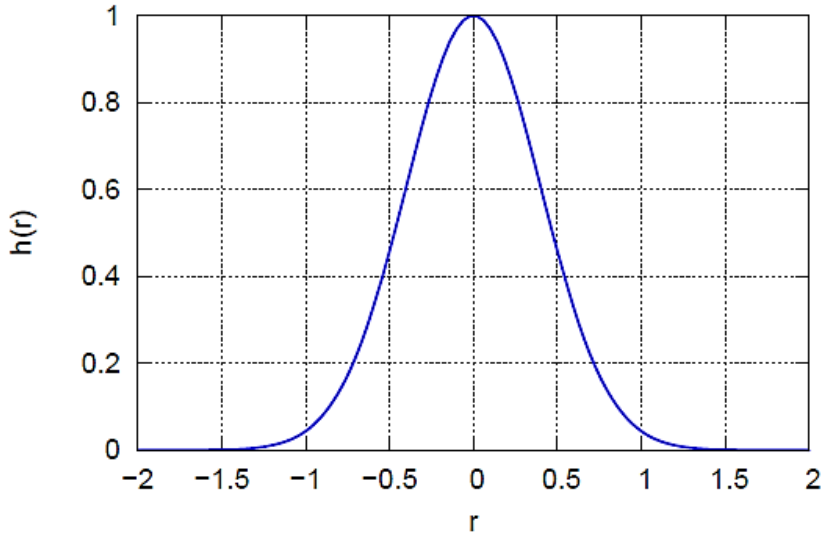
Mit g_i und g_k als Positionen der *Neuronen auf dem Gitter*, nicht im Eingaberaum.

Weitere Funktionen: Kegelfunktion, Zylinderfunktion, Mexican-Hat-Funktion

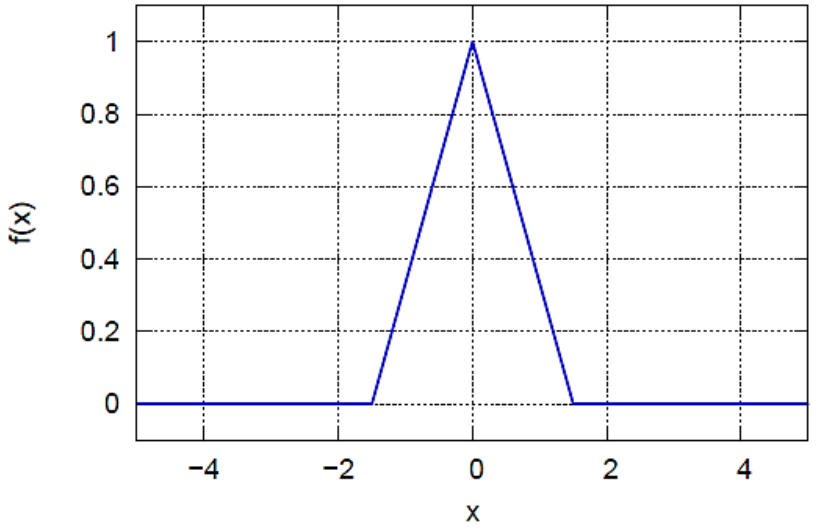
Netztopologien

SOMs – Topologiefunktion h:

