MUSTERERKENNUNG

Vorlesung im Sommersemester 2017

Prof. E.G. Schukat-Talamazzini

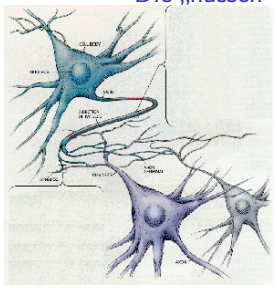
Stand: 17. März 2017

Teil VIII

Künstliche Neuronale Netze

Das menschliche Zentralnervensystem

Die „nassen“ Neuronen der anthropomorphen Blaupause



Das Neuron

ist die elementare Verarbeitungseinheit.

- ca. 10^{12} Nervenzellen
- ca. 10^{14} Verbindungen

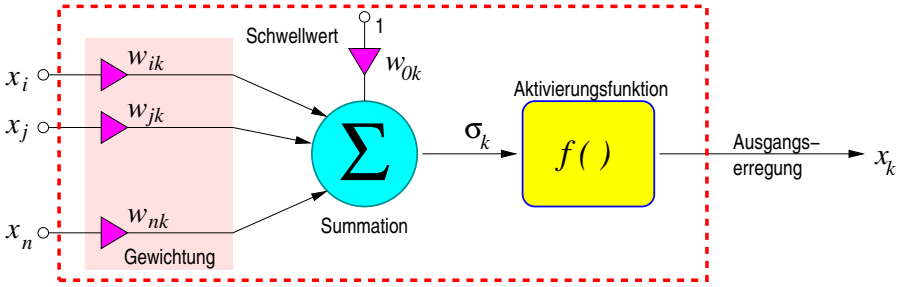
Axon Macht die **Grundaktivität** einer Nervenzelle in Form elektrischer Impulse verfügbar

Synapse Übermittlung an andere Neuronen über **synaptische** Verbindungen unterschiedlicher Stärke

Feuern Überschreitet die gesammelte **Eingangserregung** einen bestimmten Betrag, führt dies zu einer Aktivitätsänderung *(erhöhte Impulsfrequenz)*

Das Modellneuron

Grundbaustein des Paradigmas *massiv-paralleler Verarbeitung*



Parallel-Distributed Processing

Ein **Künstliches Neuronales Netz** (KNN) ist ein Arrangement von *Modellneuronen* und ihren *Verbindungen*.

Dynamisches Verhalten

bestimmt durch

- Aktivierungsbegriff
- Kombination der Eingänge
- Aktivierungsfunktion
- Verschaltungstopologie

Neuronales Aktivierungsniveau

Das Ausgabesignal des Modellneurons

Ausgangserregung

Mit $x_k(t)$ bezeichnen wir die Ausgangserregung des k-ten Modellneurons zum Zeitpunkt t.

Der **Gesamtsystemzustand** ist durch $\boldsymbol{x}(t)$ charakterisiert.

- **Wertebereich**
  Feuern oder *ruhen* $x_k(t) \in \{1, 0\}$
  Quantitatives *Erregungspotenzial* $x_k(t) \in \mathbb{R}$
- **Aktivierung**
  Schwellenwert: Neuron $\#k$ aktiv gdw. $x_k(t) \geq \theta$
  Feuerrate: Neuron $\#k$ aktiv gdw. Anzahl Spikes/Zeiteinheit $\geq \theta$
- **Zeitraster**
  dynamisch — stetige Zeitskala $t \in \mathbb{R}$
  dynamisch — diskrete Zeitskala $t \in \mathbb{Z}$
  nicht-dynamisch $x_k(t) \equiv x_k$

Neuronales Aktivierungsniveau

Die Eingabesignale des Modellneurons

Gewichtung der Eingangserregungen

Modellierung der Stärke synaptischer Verbindungen

$$\boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ \vdots & & \ddots & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{pmatrix}$$

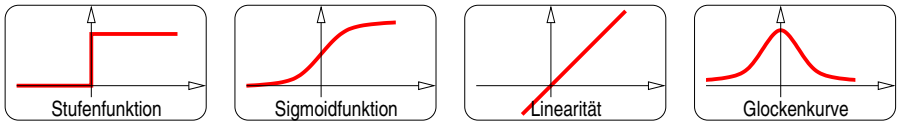
Exzitatorische Synapsen $w_{ij} > 0$
Inhibitorische Synapsen $w_{ij} < 0$

