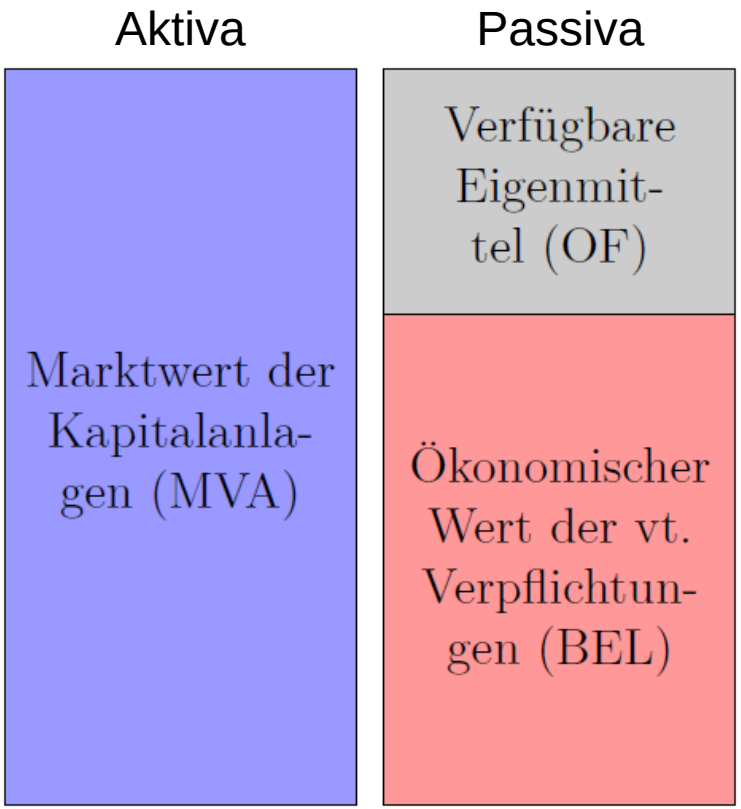
Künstliche Neuronale Netze in der Risikokapitalberechnung

Köln, 30.11.2018

Rudolf Born

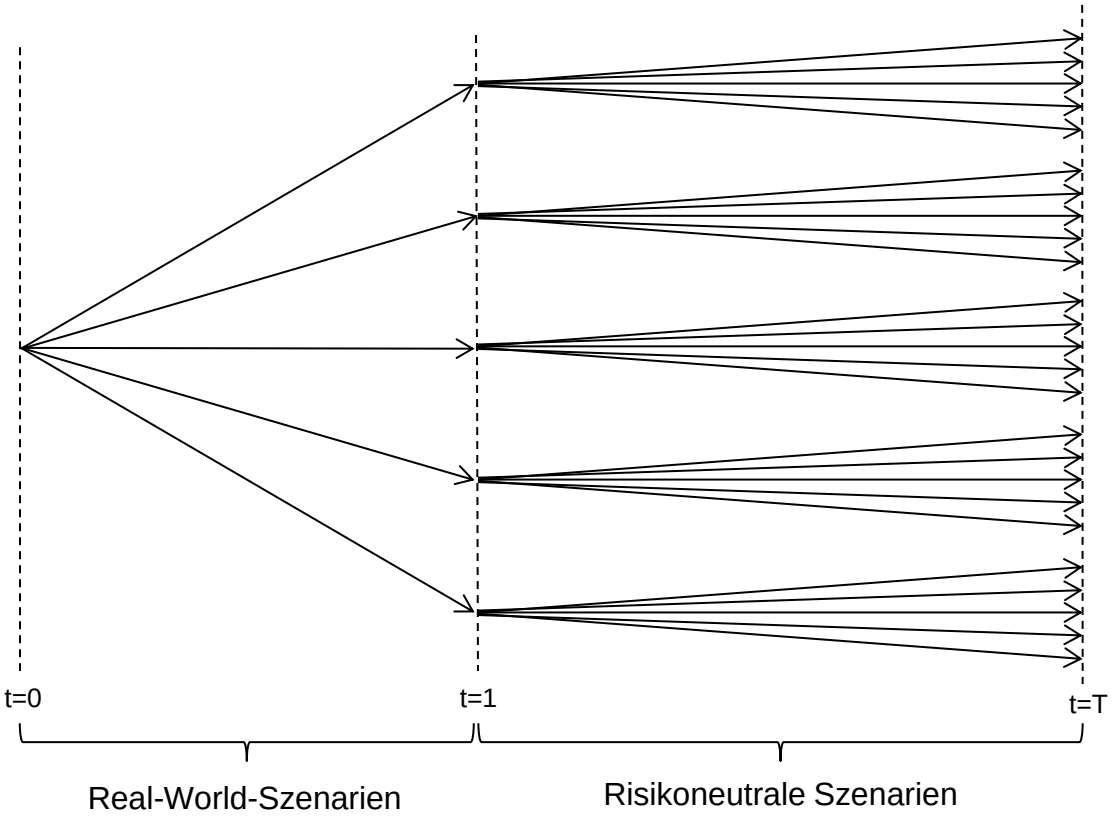
Die Solvenzkapitalanforderung

- Gibt die Höhe des anrechnungsfähigen Eigenkapitals an, über die ein Versicherer zu verfügen hat
- Dieses entspricht dem Value-at-Risk (VaR) der Basiseigenmittel zu einem Konfidenzniveau von 99,5 % über einen Zeitraum von einem Jahr
- Mark-to-Market- und Mark-to-Model-Ansätze
- Meist Barwert-Methoden:
  - Zukünftige Cashflows werden auf den Berwertungsstichtag mit einer geeigneten Zinskurve diskontiert
- Standardformel oder internes Modell



Nested Stochastics

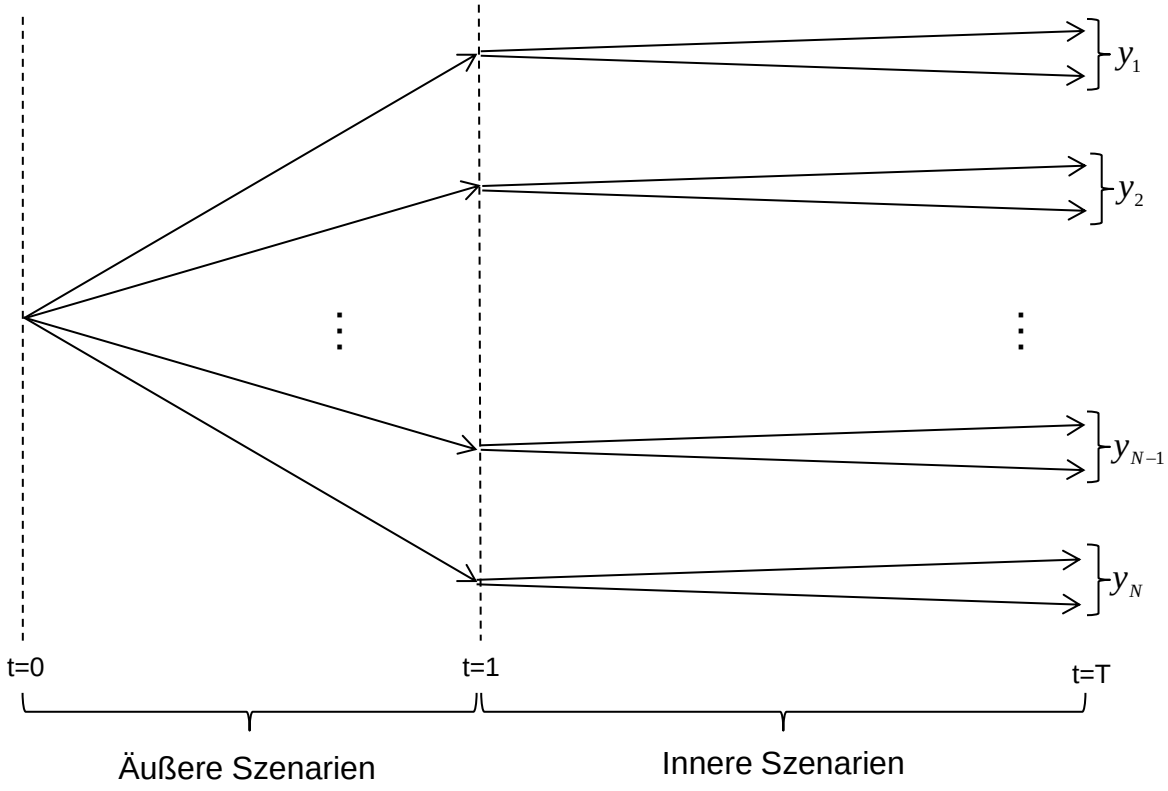
- Ermittlung der Risikotreiber (Marktrisiken und versicherungs-technische Risiken)
- Erstellung einer Vielzahl von realistischen Ein-Jahres-Prognosen
- Pro Szenario Bildung einer Vielzahl von risikoneutralen Szenarien mithilfe von Monte-Carlo-Simulationen
- Schätzung des ökonomischen Werts der Eigenmittel pro Real-World-Szenario
- Folge: enorme Laufzeit