Gauß-Glocke in 1D



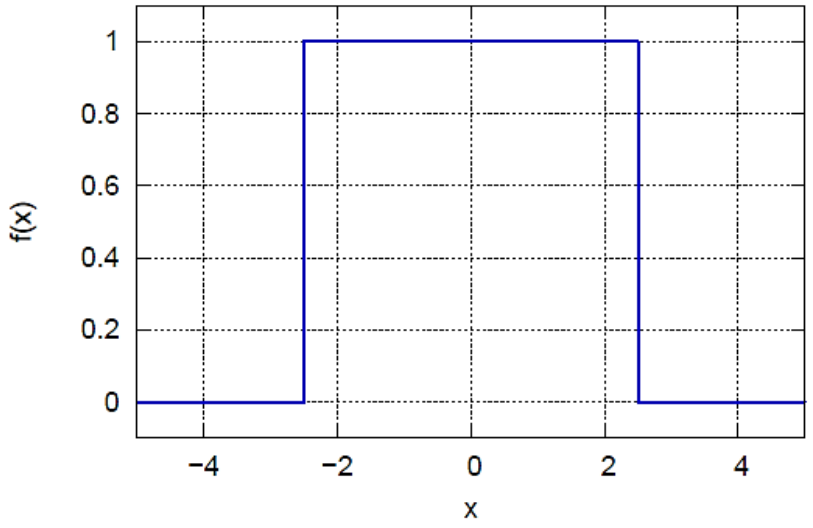
Kegelfunktion



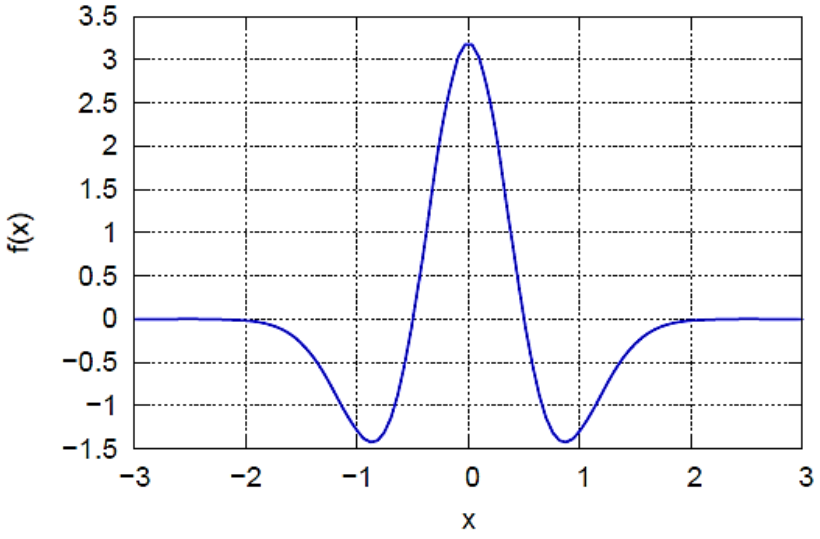
Netztopologien

SOMs – Topologiefunktion h:

Zylinderfunktion

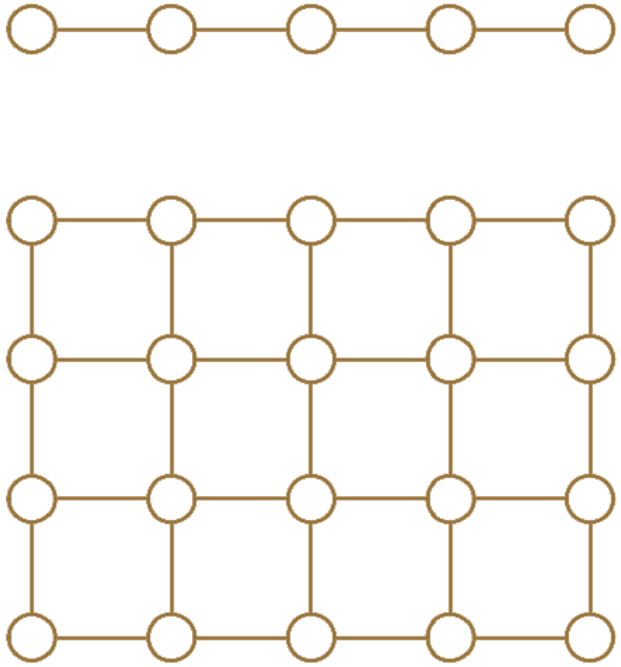


Mexican-Hat-Funktion



Netztopologien

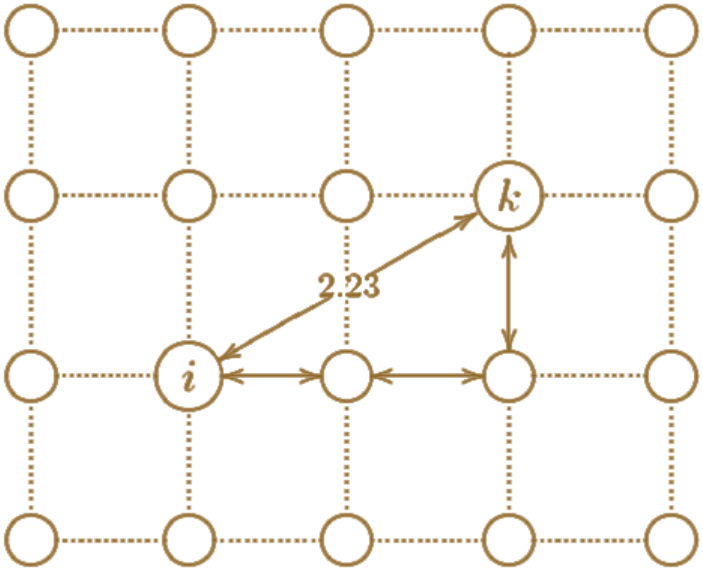
Beispiel SOM: ein- und zweidimensionale Topologie



Netztopologien

Beispiel-Abstände in SOM:

- 1-dim: Anzahl diskreter Weglänge zwischen i und k
- 2-dim: Euklidischer Abstand
- Hier: Gitterkantenlänge 1



Netztopologien

SOMs – Training:

Initialisierung: Start des Netzes mit zufälligen Neuronenzentren $c_k \in \mathbb{R}^N$ aus dem Eingangsraum

Anlegen eines Eingangsmusters: Es wird ein Stimulus, also ein Punkt p aus dem Eingangsraum $\mathbb{R}^N$ gewählt und in das Netz eingegeben.

