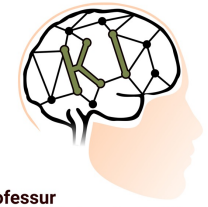


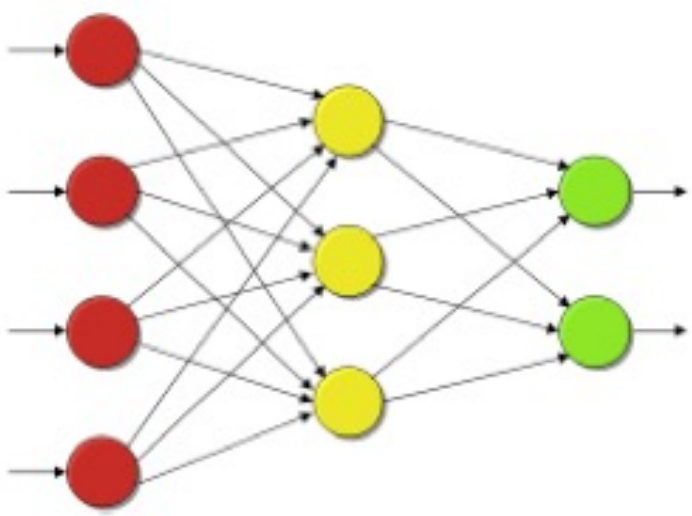
Professur
Künstliche Intelligenz

Kapitel 5:
Künstliche Neuronale Netze

CHEMNITZ UNIVERSITY
OF TECHNOLOGY

Prof. Dr. Fred Hamker,
Department of Computer Science

Künstliche Neuronale Netze



Inhalt:

- Einführung
- Aufbau künstlicher Neuronaler Netze
- Perzeptron und Trennbarkeit
- Historie und Zukunft Neuronaler Netze
- Lernen in künstlichen Neuronalen Netzen
- Lernen im Perzeptron
- Delta-Lernregel
- Backpropagation Lernregel
- Verwendung und Bewertung Neuronaler Netze
- Unüberwechtes und Hebb‘sches Lernen
- Deep Learning
- Spiking Neuronen
- Reservoir Computing und Echo-State Netze

Empfohlene Literatur:

- Haykin, S. Neural Networks and Learning Machines (3rd Edition), Prentice Hall, 2009.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press.

Was sind künstliche Neuronale Netze ?

- Der Begriff künstliche Neuronale Netze steht für mathematische Modelle, die sich an der Informationsverarbeitung des Gehirns orientieren
- Neuronale Netze bestehen aus vielen kleinen Verarbeitungseinheiten, den Neuronen, die miteinander verbunden sind und die Fähigkeit zum Lernen besitzen.
- Neuronale Netze speichern das Wissen verteilt in sogenannten Verbindungsgewichten
- Neuronale Netze verarbeiten Information subsymbolisch

Vorteile von künstlichen Neuronalen Netzen

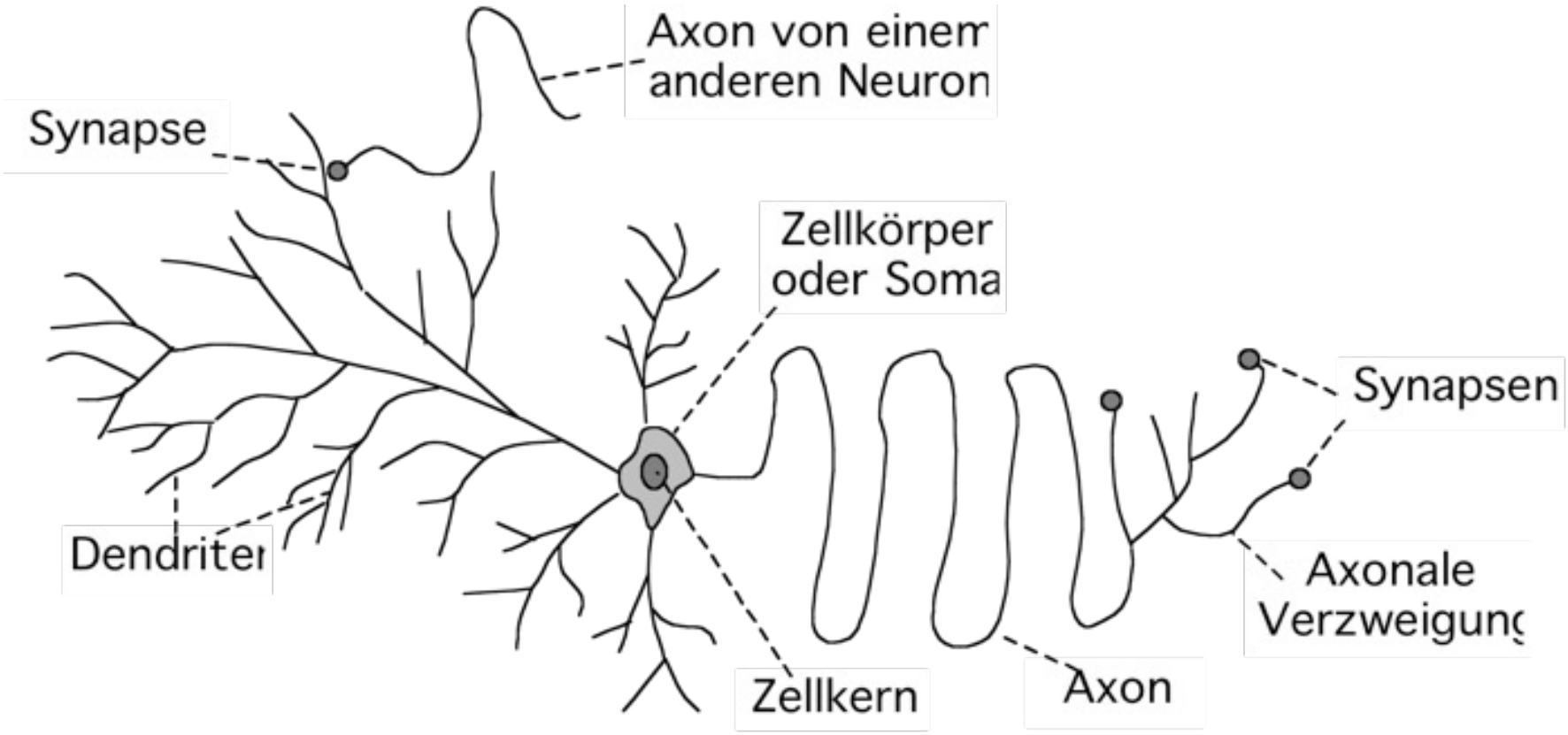
Neuronale Netze erweisen sich insbesondere dann als vorteilhaft:

- wenn es a priori schwierig oder nicht praktikabel erscheint, konkrete Annahmen über einen möglichen nichtlinearen Zusammenhang von mehreren Variablen zu formulieren
- wenn eine genaue Kenntnis der Wirkungszusammenhänge zweitrangig ist und es vorrangig auf eine möglichst präzise funktionale Approximation der Zusammenhänge ankommt.
- wenn (viele) Beispiele zum Lernen vorhanden sind.

Computational Neuroscience / Neurokognition

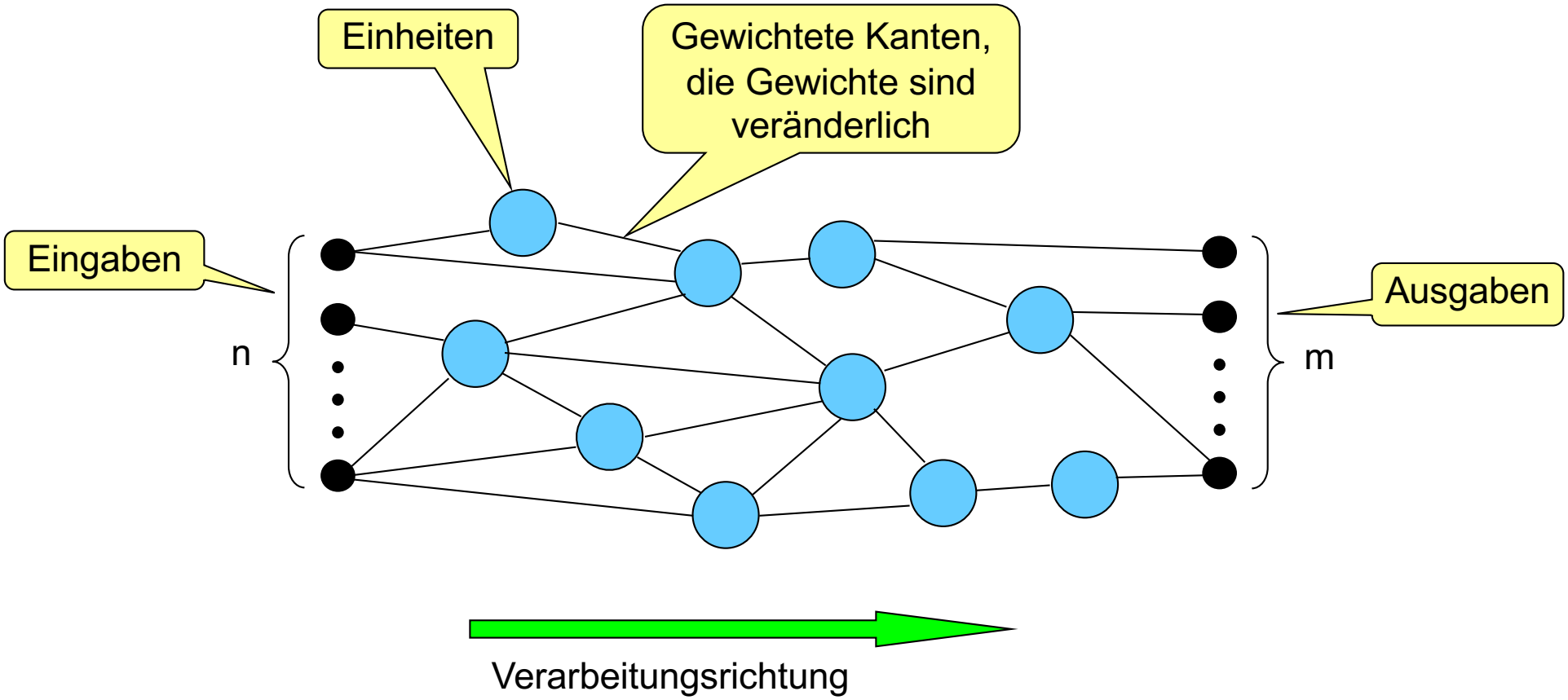
- Künstliche Neuronale Netze sind nochmals zu unterscheiden von Ansätzen, die kognitive Prozesse des Gehirns oder das Verhalten bestimmter Neurone im Detail untersuchen.
- Künstliche Neuronale Netze haben zwar ein biologisches Vorbild, versuchen aber nicht das Gehirn zu erklären oder nachzuimplementieren.
- Forschungsdisziplinen wie Computational Neuroscience oder Neurokognition sind eng an den Daten der Neurowissenschaften orientiert, während künstliche Neuronale Netze nur die Konzepte übernehmen und anwenden.

Schematischer Aufbau eines Neurons



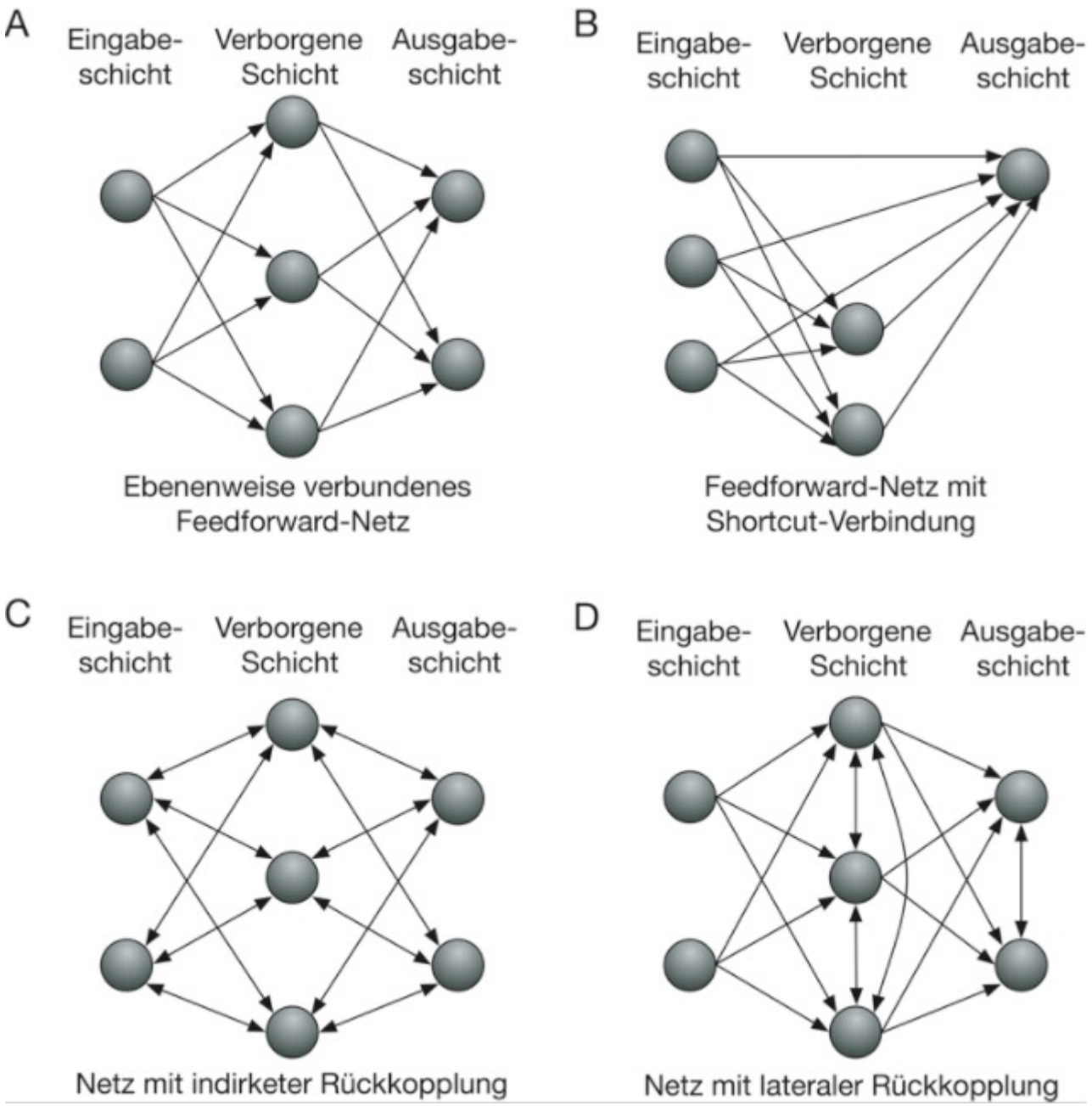
Arbeitsweise eines (künstlichen) Neurons

Aufbau künstlicher Neuronaler Netze



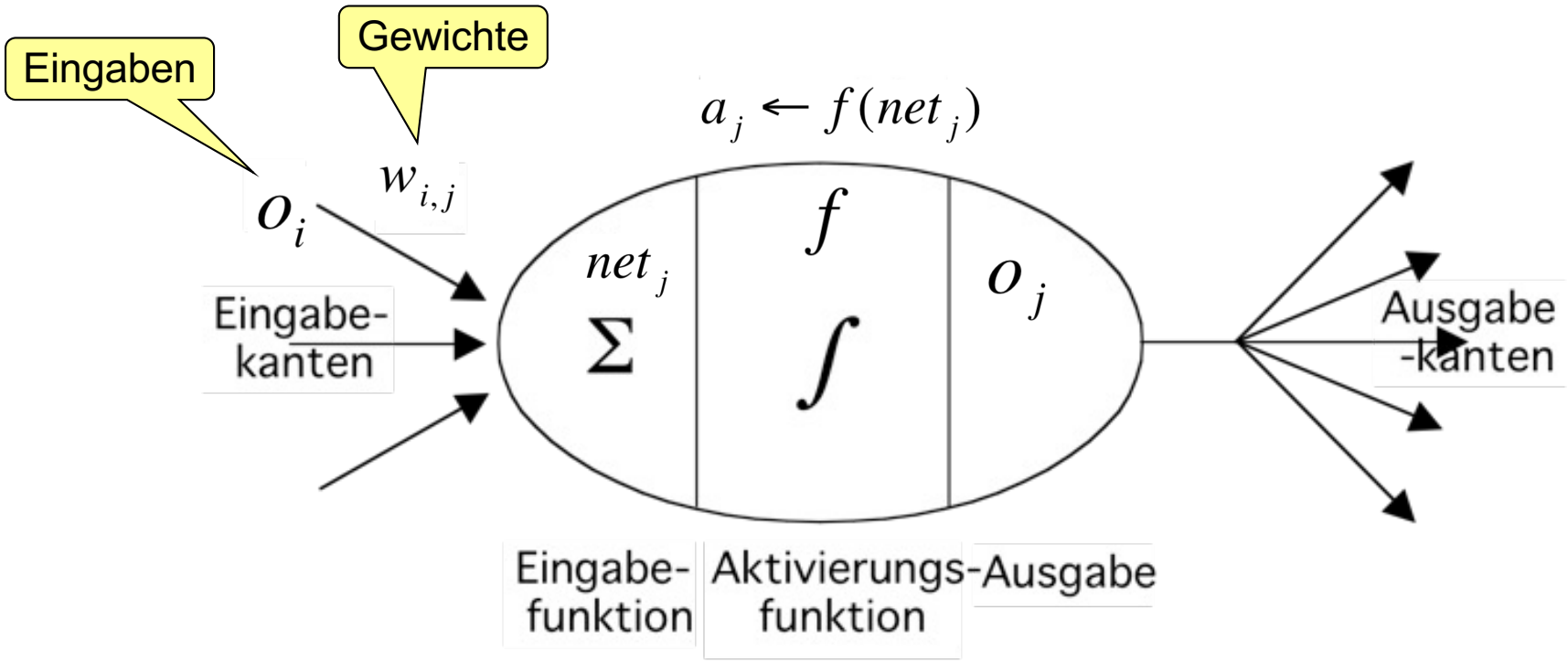
Prinzipiell können die Neuronen beliebig untereinander verbunden sein. Oft werden Neuronale Netze allerdings in Schichten organisiert, wobei die Schichten hierarchisch angeordnet sind.

Struktur künstlicher Neuronaler Netze



Aus: Hamker, F.H., Zirnsak, M. (2010) Neuronale Netze. In: H. Holling, B. Schmitz (eds.), Handbuch der Psychologischen Methoden und Evaluation, Hogrefe Verlag, Göttingen.

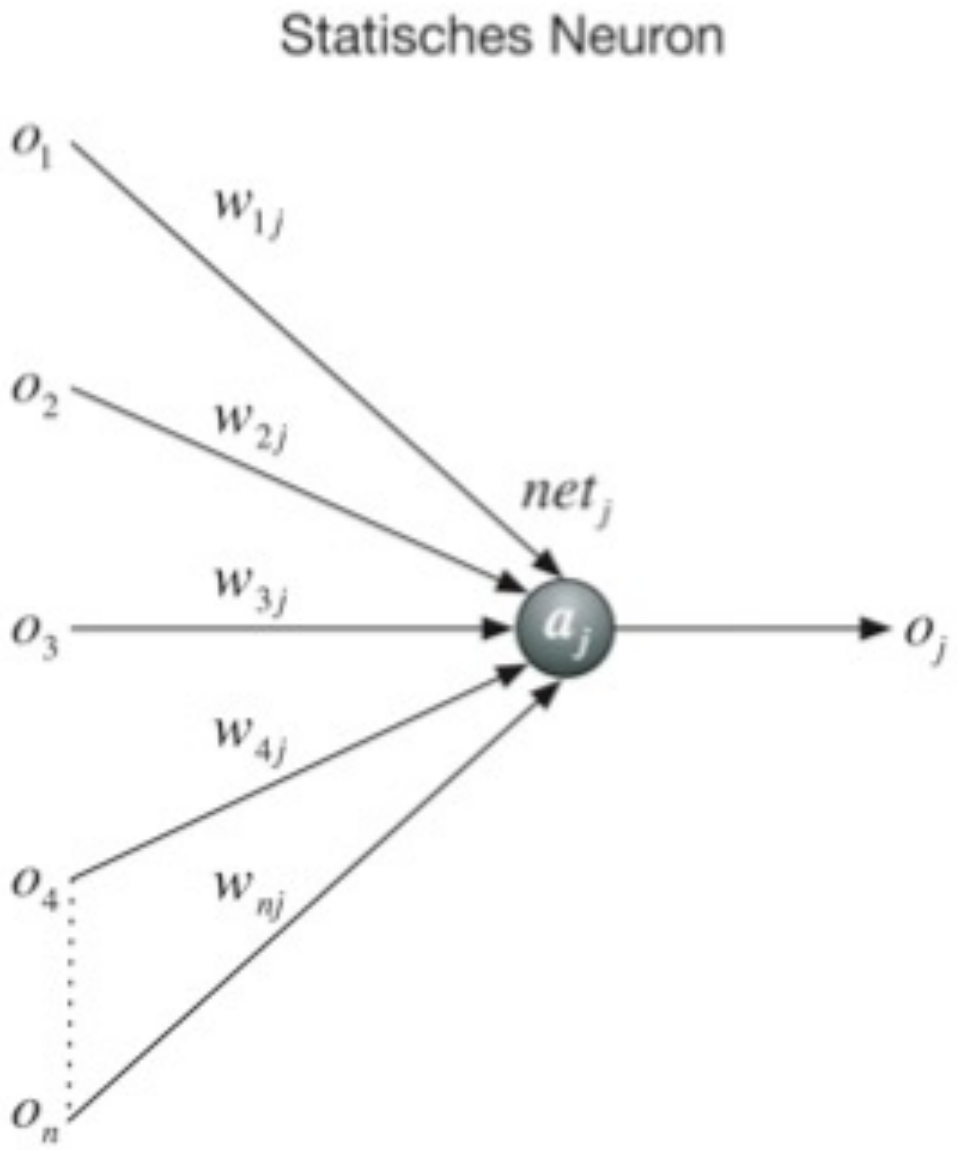
Arbeitsweise eines künstlichen Neurons

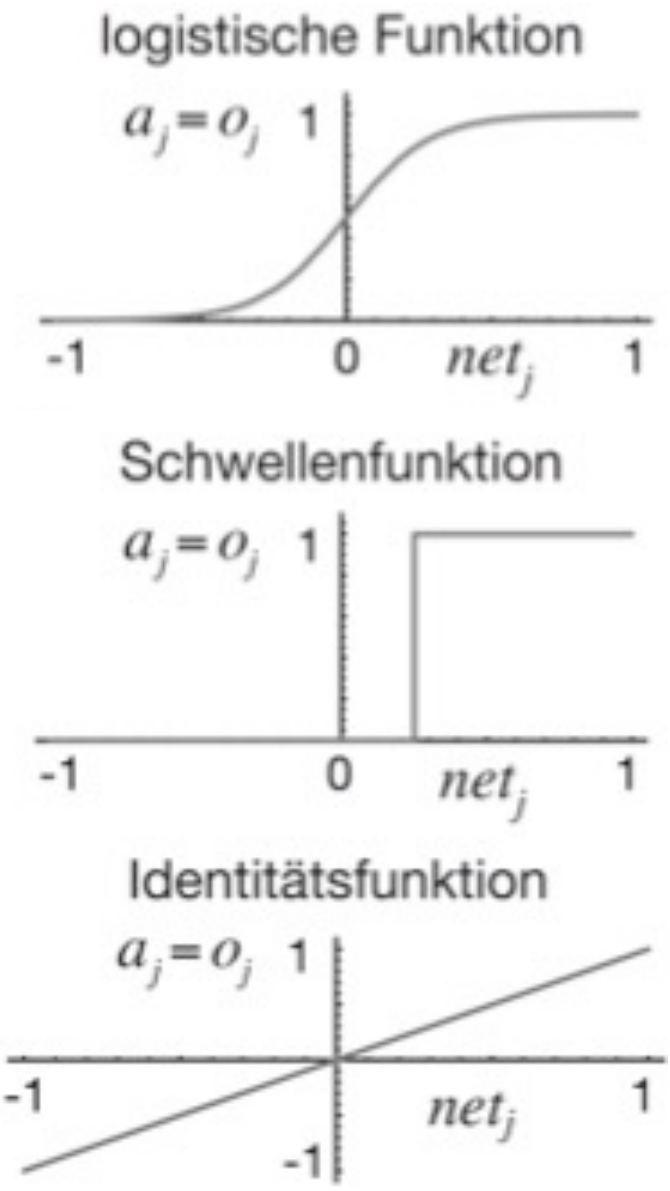


$$net_j = \sum_i w_{i,j} o_i = w_j \cdot o \qquad a_j \leftarrow f(net_j) = f\left(\sum_i w_{i,j} o_i\right)$$

Meistens wird die Ausgabe mit der Aktivierung identifiziert.