Least Squares Monte Carlo (LSMC)

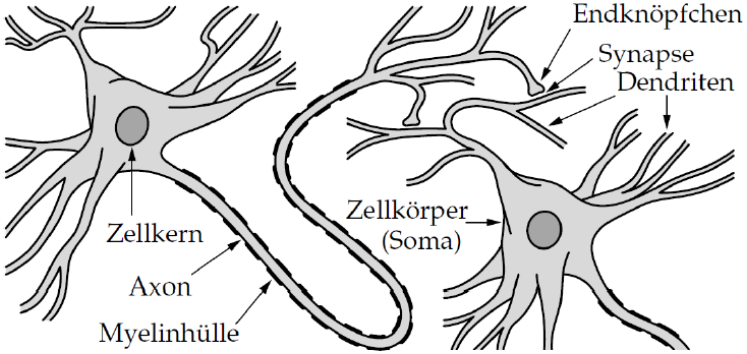
Die Grundidee

- Ziel: Herstellung eines funktionalen Zusammenhangs zwischen der betrachteten ökonomischen Variablen und den zugrundeliegenden Risikofaktoren
- Im Gegensatz zu Nested Stochastics werden pro „äußerem Szenario“ nur eine kleine Anzahl von „inneren“ Bewertungsszenarien benötigt
- Im nächsten Schritt: Durchführung einer Kleineste-Quadrate-Regression
- Idee: Durch die Regression wird der Fehler der einzelnen Schätzungen ausgemittelt



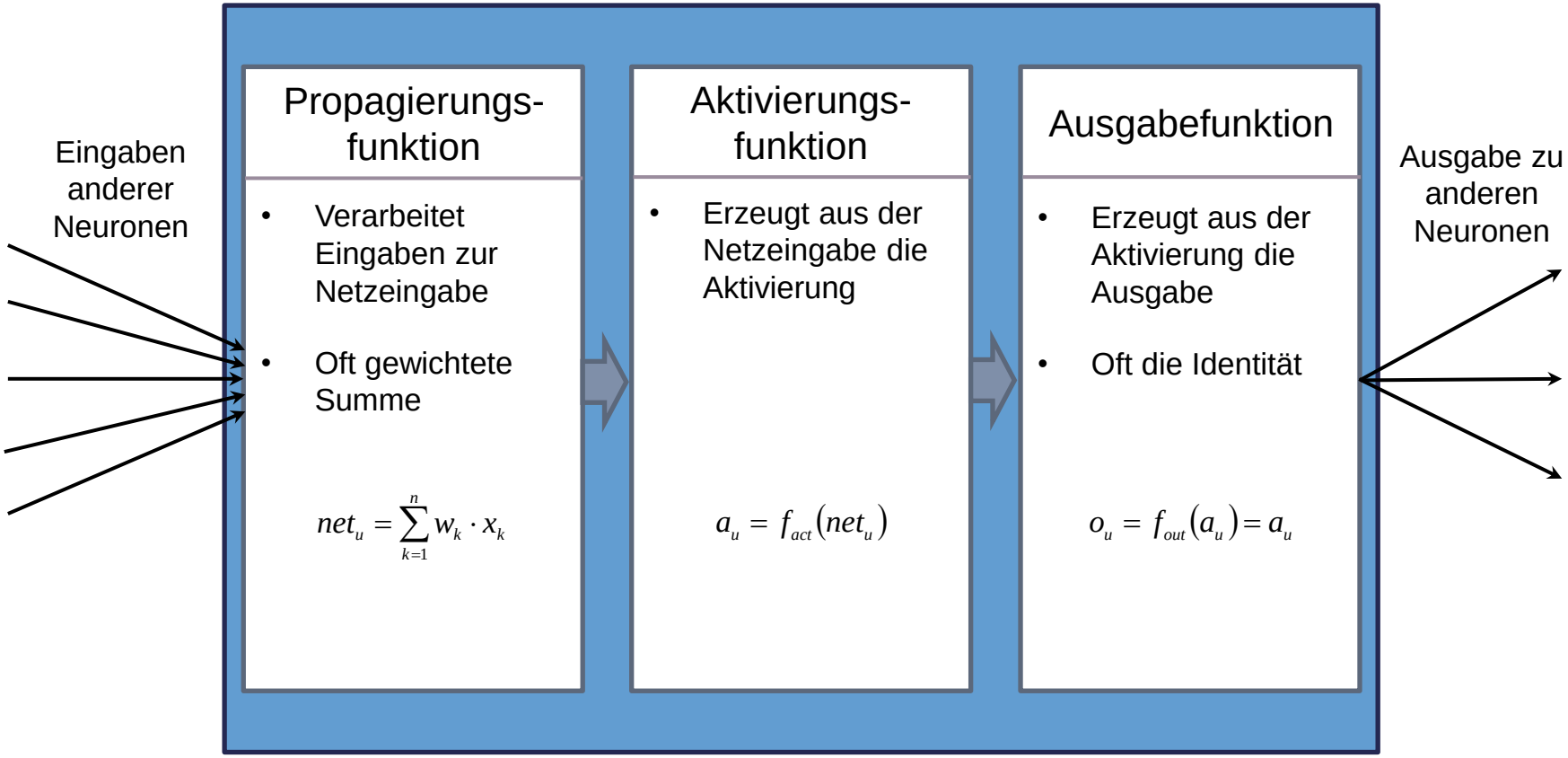
Künstliche Neuronale Netze - Grundidee

Natürliches Nervensystem als biologisches Vorbild



- Grundbausteine sind die Nervenzellen bzw. Neuronen
- Bestandteile eines Neurons: Zellkörper, Dendriten, Axon
- Verbindungen zwischen den Neuronen
- Verbindung (Synapse) zwischen Axon und Dendrite
- Signale fließen von Dendriten über den Zellkörper zu dem Axon
- Dann Weiterleitung an alle verbundenen Dendriten anderer Neuronen
- Endköpfchen: Umwandlung von elektrischer Ladung in chemische Substanz (Transmitter)
- Transport über elektrisches Potenzial an postsynaptische Membran
- Ankommende Signale werden aufsummiert und als elektrische Ladung an den Zellkern weitergeleitet
- Ab einem bestimmtet Schwellenwert „feuert“ das Neuron