Kombination der Eingangserregungen

Die gesammelte Eingangserregung des k-ten Modellneurons (zur Zeit t) wird mit σ_k bzw. mit $\sigma_k(t)$ bezeichnet und akkumuliert gemäß:

$$\sigma_k(t) = \begin{cases} \sum_j w_{jk} \cdot x_j(t) & \text{(Linearkombination)} \\ \sum_j (w_{jk} - x_j(t))^2 & \text{(Unähnlichkeit)} \end{cases}$$

Neuronale Aktivierungsfunktionen



Lineare Aktivierung

$$x_k = a \cdot \sigma_k + b$$

lineare Netzwerke

Zielwertaktivierung

$$x_k = \exp\left\{-C \cdot (\sigma_k - \theta_k)^2\right\}$$

selbstorganisierende Karten

Sprungfunktion

$$x_k = \begin{cases} 1 & \sigma_k > \theta_k \\ 0 & \sigma_k \leq \theta_k \end{cases}$$

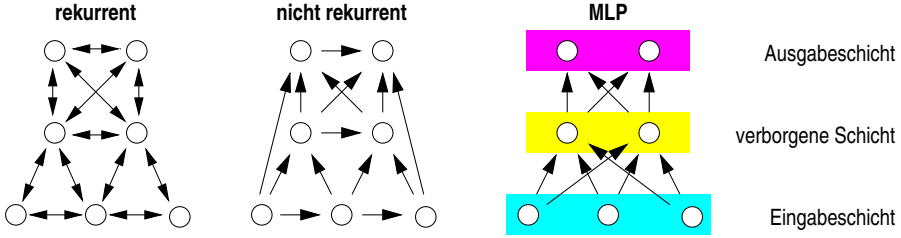
klassisches Perzeptron

Sigmoidfunktion

$$x_k = \frac{1}{1 + \exp(-\sigma_k)}$$

Mehrschichtenperzeptron

Neuronale Verschaltung



Verbindungstopologie

Die Gewichtmatrix $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ definiert einen **gerichteten Graphen** $(\mathcal{N}, \mathcal{E})$ über dem Neuronenvorrat $\mathcal{N} = \{1, \ldots, N\}$ durch:

$$j \prec k \Leftrightarrow (j,k) \in \mathcal{E} \Leftrightarrow w_{jk} \neq 0 \,, \qquad (\forall j,k \in \mathcal{N})$$

Rekurrente Netzwerke

Die Verschaltung besitzt Zyklen.

➪ *dynamisches System*

Feed-Forward-Netzwerke

Die Verschaltung ist zyklenfrei.

➪ *endliche Berechnung*

Feed-Forward-Netzwerke

Nichtrekurrente (zyklenfreie) künstliche neuronale Netze

Definition

Ein KNN mit Gewichtmatrix $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ und zyklenfreier neuronaler Verknüpfungsrelation '$\prec$' heißt **nichtrekurrentes KNN** oder **Vorwärtsnetz**.

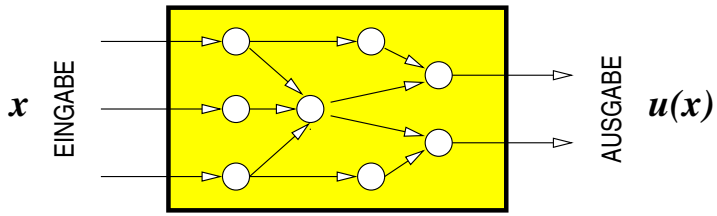
Die Menge $\mathcal{N}$ aller Netzknoten zerfällt in die Teilmengen

$$\begin{aligned} \mathcal{N}_E &= \{k \in \mathcal{N} \mid \neg\exists j : j \prec k\} \\ \mathcal{N}_A &= \{j \in \mathcal{N} \mid \neg\exists k : j \prec k\} \\ \mathcal{N}_H &= \mathcal{N} \setminus (\mathcal{N}_E \cup \mathcal{N}_A) \end{aligned}$$

der **Eingabe-**, der **Ausgabe-** bzw. der **verborgenen** Neuronen.

Feed-Forward-Netzwerke

Serielle Traversierbarkeit nichtrekurrenter KNNs



Fakt

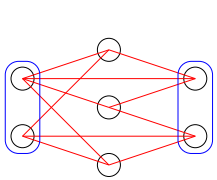
Die Knoten eines zyklenfreien KNN lassen sich in $\prec$-verträglicher Weise ***linear anordnen****, d.h., es gilt*

$$\forall j,k \in \mathcal{N}: \qquad j \prec k \;\Rightarrow\; j < k$$

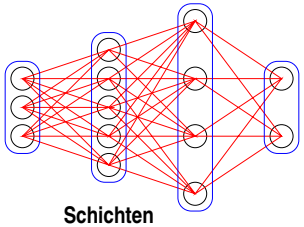
nach entsprechender Umnummerierung.