Aktivierungsfunktionen

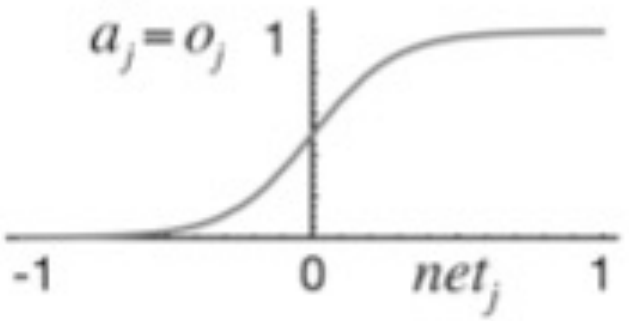




Aktivierungsfunktionen

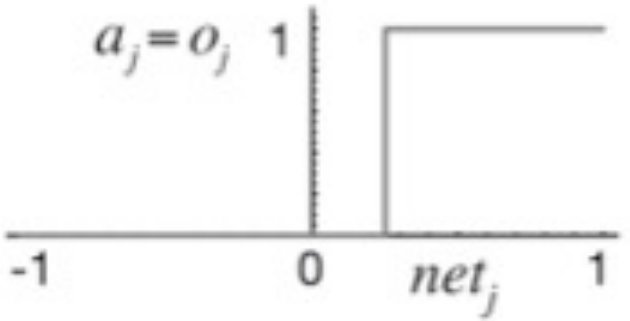
$$o_j = a_j = f(net_j) = \frac{1}{1 + \exp\left(-\beta(net_j - \theta)\right)}$$

logistische Funktion



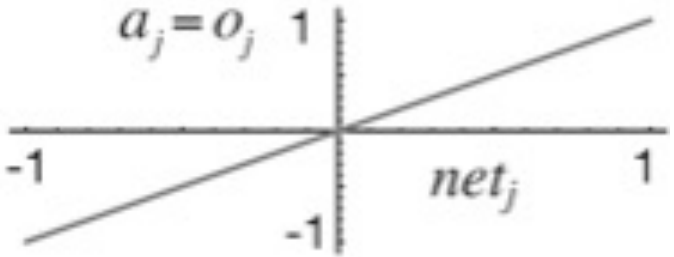
$$o_j = a_j = f(net_j) = \begin{cases} 1, & net_j > \theta \\ 0 & \end{cases}$$

Schwellenfunktion

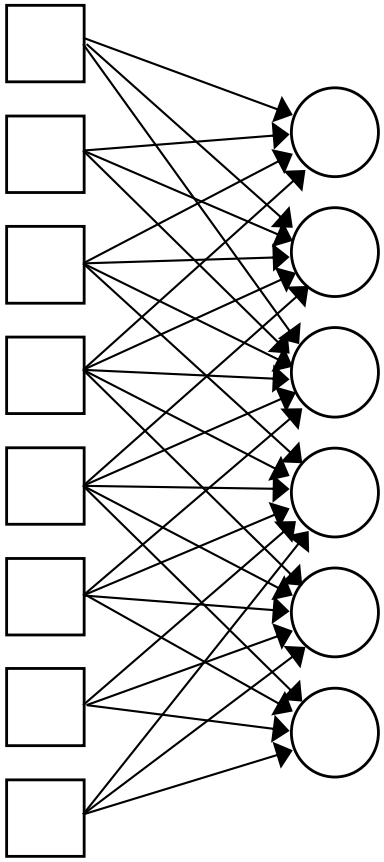


$$o_j = a_j = f(net_j) = net_j$$

Identitätsfunktion



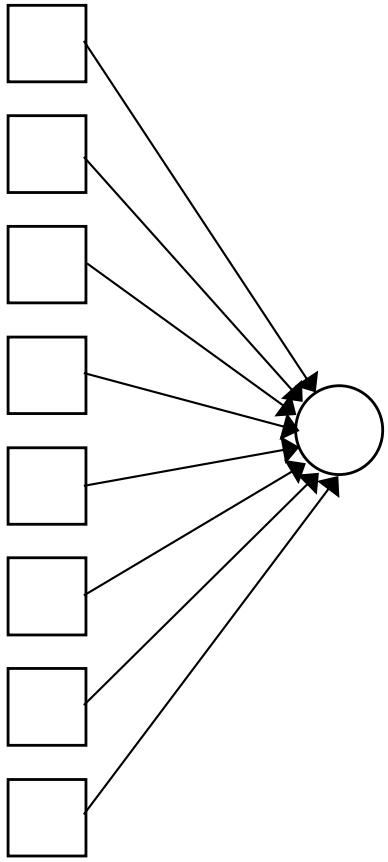
Perzeptron



I_j $W_{j,i}$ O_i

Eingabe-einheiten Ausgabe-einheiten

(a) Perzeptron-Netz



I_j W_j O

Eingabe-einheiten Ausgabe-einheit

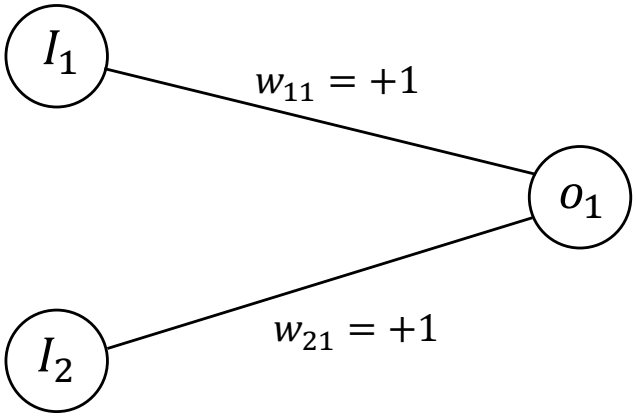
(b) Einzelnes Perzeptron

Frank Rosenblatt (1928–1969) was a computer scientist who completed the Perceptron, or MARK 1, computer at Cornell University in 1960. This was the first computer that could learn new skills by trial and error, using a type of neural network that simulates human thought processes.

http://www.bookrags.com/wiki/Frank_Rosenblatt

Elementare Boolesche Funktionen

AND



$$o_j = f(net_j) = \begin{cases} 1, & net_j > \theta \\ 0 \end{cases}$$

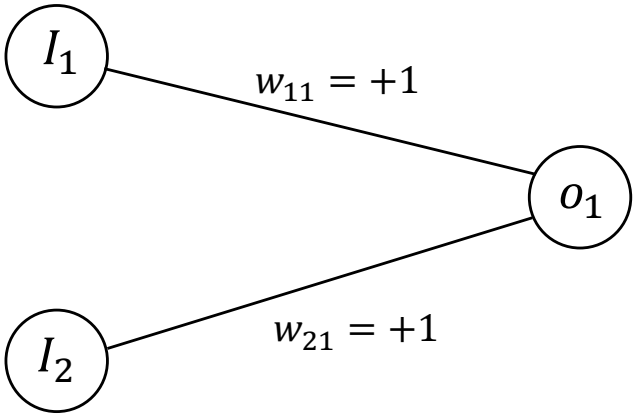
$$f(net_j) = \begin{cases} 1 & if \quad net \geq 1.5 \\ 0 & if \quad net < 1.5 \end{cases}$$

I_1	I_2	$\sum w_{ij} I_j$	o_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1

o_1 ist nur dann gleich 1 wenn I_1 und I_2 jeweils 1 sind

Elementare Boolesche Funktionen

OR



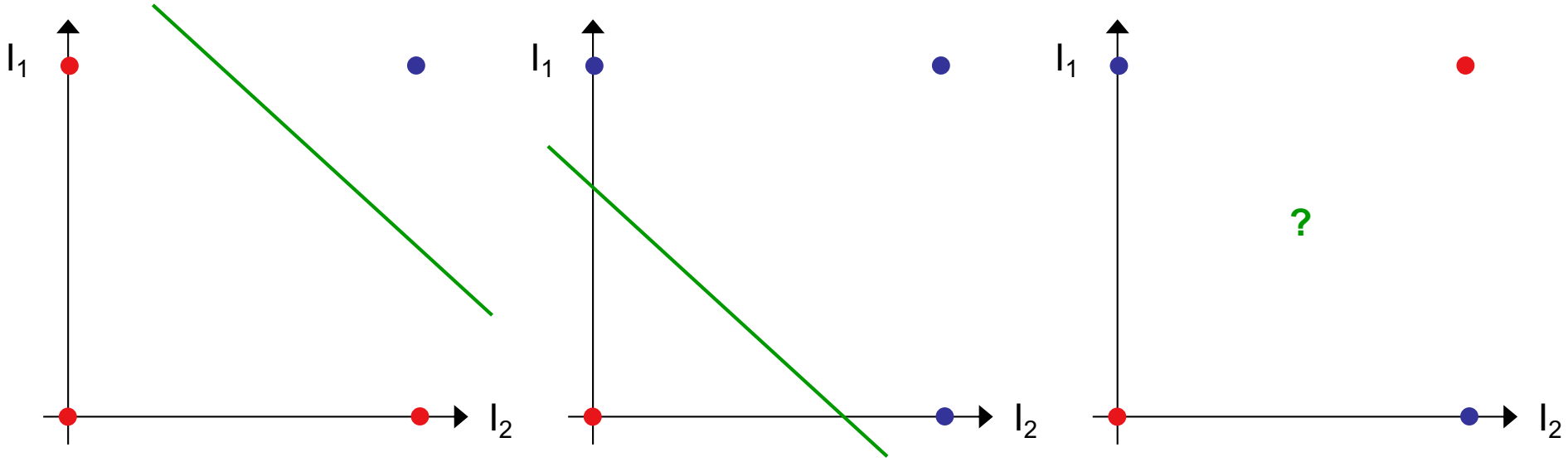
$$o_j = f(net_j) = \begin{cases} 1, & net_j > \theta \\ 0 & \end{cases}$$

$$f(net) = \begin{cases} 1 & if \;\; net \geq 0.5 \\ 0 & if \;\; net < 0.5 \end{cases}$$

I_1	I_2	$\sum w_{ij} I_j$	o_1
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1

o_1 ist dann gleich 1 wenn I_1 oder I_2 (oder beide) jeweils 1 sind

Lineare Trennbarkeit



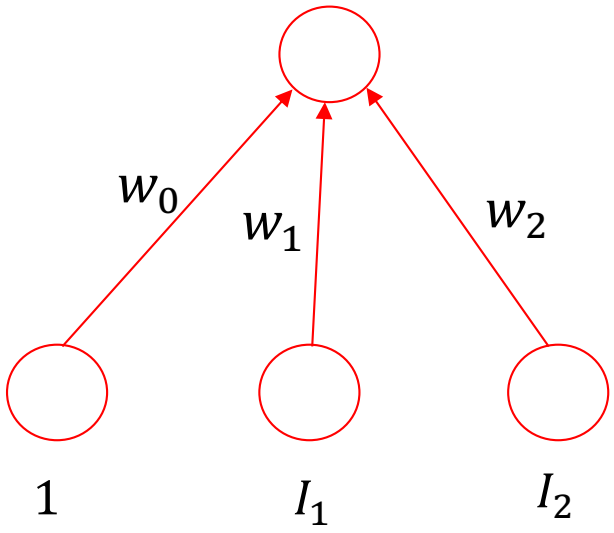
(a) I_1 **and** I_2

(b) I_1 **or** I_2

(c) I_1 **xor** I_2

Lineare Trennbarkeit

XOR

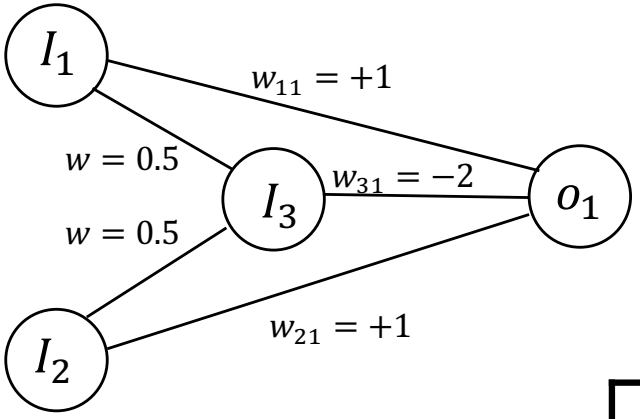


I_1	I_2	
0	0	$w_0 \leq 0$
0	1	$w_2 + w_0 > 0$
1	0	$w_1 + w_0 > 0$
1	1	$w_1 + w_2 + w_0 \leq 0$

nicht möglich (keine Lösung)

Elementare Boolesche Funktionen

XOR

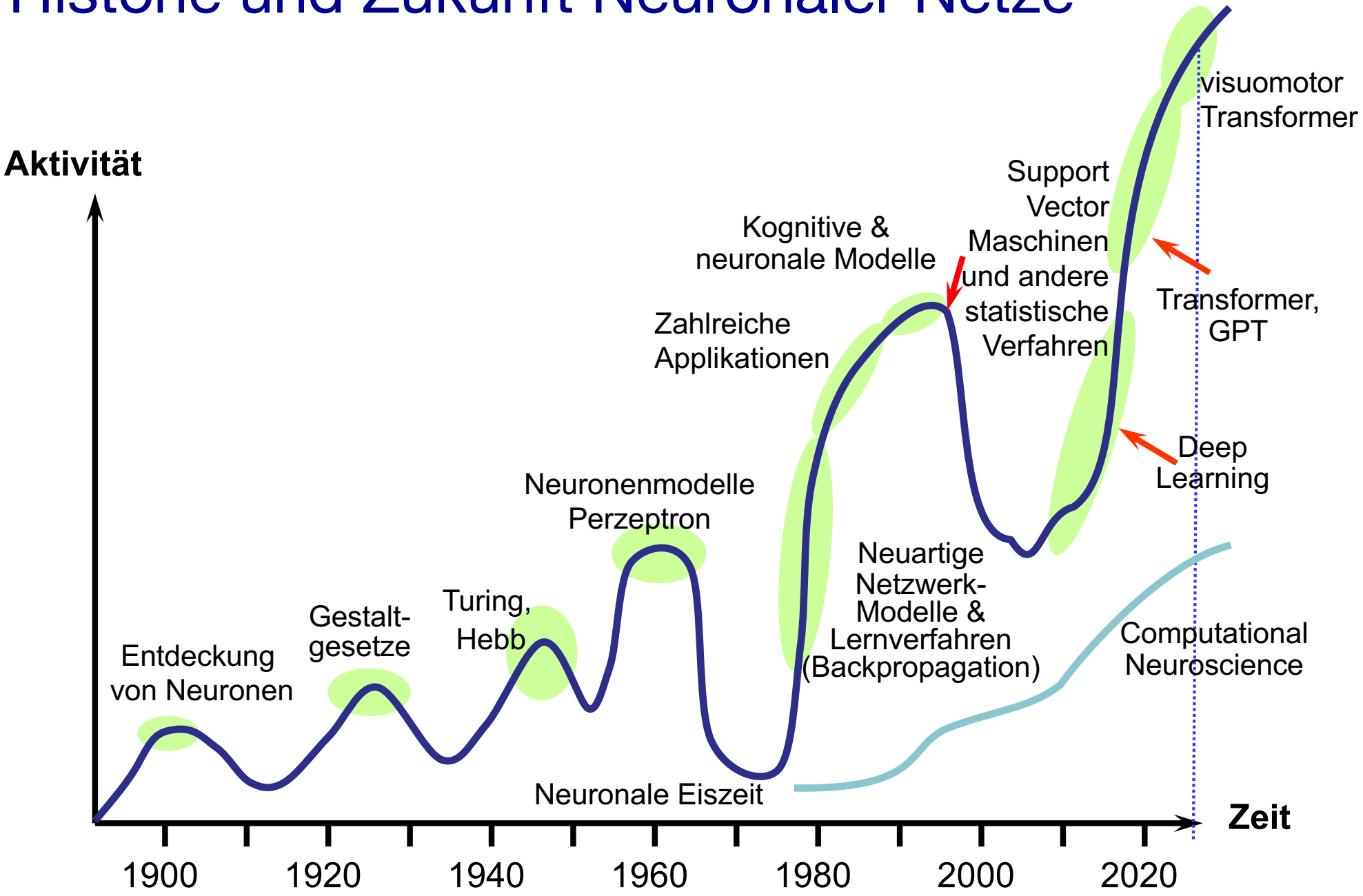


$$f(net) = \begin{cases} 1 & if \;\; net \geq 0.6 \\ 0 & if \;\; net < 0.6 \end{cases}$$

I_1	I_2	$0.5I_1 + 0.5I_2$	I_3	$\sum w_{ij} I_j$	o_1
0	0	0	0	0	0
0	1	0.5	0	1	1
1	0	0.5	0	1	1
1	1	1	1	0	0

o_1 ist dann gleich 1 wenn I_1 gleich 1 oder I_2 gleich 1 ist, aber nicht beide

Historie und Zukunft Neuronaler Netze



Lernen in künstlichen Neuronalen Netzen

Damit ein Neuronales Netz lernen kann, müssen einige Komponenten des Netzes flexibel sein.
So ist es z.B. möglich, dass im Laufe des Lernvorgangs (auch Training genannt) neue Neuronen entstehen, bestehende Neurone gelöscht oder Aktivierungsfunktionen modifiziert werden. In der Regel wird unter Lernen jedoch die Veränderung der Verbindungsstärke *w* verstanden.

Es wird zwischen drei Lernregimes unterschieden:

Unüberwachtes Lernen (unsupervised learning)

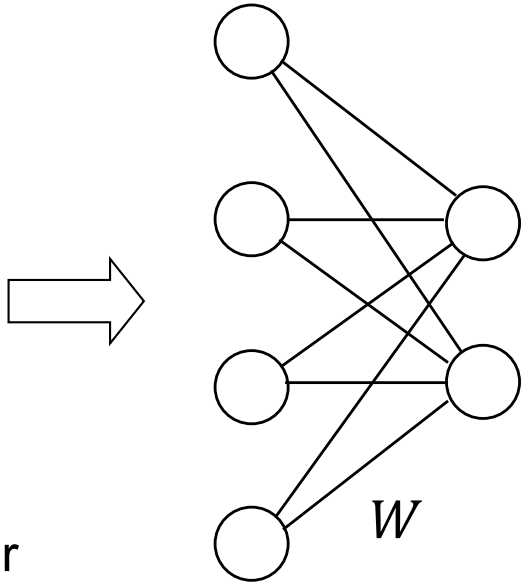
Überwachtes Lernen (supervised learning)

Verstärkendes Lernen (reinforcement learning)

Unüberwachtes Lernen

$$I = \begin{pmatrix} 0.1 \\ 0.4 \\ 0.2 \\ 0.9 \end{pmatrix}$$

Eingabemuster



$$o = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

Beim unüberwachten Lernen werden dem Netz lediglich mehrere Eingabemuster ***I*** präsentiert und das Lernverfahren nutzt statistische Eigenschaften in den Daten zur Generierung von Ausgabemustern.

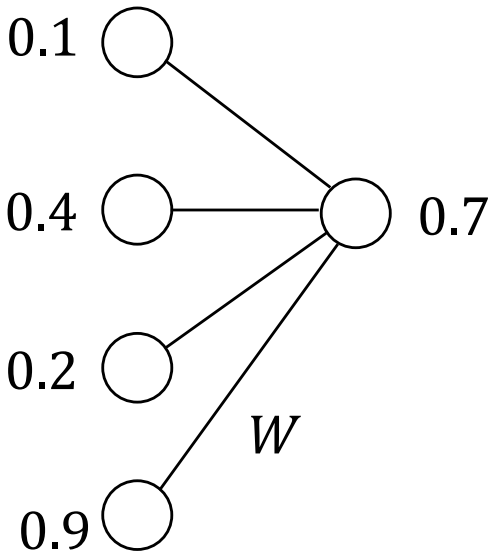
Hebb'sche Lernregel:

$$\Delta W = \eta \cdot I \cdot o$$

η Lernschrittweite

Unüberwachtes Lernen

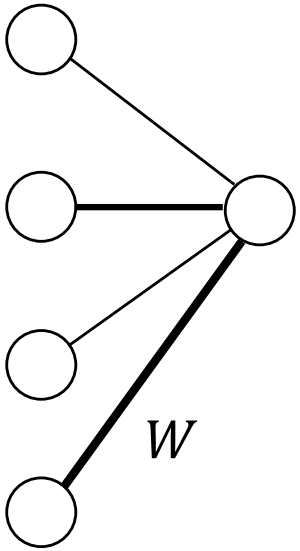
Vorher:



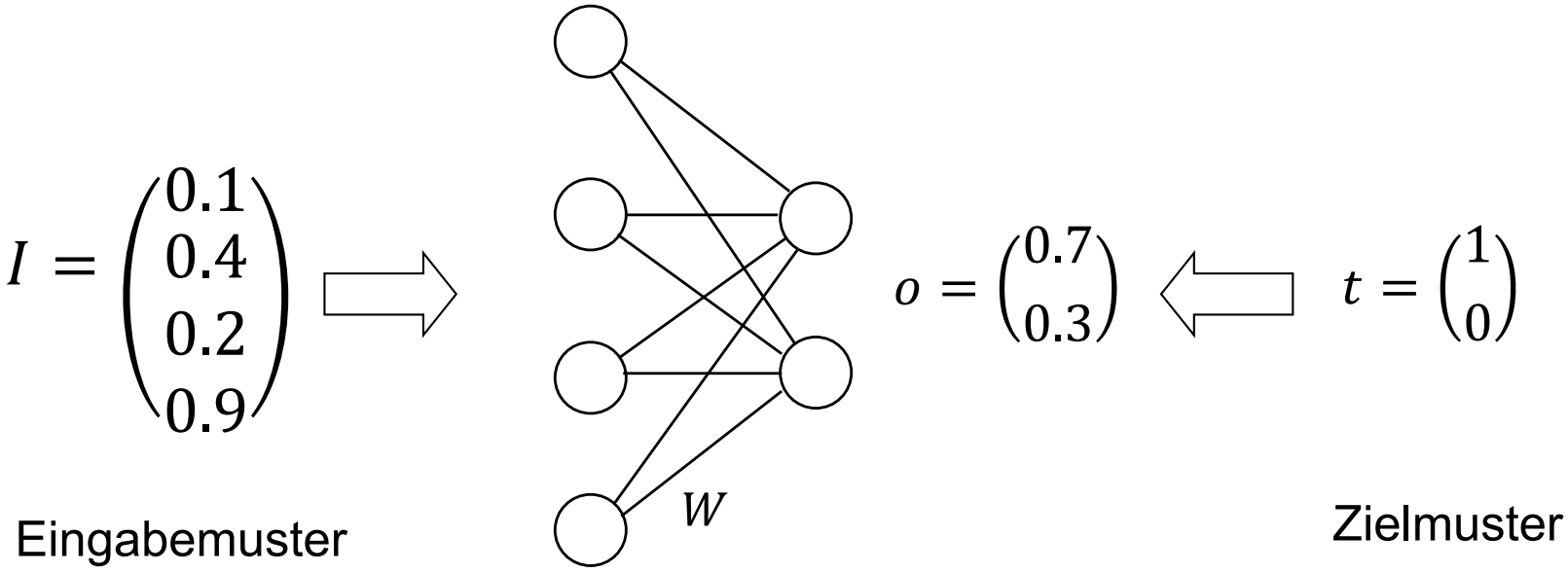
Hebb'sche Lernregel:

$$\Delta W = \eta \cdot I \cdot o$$

Nachher:



Überwachtes Lernen

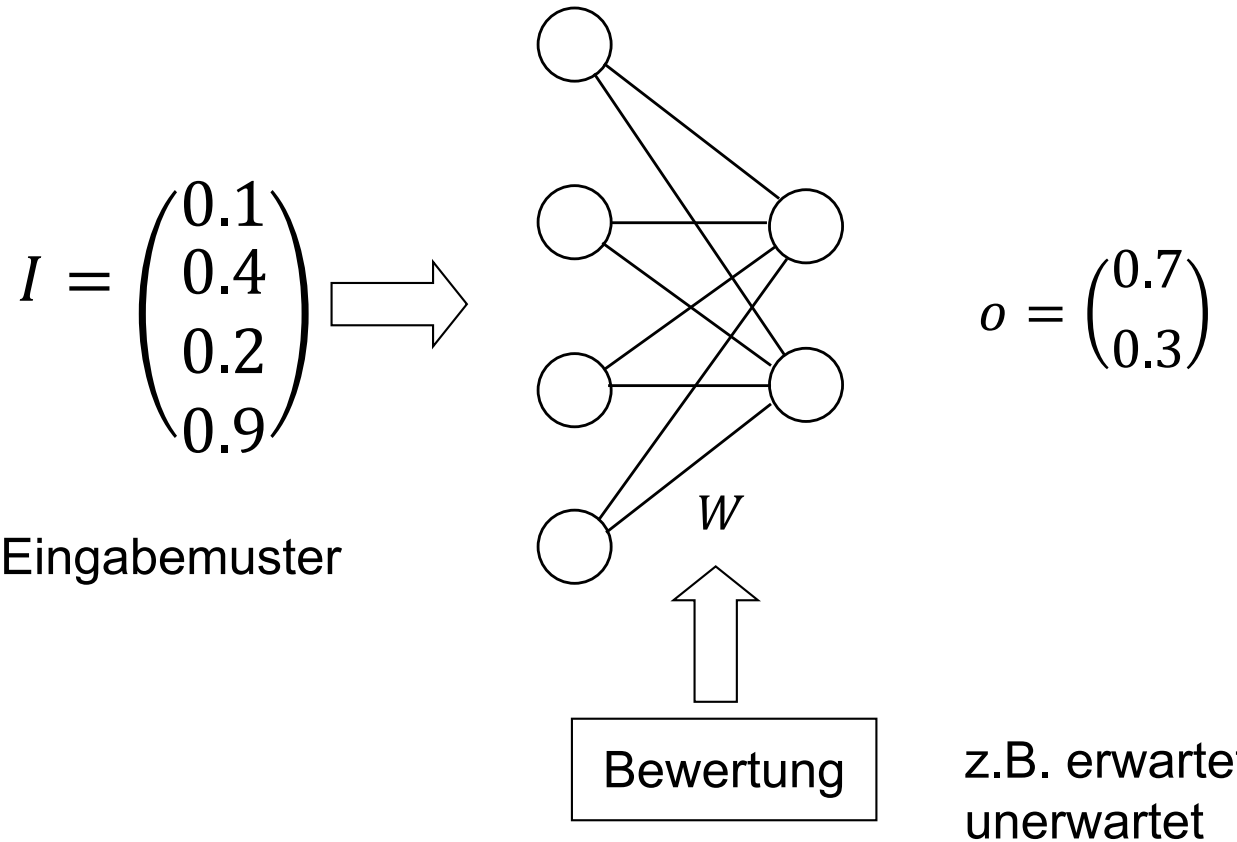


Beim überwachten Lernen werden dem Netz zu jedem Eingabemuster $\boldsymbol{I}$ ein Zielmuster **t** angegeben. Die Aufgabe des Lernverfahrens ist es, die Gewichte des Netzes so zu ändern, dass das Netz nach wiederholter Präsentation der Paare von Eingabe- und Ausgabemustern diese Assoziation selbständig vornehmen und dies auch für unbekannte, ähnliche Eingabemuster tun kann (Generalisierung).

Delta-Lernregel:

$$\Delta w_{ij} = \eta \cdot I_i (t_j - o_j)$$

Reinforcement Lernen (RL)

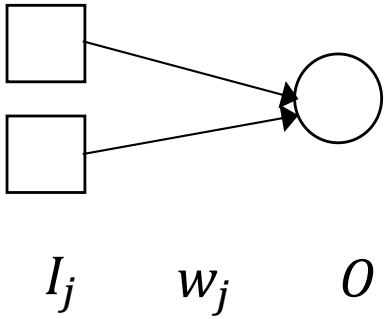


Beim verstärkenden Lernen (reinforcement learning) wird lediglich ein unspezifisches Signal z.B. eine Belohnung oder Bestrafung verwendet. Diese Bewertung wird vom Modell antizipiert und es lernt nur dann, wenn es einen Unterschied zwischen tatsächlicher und antizipierter Belohnung gibt.

Deep RL Netze verwenden die RL Optimierungsfunktion für überwachtes Lernen.

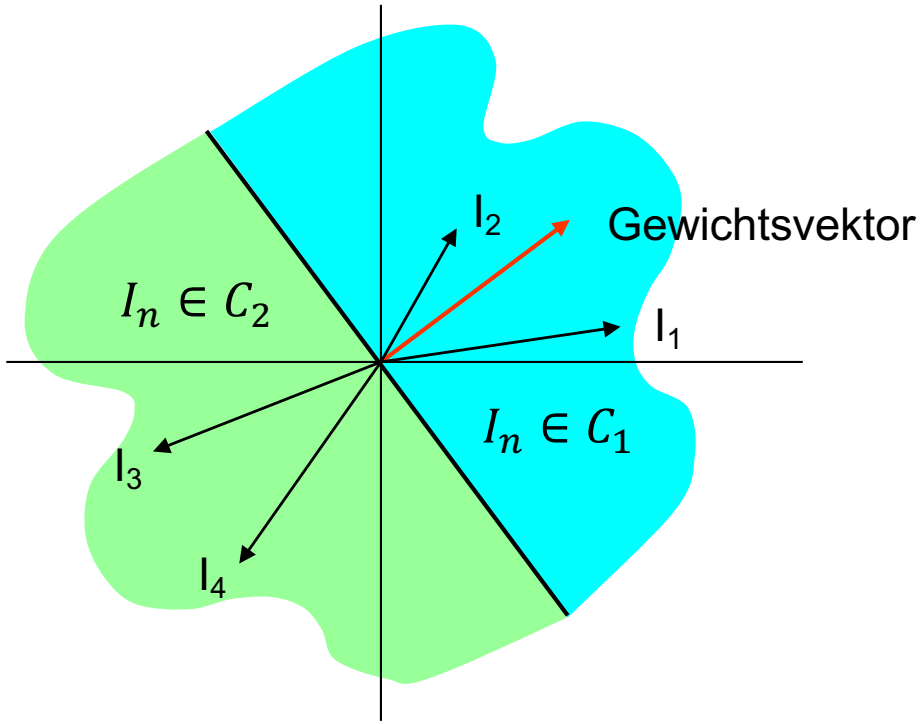
Lernen im Perzeptron

Beispiel: Muster mit 2 Merkmalen sollen 2 Klassen zugeordnet werden



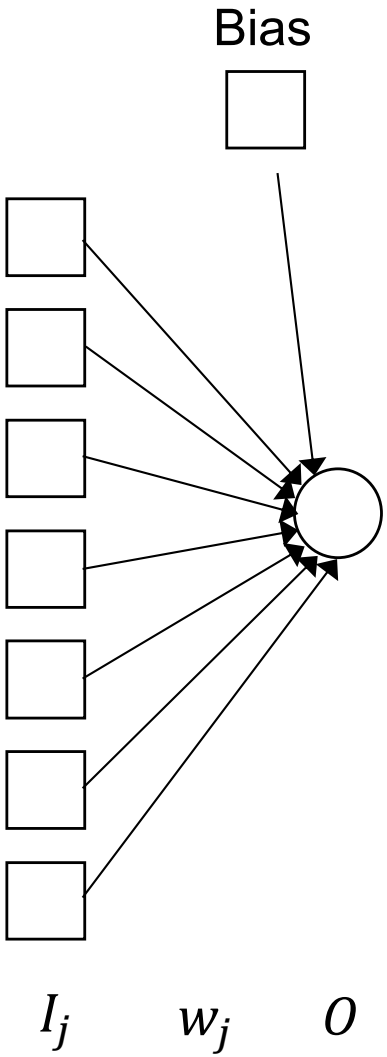
Zielvektor:

$t \in \{+1, -1\}$



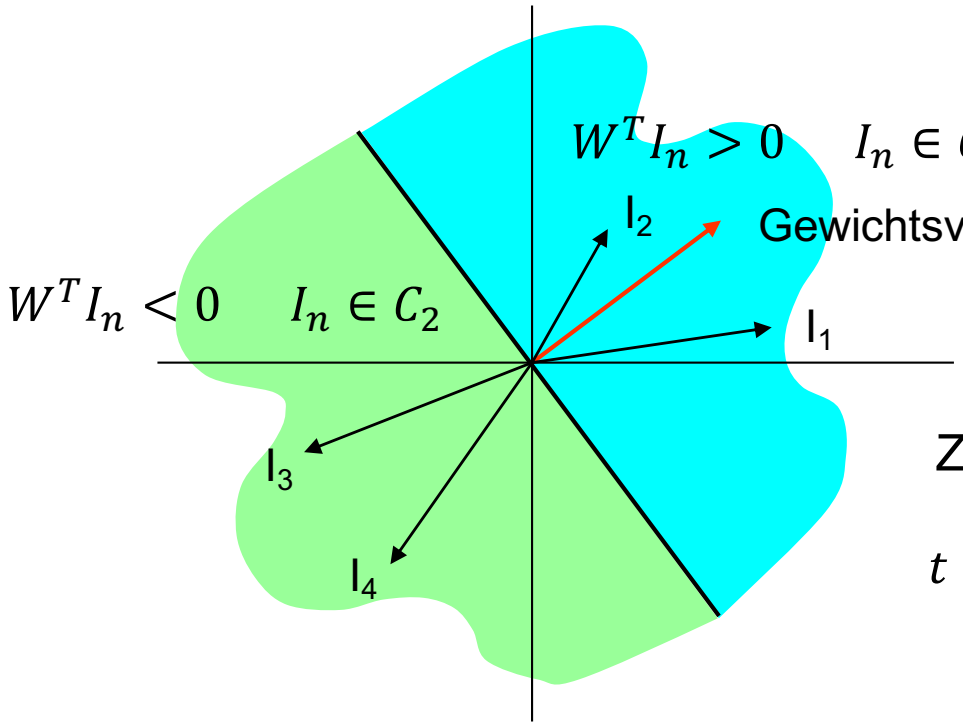
Ziel: Der Gewichtsvektor muss so eingestellt werden, dass die Beispiele richtig klassifiziert werden (die Ausgabe dem Zielvektor entspricht).