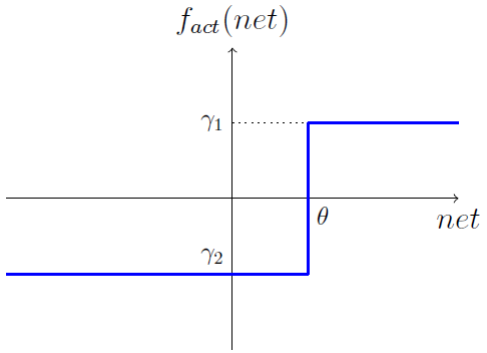
Arbeitsweise eines künstlichen Neurons



Aktivierungsfunktionen

- Binäre Schwellenwertfunktion:

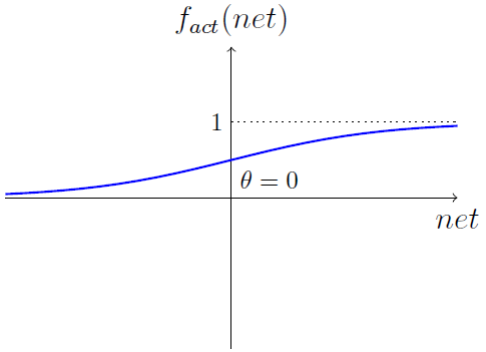
$$f_{act}(net) = \begin{cases} \gamma_1, & net \geq \theta_u \\ \gamma_2, & net < \theta_u \end{cases}$$



Binäre Schwellenwertfunktion

- Logistische Funktion:

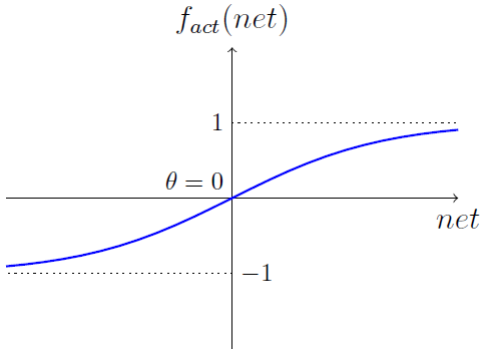
$$f_{act}(net) = \frac{1}{1 + e^{-(net - \theta_u)}}$$



Logistische Funktion

- Tangens Hyperbolicus:

$$f_{act}(net) = \frac{e^{(net - \theta_u)} - e^{-(net - \theta_u)}}{e^{(net - \theta_u)} + e^{-(net - \theta_u)}}$$

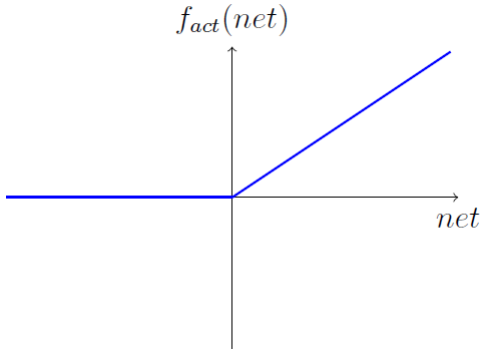


Tangens Hyperbolicus

Aktivierungsfunktionen

- Rectified Linear Unit (ReLU):

$$f_{act}(net) = \max(0, net - \theta_u)$$

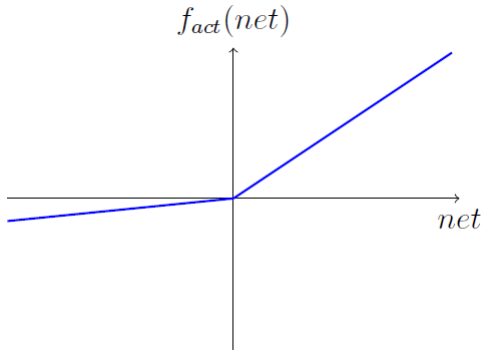


ReLU

- Leaky ReLU:

$$f_{act}(net) = \max(\alpha(net - \theta_u), net - \theta_u)$$

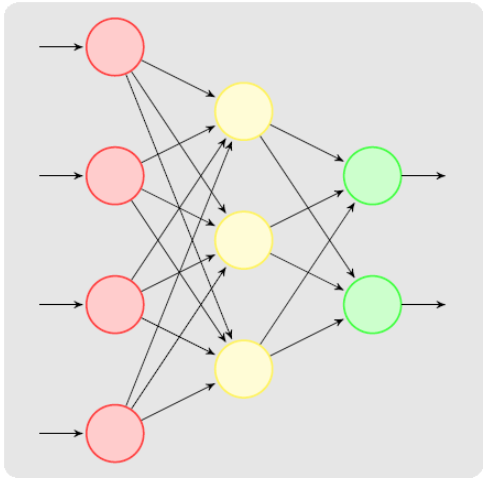
mit $0 < \alpha < 1$



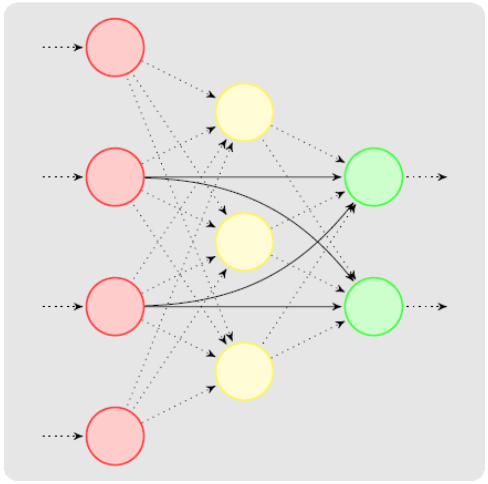
Leaky ReLU

FeedForward Neural Networks

- Input Layer (rot)
- Hidden Layer (gelb)
- Output Layer (grün)
- Verbindungen nur zu Neuronen in Richtung der Ausgabeschicht erlaubt
- Multilayer Perceptron
- Radiale Basisfunktionennetze