Spielarten nichtrekurrenter Netzwerke

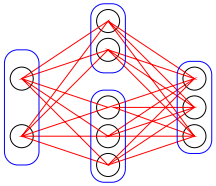
Neuronenschichten & Verbindungsebenen



nichtrekurrent
keine Anregungszyklen



Schichten
volle Verdrahtung zwischen benachbarten Schichten



Module
zyklenfreier Modulgraph
Metakanten voll verdrahtet

Definition

Zerfällt die Knotenmenge $\mathcal{N}$ eines zyklenfreien KNNs in eine Folge $\mathcal{N}_0, \ldots, \mathcal{N}_L$ disjunkter Schichten mit der Eigenschaft

$$w_{jk} \neq 0 \quad \Leftrightarrow \quad (\exists \ell)\; j \in \mathcal{N}_{\ell-1} \wedge k \in \mathcal{N}_\ell$$

(vollständige Verschaltung benachbarter Schichten), so sprechen wir von einem **Mehrschichtenperzeptron** (MLP = *multi-layer perceptron*).

Berechnungsprozeß in nichtrekurrenten Netzwerken

Zyklenfreie KNNs kommen ohne explizite *Zeitrechnung* aus

Lemma

Die Ausgangserregungen $x_k(t)$ der Neuronen eines Mehrschichtenperzeptrons sind nicht von der Zeit, sondern nur von den Zuständen $\boldsymbol{x}_E = (x_k | k \in \mathcal{N}_E)^\top$ abhängig und berechnen sich durch die Vorwärtsrekursion

$$x_k = f\left(\sum_{j \prec k} w_{jk} \cdot x_j\right) \quad \textit{bzw.} \quad x_k = f\left(\sum_{j \in \mathcal{N}_{\ell-1}} w_{jk} \cdot x_j\right)$$

für alle $\begin{cases} \textit{FFN}: & k = 1, 2, \ldots, N \\ \textit{MLP}: & k \in \mathcal{N}_\ell,\ \ell = 1..L \end{cases}$ *mit der Aktivierungsfunktion $f(\cdot)$.*

Das Netzwerk definiert folglich eine ***nichtlineare Abbildung***

$$\boldsymbol{u}: \begin{cases} \mathbb{R}^{|\mathcal{N}_E|} & \to & \mathbb{R}^{|\mathcal{N}_A|} \\ \boldsymbol{x}_E & \mapsto & \boldsymbol{u}(\boldsymbol{x}_E) = \boldsymbol{x}_A = (x_k | k \in \mathcal{N}_A)^\top \end{cases} .$$

Klassisches Perzeptron

Rosenblatt 1962

Definition

Ein einschichtiges KNN mit N Eingabeknoten, K Ausgabeknoten, der Gewichtmatrix $\boldsymbol{W} \in \mathbb{R}^{N \times K}$ und dem Systemverhalten

$$y_k(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{j=1}^{N} w_{jk} \cdot x_j\right), \quad k = 1, \ldots, K$$

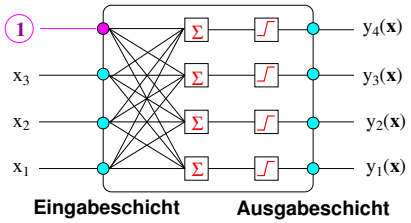
heißt **einstufiges** (oder klassisches) **Perzeptron**.

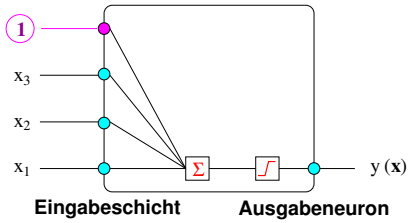
Bemerkungen

1. Das Perzeptron kann zu Klassifikationszwecken für N-dimensionale Merkmalvektoren aus K Musterklassen herangezogen werden; Prüfgrößen sind die Aktivierungsniveaus $y_k(\boldsymbol{x})$ der Ausgabeneuronen.
2. Das Verhalten der Ausgabeneuronen eines Perzeptrons ist vollständig entkoppelt; es genügt also, Perzeptren mit **einem Ausgabeneuron** zu betrachten.