Lernen im Perzeptron



$$o = f(net) = f\left(\sum_j w_j I_j\right) = f(W^T I)$$

$$f(net) = \begin{cases} 1, & net \geq 0 \\ -1, & net < 0 \end{cases}$$



Zielvektor:

$t \in \{+1, -1\}$

Lernen im Perzeptron

$W^T I_n < 0 \quad I_n \in C_2$

$W^T I_n > 0 \quad I_n \in C_1$

Der Zielvektor für die Klassen ist:

$t \in \{+1, -1\}$

dann gilt: $W^T I_n t_n > 0$

Für das Lernen wird eine Fehlerfunktion benötigt.

Anstatt einfach die Anzahl der Fehler als Fehlerfunktion zu nehmen, kann man eine stückweise glatte Fehlerfunktion gewinnen, indem man für jedes falsch klassifizierte Muster die Entfernung des Musters im Gewichtsraum ansetzt:

$$E(w) = -\sum_{n \in Fehler} W^T I_n \cdot t_n$$

Lernen im Perzeptron

$$E(w) = -\sum_{n \in Fehler} W^T I_n \cdot t_n$$

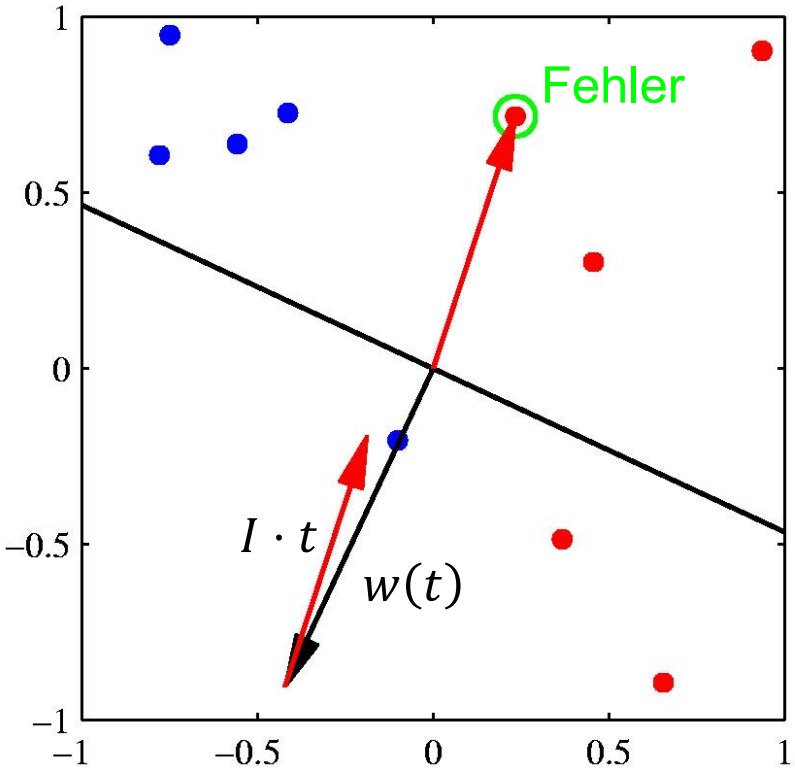
Nach dem stochastischen Gradientenabstieg erhält man:

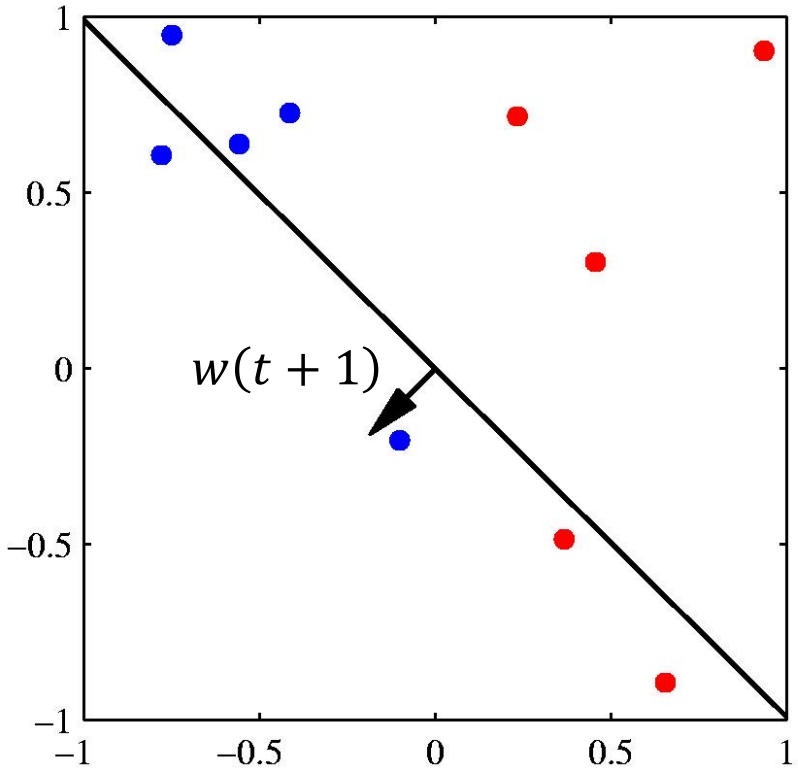
$$w(t+1) = w(t) - \eta \nabla E(w) = w(t) + \eta I_n \cdot t_n$$

Falls eine exakte Lösung existiert, garantiert diese Lernregel, dass diese in einer endlichen Anzahl an Lernschritten auch gefunden wird.

Allerdings muss das Problem linear separabel sein.

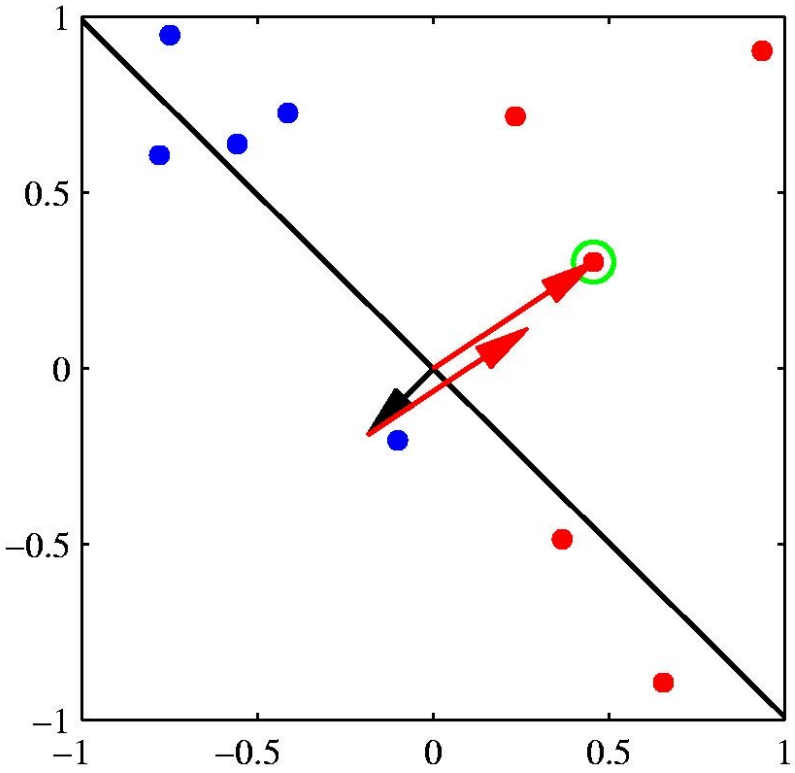
Lernen im Perzeptron

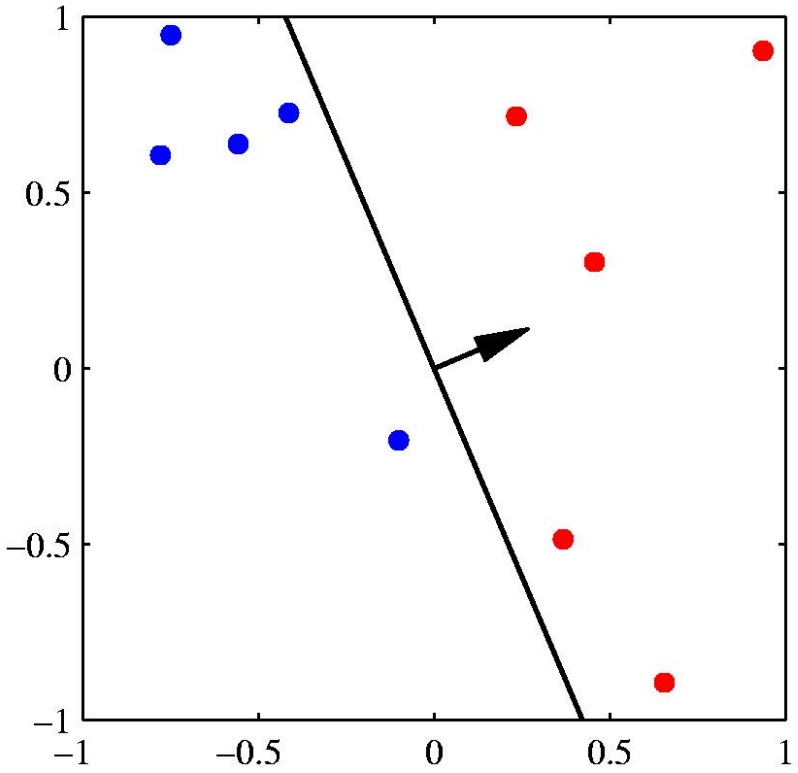




$$w(t+1) = w(t) - \eta \nabla E(w) = w(t) + \eta I_n \cdot t_n$$

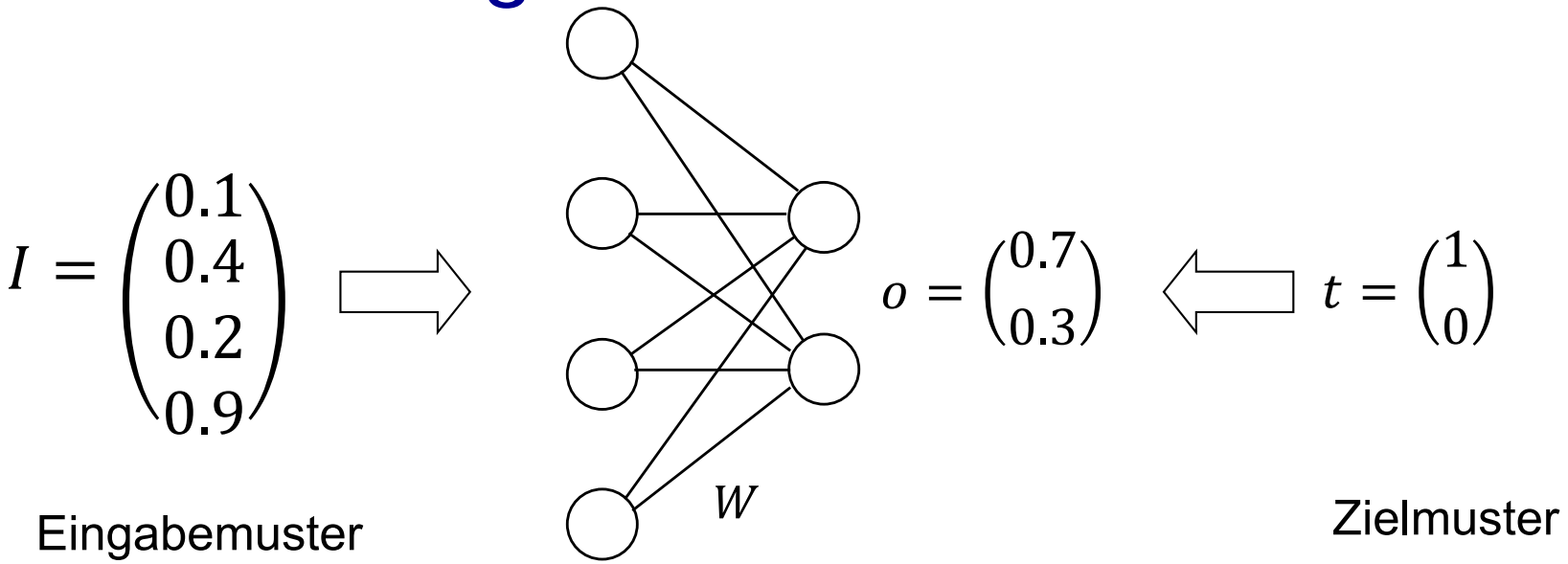
Lernen im Perzeptron





$$w(t+1) = w(t) - \eta \nabla E(w) = w(t) + \eta I_n \cdot t_n$$

Delta-Regel für einschichtige Netze mit linearen Aktivierungsfunktionen



Delta-Lernregel:

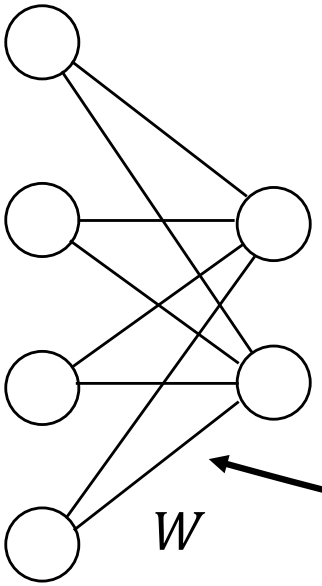
$$\Delta w_{ij} = \eta \cdot I_i \left(t_j - o_j\right) = \eta \cdot I_i \cdot \delta_j$$

Diese Lernregel, auch als Widrow/Hoff-Regel (1960) bekannt, ermöglicht es, Gewichtsänderungen so durchzuführen, dass ein bestimmter Eingabevektor mit einem gewünschten Ausgabevektor assoziiert wird. Allerdings ist sie nur für zweischichtige Netze definiert, da eine gewünschte Ausgabe nicht für Hidden-Units beschrieben wird.

Herleitung der Delta-Lernregel: Fehlerfunktion

Gegeben ist ein Input I_p aus der Menge aller Inputs und das dazugehörige Ziel $\mathbf{t}_p$:

$$I_p = \begin{pmatrix} 0.1 \\ 0.4 \\ 0.2 \\ 0.9 \end{pmatrix}$$



W

$$o_p = \begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix}$$

E_p

Fehler

$$t_p = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Ziel: Anpassung der Gewichte zur Minimierung des Fehlers

$$E_p = \frac{1}{2}(t_p - o_p)^2$$

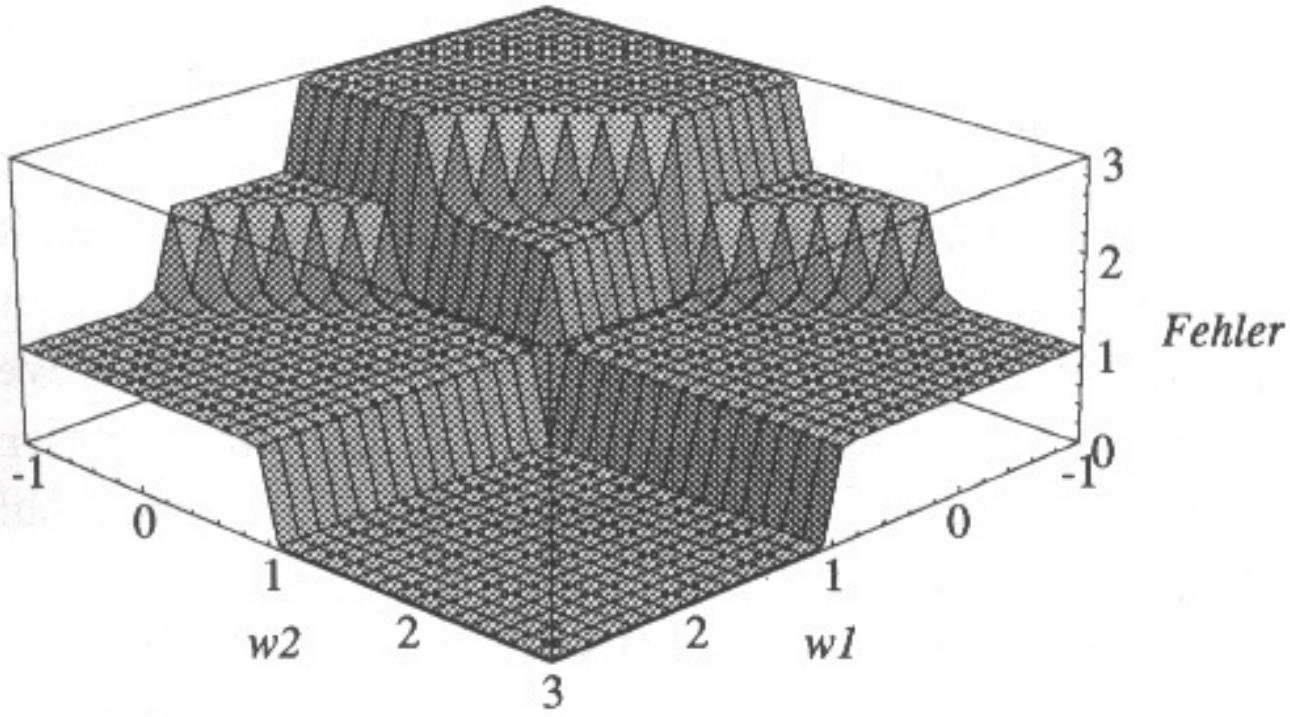
Quadrat des Euklid'schen Abstandes

Herleitung der Delta-Lernregel: Fehlerfunktion

Die Fehlerfunktion lässt sich einfach visualisieren, wenn das neuronale Netz lediglich zwei Gewichte besitzt.

Beispiel: OR Funktion

Variiere die Gewichte w1 und w2 und berechne für jedes (w1,w2)-Paar den Fehler E über alle Eingabemuster



Ziel: Finde das Minimum der Fehlerfunktion!

Herleitung der Delta-Lernregel

Bestimme zunächst eine Fehlerfunktion (Optimierungsfunktion):

$$E = \frac{1}{2}(t - o)^2$$

Da die relevanten unabhängigen Variablen lediglich die Gewichte w_i sind, wird der Fehler als Funktion der Gewichte definiert:

$$E = E(w_1, \dots, w_n)$$

Nutze zur Minimierung beispielsweise den Gradientenabstieg:

$$E(w^{neu}) = E(w^{alt} + \Delta w) \leq E(w^{alt})$$

Herleitung der Delta-Lernregel

Nach dem Gradientenabstiegsverfahren berechnet sich die Änderung eines jeden Gewichtes w_{ij} mit Hilfe der Ableitung der Fehlerfunktion nach den Gewichten:

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E(W)}{\partial w_{ij}} \qquad \text{wobei} \qquad E = \frac{1}{2}(t-o)^2$$

E ist zwar eine Funktion von w, *allerdings nur indirekt durch die Berechung der Aktivierung der Ausgabeneuronen* $o_j(w)$ jedes Neurons *j*. Die Berechung der Ableitung erfordert daher die Anwendung der Kettenregel (äußere mal innere Ableitung):

$$\frac{\partial E(W)}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}}$$

mit $E = \frac{1}{2}(t-o(W))^2$ folgt $\frac{\partial E}{\partial o_j} = -(t_j - o_j)$

Herleitung der Delta-Lernregel

Bei Verwendung einer linearen Aktivierungsfunktion f(x)=x in der Ausgabeschicht berechnet sich die Ausgabe als:

$$o_j = \sum_i I_i w_{ij}$$

so dass gilt:

$$\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial \sum_i I_i w_{ij}}{\partial w_{ij}} = I_i$$

Damit ergibt sich die Delta-Lernregel für einschichtige Netze mit linearer Aktivierungsfunktion zu:

$$\Delta w_{ij} = \eta \cdot I_i \left(t_j - o_j\right)$$

Herleitung der allgemeinen Delta-Lernregel

$$\Delta w_{ij} = -\eta \cdot \frac{\partial E(W)}{\partial w_{ij}}$$

wobei

$$E = \frac{1}{2}(t - o)^2$$

$$\frac{\partial E(W)}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

mit

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j} f(net_j) = f'(net_j)$$

$$\frac{\partial net_j}{\partial w_{ij}} = I_i$$

$$\boxed{\Delta w_{ij} = \eta \cdot I_i (t_j - o_j) f'(net_j)}$$

Backpropagation Lernregel

Die bisher dargestellten Lernregeln funktionieren nur bei Neuronalen Netzen ohne Hidden-Units.

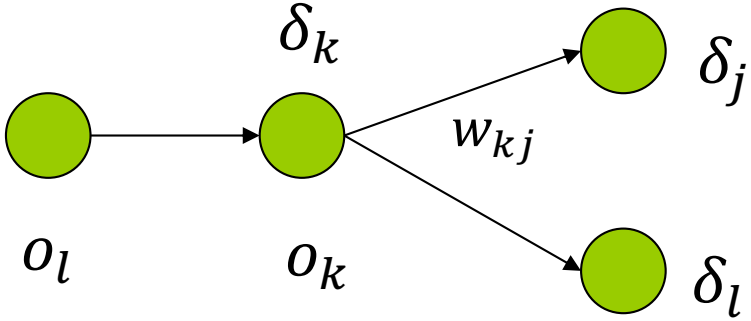
Das Backpropagation Verfahren stellt eine Rechenvorschrift dar, mit der die Gewichte zu den Hidden-Units modifiziert werden können. Entwickelt wurde das Verfahren bereits in den 70er Jahren (u. a. von Paul Werbos, 1974), allerdings geriet es zunächst für über ein Jahrzehnt in Vergessenheit.

Paul Werbos

Besonders bekannt wurde der Backpropagation Ansatz von Rumelhart, Hinton und Williams (1986).
Es ist allerdings sehr umstritten, ob diese Form des Lernens in Gehirn verwendet wird.

David Rumelhart

Herleitung der Backpropagation Lernregel



$$\frac{\partial E(W)}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

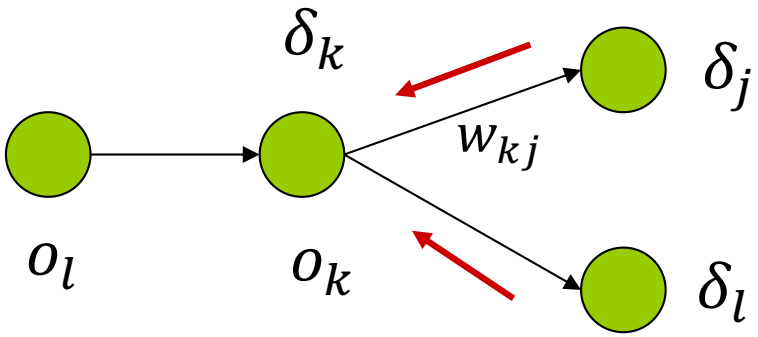
$$\delta_j = \frac{\partial E}{\partial net_j}$$

Um den Beitrag eines Hidden Neurons zum Fehler zu bestimmen, nutzen wir den Zusammenhang, dass sich eine Variation der Aktivierung net_k nur durch Änderungen der Aktivierungen net_j in der Fehlerfunktion bemerkbar macht.

$$\delta_k = \frac{\partial E}{\partial net_k} = \sum_j \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial net_k} = \sum_j \delta_j \frac{\partial w_{kj} o_k}{\partial net_k} = \sum_j \delta_j \frac{\partial w_{kj} f(net_k)}{\partial net_k}$$

$$\delta_k = f'(net_k)\sum_j w_{kj}\delta_j$$

Herleitung der Backpropagation Lernregel



$$\delta_k = f'(net_k) \sum_j w_{kj} \delta_j$$

Die Backpropagation Lernregel besagt also, dass wir den Fehler (Delta) eines Hidden Neurons bekommen, wenn wir die Fehler der Ausgabeneuronen rückwärts propagieren.

$$\Delta w_{lk} = -\eta \cdot o_l \delta_k$$

Die Backpropagation Lernregel kann für beliebige Netze verwendet werden und gilt nicht nur für das Multi-Layer Perzeptron (MLP).

Multi-Layer Perzeptron mit Backpropagation

Algorithmus:

Lege eine Eingabe an und berechne die Ausgabe.

Bestimme δ_j für alle Ausgabeneuronen.

$$\delta_j = o_j - t_j$$

Rückwärtspropagierung der δ_j , um die δ-Werte der Hidden-Neuronen zu bestimmen:

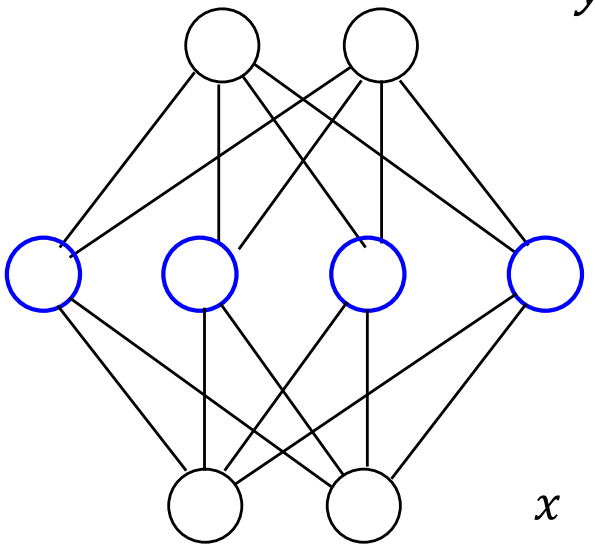
$$\delta_k = f'(net_k) \sum_j w_{jk} \delta_j$$

Bestimmung der Gewichtsänderung unter Verwendung der δ-Werte:

$$\Delta w_{ij} = -\eta \cdot \delta_j I_i$$

Radiale Basis-Funktion Netze

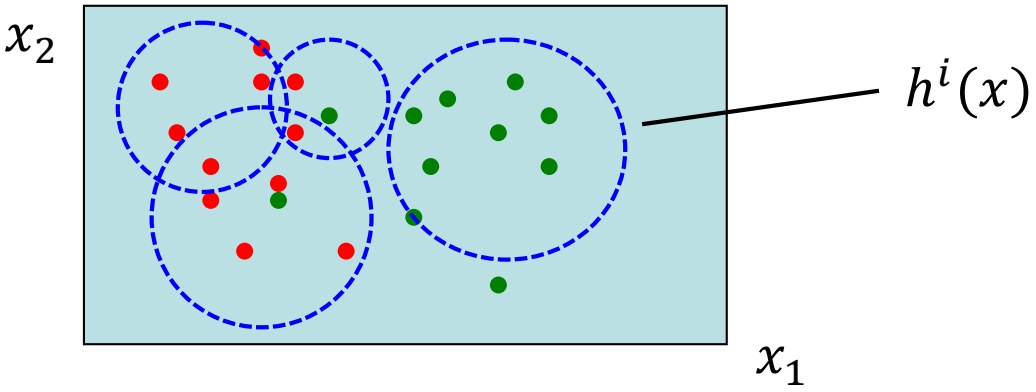
Als Variante zum MLP eignen sich auch Radiale Basis Funktion Netze. Hier wird als Maß der Aktivierung der Neurone in der ersten Schicht der Abstand verwendet.



$$y_i(x) = \sum_i w_{ij} h^i(x)$$

Während das MLP eine rein verteilte Repräsentation lernt, besitzen RBF-Netze eher eine lokale Repräsentation.

$$h^i(x) = exp\left(-\frac{\|x - c_i\|}{2\sigma_i^2}\right)$$



Oft werden die Zentren c_j nach der Verteilung der Daten initialisiert. Sie können allerdings auch nach dem Backpropagation Lernalgorithmus gelernt werden. Es gibt auch Verfahren, die Knoten neu einfügen oder löschen.

Die Gewichte der zweiten Schicht werden typischerweise nach der Delta Lernregel bestimmt.

Verwendungsweise Neuronaler Netze

Die Anwendung von Neuronalen Netzen ist im Grunde sehr einfach, allerdings kann sich die Suche nach dem “besten” Netz und Lernverfahren als langwierig gestalten, da der Anwender sehr viele Freiheitsgrade hat.

- Anzahl der Neuronen
- Anzahl der Schichten
- Wahl der Nichtlinearitäten
- Lernstrategien (Batch, on-line, Momentum-Term, weight-decay)

Neben dem Gradientenabstiegsverfahren wurden wiele weitere Lernverfahren entwickelt:

- Quickprop (nur für kleine NN)
- Conjugate Gradient Descent (nur für kleine NN)

- Adaptive Moment Estimation (Adam), nutzt ein Momentum und adaptive Lernraten je Parameter

Verwendungsweise Neuronaler Netze

Es soll ein Neuronales Netz für eine bestimmte Aufgabe konstruiert werden.
Dazu geht man in folgenden Schritten vor:

Konstruktionsphase

1. Lege Anzahl und Typ der benötigten Einheiten fest.
2. Lege die Verbindungen zwischen den Einheiten fest.
3. Initialisiere die Gewichte an den Kanten.