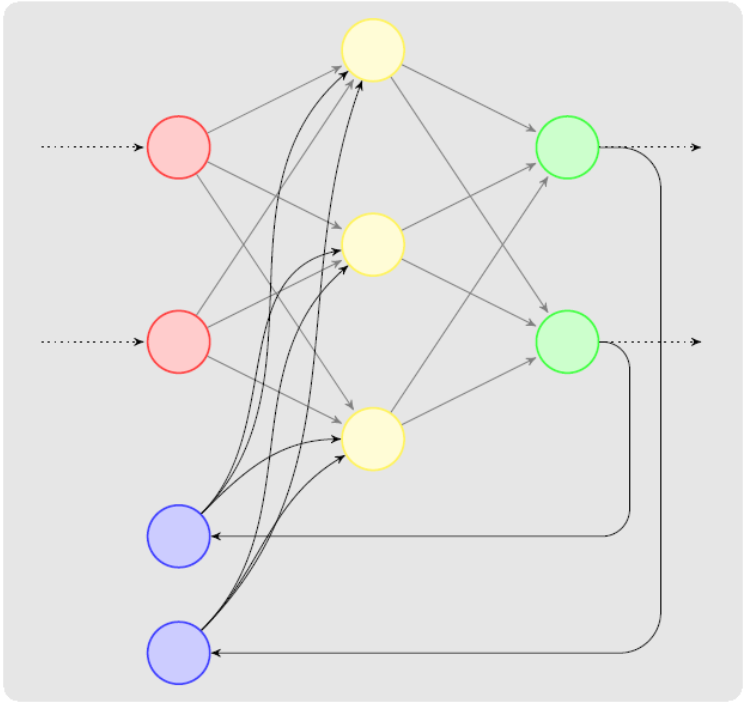


FeedForward-Netz ohne ShortCut-Verbindungen

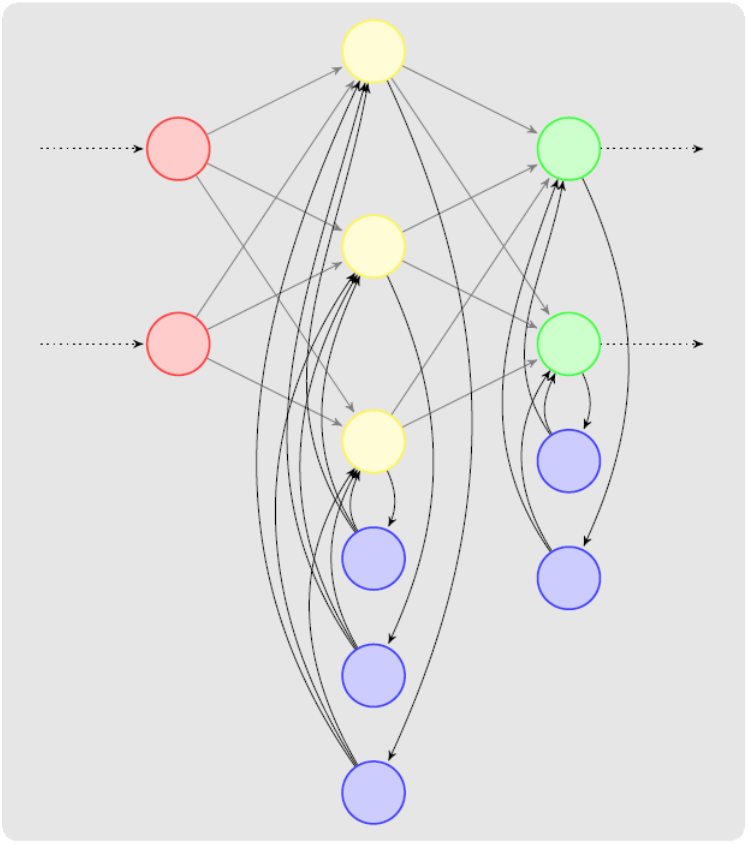


FeedForward-Netz mit ShortCut-Verbindungen

Weitere Netz-Architekturen



Jordan-Netz mit zwei Kontextneuronen (blau)



Elman-Netz mit drei Kontextneuronen in der versteckten Schicht und zwei Kontextneuronen in der Ausgabeschicht

Das Bias-Neuron

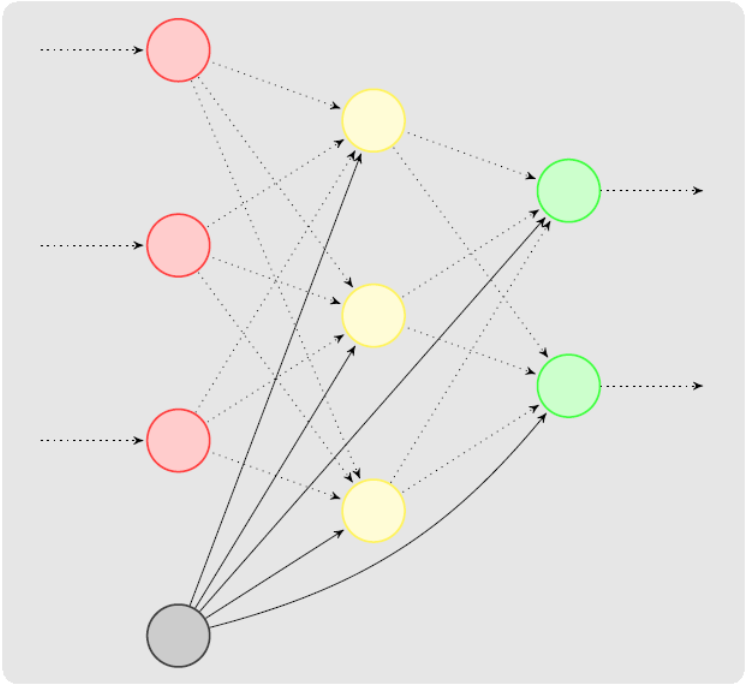
Schwellenwert als Verbindungsgewicht

- Neurone besitzen Schwellenwert θ
- Diese können während des Trainings verändert werden
- Biasneuron hat Verbindungen zu allen anderen Neuronen außer Inputneuronen
- Ausgabe des Biasneurons ist -1
- Im Fall der gewichteten Summe beträgt die Netzeingabe für ein Neuron u:

$$net_u - \theta_u = \sum_{i \in I} \left(w_{i,u} \cdot o_i \right) - \theta_u$$

$$= \sum_{i \in I \cup bias} \left(w_{i,u} \cdot o_i \right) \qquad \text{wobei } w_{bias,u} = \theta_u,\ o_{bias} = -1$$

- Behandlung der Schwellenwerte wie Gewichte



Training Künstlicher Neuronaler Netze

Arten des Trainings

Überwachtes Lernen (supervised learning)

- Existenz einer Trainingsmenge von Eingabemustern sowie deren korrekten Ergebnissen in Form der genauen Aktivierung sämtlicher Ausgabeneurone
- Vergleich der Ausgabe des Netzes für jedes Trainingsmuster mit der korrekten Lösung
- Anhand deren Differenz Veränderung der Gewichte
- Ziel: Generalisierung

Unüberwachtes Lernen (unsupervised learning)

- Trainingsmenge enthält nur Eingabemuster
- Ziel: Identifizierung von ähnlichen Mustern und Klassifizierung in Kategorien

Bestärkendes Lernen (reinforcement learning)

- Nach erfolgtem Durchlauf des Eingabemusters Lieferung eines Wahrheits- oder reellem Wert, der angibt, ob das Ergebnis richtig oder falsch war

Training Künstlicher Neuronaler Netze

Arten des Trainings

Offline

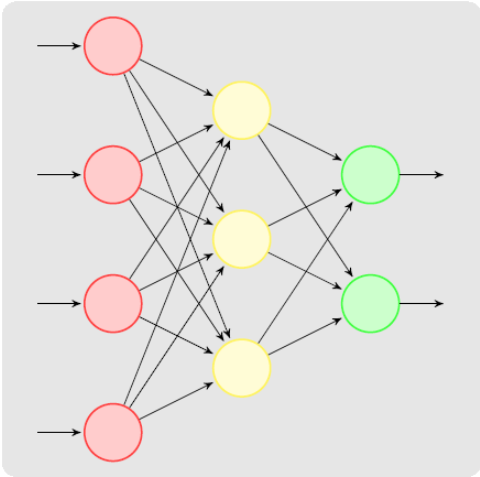
- Präsentierung einer Stapels (Batch) von Trainingsmustern
- Ermittlung des Gesamtfehlers mit Hilfe der Fehlerfunktion
- Modifizierung der Gewichte anhand des Gesamtfehlers

Online

- Modifizierung der Gewichte nach jedem Trainingsbeispiel

Training Künstlicher Neuronaler Netze

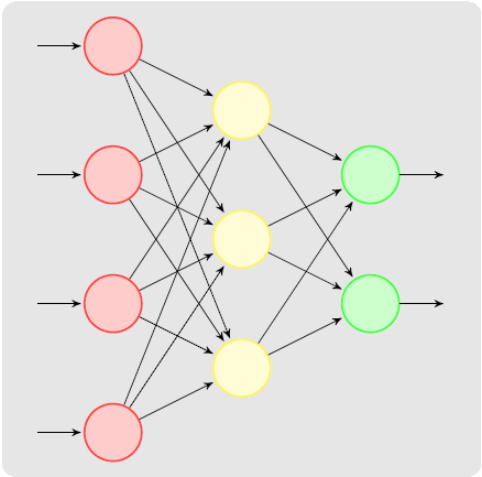
Trainingsmenge und Fehlermessung



- Eingabevektor: $p = (p_1, p_2, ..., p_n) \in \Re^n$
- Gewünschte Ausgabe: $t = (t_1, t_2, ..., t_m) \in \Re^m$
- Menge der Trainingsmuster: $P = \{(p^{(i)}, t^{(i)}) \in \Re^n \times \Re^m \mid i = 1, ..., N\}$
- Spezifischer Fehler:
  - Quadratische Abstand: $Err_p = \frac{1}{2} \sum_{\Omega \in O} (t_\Omega - y_\Omega)^2$
  - Euklidische Abstand: $Err_p = \sqrt{\sum_{\Omega \in O} (t_\Omega - y_\Omega)^2}$
  - Root-Mean-Square: $Err_p = \sqrt{\frac{\sum_{\Omega \in O} (t_\Omega - y_\Omega)^2}{|O|}}$
- Gesamtfehler: $Err = \sum_{p \in P} Err_p$