Abstandsmessung: Für jedes Neuron k im Netz wird nun der Abstand $\| p - c_k \|$ bestimmt.

Netztopologien

SOMs – Training:

Winner takes all: Gewinnerneuron i mit dem kleinsten Abstand zu p wird ermittelt

$$\| p - c_i \| \leq \| p - c_k \| \qquad \forall k \neq i$$

Adaption der Zentren: Die Zentren der Neuronen werden innerhalb des Eingangsraumes versetzt

$$\Delta c_k = \eta(t) \cdot h(i,k,t) \cdot (p - c_k)$$
$$\Delta c_k(t+1) = c_k(t) + \Delta c_k(t)$$

Netztopologien

SOMs – Training:

Typische Größenordnungen für den Zielwert der Lernrate sind zwei Größenordnungen kleiner als der Startwert.

Beispiel:

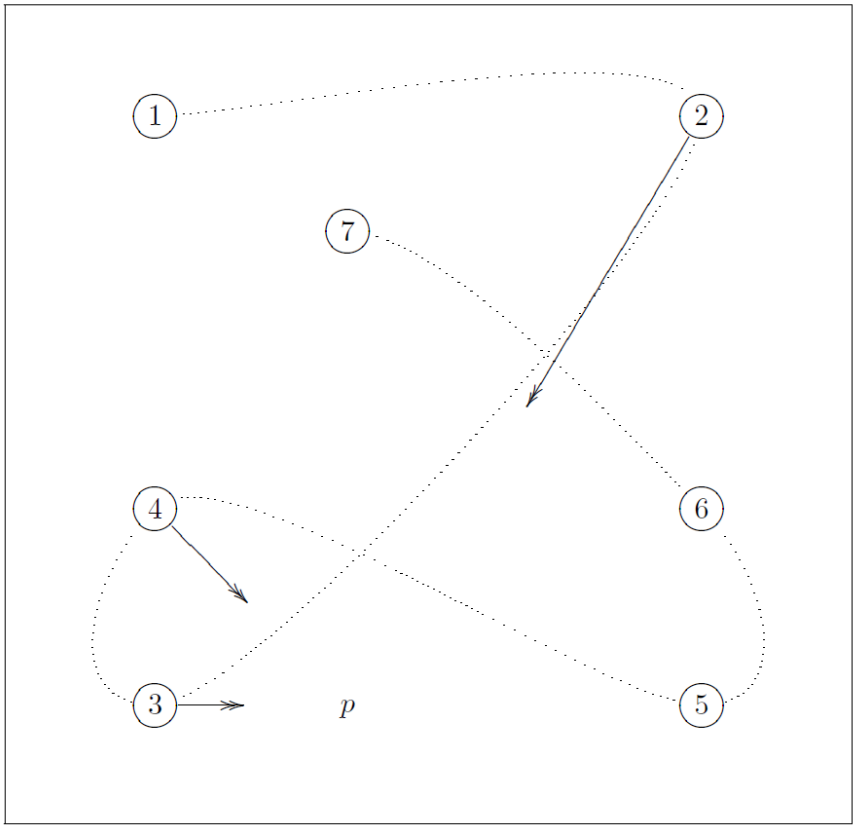
$$0.01 < \eta < 0.6$$

Es gilt stets:

$$h \cdot \eta \leq 1$$

Netztopologien

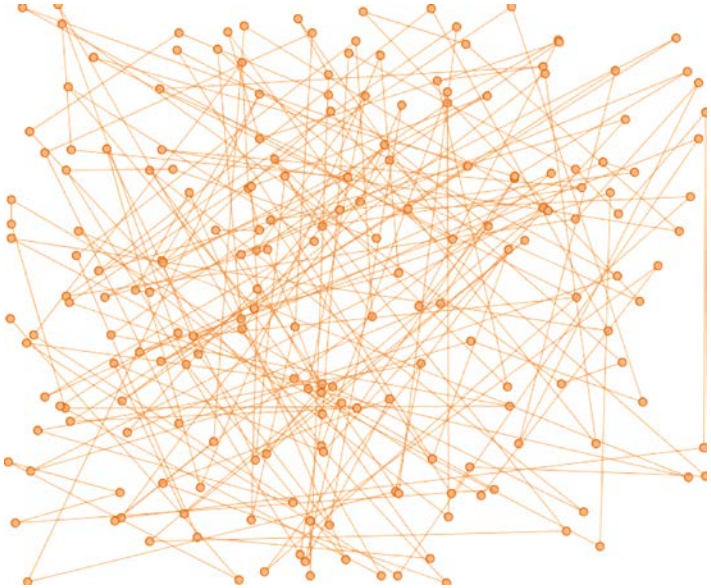
Beispiel SOMs – Training: h(i,k,t)= 1 für k=i oder k = Nachbar von i, 0 sonst



Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

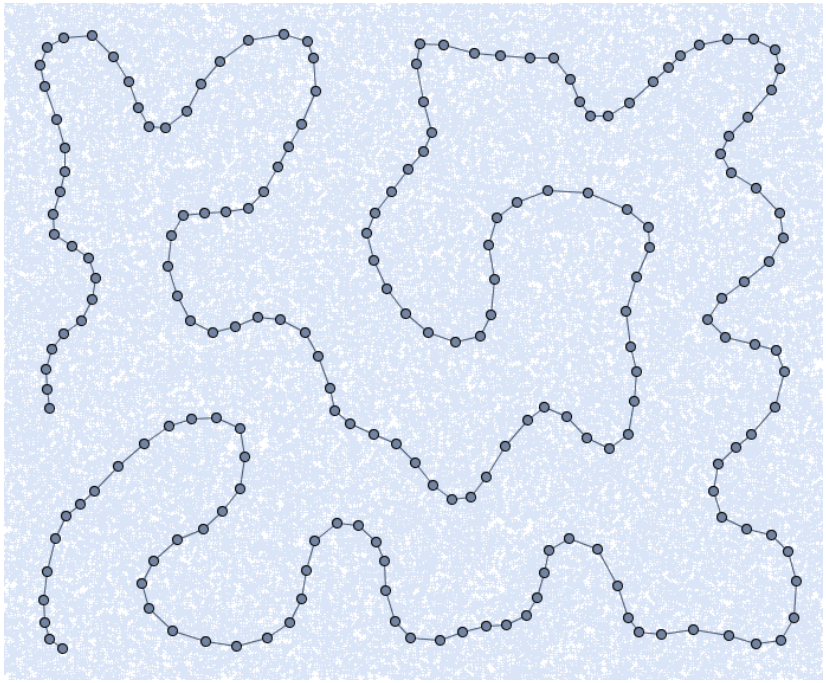
Beispiel SOMs – Training:

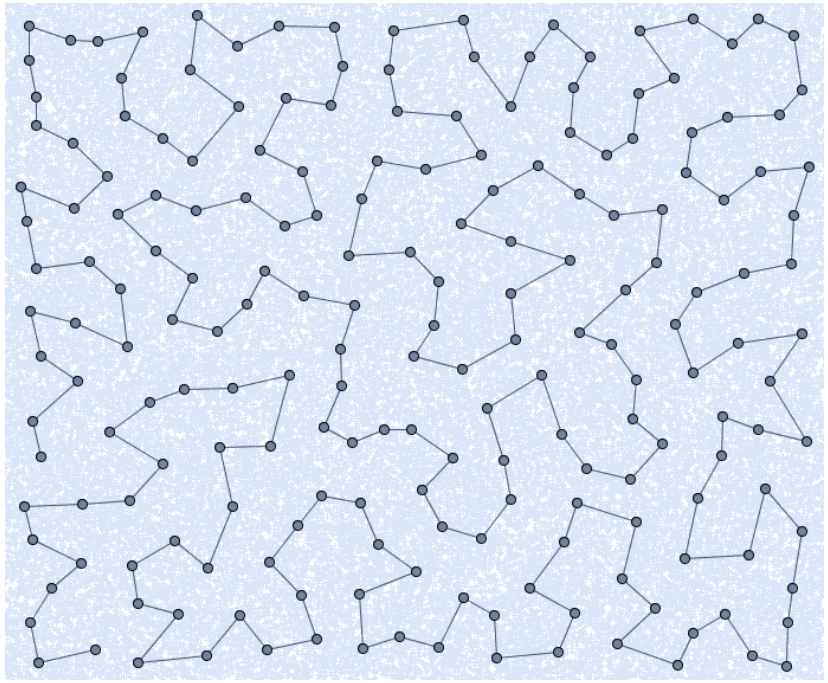


Verhalten einer SOM mit eindimensionaler Topologie ($G = 1$) nach Eingabe von 0, 100 zufällig verteilten Eingabemustern $p \in \mathbb{R}^2$. η fiel während des Trainings von 1.0 auf 0.1, der σ-Parameter der als Nachbarschaftsmaß eingesetzten Gauß-Funktion von 10.0 auf 0.2.