Trainingsphase

4. Wähle eine Menge von Beispielen (Paare von Ein- und Ausgaben).
5. Lege fest, in welcher Reihenfolge die Eingaben der Beispiele in das Netz eingegeben werden sollen.
6. Bestimme für jedes Beispiel die Differenz zwischen gewünschter und tatsächlicher Ausgabe.
7. Passe die Gewichte in geeigneter Weise an $\Rightarrow$ Lernalgorithmus.

Verwendungsweise Neuronaler Netze

Da Neuronale Netze dazu neigen, die Trainingsdaten „auswendig“ zu lernen, wurden verschiedene Kriterien entwickelt dieses „overfitting“ zu vermeiden. Eine einfache Strategie ist die folgende:

Abbruchkriterium

1. Bestimme den Fehler auf einem Validierungs-Datensatz (VD).
2. Beende das Training, wenn der Fehler auf dem VD nicht weiter fällt.

3. Ein Vergleich der Netze kann auf Basis eines unbekannten Testdatensatzes erfolgen (Leave-k-out).

Bewertung Neuronaler Netze

Um den Fehler des neuronalen Netzwerkes zu bestimmen gibt es verschiedene Maße und Anforderungen welche die Validierungs und Testdaten erfüllen sollten.

Ziel der Bewertung eines Algorithmus ist es zu bestimmen, inwieweit er korrekte Ausgaben für neue, unbekannte Beispiele erzeugen kann, d.h. wie gut er von den Trainingsdaten auf das zu lösende Problem **generalisiert**. Da es typischerweise unmöglich ist, alle Eingaben zu testen muss eine geeignete Auswahl getroffen werden, um den **Generalisierungsfehler** abschätzen zu können. Der so ermittelte Fehler wird **empirischer Fehler** genannt.

Anforderungen an Validierungs- und Testdaten

1. Sie dürfen nicht bereits in den Trainingsdaten enthalten sein.
2. Sie müssen das Problem ausreichend abdecken (Probenahmefehler), z.B. indem eine ausreichend große Menge zufällig gewählter Beispiele genutzt wird.
3. Testdaten dürfen nicht in die Modellentwicklung einfließen (Meta-Overfitting).

Bewertung Neuronaler Netze

Ein Verfahren, um den Generalisierungsfehler abzuschätzen ist das **Kreuzvalidierungsverfahren**.

Das einfachste Vorgehen ist die **hold-out cross-validation**. Hierbei werden die Daten entsprechend eines gewählten Verhältnisses in Trainings-, Validierungs-, und Testdaten geteilt (z.B. Training 60%, Validierung 20%, Test 20%).

Optimal wäre es, alle möglichen Teilungen mit p Elementen zu betrachten: **leave-p-out cross-validation**. Diese benötigt allerdings C_p^n Auswertungen, was meist nicht berechenbar ist. Eine Approximation hiervon stellt **k-fold cross-validation** dar, bei der die Daten in k feste Blöcke geteilt werden, anstatt alle Teilmengen der Größe p zu betrachten.

Vorgehen k-fold cross-validation

1. Teile die Daten in k Blöcke.
2. Trainiere das Netzwerk auf k-1 Blöcken und teste auf dem verbliebenen Block.
3. Wiederhole dieses Vorgehen so das jeder Block einmal zum Testen genutzt wird.
4. Bilde den Durchschnitt der k ermittelten empirischen Fehler.

Bewertung Neuronaler Netze

Je nach Aufgabe des Neuronalen Netzes kann man seine Güte mit unterschiedlichen Maßen bewerten. Grundsätzlich kann zwischen Maßen für Regression und Klassifikation unterscheiden werden.

Maße für Regression:

1. Fehlerfunktion des Lernalgorithmus (Loss)
2. Mittlere Quadratische Abweichung (MSE)
3. Mittlere Absolute Abweichung (MAE)

Bewertung Neuronaler Netze

Die Fehlermaße für die Klassifikation basieren auf der Wahrheitsmatrix (Konfusionmatrix).

Gesamtzahl = P+N	Vorhersage Positiv (PP)	Vorhersage Negativ (PN)
Positiv (P)	Echt Positiv (TP)	Falsch Negativ (FN)
Negativ (N)	Falsch Positiv (FP)	Echt Negativ (TN)

https://cs230.stanford.edu/section/8/

Bewertung Neuronaler Netze

Maße für Klassifikation

1. Einfacher Fehler $\epsilon = \frac{\#\text{Fehler}}{\#\text{Beispiele}}$

2. Accuracy $\text{acc} = 1 - \epsilon = \frac{TP+TN}{TP+FP+TN+FN}$
   (Korrektklassifikationsrate, Treffergenauigkeit)

3. Precision (Spezifität) $P = \frac{TP}{TP+FP}$
   Hohe Precision bedeutet, eine niedrige Anzahl an fälschlich korrekt klassifizierten positiven Beispielen.

4. Recall (Sensitivität) $R = \frac{TP}{TP+FN}$
   Hoher Recall bedeutet, eine niedrige Anzahl an fälschlicherweise nicht erkannten positiven Beispielen.

5. F1-Score $\text{F1} = \frac{2 \cdot P \cdot R}{P+R}$
   (harmonischer Mittelwert aus P und R)

https://cs230.stanford.edu/section/8/

Bewertung Neuronaler Netze

Beispiel: In einer Studie für ein neue Methode zur Erkennung von Typ-2 Diabetes nehmen 500 Probanden teil. Die Methode erkennt bei 250 Probanden Diabetes. Eigentlich sind es aber 100 Probanden mit Diabetes und bei 10 wurde der vorhandene Diabetes nicht erkannt.

Gesamtzahl = P+N	Vorhersage Positiv (PP)	Vorhersage Negativ (PN)
Positiv (P)	TP = 100	FN = 10
Negativ (N)	FP = 150	TN = 240

Precision $$P = \frac{TP}{TP+FP} = \frac{100}{100+150} = 0{,}4$$

- Bei vielen fälschlich erkannte Diabetes führt zu einer niedrigen Precision.

Recall $$R = \frac{TP}{TP+FN} = \frac{100}{100+10} = 0{,}91$$

- Wenige nicht erkannte Diabetes Erkrankungen führen zu einem hohen Recall.

F1-Score $$\mathrm{F1} = \frac{2 \cdot P \cdot R}{P+R} = \frac{2 \cdot 0{,}4 \cdot 0{,}91}{0{,}4+0{,}91} = 0{,}56$$

- Eine niedrige Precision führt zu einem niedrigem F1-Score.

Bewertung Neuronaler Netze

Da der Fehler von der Statistik der Daten abhängt, spiegelt er diese wieder. Das kann ungewollte Folgen haben. Unter anderem wenn die Häufigkeit der Klassen im Datensatz stark **unausgewogen** ist.

Man kann Bewertungsmaße nutzen, die Unausgewogenheit zu berücksichtigen.

Maße für unausgewogene Daten:

1. Macro-averaging
   (Mittelt das jeweilige Maß welches pro Klasse erzielt wurden. Wobei jede Klasse gleich eingeht.)
2. Balanced Accurracy (bACC)
   (Mittel von Presition und Recall) $BA = \frac{P+R}{2}$
3. F1-Score

Weitere Maße

1. Average Precision (AP) und Area Under the ROC (AUC)
   (Mittlere Precision für jeden Schwellwert. Entspricht der Fläche unter der ROC Kurve.)
2. Mean Average Precision (mAP)
   (Pro Klasse gemittelte AP.)

https://cs230.stanford.edu/section/8/

Unüberwachtes Lernen

Was ist unüberwachtes Lernen?

- Lernen von Daten ohne Kenntnis von Zielwerten oder Belohnungen
- Nur Muster und Beziehungen innerhalb der Eingabedaten dienen als Grundlage für das Lernen
- Es kann verwendet werden, um die zugrunde liegende Struktur und die Merkmale von Daten zu identifizieren
- Typische Anwendungen sind:
  - Clustering: Auffinden von Gruppen ähnlicher Daten
  - Dimensionalitätsreduzierung: Abbildung auf einen weniger dimensionalen Raum unter Beibehaltung der wichtigsten Informationen
  - Anomalieerkennung: Auffinden ungewöhnlicher Muster in Daten
  - Datengenerierung: Erzeugen neuer Daten aus der Datenstatistik

Unüberwachtes Lernen

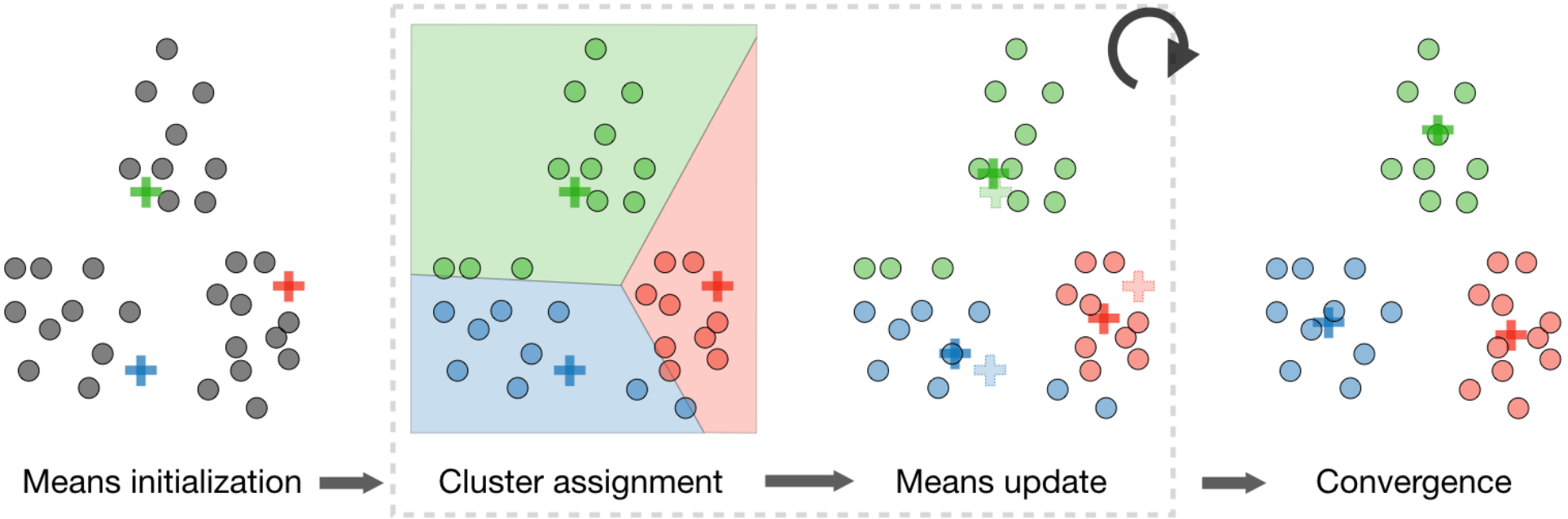
Was sind typische Algorithmen des unüberwachten Lernens?

- k-means
- Self-organizing maps (SOM)
- Hierarchical clustering
- Principle component analyzis (PCA)
- Independent component analyzis (ICA)
- Bolzmann machine (BM)
- Autoencoder (AE)
- Generative pre-trained transformers (GPT)
- Hebbsches Lernen

Unüberwachtes Lernen

Typische Algorithmen des unüberwachten Lernens:

- k-means

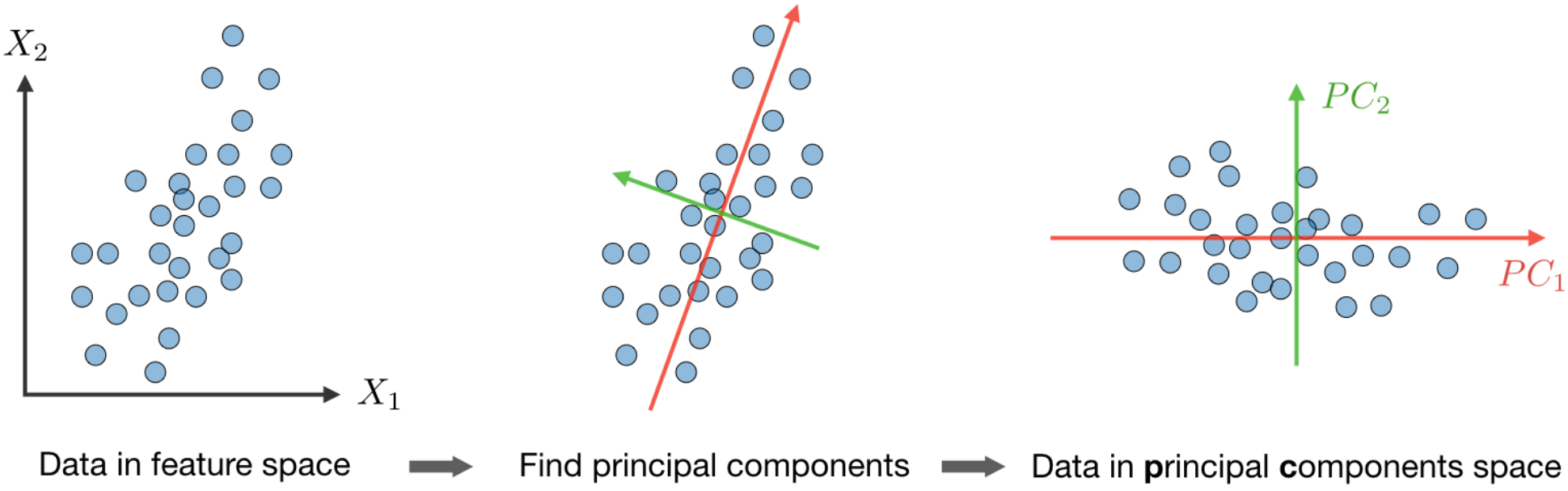


https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning

Unüberwachtes Lernen

Typische Algorithmen des unüberwachten Lernens:

- Principle component analyzis (PCA)



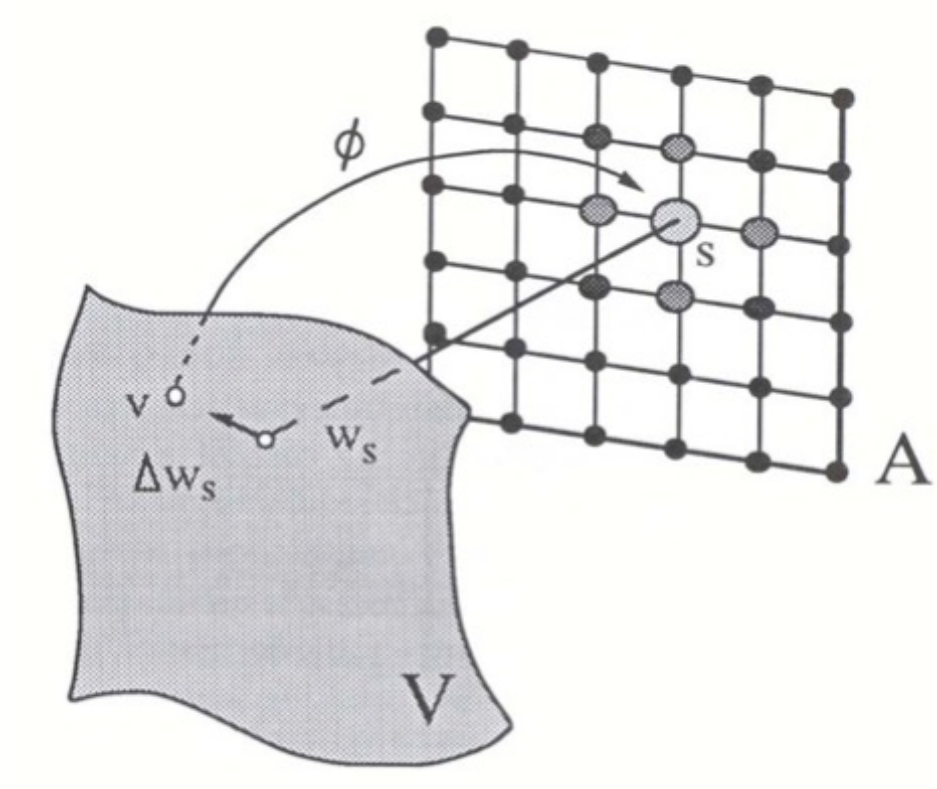
https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-unsupervised-learning

Unüberwachtes Lernen

Teuvo Kohonen

Typische Algorithmen des unüberwachten Lernens:

- Selbstorganisierende Merkmalskarten



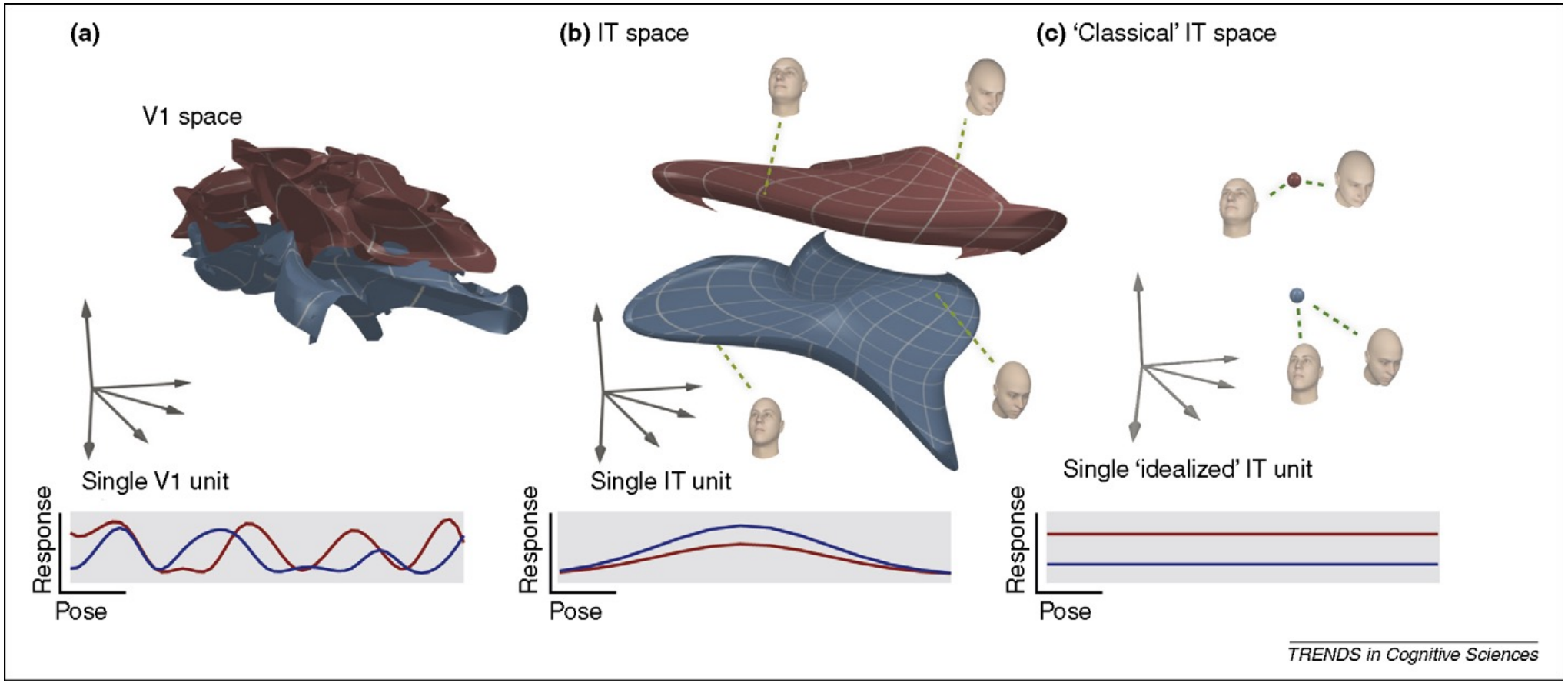
Aus: H. Ritter, T. Martinetz, and K. Schulten. Neuronale Netze - eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke. Addison-Wesley, Bonn, 1990.

Idee:
Die Neuronen stehen durch eine Nachbarschaftsfunktion miteinander in Beziehung.

Das Neuron mit dem stärksten Input sowie seine Nachbarn dürfen den Input repräsentieren, indem ihre Gewichtsvektoren adaptiert werden.

Unüberwachtes Lernen

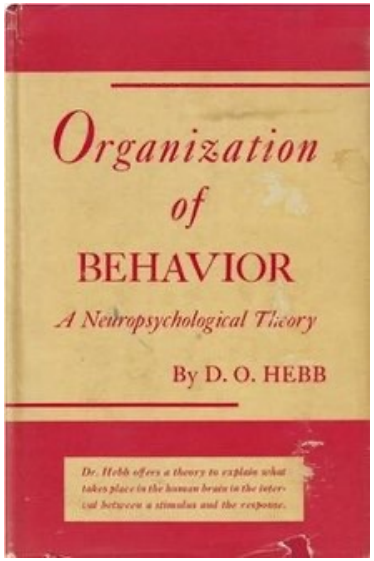
Unüberwachtes Lernen könnte helfen, „bessere“ Repräsentationen zu erhalten. Eine Hypothese über die Wahrnehmung ist, dass neuronale Systeme die neuronalen Repräsentationen verschiedener Elemente entwirren müssen, um sie leicht unterscheiden zu können.



DiCarlo, J. J., & Cox, D. D. (2007). Untangling invariant object recognition. *Trends in Cognitive Sciences*, *11*(8), 333–341.

Was ist Hebb‘sches Lernen?

Donald O. Hebb



The Organization of Behavior (1949)

Donald Hebb postuliert 1949 in seinem Buch *The Organization of Behavior*, wie lang anhaltende zelluläre Veränderungen im Nervensystem hervorgerufen werden:

> *"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.“*

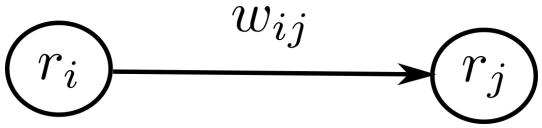
Häufig vereinfacht zu:

> *Neurons wire together if they fire together.*

Was ist Hebb‘sches Lernen?

Auf diesem Prinzip aufbauend kann eine grundlegende Gleichung formuliert werden, bei der die Gewichtsänderung proportional zum Produkt der Aktivierungswerte erfolgt:

$$\Delta w_{ij} = \eta \cdot r_i \, r_j$$



Zwei verbundene Neuronen.

- Hebb‘sches lernen erfordert keine anderen Informationen als die Aktivitäten, wie z.B. Labels oder Fehlersignale: es ist ein unüberwachtes Lernverfahren.
- Hebb‘sches Lernen bezeichnet keine konkrete Lernregel, sondern ist ein Postulat über das grundlegende Prinzip des biologischen Lernens.

→ Es lernt häufig vorkommende Merkmale der Eingabedaten. Es wird auch als **korrelationsbasierte Lernregel** bezeichnet.

Eigenschaften Hebb‘scher Lernregeln

Eine sinnvolle Hebb‘sche Lernregel muss mehrere Kriterien erfüllen:

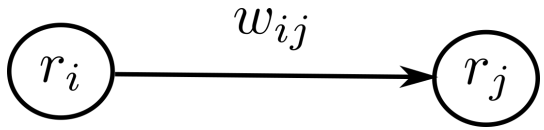
1. **Lokalität:** Die Gewichtsänderung sollte nur von der Aktivität der beiden Neuronen und dem synaptischen Gewicht selbst abhängen.
$$\Delta w_{ij} = F(w_{ij}; r_i; r_j)$$
2. **Kooperativität:** Das Hebb’sche Postulat besagt, dass Zelle **A** *"takes part in firing it",* was bedeutet, dass beide Neuronen aktiv sein müssen, um eine Gewichtszunahme zu bewirken.
3. **Synaptische Hemmung:** In der Regel ist ein Mechanismus zur Verringerung der Gewichte erforderlich.
4. **Begrenztheit:** Gewichte sollten in einem bestimmten Bereich begrenzt sein.
5. **Wettbewerb:** Gewichte wachsen auf Kosten anderer Gewichte. Dies kann durch eine Normalisierung des Gewichtsvektors erreicht werden.
6. **Langzeitstabilität:** Bei adaptiven Systemen muss darauf geachtet werden, dass zuvor gelernte Informationen nicht verloren gehen. Dies wird als "Stabilitäts-Plastizitäts-Dilemma" bezeichnet.

Gerstner, W. and Kistler, W. M. (2002). Mathematical formulations of Hebbian learning. Biological Cybernetics, 87(5–6), 404–415

Implementierungen Hebb‘scher Lernregeln

Einfachste Form: Lernen hängt nur von der präsynaptischen r_i und postsynaptischen r_j Feuerrate und einer Lernrate η ab (korrelationsbasiertes Lernprinzip):

$$\Delta w_{ij} = \eta \cdot r_i \, r_j$$



Two connected neurons.

Wenn die postsynaptische Aktivität über mehrere Eingangssynapsen berechnet wird:

$$r_j = \sum_i r_i w_{ij} = \boldsymbol{r}^T \times \boldsymbol{w}_j$$

dann akkumuliert die Lernregel die **Autokorrelationsmatrix** $\mathbf{Q}$ der Eingabevektoren $\boldsymbol{r}$:

$$\Delta \boldsymbol{w}_j = \eta \boldsymbol{r} r_j = \eta \boldsymbol{r} \times \boldsymbol{r}^T \times \boldsymbol{w}_j = \eta \, \mathbf{Q} \times \boldsymbol{w}_j$$

wobei $\mathbf{Q}$ den Mittelwert des Vektorprodukts der verschiedenen Eingaben bezeichnet:

$$\mathbf{Q} = \mathbb{E}_r(\boldsymbol{r} \times \boldsymbol{r}^T)$$

→ Folglich erhalten häufig gemeinsam auftretende Eingabewerte eine starke Gewichtung.

Dayan, P., & Abbott, L. F. (2005). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press Cambridge, MA, USA. S. 286, Abschnitt 8.2.

Implementierungen Hebb‘scher Lernregeln

Kovarianzbasiertes Hebb’sches Lernen

Kovarianz zwischen zwei Zufallsvariablen X und Y ist definiert durch:

$$cov(X,Y) = \mathbb{E}\big(X - \mathbb{E}(X)\big)\mathbb{E}\big(Y - \mathbb{E}(Y)\big)$$
$$= \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$$

Ihre Korrelation ist definiert durch:

$$corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

- Eine Eigenschaft der Kovarianz ist, dass sie für unabhängige Variablen Null ist und für abhängige Variablen positiv ist.
- Allerdings misst sie nur die lineare Unabhängigkeit und ignoriert Abhängigkeiten höherer Ordnung.

→ Sie ist nützlich, um sinnvolle Gewichte zwischen statistisch abhängigen Eingaben zu lernen.

Implementierungen Hebb‘scher Lernregeln

Kovarianzbasiertes Hebb’sches Lernen

$$cov(X,Y) = \mathbb{E}\big(X - \mathbb{E}(X)\big)\mathbb{E}\big(Y - \mathbb{E}(Y)\big) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y)$$

Bei der folgenden Formulierung ist die Gewichtsänderung relativ zur Kovarianz der Aktivitäten der verbundenen Neuronen:

$$\Delta w_{ij} = \eta(r_i - \theta_i)(r_j - \theta_j)$$

Wobei θ_i und θ_j die Schätzungen der Erwartungswerte von prä- und postsynaptischer Aktivität sind. θ kann z.B. als exponentiell geglätteter Durchschnitt berechnet werden:

$$\theta_i = \alpha r_i + (1-\alpha)\theta_i$$
$$\theta_j = \beta r_j + (1-\beta)\theta_j$$

- Beim kovarianzbasierten Lernen können die Gewichte sowohl zunehmen als auch abnehmen.

Varianten:

$$\Delta w_{ij} = \eta r_i(r_j - \theta_j) = \eta(r_i r_j - \theta_j r_i)$$

$$\Delta w_{ij} = \eta(r_i - \theta_i) r_j = \eta(r_i r_j - r_j \theta_i)$$

Die Lernregel akkumuliert die Kovarianzmatrix $\mathbf{C}$ der Eingabevektoren $\boldsymbol{r}$:

$$\Delta \boldsymbol{w}_j = \eta(\boldsymbol{r} - \boldsymbol{\theta})\big(r_j - \theta_j\big) = \eta(\boldsymbol{r} - \mathbb{E}_r(\boldsymbol{r}))\times(\boldsymbol{r}^T - \mathbb{E}_r(\boldsymbol{r}^T))\times\boldsymbol{w}_j = \eta\,\mathbf{C}\times\boldsymbol{w}_j$$

Dayan, P., & Abbott, L. F. (2005). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press Cambridge, MA, USA. S. 288, Abschnitt 8.2.

Implementierungen Hebb‘scher Lernregeln

Begrenztheit – Normalisierung

- Da die Korrelation von Eingabe und Ausgabe durch das Lernen zunimmt, würde das Gewicht ohne Grenzen wachsen.
- Im Falle von antikorrelierten Neuronen könnten die Gewichte beim kovarianzbasierten Lernen auch negativ werden.