Training Künstlicher Neuronaler Netze

Backpropagation of Error



- Ziel des Trainings:

$$\forall p \in P : y_p \approx t_p \quad \text{bzw.} \quad \forall p \in P : Err_p \approx 0$$

- Betrachte den Gesamtfehler als Funktion der Gewichte:

$$Err : W \to \Re$$

- Veränderung der Gewichte:

$$\Delta W = -\eta \nabla Err(W)$$

- Für ein Ausgabeneuron Ω gilt:

$$\Delta w_{i,\Omega} = -\eta \frac{\partial Err(W)}{\partial w_{i,\Omega}} = -\eta \sum_{p \in P} \frac{\partial Err_p(W)}{\partial w_{i,\Omega}}$$

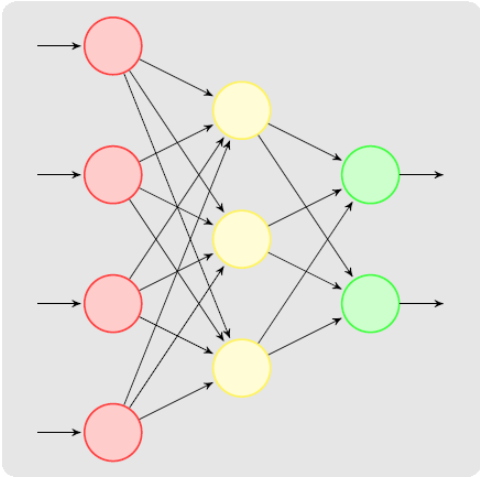
$$\frac{\partial Err_p(W)}{\partial w_{i,\Omega}} = \frac{\partial Err_p(W)}{\partial o_{p,\Omega}} \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}}$$

- Für den spezifischen Fehler $Err_p = \frac{1}{2} \sum_{\Omega \in O} (t_\Omega - y_\Omega)^2$ erhält man

$$\frac{\partial Err_p(W)}{\partial w_{i,\Omega}} = -(t_{p,\Omega} - o_{p,\Omega}) \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}} = -\delta_{p,\Omega} \cdot \frac{\partial o_{p,\Omega}}{\partial w_{i,\Omega}}$$

Training Künstlicher Neuronaler Netze

Backpropagation of Error



$$o_{p,\Omega} = a_{p,\Omega} = net_{p,\Omega} = \sum_{j \in I} \left(w_{j,\Omega} \cdot o_{p,j} \right)$$

$$\Rightarrow \frac{\partial Err_p(W)}{\partial w_{i,\Omega}} = -\delta_{p,\Omega} \cdot \frac{\partial \sum_{j \in I} \left(w_{j,\Omega} \cdot o_{p,j} \right)}{\partial w_{i,\Omega}}$$

$$\frac{\partial \sum_{j \in I} \left(w_{j,\Omega} \cdot o_{p,j} \right)}{\partial w_{i,\Omega}} = o_{p,i}$$

$$\Rightarrow \frac{\partial Err_p(W)}{\partial w_{i,\Omega}} = -\delta_{p,\Omega} \cdot o_{p,i}$$

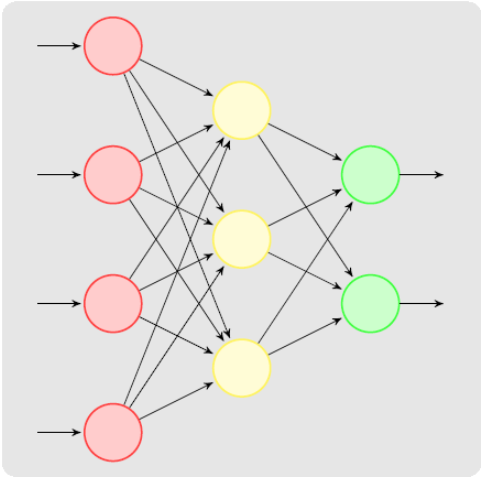
$$\Rightarrow \Delta w_{i,\Omega} = \eta \sum_{p \in P} o_{p,i} \cdot \delta_{p,\Omega}$$

- In der Online-Variante werden die Gewichte nach jedem Trainingsbeispiel modifiziert:

$$\Delta w_{i,\Omega} = \eta \cdot o_i \cdot \delta_\Omega$$

Training Künstlicher Neuronaler Netze

Backpropagation of Error



- Betrachte nun ein inneres Neuron h
- $K =$ Menge der Vorgängerneurone k
- $L =$ Menge der Nachfolgerneurone l
- Man erhät:

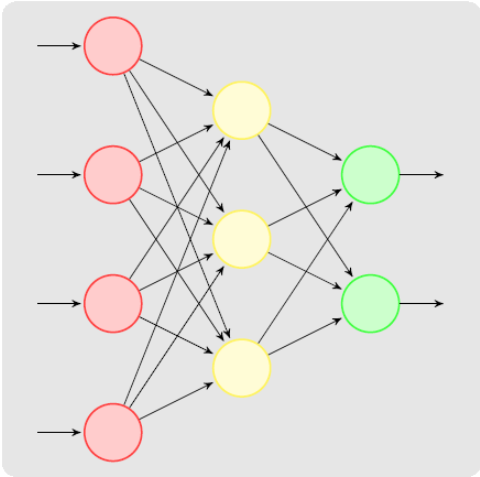
$$\frac{\partial Err(W)}{\partial w_{k,h}} = \underbrace{\frac{\partial Err}{\partial net_h}}_{=-\delta_h} \cdot \frac{\partial net_h}{\partial w_{k,h}}$$

$$\delta_h = -\frac{\partial Err}{\partial net_h} = -\frac{\partial Err}{\partial o_h} \cdot \frac{\partial o_h}{\partial net_h}$$

- Für den zweiten Faktor gilt: $\frac{\partial o_h}{\partial net_h} = \frac{\partial f_{act}(net_h)}{\partial net_h} = f'_{act}(net_h)$
- Für den ersten Faktor gilt: $-\frac{\partial Err}{\partial o_h} = -\frac{\partial Err(net_{l_1}, ..., net_{l_{|L|}})}{\partial o_h} = -\sum_{l \in L} \left(\frac{\partial Err}{\partial net_l} \cdot \frac{\partial net_l}{\partial o_h} \right)$

Training Künstlicher Neuronaler Netze

Backpropagation of Error



$$-\frac{\partial Err}{\partial o_h} = -\frac{\partial Err(net_{l_1},...,net_{l_{|L|}})}{\partial o_h} = -\sum_{l\in L}\left(\frac{\partial Err}{\partial net_l}\cdot\frac{\partial net_l}{\partial o_h}\right)$$

$$\frac{\partial net_l}{\partial o_h} = \frac{\partial \sum_{j\in H} w_{j,l}\cdot o_j}{\partial o_h} = w_{h,l}$$

$$-\frac{\partial Err}{\partial net_l} = \delta_l$$

$$\Rightarrow -\frac{\partial Err}{\partial o_h} = \sum_{l\in L}\delta_l w_{h,l}$$

$$\Rightarrow \delta_h = f'_{act}(net_h)\cdot\sum_{l\in L}(\delta_l\cdot w_{h,l})$$