Netztopologien

Beispiel SOMs – Training:



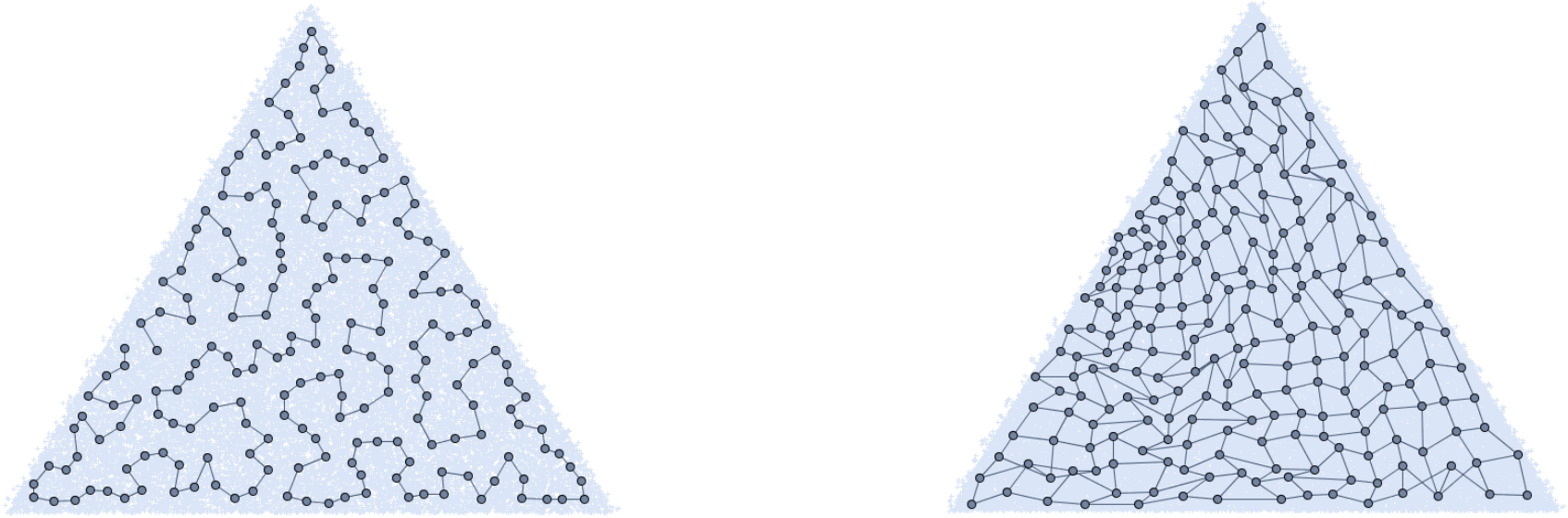


Verhalten einer SOM mit eindimensionaler Topologie ($G = 1$) nach Eingabe von 70000, 80000 zufällig verteilten Eingabemustern

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOMs – Endzustände:

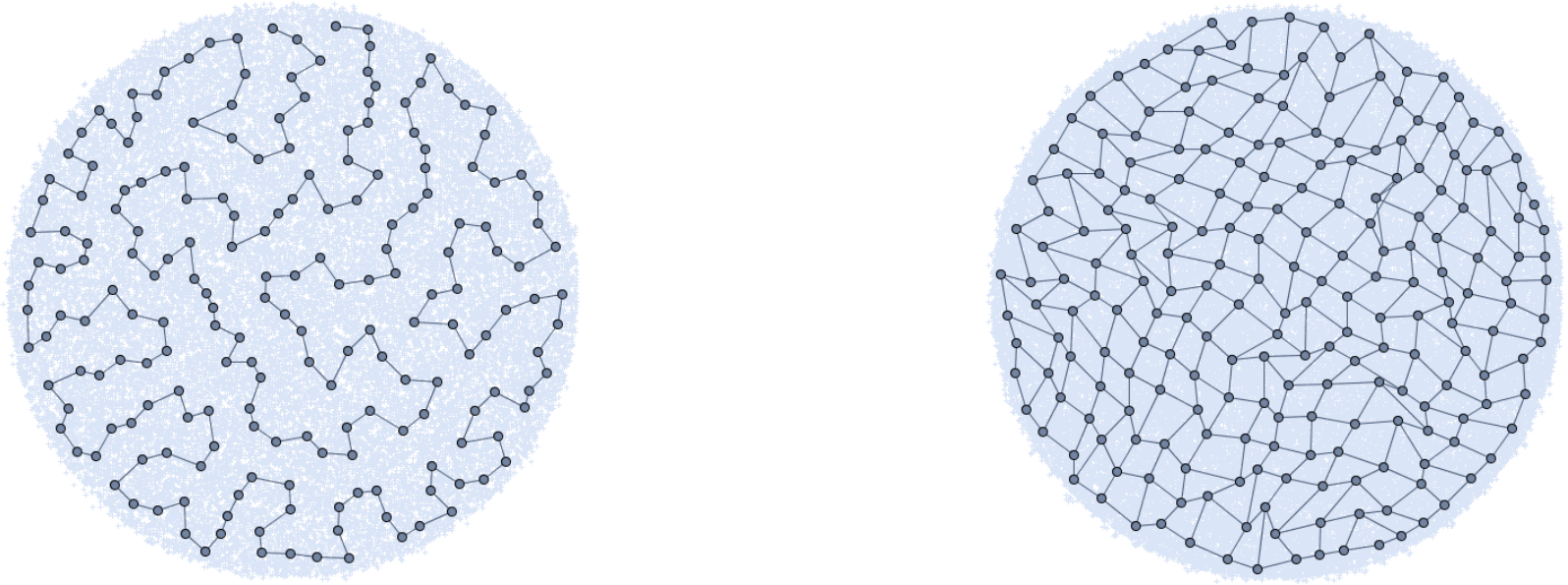


Endzustände von SOMs auf verschieden abgedeckten Inputräumen. Genutzt wurden bei eindimensionaler Topologie (links) 200 Neuronen, bei zweidimensionaler (rechts) 10 × 10 Neuronen und bei allen Karten 80.000 Eingabemuster.

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOMs – Endzustände:

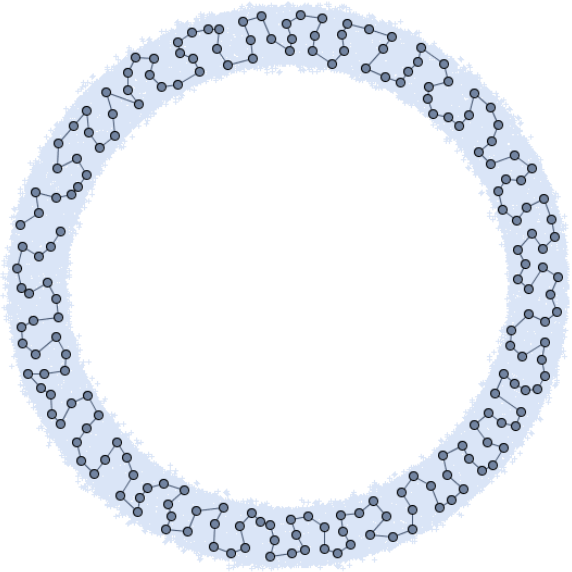


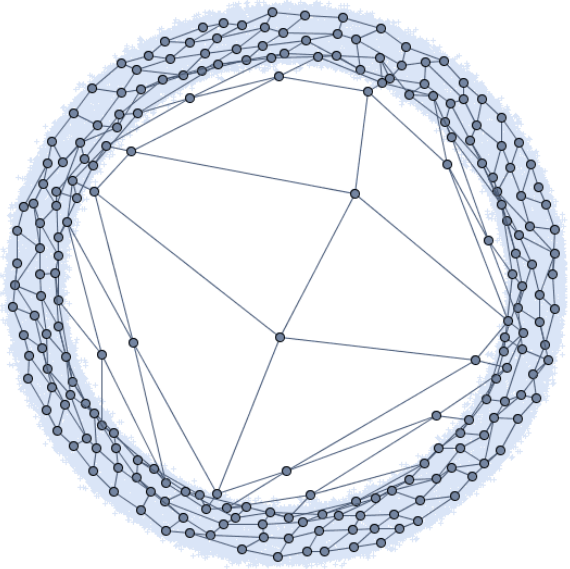
Endzustände von SOMs auf verschieden abgedeckten Inputräumen. Genutzt wurden bei eindimensionaler Topologie (links) 200 Neuronen, bei zweidimensionaler (rechts) 10 × 10 Neuronen und bei allen Karten 80.000 Eingabemuster.

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOMs – Endzustände:



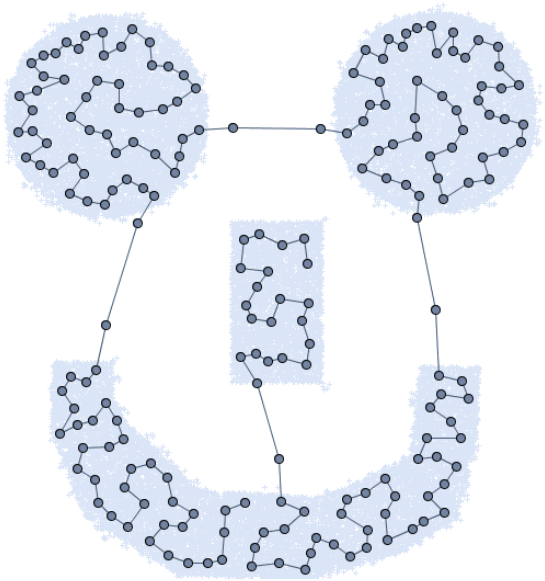


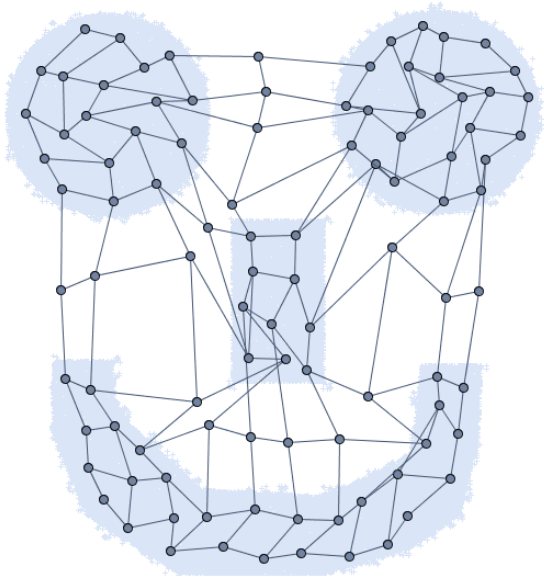
Endzustände von SOMs auf verschieden abgedeckten Inputräumen. Genutzt wurden bei eindimensionaler Topologie (links) 200 Neuronen, bei zweidimensionaler (rechts) 10×10 Neuronen und bei allen Karten 80.000 Eingabemuster.

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOMs – Endzustände:



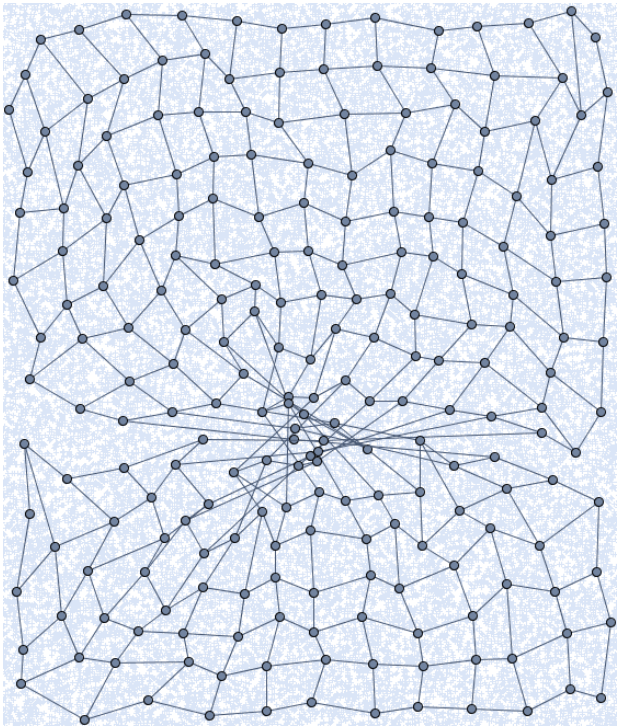


Endzustände von SOMs auf verschieden abgedeckten Inputräumen. Genutzt wurden bei eindimensionaler Topologie (links) 200 Neuronen, bei zweidimensionaler (rechts) 10×10 Neuronen und bei allen Karten 80.000 Eingabemuster.

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOMs – Topologische Defekte:

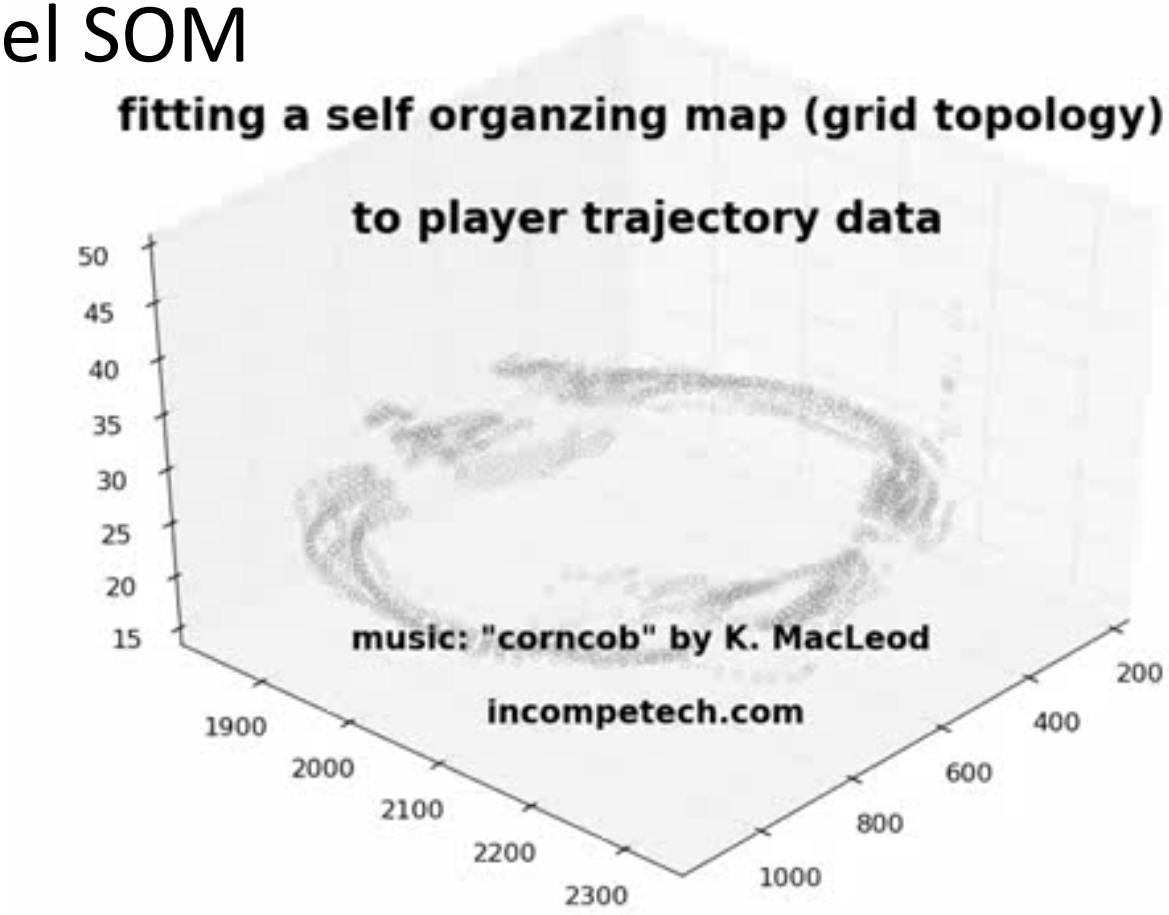


„Verknotung“ durch Fehlentfaltung in einer zweidimensionalen SOM.

Quelle: http://www.dkriesel.com/science/neural_networks

Netztopologien

Beispiel SOM



Quelle: http://www.youtube.com/user/bitLectures

Netztopologien

Kohonennetze/ SOMs — Anwendungen:

- Approximation von Funktionen (u.a. bei der keine analytische Lösung existiert)
- Inverse Kinematik, z.B. bei mechanischen Armen bei Robotern im 2-dim. Raum
  → Finden kürzesten Weg zwischen 2 Punkten
- Assoziative Speicherung von Daten
- Kontextbasierte Suche

Netztopologien

Kohonennetze/ SOMs – Anwendungen:

- Traveling Salesman Problem: Elastischer Netzalgorithmus
- Spracherkennung, Unterschriftenerkennung, Gesichtserkennung, …

Netztypen – Zusammenfassung

Vorwärtsgerichtete Netze:

	Pattern Associator	Kompetitive Netze	Kohonennetze
Kernkonzept	Assoziationen zwischen verschiedenen Reizpaaren bilden	1.Erregung 2.Wettbewerb 3.Gewicht-modifikation	Wie kompetitive Netze, nur mit mehrdimensionaler Output-Schicht
Lernregel	Hebb-Regel; Delta-Regel	Competitive Learning	Konzeptuell: Competitive Learning
Rückkopplungen?	Nein	Nein	Nein
Hidden-Units?	Nein	Können vorhanden sein	In der Regel nicht
Art der Lernregel	Supervised learning	Unsupervised learning	Unsupervised learning
Vorteile	Einfachheit	Biologische Plausibilität	Biologische Plausibilität
Nachteile	Keine Hidden-Units →biologisch eher unplausibel	„Erstarken“ einzelner Output-Units verhindert „sinnvolle“ Kategorisierung	Wahl zahlreicher Parameter entscheidend für adäquate Clusterung

Aufgabe 1

Implementieren Sie ein einfaches NN, das wie folgt funktioniert:

- Eingabe: eine 2 dimensionale Matrix.
- Multiplikation der Eingabe mit festen Gewichten
  - Matrixmultiplikation.
- Aktivierungsfunktion: Sigmoid oder tanh().
- Eine Ausgabe zurückgeben.
- Berechnung des Fehlers, indem die Differenz aus der gewünschten Ausgabe der Daten und der vorhergesagten Ausgabe genommen wird.
- Leichte Anpassung der Gewichte basierend auf dem Fehler.
- Das NN für 3000 Iterationen trainieren.