Es gibt mehrere Möglichkeiten, die Gewichte zu beschränken:

1. Harte Grenzen

$$w_{min} \leq w_{ij} \leq w_{max}$$

2. Weiche Grenzen

$$\Delta w_{ij} = \eta r_i r_j (w_{ij} - w_{min})(w_{max} - w_{ij})$$

3. Normalisierung der Gewichtsvektorlänge (Oja, 1982)

$$\Delta w_{ij} = \eta (r_i r_j - \alpha r_j^2 w_{ij})$$

4. Feuerraten basierte Schwellwertanpassung (Bienenstock, Cooper, & Monroe, 1982; Interator, Cooper, 1992)

$$\Delta w_{ij} = \eta r_i \big(r_j - \theta\big) r_j \, f'(net)$$
$$\theta = \mathbb{E}\big(r_j^2\big)$$

Implementierungen Hebb‘scher Lernregeln

Oja Lernregel

Normalisierung der Länge von Gewichtsvektoren (Oja, 1982)

$$\Delta w_{ij} = \eta(r_i r_j - \alpha r_j^2 w_{ij})$$

Erkki Oja fand eine Formulierung, welche die Länge eines Gewichtsvektors durch eine lokale Operation normalisiert und damit das erste Kriterium für Hebb‘sches Lernen erfüllt.

$\alpha r_j^2 w_{ij}$ ist ein Regularisierungsterm: wenn das Gewicht w_{ij} oder die postsynaptische Aktivität r_j zu hoch werden, dann übersteigt dieser Term den „Hebb’schen” Teil $r_i r_j$ und reduziert das Gewicht.

Oja hat gezeigt, dass mit dieser Gleichung die Norm des Gewichtsvektors gegen einen konstanten Wert konvergiert, der durch den Parameter α bestimmt wird:

$$\|\boldsymbol{w}\|^2 = \frac{1}{\alpha}$$

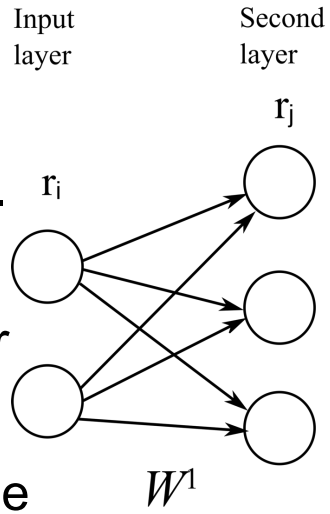
Um dies zu ermitteln, wurde die Beziehung zwischen Eingabe und Ausgabe genutzt $r_j = \boldsymbol{r^T} \times \boldsymbol{w}$ zusammen mit einer Taylor-Erweiterung über $\boldsymbol{w}$.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. Journal of Mathematical Biology, 15(3), 267–273. Dayan, P., & Abbott, L. F. (2005). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press Cambridge, MA, USA. S. 290, Abschnitt 8.2.

Neuronale Netze mit Hebb‘schem Lernen

Perzeptron

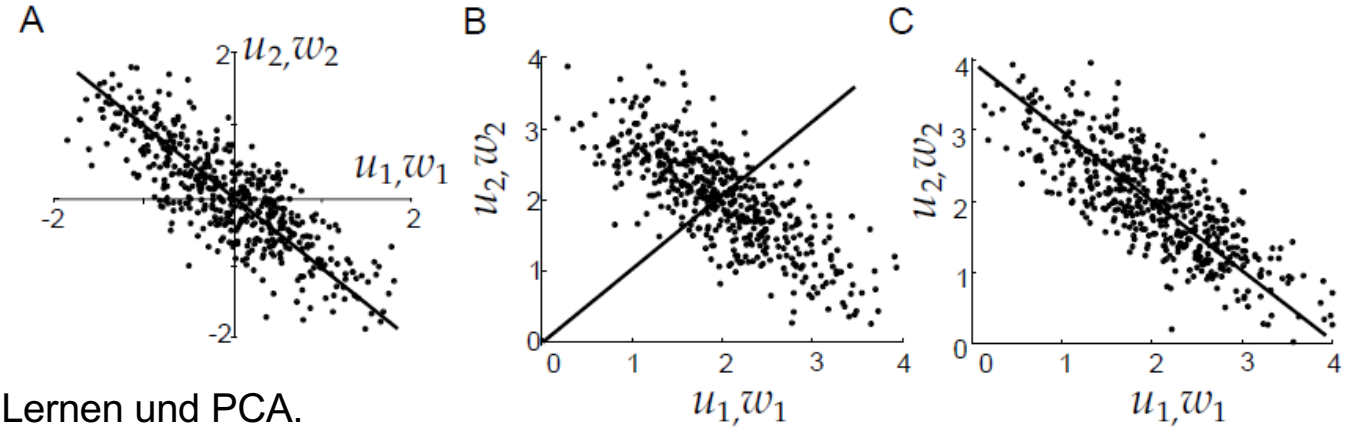
Was lernt eine Schicht mehrerer Neuronen mittels Hebb’schen Lernens?



- Erkki Oja (1982) zeigte, dass seine Lernregel für lineare Neuronen gegen die **erste Hauptkomponente** der Eingabedaten konvergiert.

Eine Hauptkomponente ist ein Eigenvektor der Kovarianzmatrix der Eingabedaten. Die erste Hauptkomponente ist der Eigenvektor mit der größten Varianz, der den höchsten Eigenwert hat.

- Allerdings erscheint ein solches Netzwerk nicht sehr nützlich, da alle Neuronen nur die erste Hauptkomponente lernen würden.

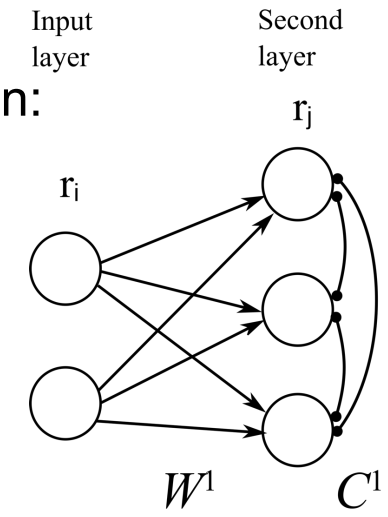


Unüberwachtes Lernen und PCA.

Dayan, P., & Abbott, L. F. (2005). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press Cambridge, MA, USA. S. 297, Abschnitt 8.3.

Neuronale Netze mit Hebb‘schem Lernen

Wettbewerb – Perzeptron



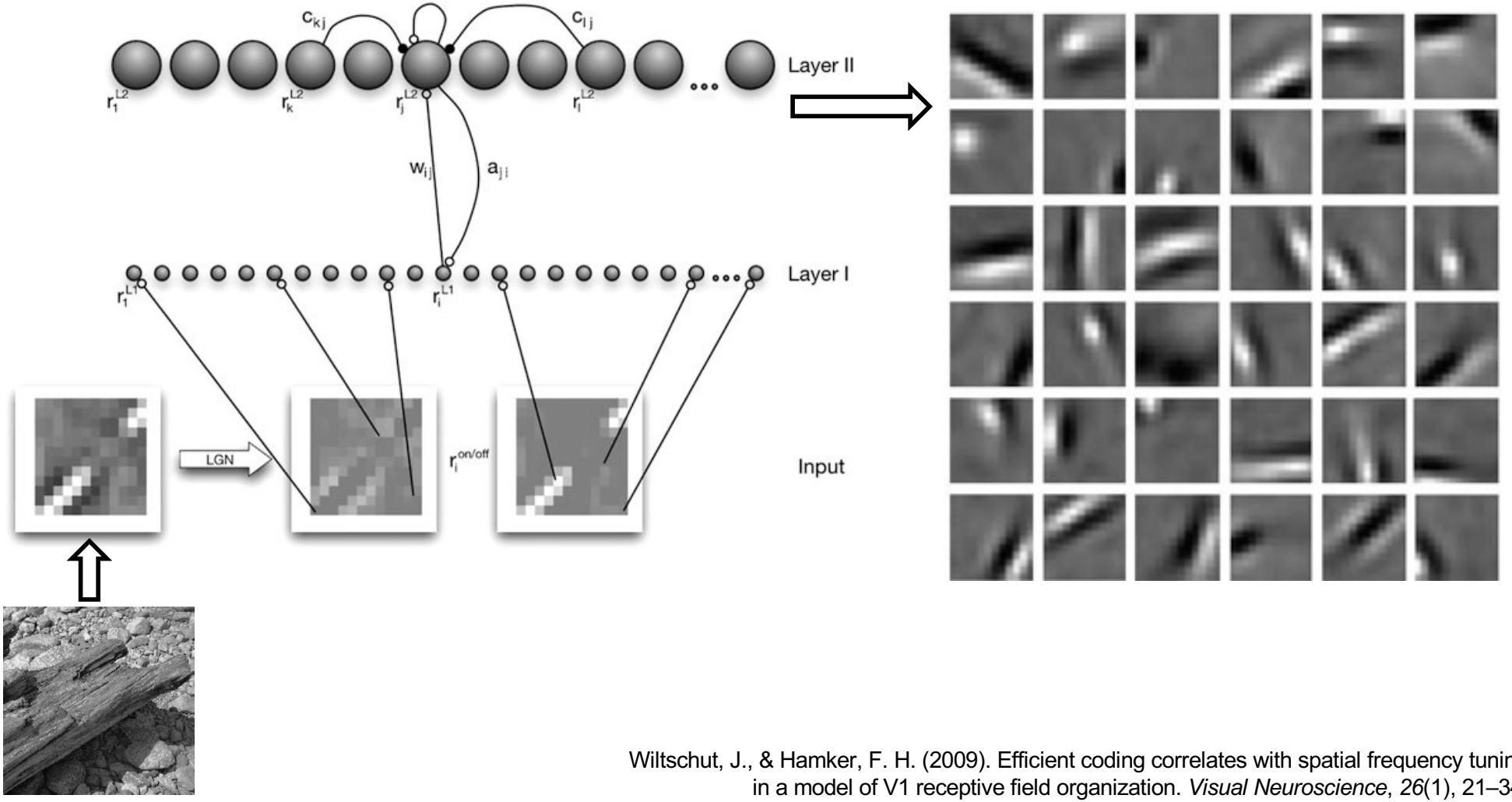
Es gibt mehrere Methoden zur Differenzierung der Neuronenantworten:

1. Winner-take-all Wettbewerb
   - Nur das Neuron mit der höchsten Antwort wird zum Lernen ausgewählt.
   - In der Praxis wird oft k-winner-take-all verwendet, wobei die k stärksten Neuronen lernen.
2. Ein rekurrenter Schaltkreis, welcher ein Wettbewerbssignal sendet
   - Die Neuronen konkurrieren mit ihren Nachbarn darum aktiv zu werden und zu lernen.
   - Im Gehirn geschieht dies nicht direkt, sondern über eine besondere Art von Neuronen, die so genannten **inhibitorische Neuronen**.
   - Inhibitorische Neuronen bilden nur Synapsen, die die Aktivität des postsynaptischen Neurons reduzieren (Dale's Gesetz).
   - Inhibition kann auf unterschiedliche Art und Weise implementiert werden.

Neuronale Netze mit Hebb‘schem Lernen

Was lernt ein solches Netz?

Beim Training mit Bildern natürlicher Szenen lernen die Neuronen, auf kantenähnliche Merkmale zu reagieren.



Wiltschut, J., & Hamker, F. H. (2009). Efficient coding correlates with spatial frequency tuning in a model of V1 receptive field organization. *Visual Neuroscience*, *26*(1), 21–34.

Hebb’sches Lernen

Zusammenfassung

- Hebb’sches Lernen postuliert Eigenschaften des biologischen Lernens.
- Algorithmen des Hebb'schen Lernens müssen folgende Eigenschaften erfüllen: Lokalität, Kooperativität, synaptische Hemmung, Begrenztheit, Wettbewerb und langfristige Stabilität.
- Hebb'sches Lernen nutzt die Statistik des Inputs und lernt häufige Muster. Wie zum Beispiel die erste Hauptkomponente.
- Hebb'sches Lernen erfordert einen Mechanismus des Wettbewerbs, um eine Differenzierung zu erreichen.
- Rekurrente inhibitorische Verbindungen erzeugen Wettbewerb, indem sie ähnliche Aktivitäten der Neuronen hemmen. Dadurch werden Abhängigkeiten reduziert und der neuronale Code wird effizienter in Bezug auf die enthaltenen Informationen.
- Ein neuronales Netz, welches auf natürlichen Bildern mit Wettbewerb trainiert wird, lernt die grundlegenden Komponenten dieser Bilder. Seine Ausgabe kann zur Objektklassifizierung verwendet werden.

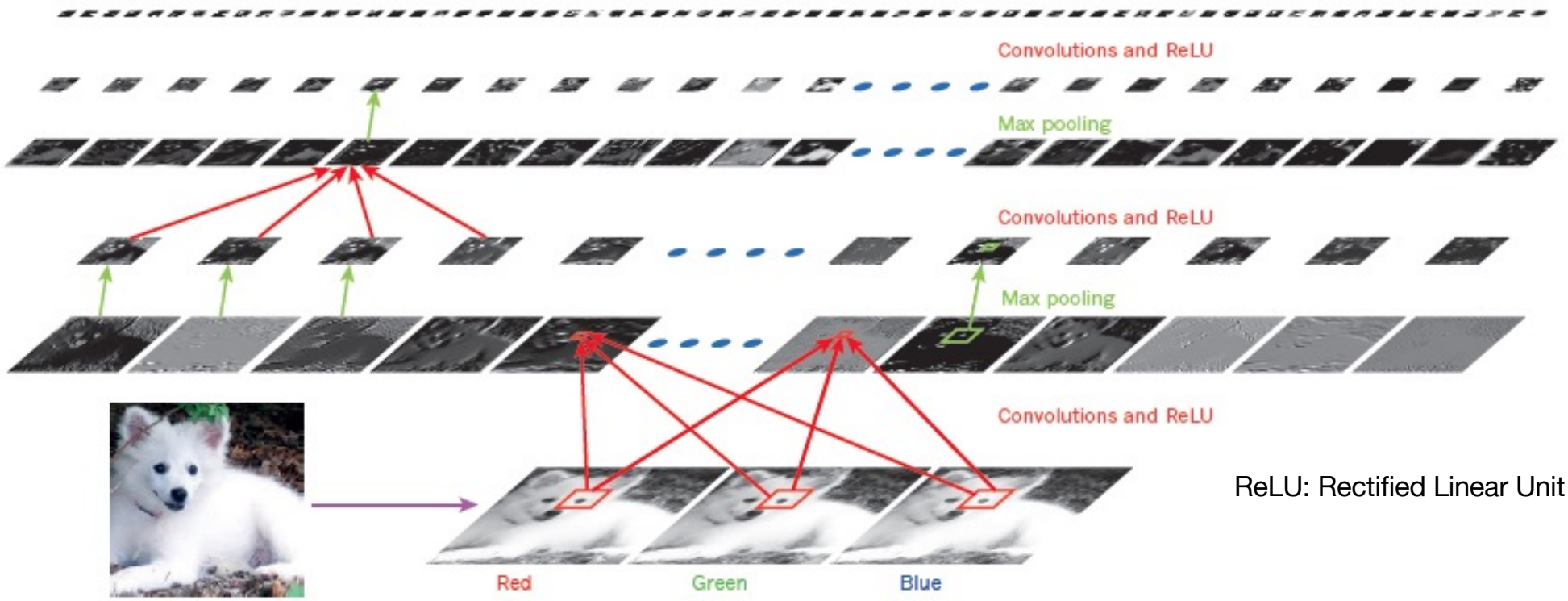
Deep Learning

- Deep learning bedeutet Lernen innerhalb einer großen Anzahl von Schichten (layers).
- Deep learning wird oft für Klassifikations-Aufgaben verwendet, wobei höhere Schichten Aspekte der Eingabe betonen, die für die Diskrimination wichtig sind.
- Wichtig ist, dass die Merkmale nicht vom Menschen bestimmt, sondern gelernt werden.
- Deep learning kann in drei Vorgehensweisen unterteilt werden:
  - Unüberwachtes Lernen in generativen Modellen und Autoencodern
  - Überwachtes Lernen mit stochastischem Gradientenabstieg
  - Hybride Modelle, die zunächst unüberwacht lernen und dann überwacht.

Deep Learning

- Deep learning wurde durch verschiedene Entdeckungen und “Tricks” möglich:
  - „Rectified linear units“ und nicht sigmoide Funktionen
  - Dropout Regularisierung (zufällig Neuronen deaktiviert) gegen Overfitting
  - Deformation von Trainingsdaten, um größere Datensätze zu erzeugen
  - Große Datensätze mit „Label“
- Deep Learning Methoden haben die meisten Wettbewerbe zur Klassifikation von Sprach- und Bilddaten gewonnen.
- Große Technologie Firmen wie Google, Facebook, Microsoft, IBM, Twitter, Adobe und Start-Ups investieren in Deep Learning, inzwischen auch die deutsche Auto-Industrie.

Deep Learning mit Convolutional Networks



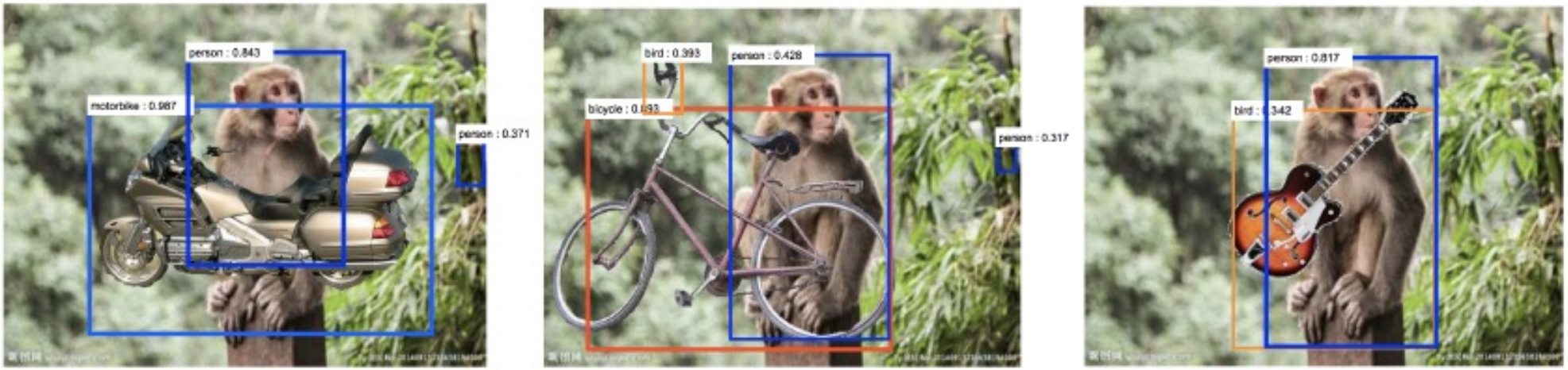
ReLU: Rectified Linear Unit

Aus: Yann LeCun, Yoshua Bengio & Geoffrey Hinton, *Deep learning*. Nature, 2015

Convolutional Netze verwenden einen Kernel, der unabhängig vom Ort ist und lokale Merkmale extrahiert.
Tiefe Neuronale Netze müssen anhand von sehr großen Datenbanken trainiert werden.

Deep Learning mit Convolutional Networks

Aufgrund der hohen Anzahl freier Parameter und der vergleichsweise wenigen Trainingsdaten sind tiefe Neuronale Netze anfällig.



Zusätzliche Objekte erzeugen einen anderen Kontext.

Links: Das Motorrad macht aus dem Affen einen Menschen.
Mitte: Auch das Fahrrad macht aus dem Affen einen Menschen und der Urwald verändert die Klassifikation des Lenkers in einen Vogel.
Rechts: Die Gitarre macht aus dem Affen einen Menschen und und der Urwald macht aus der Gitarre einen Vogel.

Yuille, AL, Liu, C (2018) Deep Nets: What have they ever done for Vision?
arXiv: https://arxiv.org/abs/1805.04025

Spiking (pulscodierte) Neurone

Warum Spiking Neuronale Netze (SNNs)?

Obwohl KNNs ursprünglich von der Funktion des Nervensystems inspriert waren, dominierte in der weiteren Entwicklung eher der praktische Nutzen die Forschung.

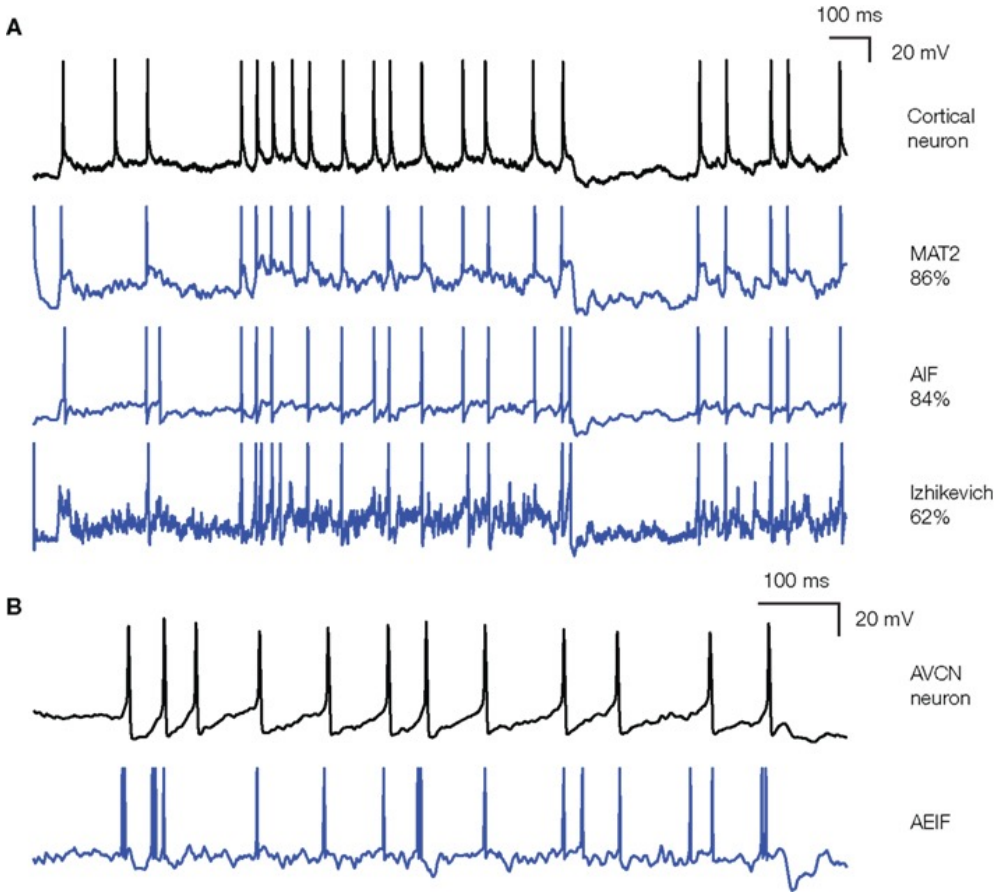
Ein sichtbarer Unterschied von biologischen zu künstl. NN ist folgender:

- Biologische Neurone tranferieren Signale über Pulse (Aktionspotentiale).

Chips könnten dieses Nutzen, um sehr energieeffizient zu operieren → **neuromorphic hardware**.

Spiking Neurone

Biologische Neurons kommunizieren mittels Spikes



- Zwei wichtige Dimensionen des Transfers von Information sind:
  - Instantane **Frequenz** oder **Feuerrate,** d.h. Anzahl von Spikes pro Sekunde (Hz).
  - Das **Timing** der Spikes.
- Spikes lassen sich als binäre Ereignisse (0 or 1) auffassen.
- **Spiking Neurone** repräsentieren spike timing, but ignorieren Details von Aktionspotentialen.

Rossant, C., Goodman, D. F. M., Fontaine, B., Platkiewicz, J., Magnusson, A. K., & Brette, R. (2011). Fitting Neuron Models to Spike Trains. *Frontiers in Neuroscience*, *5*.

Spiking Neurone

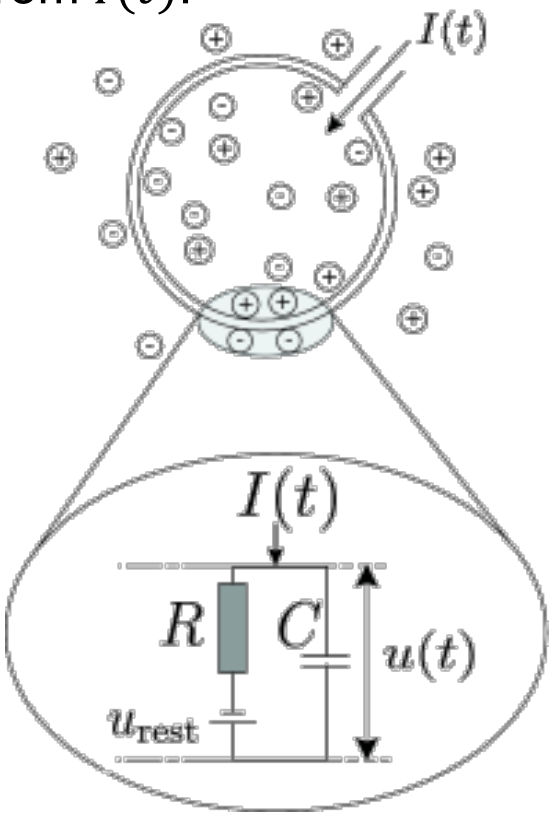
Leaky integrate-and-fire (LIF)

Das **Membranpotential** $v(t)$ integriert den Eingabestrom $I(t)$:

$$C\frac{dv(t)}{dt} = -g_L(v(t) - V_L) + I(t)$$

- C ist die Membran-Kapazität, g_L die leak Leitfähigkeit und V_L das Ruhepotential.
- Ohne Eingabestrom ($I = 0$), ist das Membranpotential gleich des Ruhepotentials.
- Wenn das Membranpotential die Schwelle V_T erreicht, dann sendet das Neuron einen Spike und das Membranpotential wird auf einen Reset-Potential V_R für eine kurze Refraktärzeit t_{ref} gesetzt.

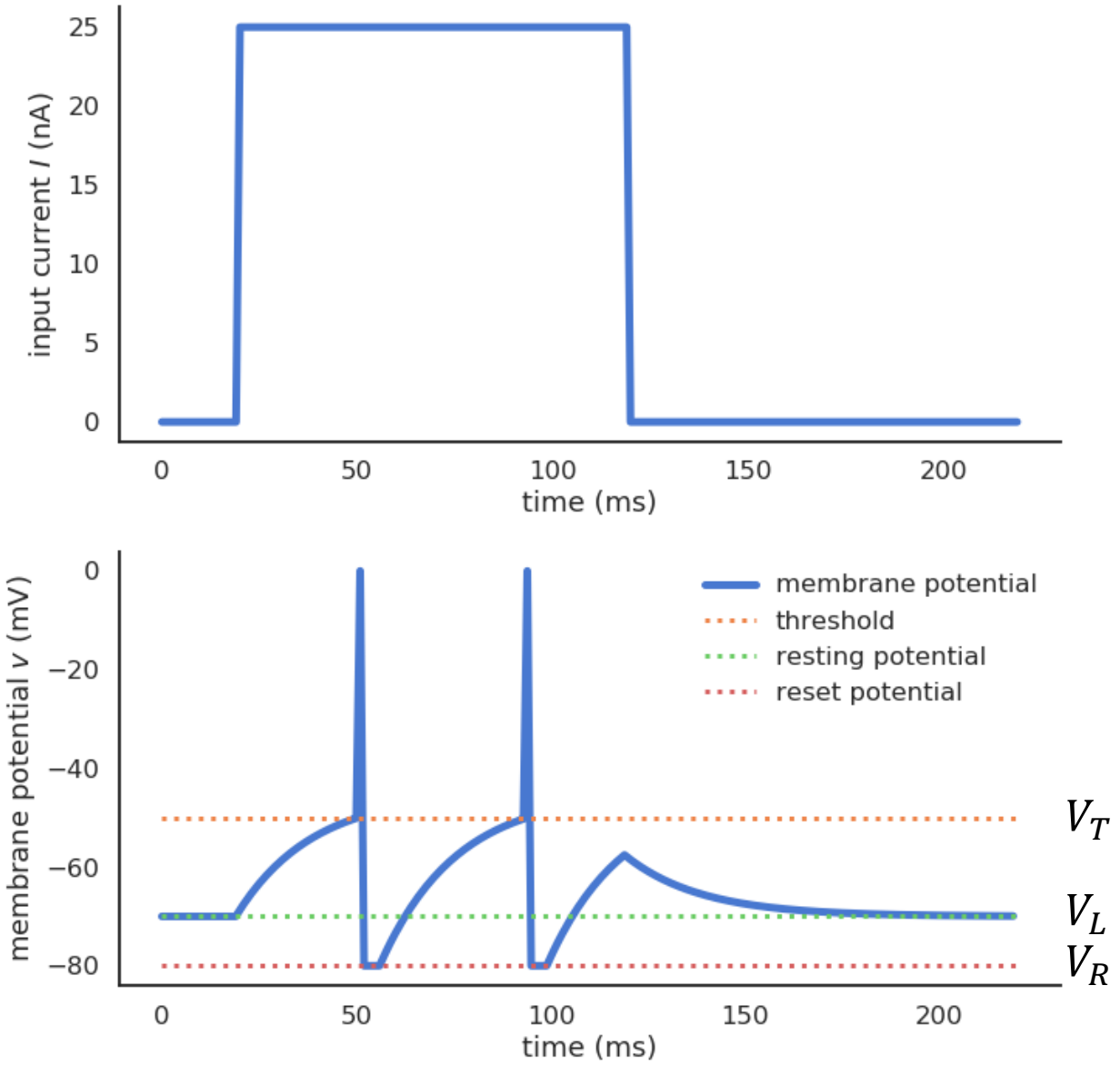
$v(t) > V_T$: sende einen Spike
setzte $v(t) = V_R$ für t_{ref} ms.