- Allgemein gilt also: $\Delta w_{k,h} = \eta\cdot o_k\cdot\delta_h$

$$\delta_h = \begin{cases} f'_{act}(net_h)\cdot(t_h - y_h) & (h \text{ außen}) \\ f'_{act}(net_h)\cdot\sum_{l\in L}(\delta_l\cdot w_{h,l}) & (h \text{ innen}) \end{cases}$$

Training Künstlicher Neuronaler Netze

Backpropagation of Error

Algorithmus 4.2 Backpropagation (online)

Initialisiere die Gewichte w, die Lernrate η und die Anzahl an Epochen $t = 0$;
while Stoppkriterien nicht erfüllt **do**
  $Err = 0$;
  for all $p \in P$ **do**
    Bestimme den Ausgabevektor y_p;
    Bestimme das Delta für die Ausgabeschicht;
    Bestimme das Delta für die versteckten Schichten;
    Modifiziere die Gewichte
    $Err = Err + Err_p$;
  end for
  $t = t + 1$;
end while

Training Künstlicher Neuronaler Netze

Backpropagation of Error

Algorithmus 4.3 Backpropagation (offline)

Initialisiere die Gewichte w, die Lernrate η und die Anzahl an Epochen $t = 0$;
Unterteile P in K Batches B_k $(P = \{B_1, ..., B_K\})$;
while Stoppkriterien nicht erfüllt **do**
  $Err = 0$;
  for $i = 1, ...k$ **do**
    for all $w \in W$ **do**
      $\Delta w = 0$;
    end for
    for all $p \in B_i$ **do**
      Bestimme den Ausgabevektor y_p;
      for all $w \in W$ **do**
        Aktualisiere Δw;
      end for
      $Err = Err + Err_p$;
    end for
    for all $w \in W$ **do**
      $w = w + \Delta w$;
    end for
    $t = t + 1$;
  end for
end while

Training Künstlicher Neuronaler Netze

Backpropagation of Error

- Gewichtsmodifikation proportional zur Lernrate η
- Wahl von η sehr entscheidend für das Verhalten von Backpropagation
- Bei zu großem η können enge Täler in der Fehlerfunktion übersprungen werden
- Bei zu kleinem η lernt das Netz in flachen Gebieten der Fehlerfunktion sehr langsam
- Wahl von η hängt maßgeblich von Problem, Netz und Trainingsdaten ab
- Weitere Ansätze:
  - Halte die Lernrate während des Trainings variabel
  - Definiere für jede Gewichtsschicht eine eigene Lernrate (nahe der Eingabeschicht größer als diejenigen nahe der Ausgabeschicht)
  - Hinzufügen eines Momentumterms: $\Delta w_{i,j}(t) = \eta o_i \delta_j + \alpha \cdot \Delta w_{i,j}(t-1)$

    mit $\alpha \in [0.5, 0.95]$

Training Künstlicher Neuronaler Netze

Resilient Backpropagation (Rprop)

- Jedes einzelne Gewicht $w_{i,j}$ hat seine eigene Lernrate $\eta_{i,j}$
- Die Lernraten $\eta_{i,j}$ werden von Rprop selbst festgelegt
- In jedem Zeitschritt werden die Lernraten von Rprop angepasst
- Der Betrag der Gewichtsmodifikation $\Delta w_{i,j}$ entspricht direkt der zugehörigen, automatisch angepassten Lernrate $\eta_{i,j}$

$$\Delta w_{i,j}(t) = \begin{cases} -\eta_{i,j}(t), & \text{wenn } g(t) > 0 \\ +\eta_{i,j}(t), & \text{wenn } g(t) < 0 \\ 0, & \text{wenn } g(t) = 0 \end{cases} \qquad \text{wobei } g(t) = \frac{\partial Err}{\partial w_{i,j}}(t)$$

$$\eta_{i,j}(t) = \begin{cases} \eta^{\uparrow}\eta_{i,j}(t-1), & \text{wenn } g(t)g(t-1) > 0 \\ \eta^{\downarrow}\eta_{i,j}(t-1), & \text{wenn } g(t)g(t-1) < 0 \\ \eta_{i,j}(t-1), & \text{wenn } g(t)g(t-1) = 0 \end{cases} \qquad \text{mit } \begin{array}{l} \eta^{\uparrow} \in [1.05, 1.2] \\ \eta^{\downarrow} \in [0.5, 0.7] \end{array}$$

Initialkonfiguration

Netzarchitektur

- Wahl Anzahl versteckter Schichten
- Wahl Anzahl Neuronen in den versteckten Schichten
- Anzahl der Eingabe- und Ausgabeneurone sind durch die Problemstellung vorgegeben
- Mit nur einer versteckten Schicht lassen sich beliebige Funktionen approximieren
- Mehr versteckte Schichten können die Trainierbarkeit des Netzes verbessern
- Jede weitere Schicht erzeugt weitere Nebenminima der Fehlerfunktion
- Möglicher Ansatz: Beginne mit einer versteckten Schicht und erhöhe die Anzahl der Schichten, falls der erste Ansatz zu keinem akzeptablen Ergebnis führen sollte