Membrane potential of a leaky integrate-and-fire neuron.

https://neuronaldynamics.epfl.ch/online/Ch1.S3.html

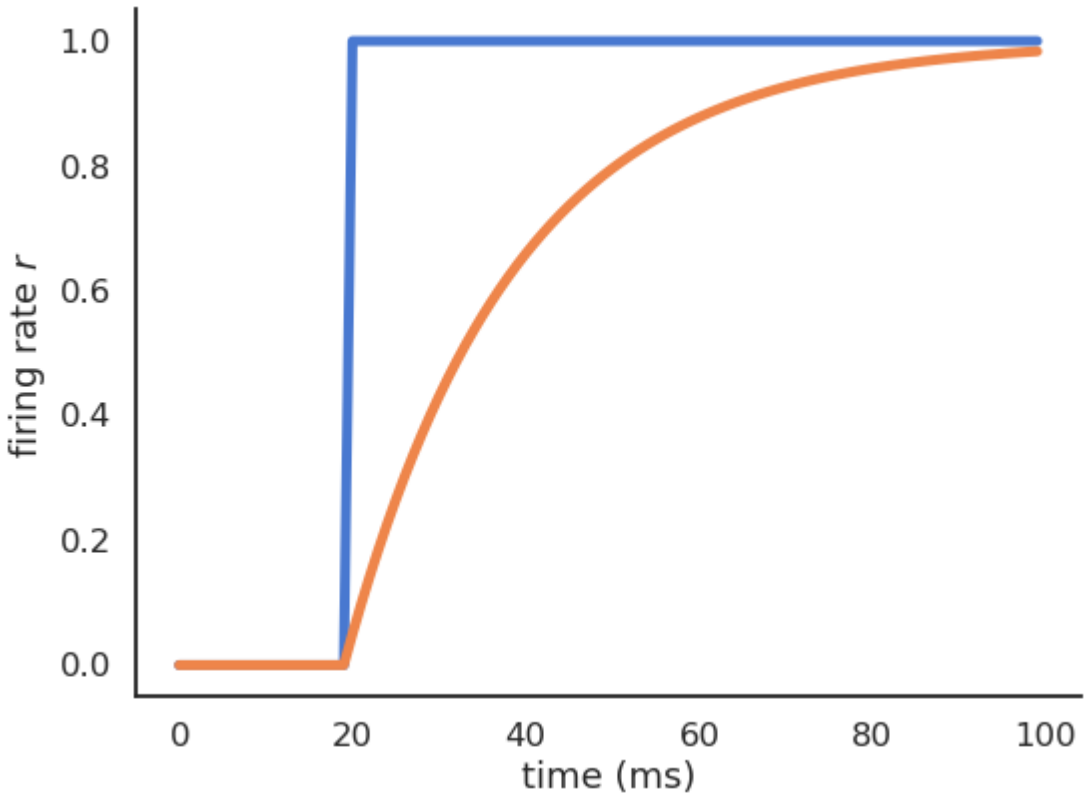
Spiking Neurone



Spike emission of a leaky integrate-and-fire neuron.

Spiking Neurone

Erklärung zu Differentialgleichungen:



Membranpotential eines einfachen ratenkodtierten Neurons mit Eingabe $I(t)$:

$$\tau \frac{dv(t)}{dt} + v(t) = I(t)$$

$$r(t) = [v(t)]^+$$

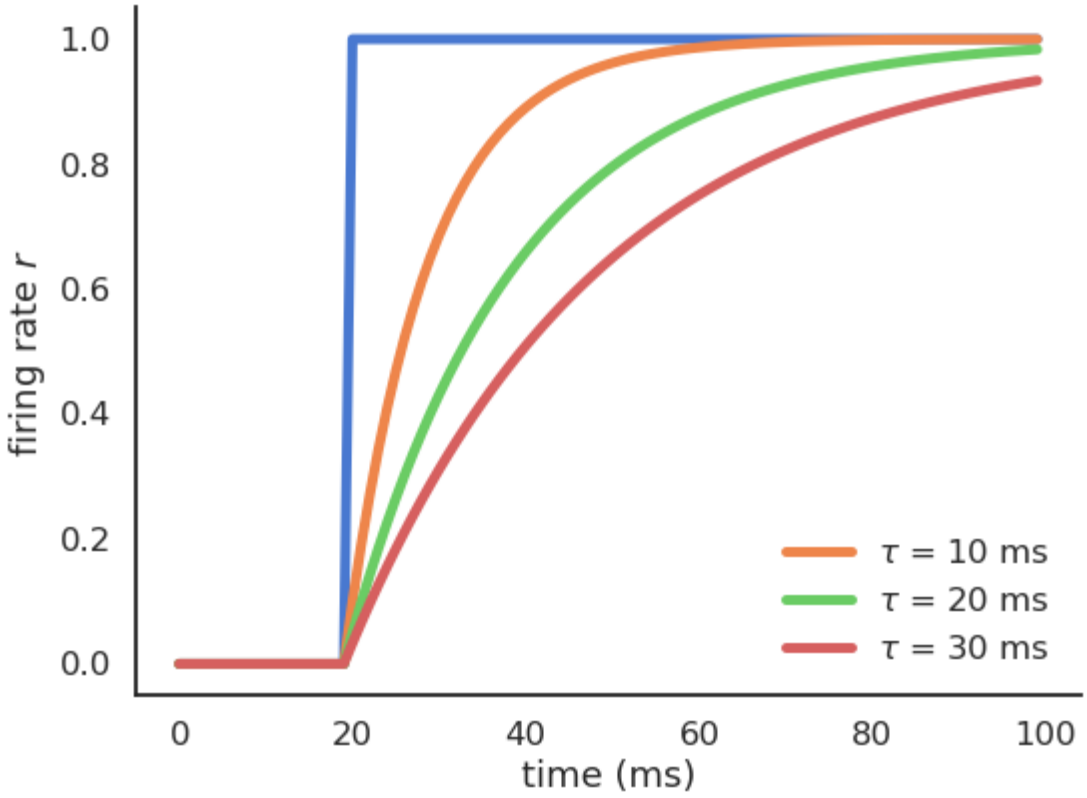
Die Anpassung von $v(t)$ ist durch seine zeitliche Ableitung bestimmt:

$$\frac{dv(t)}{dt} = \frac{I(t) - v(t)}{\tau}$$

Ein Unterschied $v(t)$ und $I(t)$, wird durch Anpassung des Membranpotentials schnell ausgeglichen. Die Schnelligkeit der Anpassung erfolgt durch τ.

Spiking Neurone

Erklärung zu Differentialgleichungen:



Biologische Neurone haben eine Zeitkonstante zwischen 5 und 30 ms abhängig vom Zelltyp.

Spiking Neurone

Beispiele von Spiking Neuronen

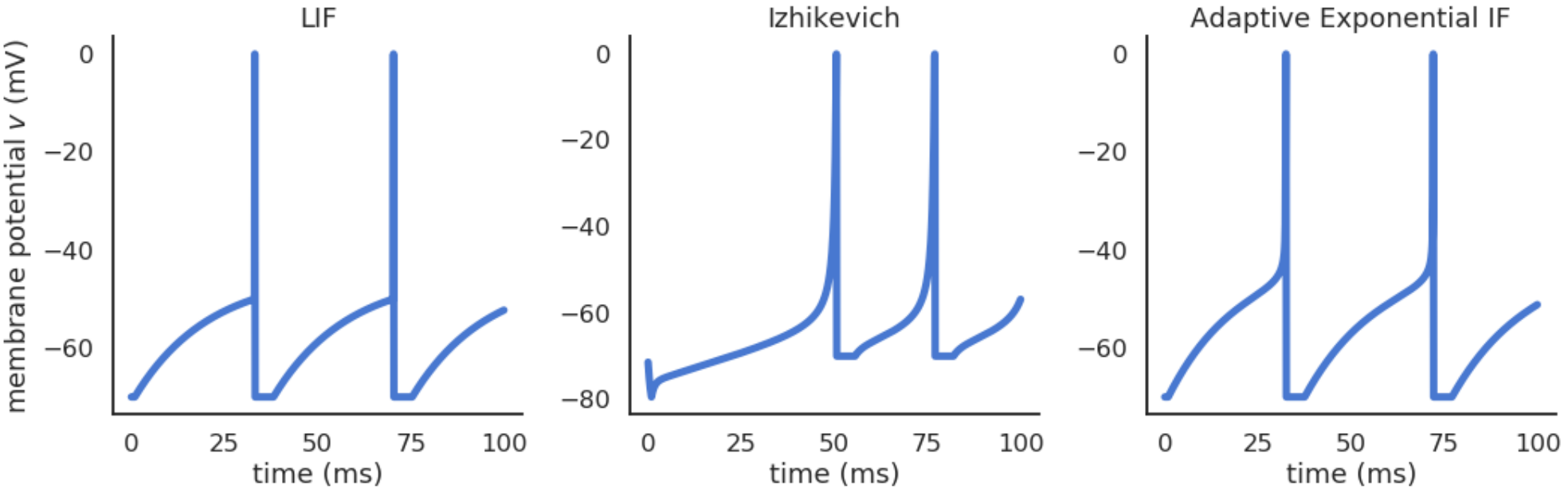
LIF $$C\frac{dv(t)}{dt} = -g_L(v(t) - V_L) + I(t)$$

- Izhikevich quadratic integrate-and-fire

$$\frac{dv(t)}{dt} = 0.04\,v(t)^2 + 5v(t) + 140 - u(t) + I(t) \qquad \frac{du(t)}{dt} = a(bv(t) - u(t))$$

- Adaptive exponential integrate-and-fire (AdEx)

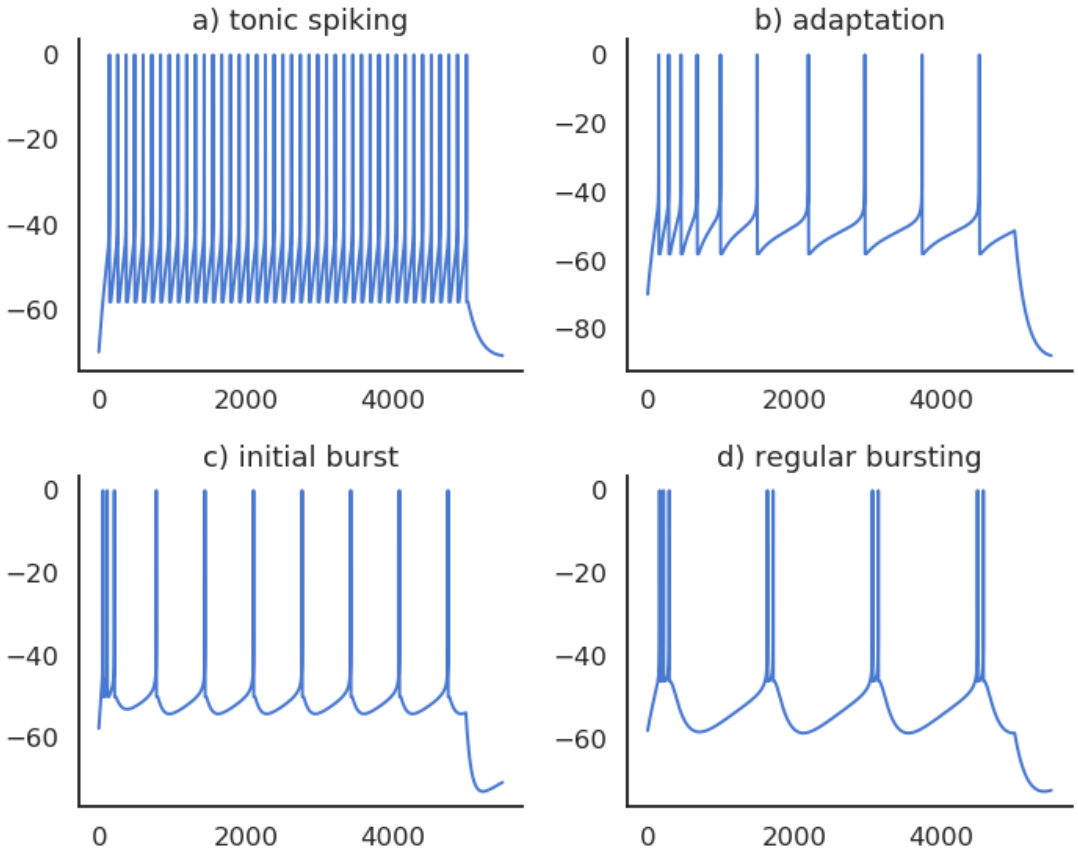
$$C\frac{dv(t)}{dt} = -g_L(v(t) - E_L) + g_L\Delta_T \exp\left(\frac{v(t) - V_T}{\Delta_T}\right) - w(t) + I(t) \qquad \tau_w\frac{dw(t)}{dt} = a(v(t) - E_L) - w(t)$$



LIF, Izhikevich und AdEx Neuronen.

Spiking Neurone

Neuronmodelle können anhand weniger Parameter eine diverse Dynamik an Feuermustern erzeugen



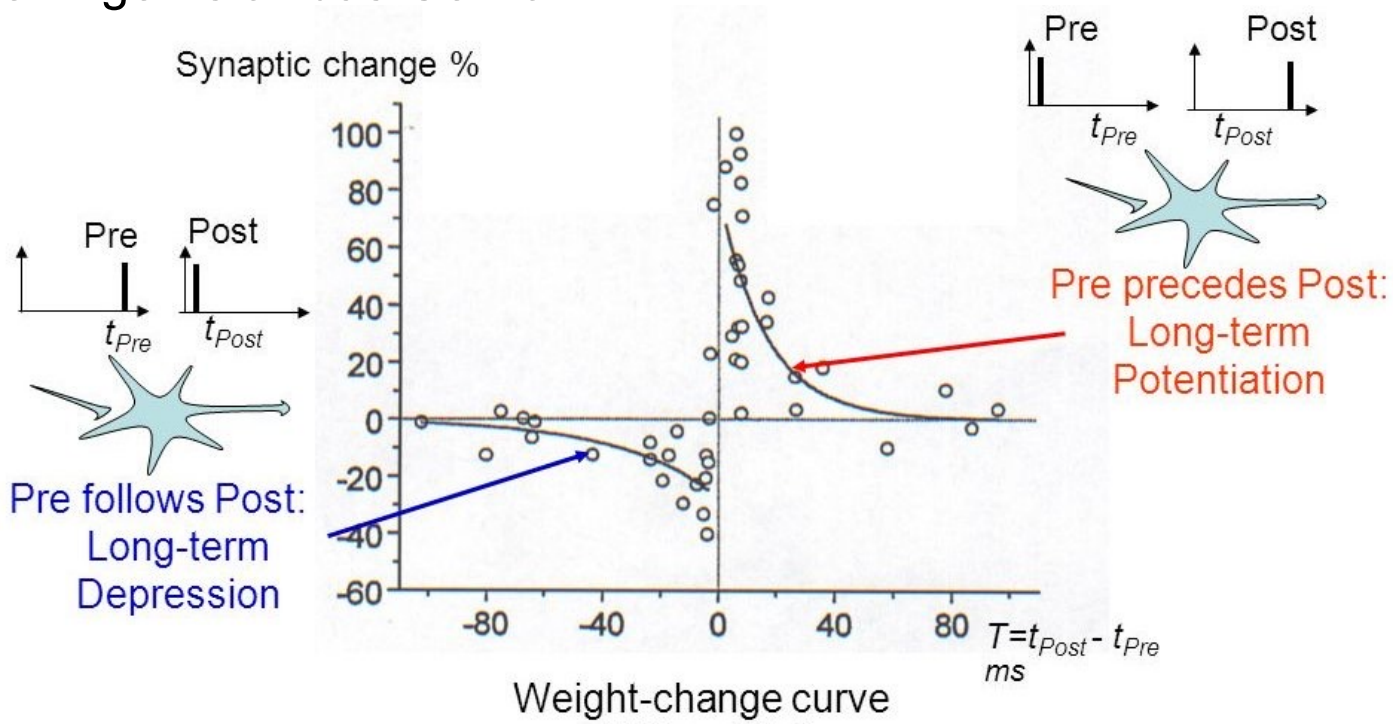
AdEx Modell mit verschiednen Parametern.

Lernen mit Spiking Neuronen

Spike-timing-dependent plasticity (STDP)

Zur Änderung des Übertragungsgewichtes werden die Zeitpunkte der Spikes berücksichtigt:

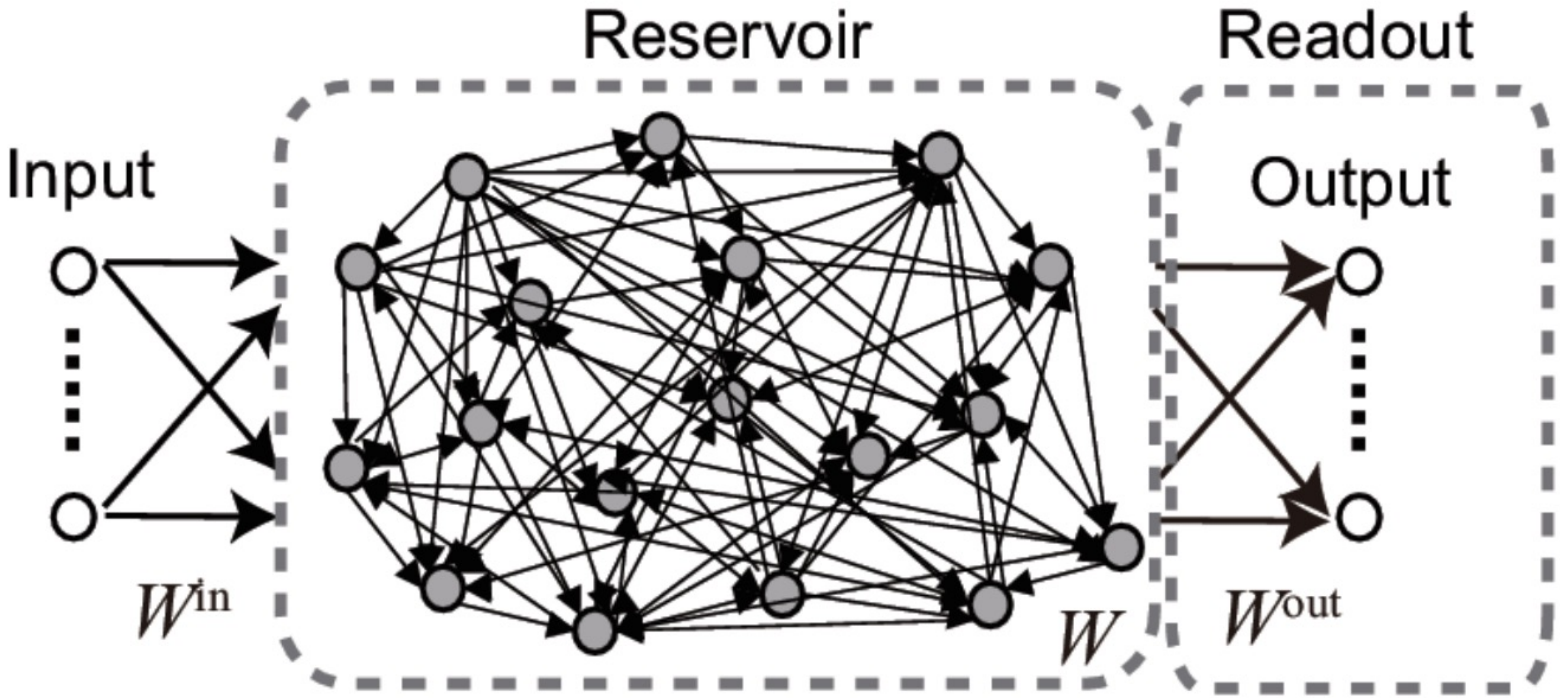
- Wenn ein prä-synaptisches Neuron **vor** dem post-synaptischen Neuron feuert, dann verstärkt sich das Gewicht. Kausalität im Feuern.
- Wenn ein prä-synaptisches Neuron **nach** dem post-synaptischen Neuron feuert, dann, verringert sich das Gewicht.



Bi, G. Q., & Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of Neuroscience*, *18*(24), 10464–10472.

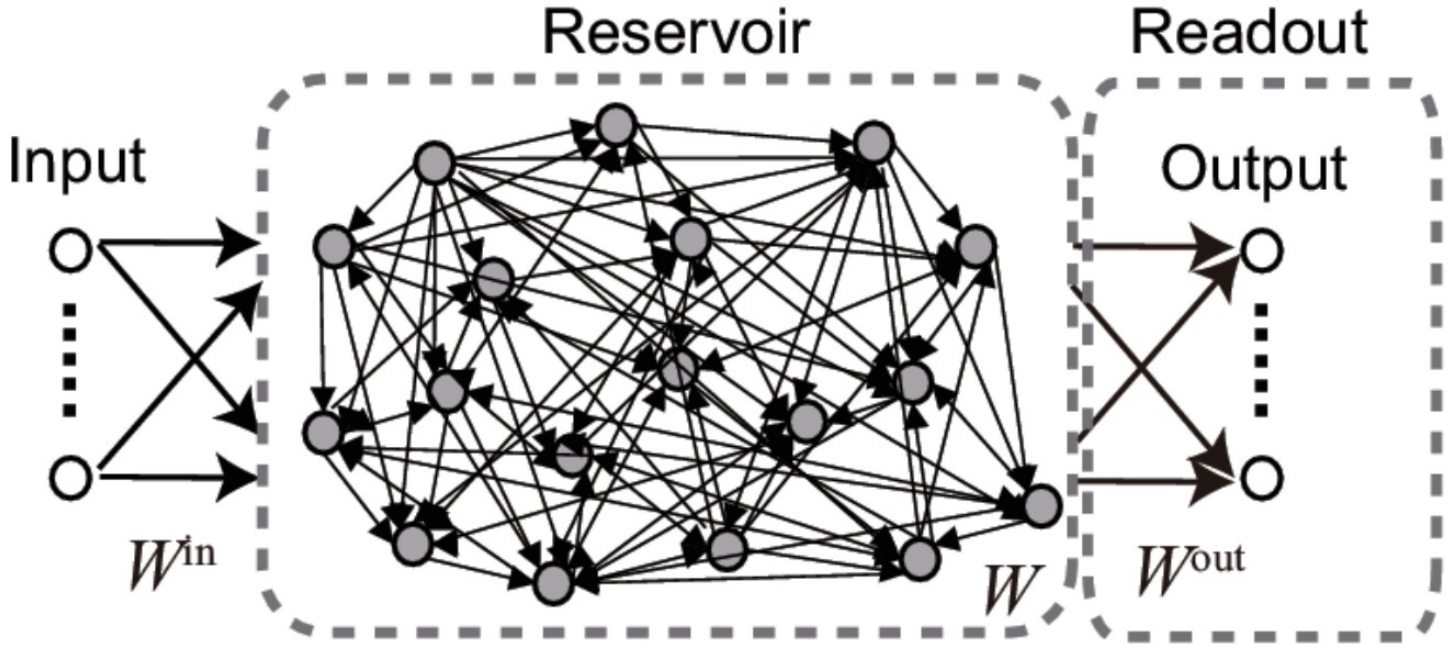
Reservoir Computing

Konzepte des **Reservoir Computing** (RC) wurden gleichzeitig von zwei Forschern um die Jahre 2000 entwickelt. Reservoir Computing beschreibt Prinzipien die unabhängig voneinander einführt wurden. Herbert Jaeger (Uni Bremen), der an **Echo-State Networks** (ESN), basierend auf ratenkodierten Neuronen (Jaeger, 2001) arbeitet, und Wolfgang Maass (TU Graz), der an **Liquid State Machines** (LSM), unter Verwendung von Spiking Neurone (Maass et al., 2002) arbeitet.



Tanaka, G., Yamane, T., Benoit, J., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A. (2019). Recent advances in physical reservoir computing : A review. *Neural Networks*, *115*, 100–123.

Reservoir Computing



Ein Reservoir ist ein rekurrentes neuronales Netzwerk (RNN), das komplexe zeitliche Dynamiken entwickeln kann.

Eigenschaften:

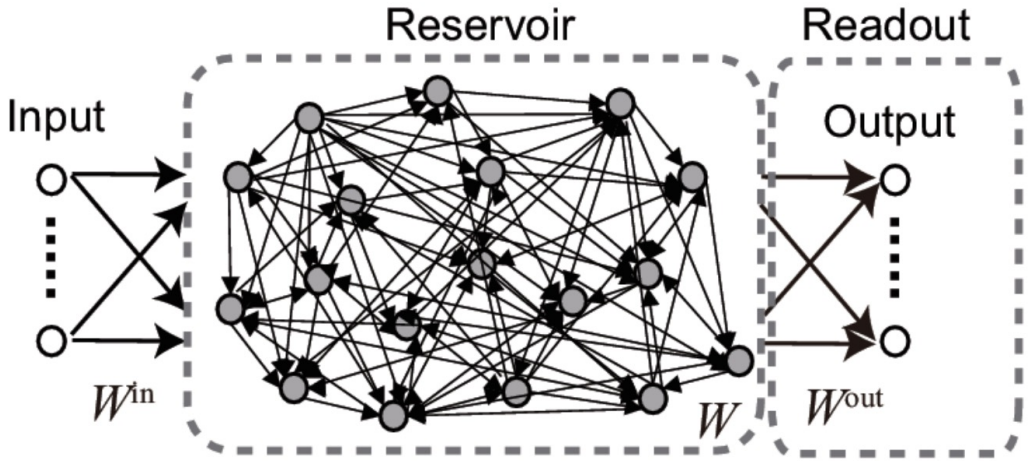
- Transformation in einen hochdimensionalen raumzeitlichen Raum (Reservoir)
- Lediglich die Ausgabe muss trainiert werden. Gewichte im Reservoir sind fixiert.
- Das Auslesen kann durch einfache Lernalgorithmen effizient realisiert werden
- Geeignet für zeitliche und sequentielle Datenverarbeitung

Tanaka, G., Yamane, T., Benoit, J., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A. (2019). Recent advances in physical reservoir computing : A review. *Neural Networks*, *115*, 100–123.

Echo-State Netzwerke

Echo-state networks (ESNs)

Ein Echo-State Netzwerk (ESN) enthält **recurrent** verknüpfte units (sparsely connected).



Ratenkodierte Neuronen im Reservoir integrieren externe und rekurente Eingaben unter Verwendung von ODEs:

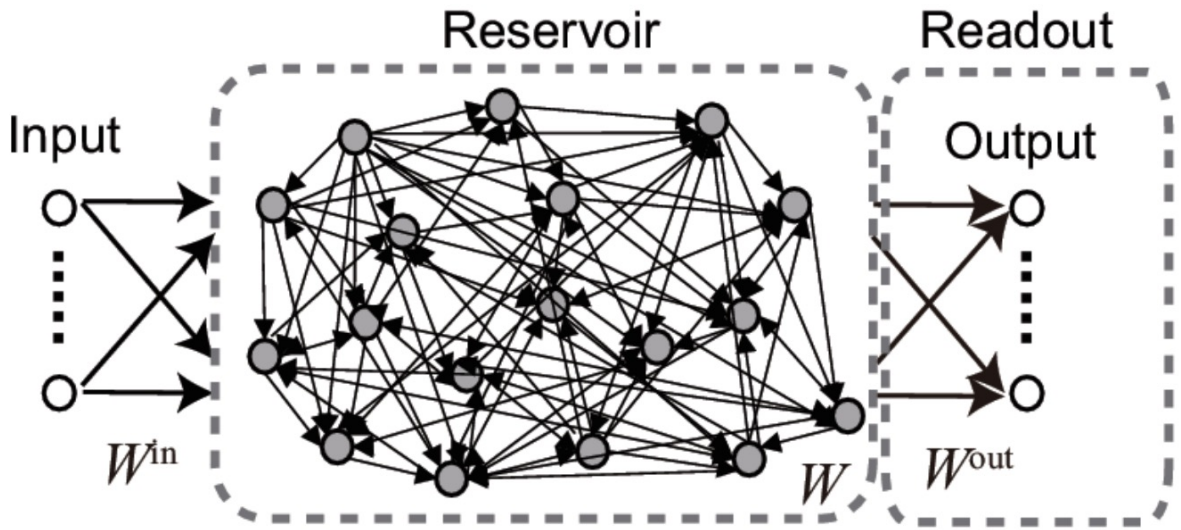
$$\tau \frac{dx_j(t)}{dt} + x_j(t) = \sum_i W_{ij}^{IN} I_i(t) + \sum_i W_{ij} r_i(t) + \xi(t)$$

Die Ausgabefunktion eines Neurons ist die tanh function [-1, 1]:

$$r_j(t) = \tanh\left(x_j(t)\right)$$

Tanaka, G., Yamane, T., Benoit, J., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A. (2019). Recent advances in physical reservoir computing : A review. *Neural Networks*, *115*, 100–123.

Echo-State Netzwerke



Readout Neurone (Output Neurone) transformieren die Aktivität des Reservoirs linear:

$$z_k(t) = \sum_j W_{jk}^{OUT} r_j(t)$$

Zum Lernen kann eine überwacht lernende Lernregel verwendet werden, um die gewünschten Outputs zu erhalten.

Tanaka, G., Yamane, T., Benoit, J., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A. (2019). Recent advances in physical reservoir computing : A review. *Neural Networks*, *115*, 100–123.

Echo-State Netzwerke

Reservoirs benötigen nur wenige hundert Iterationen, um komplexe Funktionen zu erlernen. Die lateralen Gewichte werden häufig nach dem Zufallsprinzip unter Verwendung einer Normalverteilung mit Mittelwert 0 und Abweichung initialisiert $\frac{g}{\sqrt{N}}$:

$$w_{ij} \sim N\left(0, \frac{g}{\sqrt{N}}\right)$$

g ist ein Skalierungsfaktor, der die Stärke der wiederkehrenden Verbindungen charakterisiert, was zu unterschiedlichen Dynamiken führt.

Die rekurrente Gewichtsmatrix ist oft spärlich. In der Regel werden nur 10 % der möglichen Verbindungen erstellt.

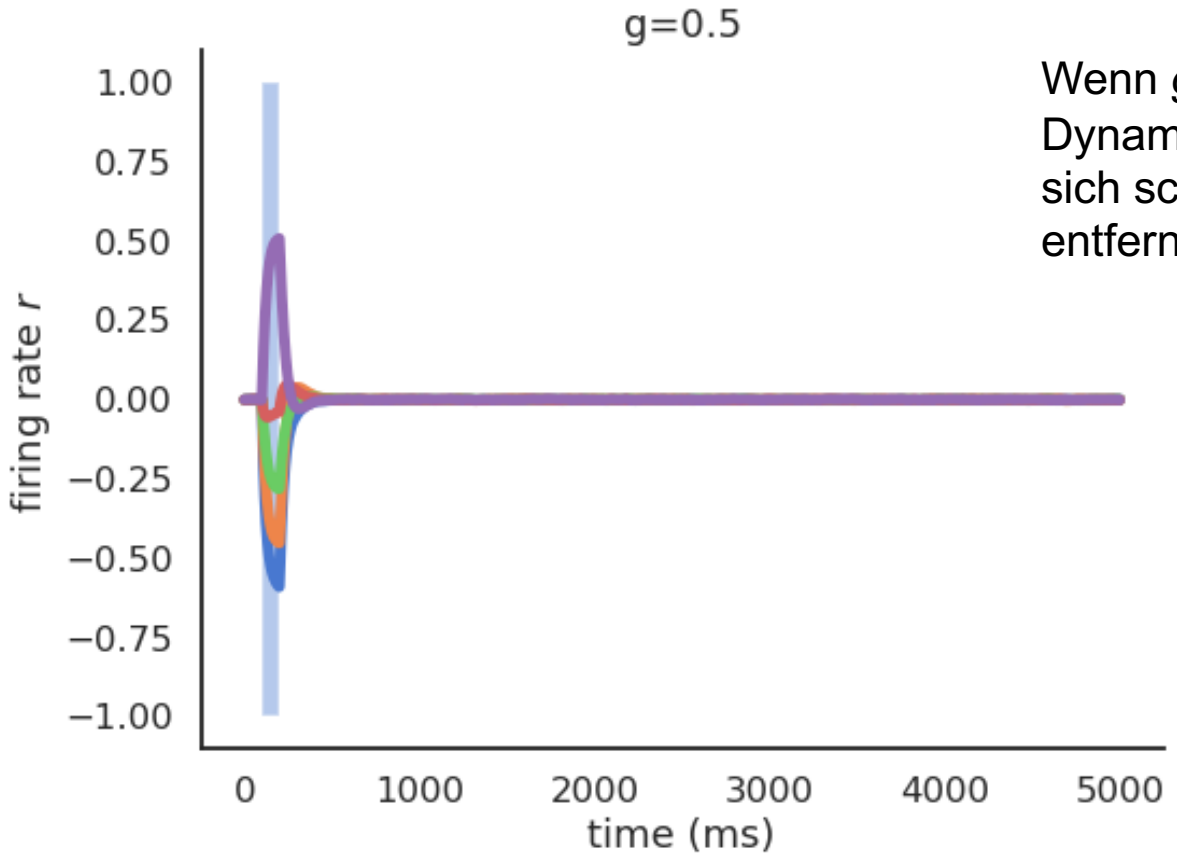
Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

Tanaka, G., Yamane, T., Benoit, J., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., & Hirose, A. (2019). Recent advances in physical reservoir computing : A review. *Neural Networks*, *115*, 100–123.

Echo-State Netzwerke

Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

Beispielaktivität nach der Präsentation eines kurzen Inputs:

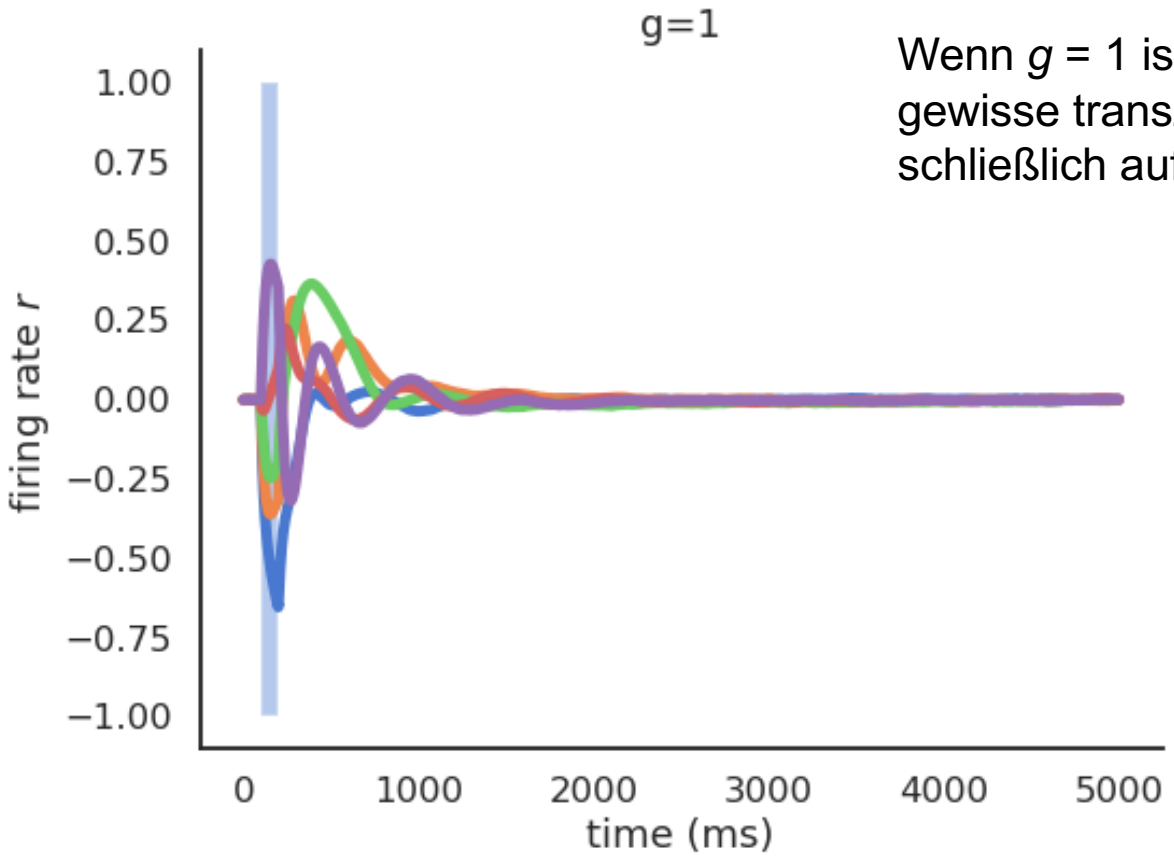


Wenn *g*<1, weist das Netzwerk wenig Dynamik auf und die Aktivität reduziert sich schnell auf 0, wenn die Eingabe entfernt wird.

Echo-State Netzwerke

Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

Beispielaktivität nach der Präsentation eines kurzen Inputs:

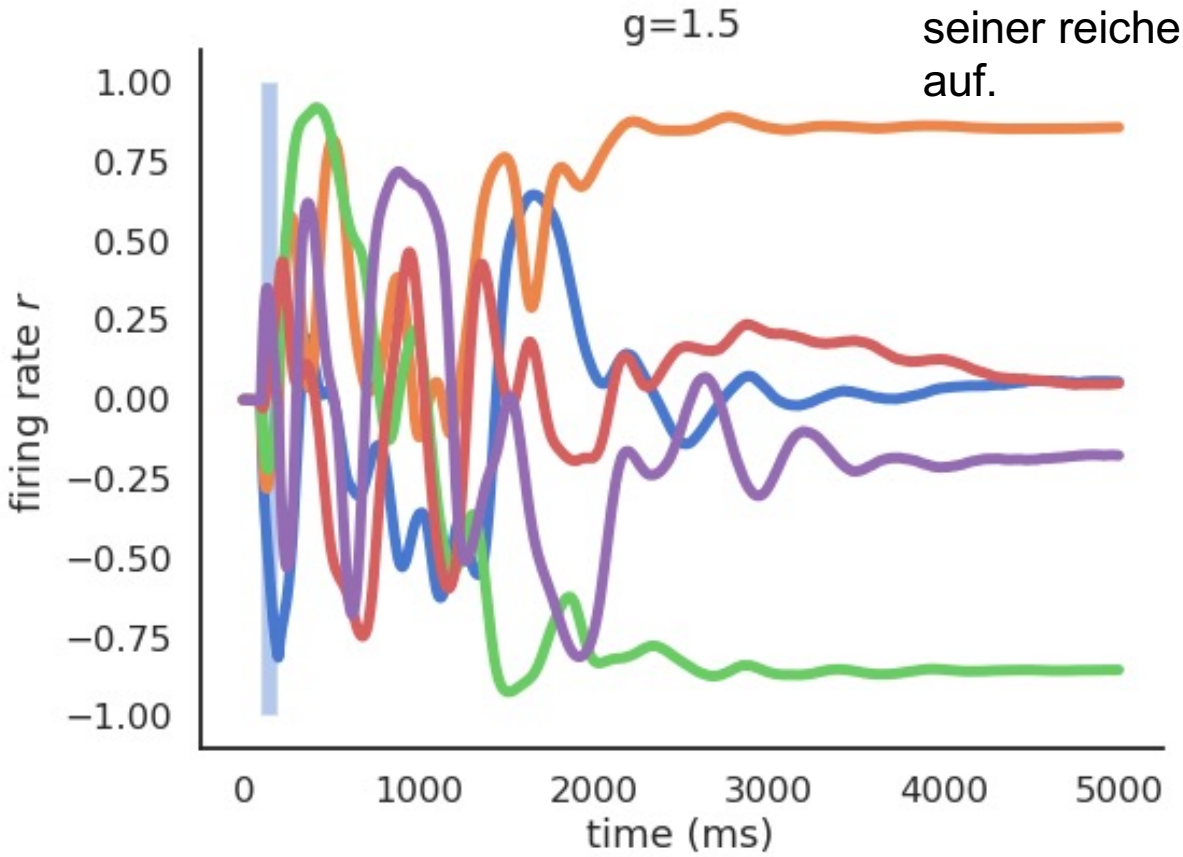


Wenn *g* = 1 ist, zeigt das Reservoir eine gewisse transzitente Dynamik, sinkt aber schließlich auf 0.

Echo-State Netzwerke

Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

Beispielaktivität nach der Präsentation eines kurzen Inputs:

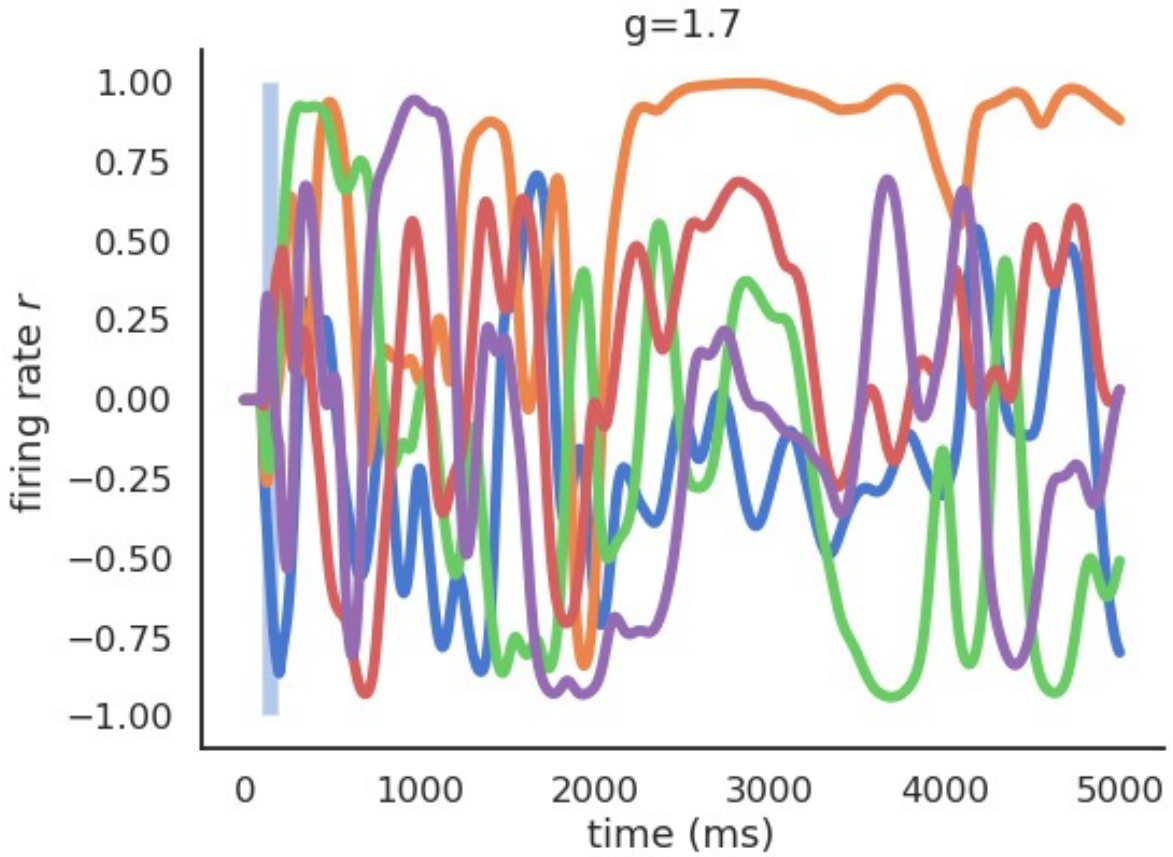


Wenn *g* = 1,5 ist, weist das Reservoir aufgrund seiner reichen Dynamik viele stabile Attraktoren auf.

Echo-State Netzwerke

Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

Beispielaktivität nach der Präsentation eines kurzen Inputs:



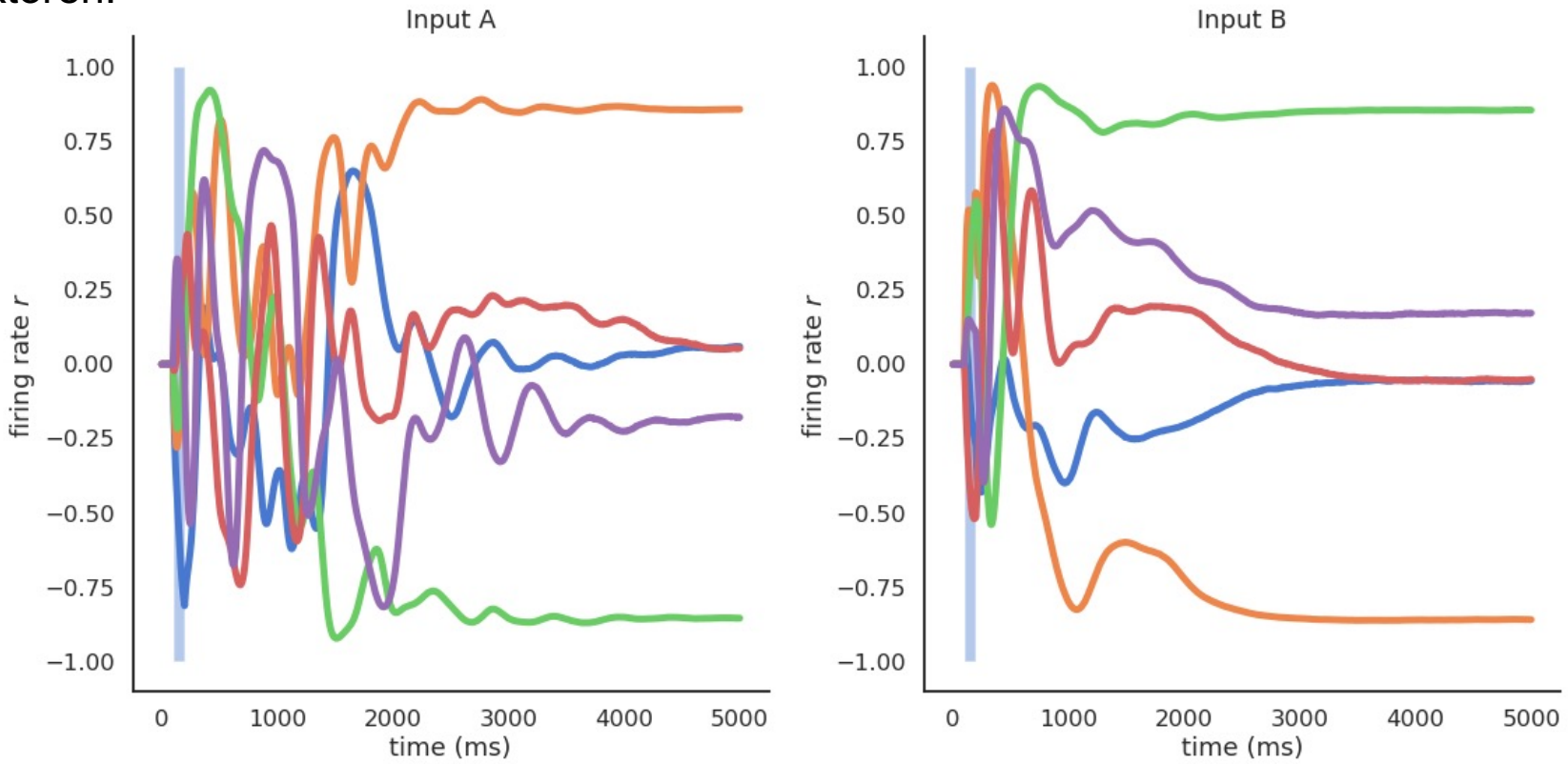
Für höhere Werte von *g* gibt es keine stabilen Attraktoren mehr: chaotisches Verhalten.

Echo-State Netzwerke

Je nach Wert von *g* weist die Dynamik des Reservoirs unterschiedliche Attraktoren auf.

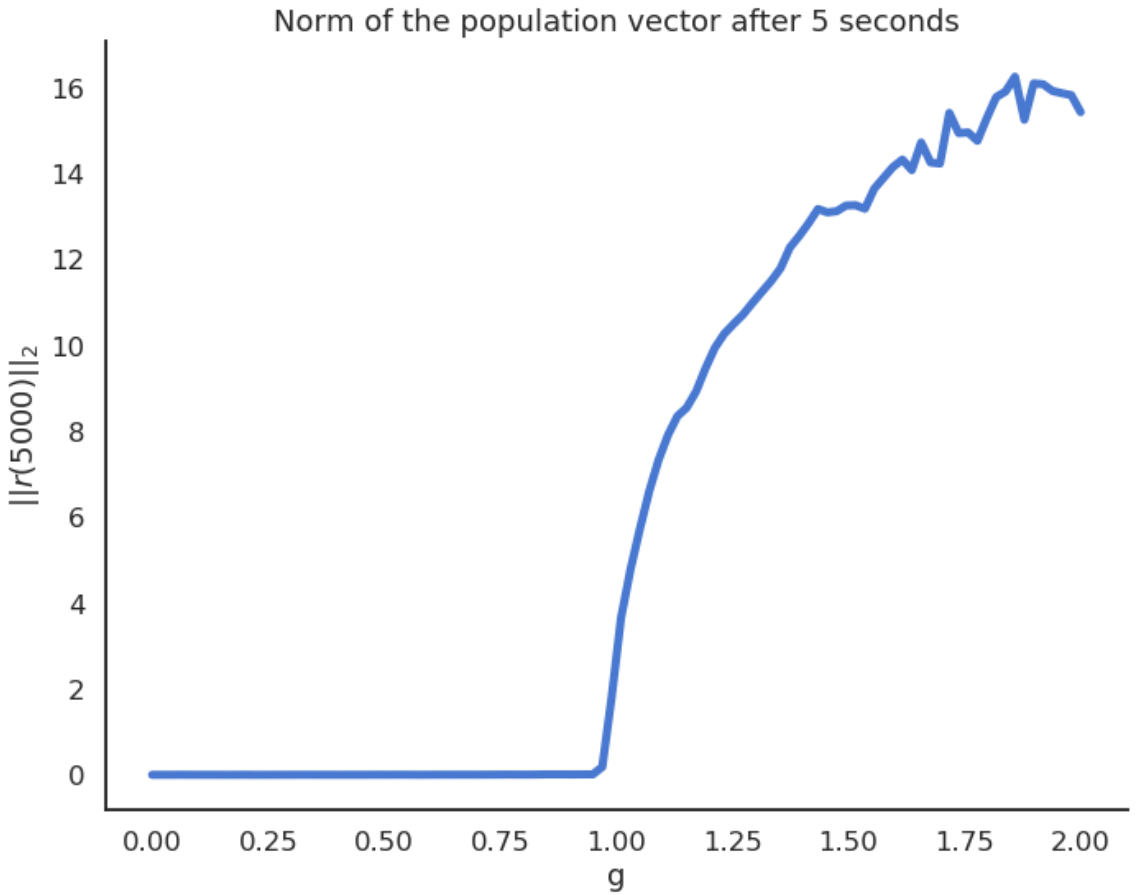
Beispielaktivität nach der Präsentation eines kurzen Inputs:

Für *g*=1,5 führen unterschiedliche Eingaben (Anfangszustände) zu unterschiedlichen Attraktoren.



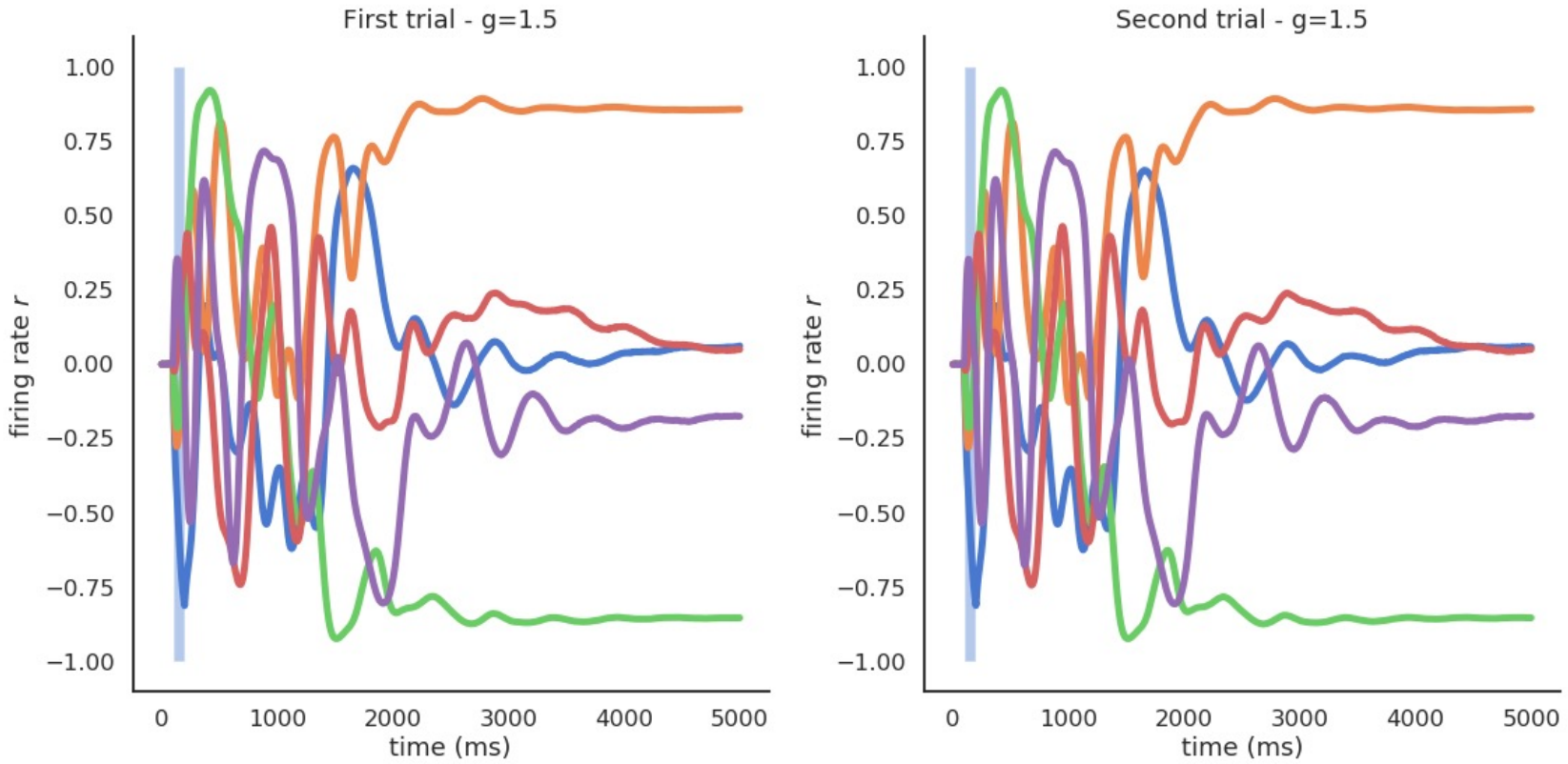
Echo-State Netzwerke

Die Gewichtsmatrix muss einen Skalierungsfaktor über 1 haben, um Attraktoren ungleich Null zu zeigen.



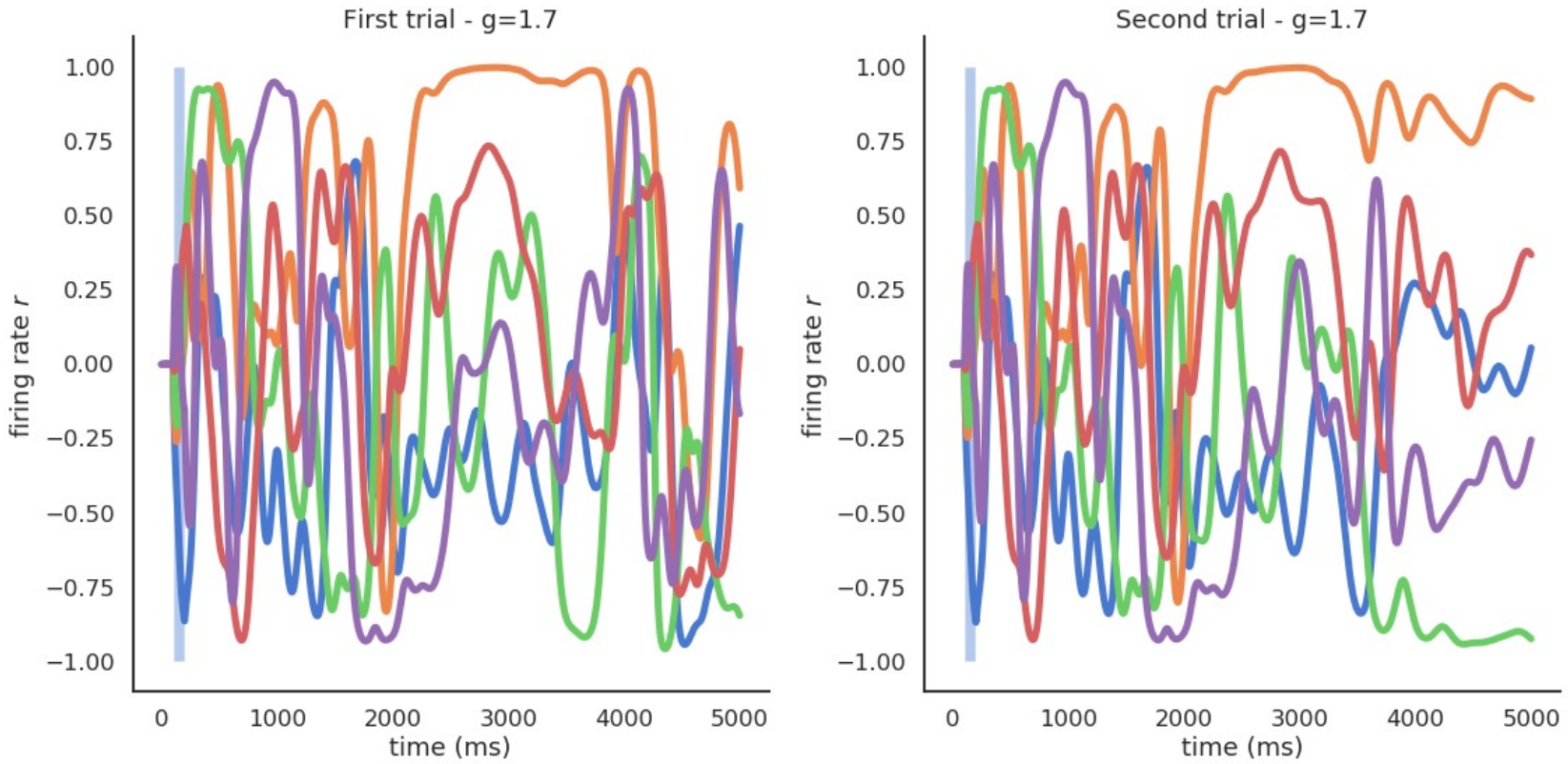
Echo-State Netzwerke

Beim gleichen Input ist der Attraktor immer derselbe, auch bei Rauschen oder Störungen.