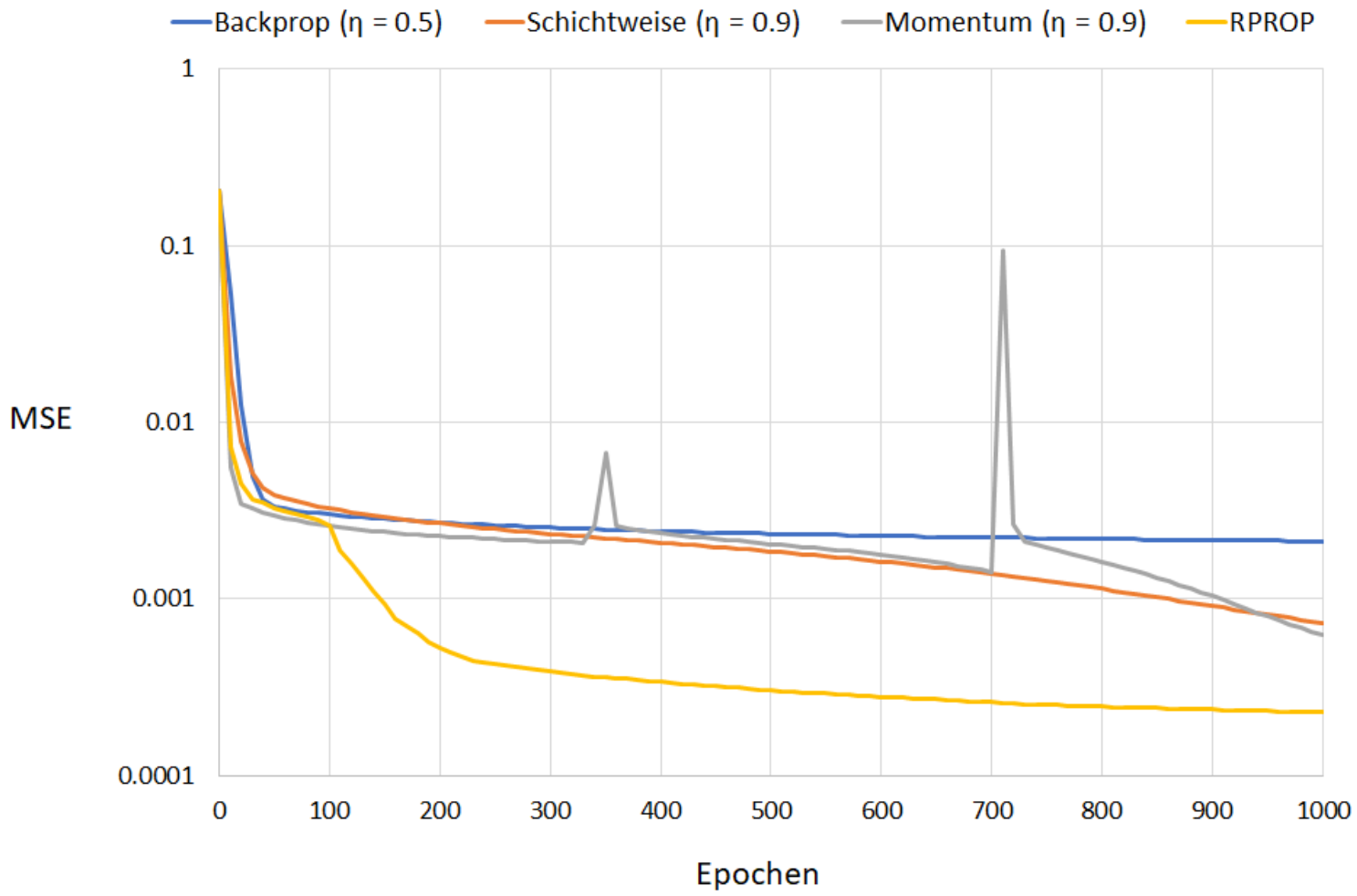
Initialkonfiguration

Wahl der Aktivierungsfunktion / Initialisierung der Gewichte

- Inputneurone nicht informationsverarbeitend
- Üblicherweise ist für alle versteckten Neurone die Aktivierungsfunktion gleich
- Für den Zweck der Funktionsapproximation: lineare Aktivierungsfunktion für die Ausgabeschicht
- Tiefere Netze lassen sich mit der ReLU-Funktion besser trainieren
- Initialisiere die Gewichte mit kleinen, zufälligen Werten

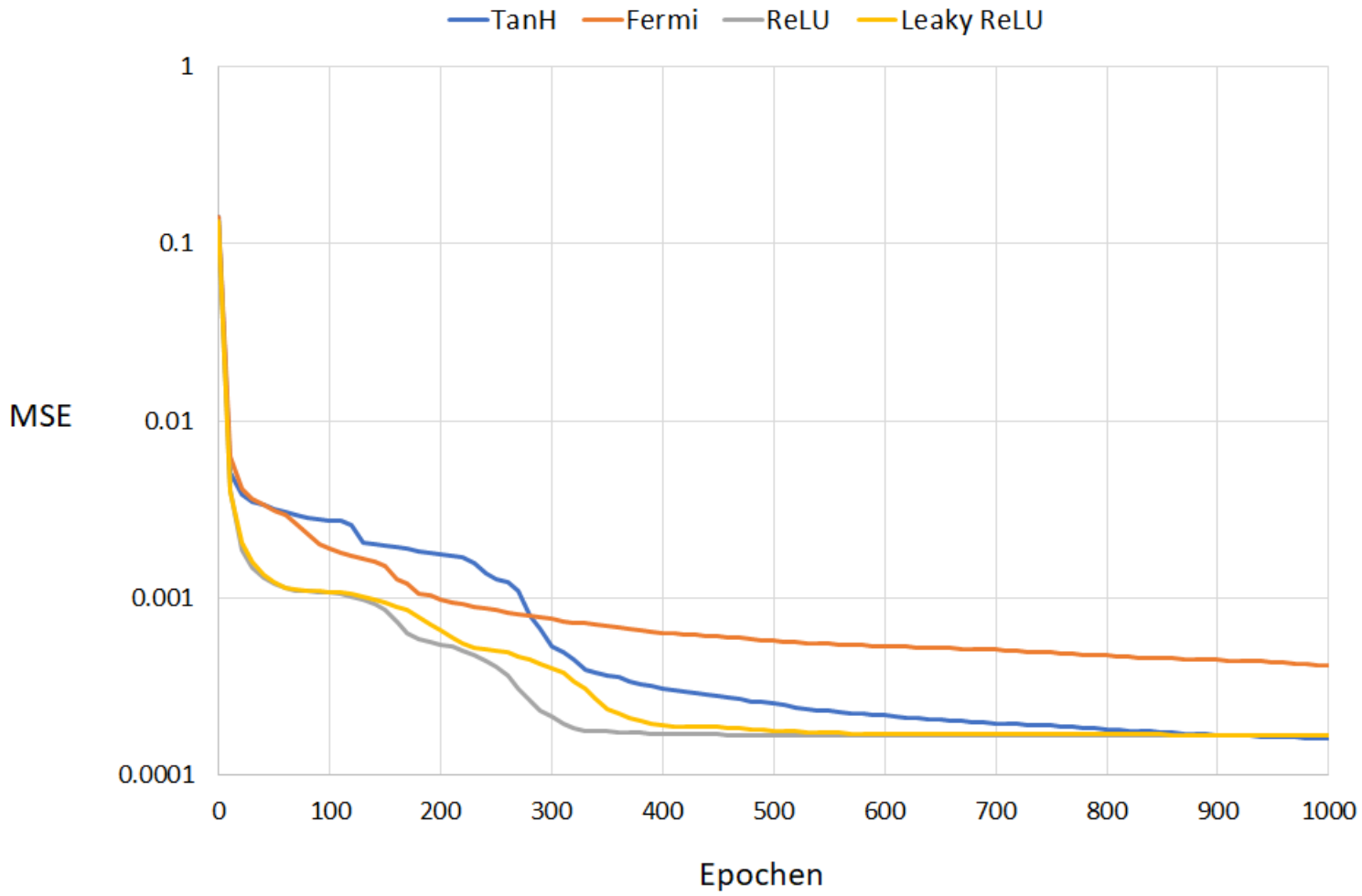
Praxis an LSCE-Daten

Vergleich Trainigsmethoden



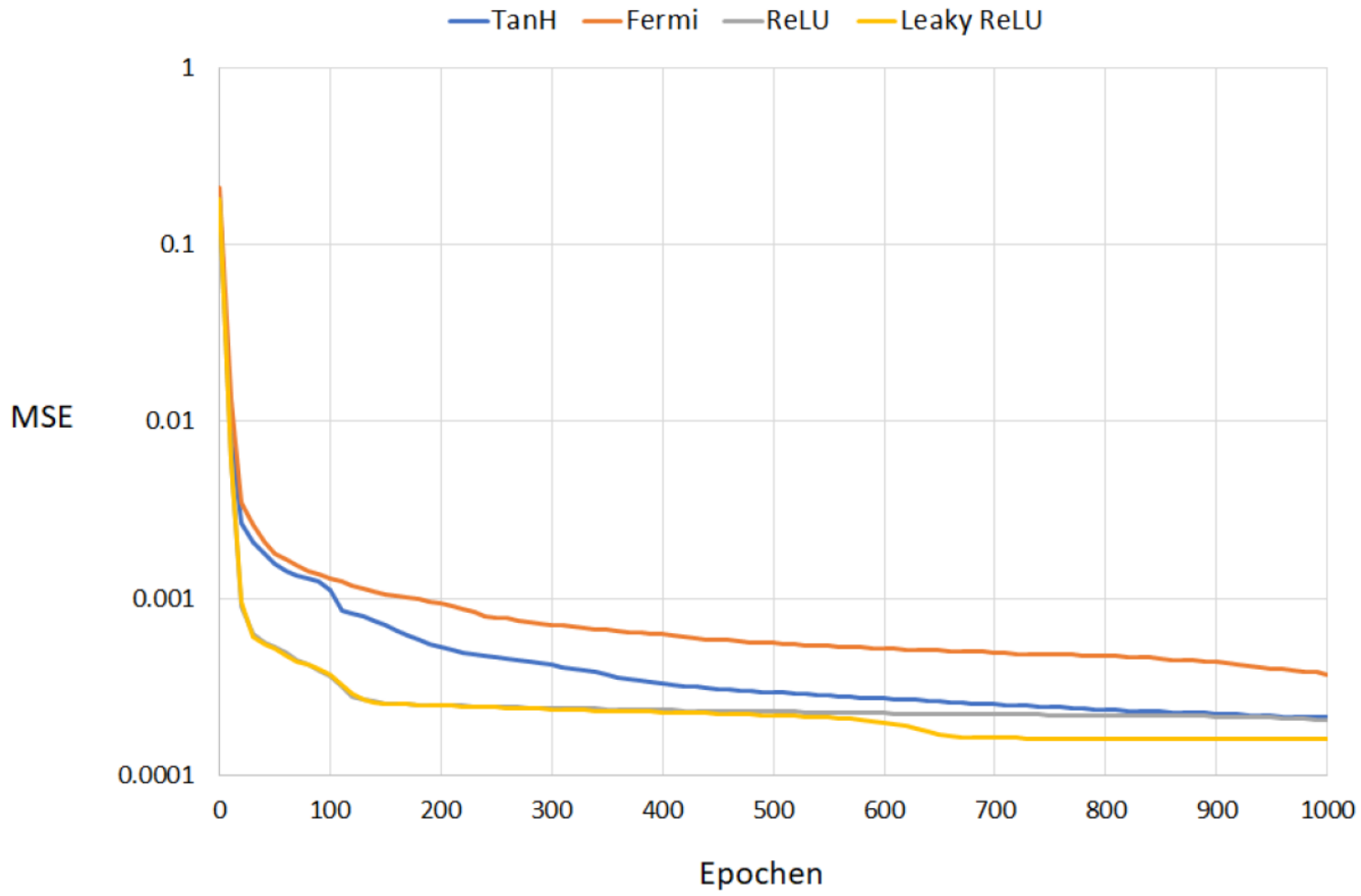
Praxis an LSCE-Daten

Vergleich Aktivierungsfunktionen Unternehmen A



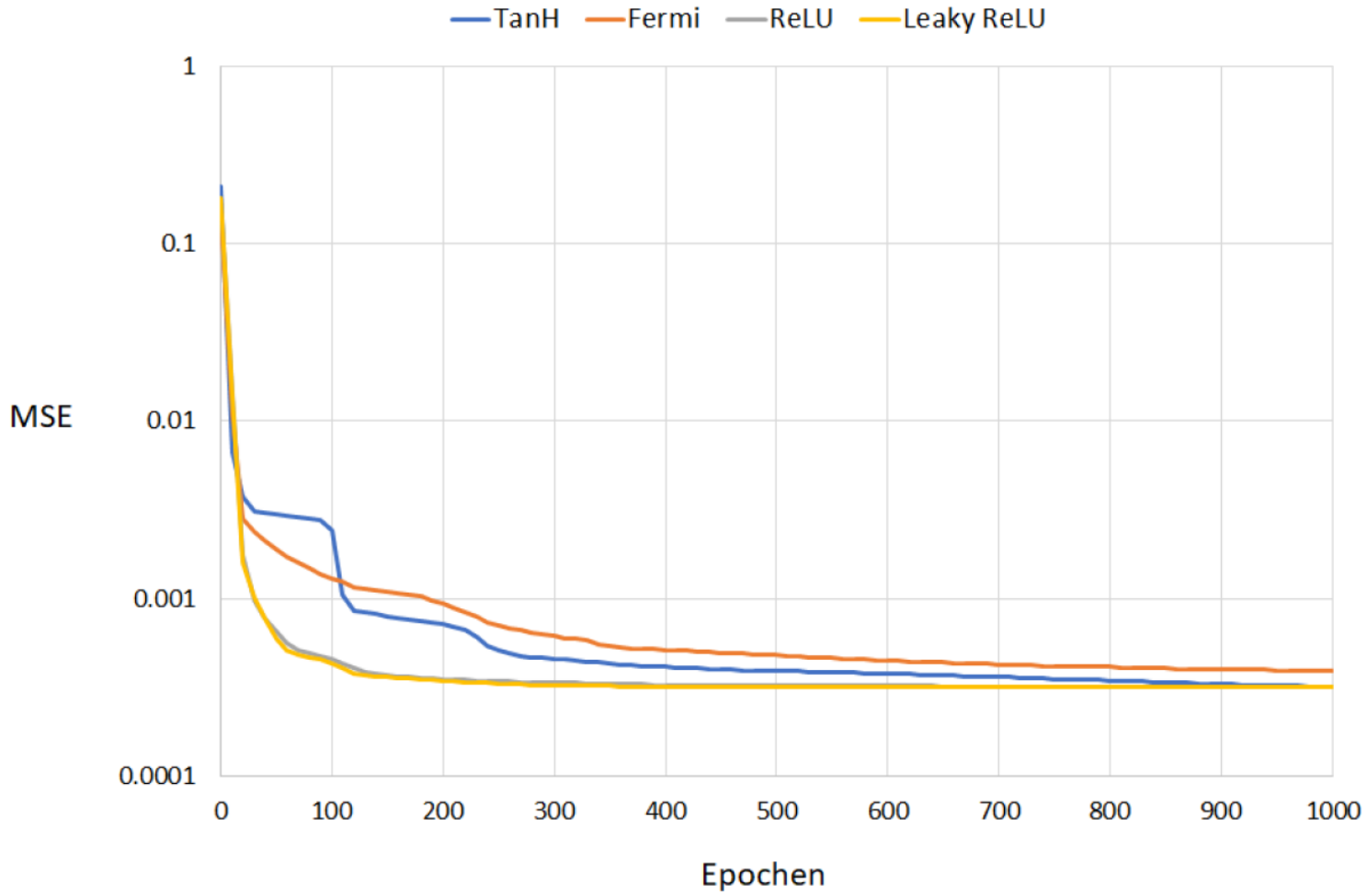
Praxis an LSCE-Daten

Vergleich Aktivierungsfunktionen Unternehmen B



Praxis an LSCE-Daten

Vergleich Aktivierungsfunktionen Unternehmen C



Praxis an LSCE-Daten

Ergebnisse

- Pro Netzarchitektur 10 Durchläufe des Trainings
- In jedem Durchlauf unterschiedliche Initialisierung der Gewichte
- Für NNs mit einer versteckten Schicht: Aktivierungsfunktion = Tangens Hyperbolicus
- Für NNs mit zwei versteckten Schichten: Aktivierungsfunktion = ReLU
- Gütekriterium: Weighted Relative Error (WRE):

$$WRE = \sum_{i=1}^{N_V} \left(\underbrace{\frac{|y_i - f(x_i)|}{y_i}}_{\text{absoluter relativer Fehler}} \times \underbrace{\frac{|y_i|}{\sum_{j=1}^{N_V} |y_j|}}_{\text{Gewicht des Fehlers}} \right) = \frac{\sum_{i=1}^{N_V} |y_i - f(x_i)|}{\sum_{j=1}^{N_V} |y_j|}$$

	Polynom	5	10	15	20	5-5	10-10
A	0,27%	0,53%	0,27%	0,24%	0,19%	0,22%	0,17%
B	0,92%	1,51%	0,88%	0,87%	0,91%	1,33%	0,79%
C	0,30%	0,58%	0,31%	0,29%	0,29%	0,38%	0,29%

Vielen Dank für Ihre Aufmerksamkeit