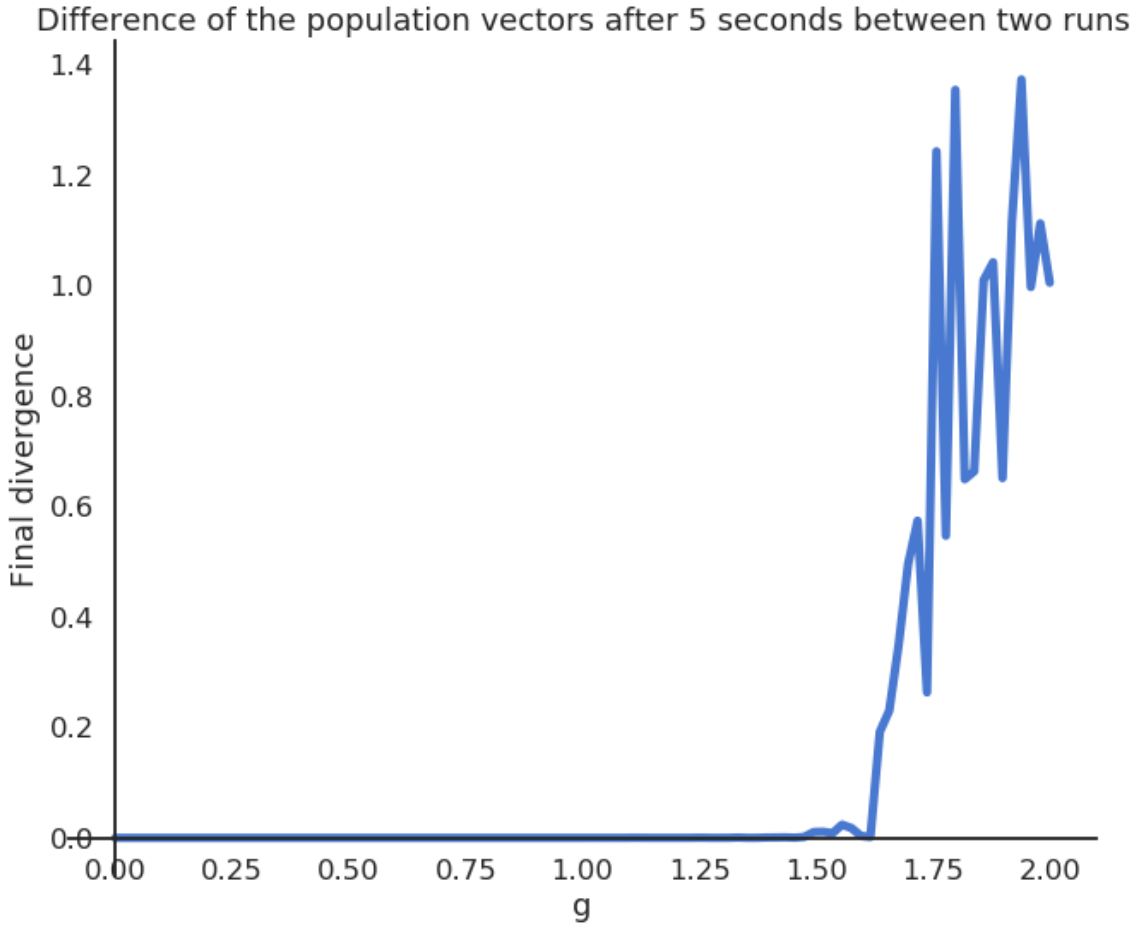
Echo-State Netzwerke

Im chaotischen Regime führt die geringste Unsicherheit über die Anfangsbedingungen (oder das Vorhandensein von Rauschen) langfristig zu sehr unterschiedlichen Trajektorien.



Echo-State Netzwerke

- Das chaotische Regime erscheint ab g>1.5.
- Bei g=1,5 ist der Rand zum Chaos: Die Dynamik ist sehr reichhaltig, aber das Netzwerk ist noch nicht chaotisch.



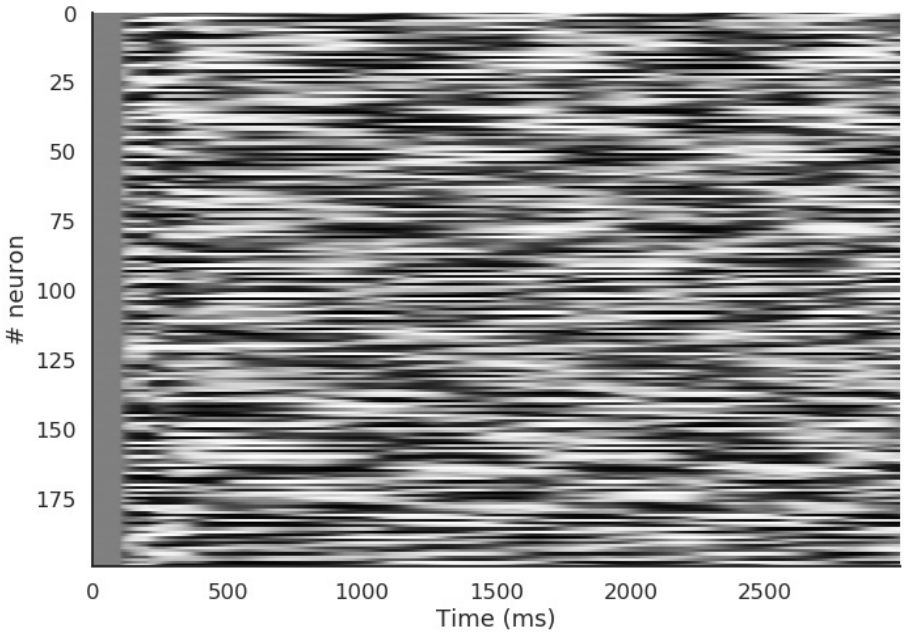
Universelle Approximation

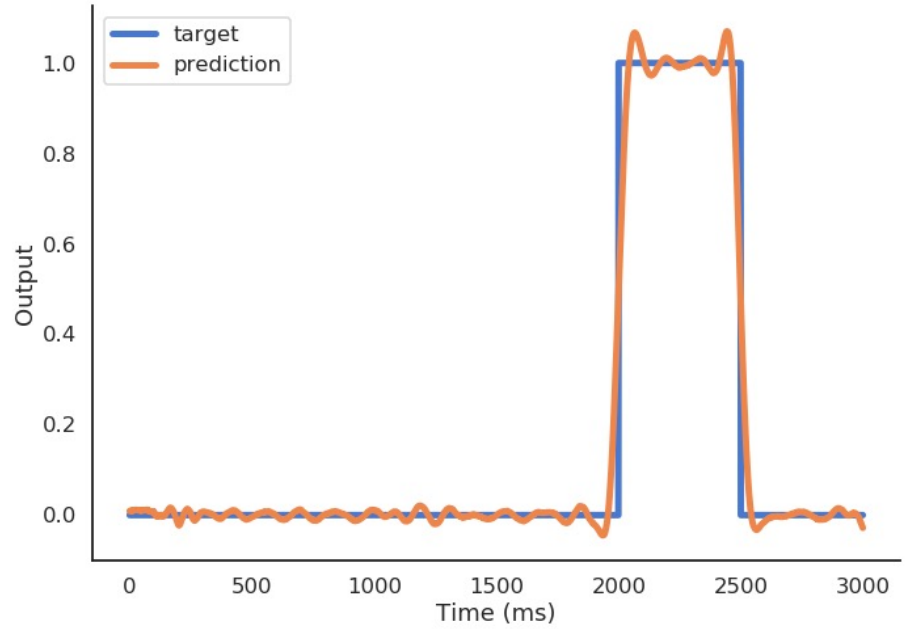
Aufgrund der Dynamik im Reservoir können die linearen Ausgabeneurone so trainiert werden, dass sie jedes nichtlineare Zielsignal über die Zeit reproduzieren:

$$z_k(t) = \sum_j W_{jk}^{OUT} r_j(t)$$

Da es sich um ein Regressionsproblem handelt, reicht oft die Delta-Lernregel (LMS) aus.

$$\Delta W_{jk}^{OUT} = \eta\big(t_k(t) - z_k(t)\big) r_j(t)$$

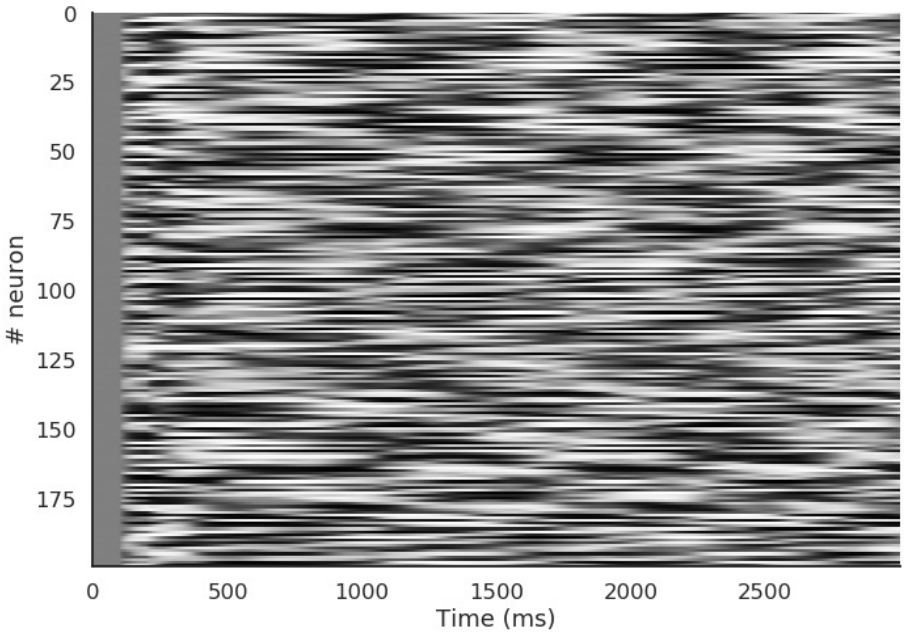


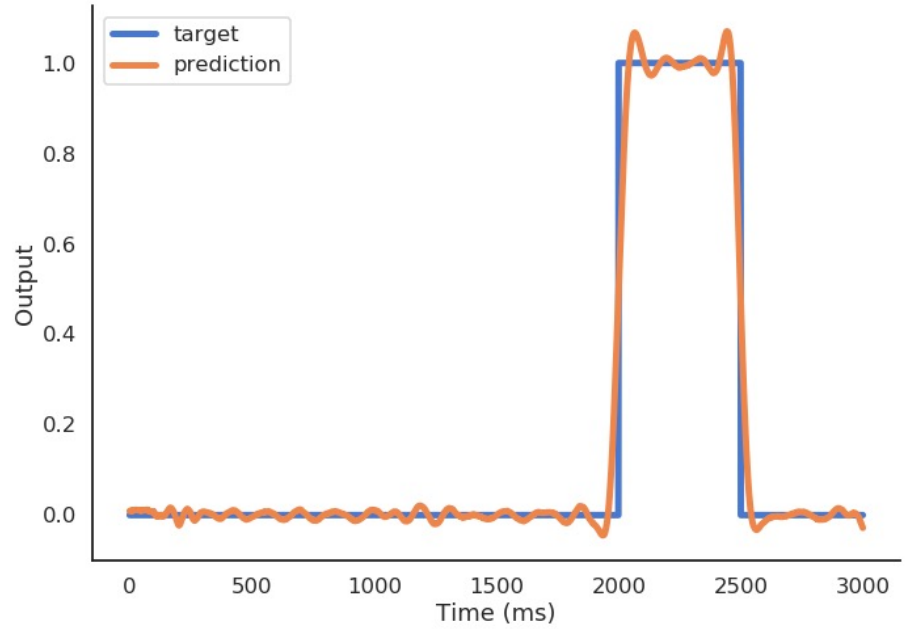


Universelle Approximation

Reservoirs sind universelle Approximatoren. Ein RC-Netzwerk kann jede nichtlineare Funktion zwischen einem Eingangssignal I(t) und einem Zielsignal t(t) approximieren:

- bei genügend Neuronen im Reservoir,
- bei Dynamik am Rande des Chaos.

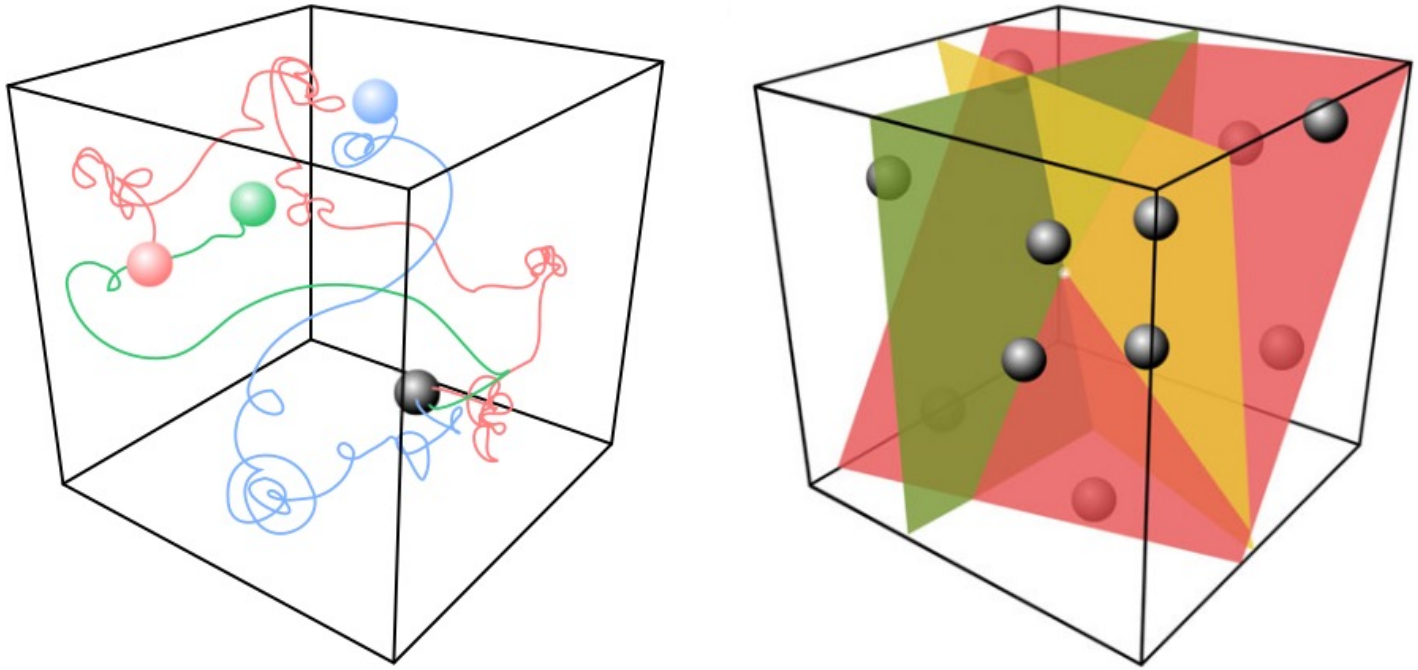




Universelle Approximation

- Das Reservoir projiziert einen niedrigdimensionalen Input in einen hochdimensionalen raum-zeitlichen Merkmalsraum, in dem Trajektorien linear trennbar werden.
- Das Reservoir vergrößert den Abstand zwischen den Eingabemustern.
- Die Eingabemuster sind sowohl räumlich (Neuronen) als auch zeitlich getrennt: Die ausgelesenen Neuronen benötigen viel weniger Gewichte als ein entsprechendes MLP: bessere Verallgemeinerung und ggf. schnelleres Lernen.
- Ein Nachteil ist, dass es nicht gut mit hochdimensionalen Eingaben (Bildern) umgehen kann.

Räumlich-zeitliche Mustertrennung.



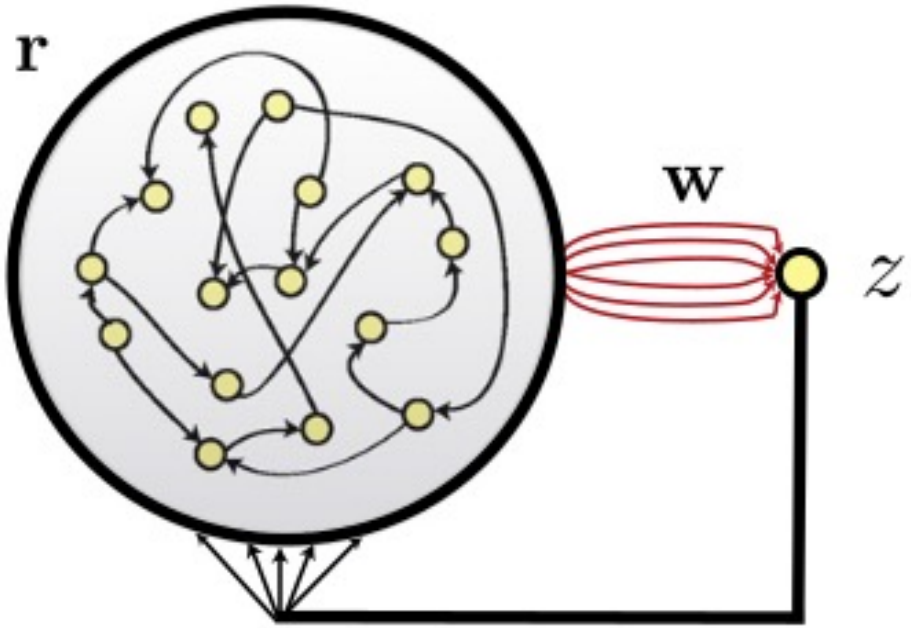
Seoane, L. F. (2019). Evolutionary aspects of reservoir computing. Philosophical Transactions of the Royal Society B.

Reservoir Computing

Feedback-Verbindungen

Der Ausgang der Ausgabeneuronen kann in das Reservoir zurückgeführt werden, umTrajektorien zu stabilisieren:

$$\tau \frac{dx_j(t)}{dt} + x_j(t) = \sum_i W_{ij}^{IN} I_i(t) + \sum_i W_{ij} r_i(t) + \sum_k W_{kj}^{FB} z_k(t) + \xi(t)$$

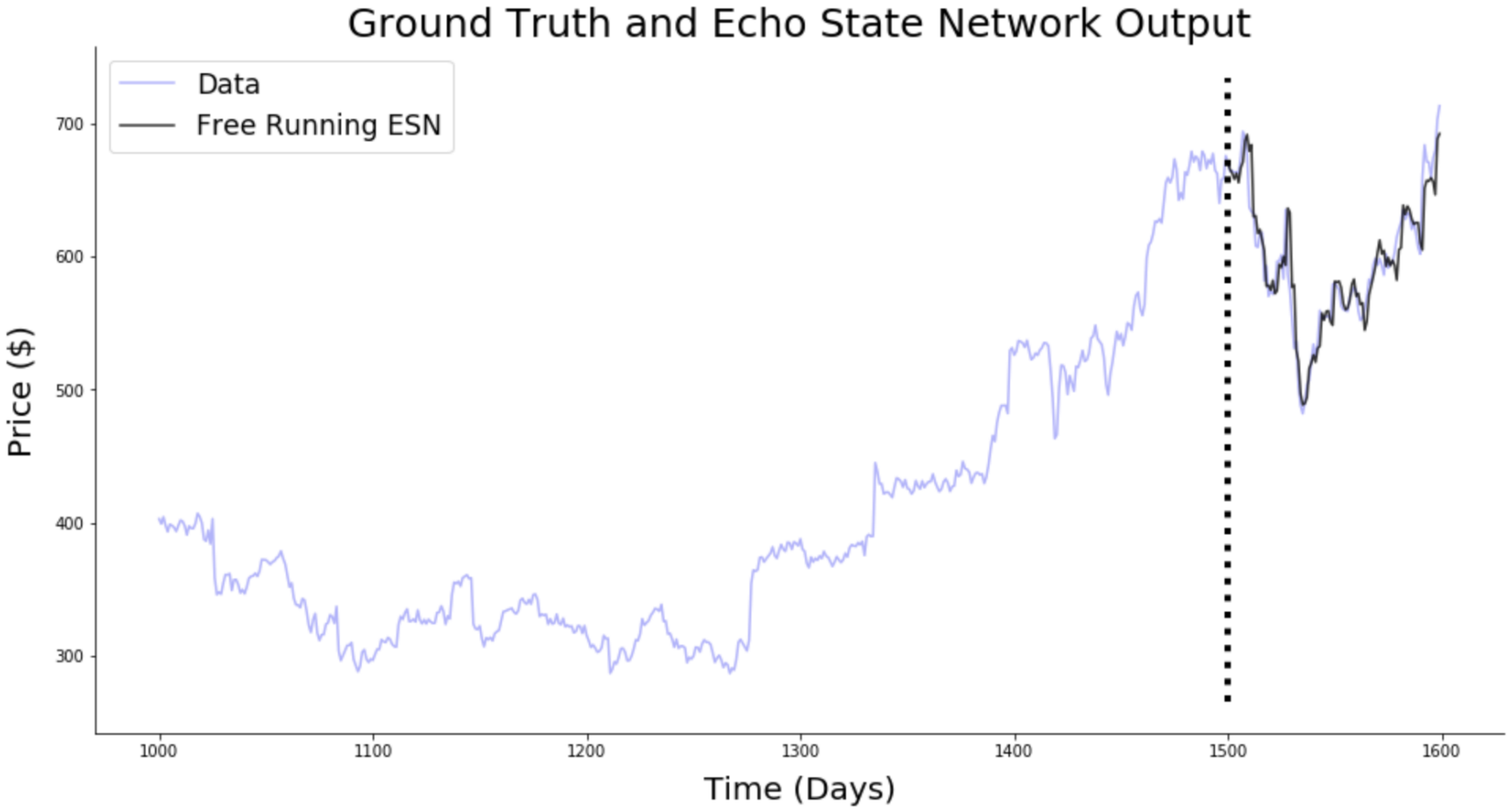


- Dies macht das Reservoir robuster gegenüber Störungen, insbesondere am Rande des Chaos.
- Die Trajektorien sind stabiler (aber immer noch hochdynamisch), was die Arbeit der ausgelesenen Neuronen erleichtert.

Reservoir Computing

Applikationen

Vorhersage: ESN sind in der Lage, die Zukunft chaotischer Systeme (Börse, Wetter) viel besser vorherzusagen als statische NN.

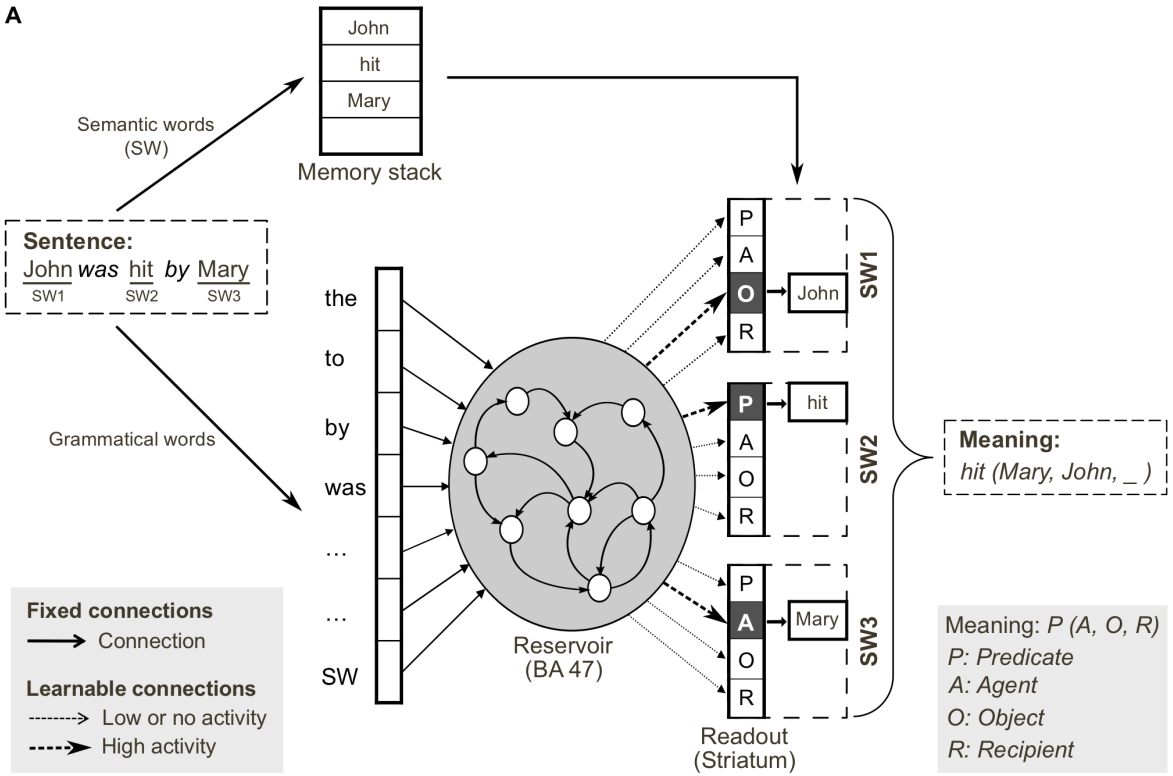


https://towardsdatascience.com/predicting-stock-prices-with-echo-state-networks-f910809d23d4

Reservoir Computing

Applikationen

NLP: RC-Netzwerke können die Dynamik der Sprache, d.h. ihre Grammatik, erfassen. RC-Netzwerke können darauf trainiert werden, Prädikate ("hit(Mary, John)") aus Sätzen ("Mary hit John" oder "John was hit by Mary") zu erzeugen



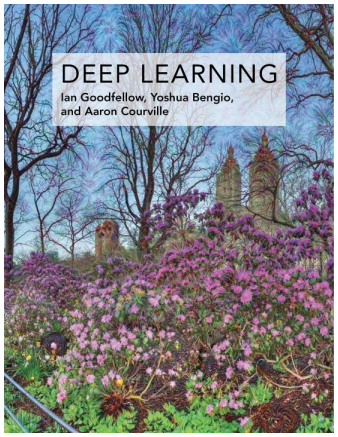
Reservoirs for language understanding.

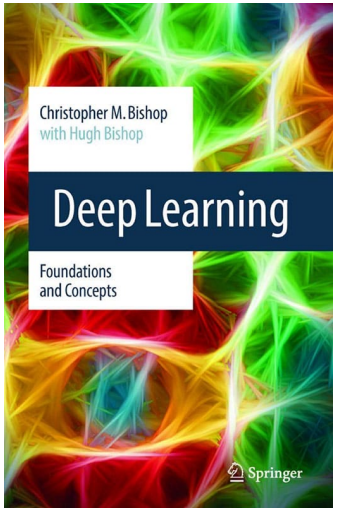
Hinaut, X., and Dominey, P. F. (2013). Real-Time Parallel Processing of Grammatical Structure in the Fronto-Striatal System: A Recurrent Network Simulation Study Using Reservoir Computing. PLoS One, 8(2):e52946. doi: 10.1371/journal.pone.0052946.

Literatur

Bücher:

- Haykin, S. Neural Networks and Learning Machines (3rd Edition), Prentice Hall, 2009
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press. (Online https://www.deeplearningbook.org)
- Christopher M. Bishop , Hugh Bishop Deep Learning: Foundations and Concepts. 2024. Springer. https://www.bishopbook.com





Zeitschriften:

- Neural Networks, Neural Computing, Neural Computation
- IEEE Transactions on Neural Networks
- Neural Processing Letters