Klassisches Perzeptron mit einem Ausgabeneuron

ADALINE — *adaptive linear engine*





Schwellenwertneuron

Feuerschwelle als Gewicht des 1-Neurons:

$$y = \operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{x} - \theta) = \left(\theta, \boldsymbol{w}^\top\right) \cdot \begin{pmatrix} -1 \\ \boldsymbol{x} \end{pmatrix}$$

Überwachtes Lernen

Ideale Zielgrößenvorgabe $y^* \in \{-1, +1\}$ lautet:

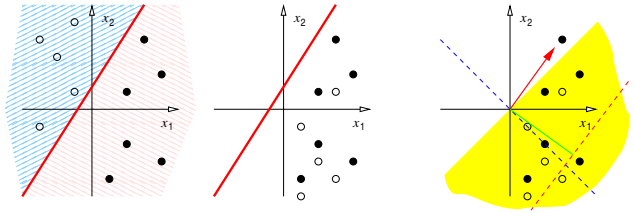
$$\operatorname{sign}(\boldsymbol{w}^\top \boldsymbol{x}) \stackrel{!}{=} y^*$$

Mit der Hilfsgröße $\boldsymbol{z} := y^* \boldsymbol{x}$ übersetzt in die Bedingung

$$\boldsymbol{w}^\top \boldsymbol{z} \stackrel{!}{\geq} 0$$

Die Perzeptron-Lernregel

Stapelweise versus instantane Auffrischung der Gewichte



Gradientenabstieg

$w_{jk} \leftarrow w_{jk} - \eta \Delta w_{jk}$

Instantan:

$$\Delta w_{jk} = \frac{\partial \varepsilon(\boldsymbol{z})}{\partial w_{jk}}$$

Stapelweise:

$$\Delta w_{jk} = \sum_{\boldsymbol{z} \in \omega} \frac{\partial \varepsilon(\boldsymbol{z})}{\partial w_{jk}}$$

Auffrischungsvorschrift

Korrektur von $\boldsymbol{w}$ proportional zur *negativen Projektion* von $\boldsymbol{w}$ auf $\boldsymbol{z}$:

$$\boldsymbol{w} \leftarrow \begin{cases} \boldsymbol{w} & \boldsymbol{w}^\top \boldsymbol{z} > 0 \\ \boldsymbol{w} - \eta \cdot (\boldsymbol{w}^\top \boldsymbol{z}) \cdot \boldsymbol{z} & \text{sonst} \end{cases}$$

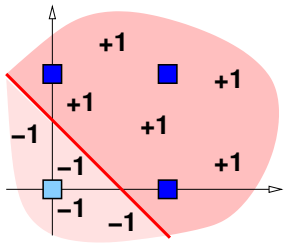
Der Koeffizient η heißt **Lernrate**.

Was ein Perzeptron lernen kann

... und was ein Perzeptron leider nicht lernen kann

Inklusive ODER-Funktion

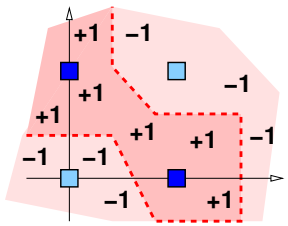
$$y(x_1, x_2) = \begin{cases} -1 & x_1 = 0 = x_2 \\ +1 & \text{sonst} \end{cases}$$



affin separierbare Muster

Exklusive ODER-Funktion

$$y(x_1, x_2) = \begin{cases} -1 & x_1 = x_2 \\ +1 & x_1 \neq x_2 \end{cases}$$



unmöglich durch Gerade zu trennen

Bemerkung

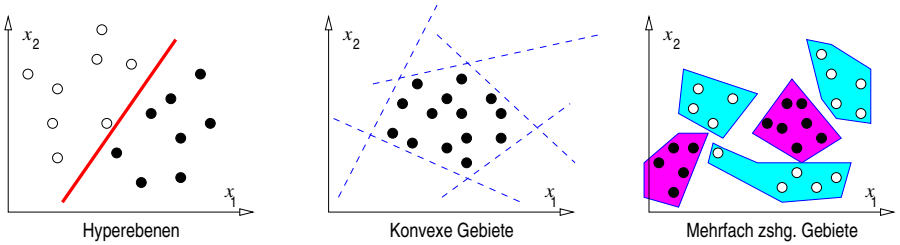
Nach dieser Hiobsbotschaft (1962) ruhte die Perzeptronforschung in Frieden bis zur (Wieder-)Entdeckung des Backpropagation-Algorithmus (1972/85).

Das Mehrschichtenperzeptron (MLP)

Asymptotisch universeller Funktionsapproximator

Satz

*Mehrschichtennetzwerke mit nichtlinearer Aktivierungsfunktion (z.B. Signum oder Sigmoid) und mindestens zwei verborgenen Neuronenschichten ($L \geq 3$ Stufen) können — **asymptotisch** mit wachsender Neuronenzahl — beliebige Klassengebiete bzw. Funktionen approximieren.*

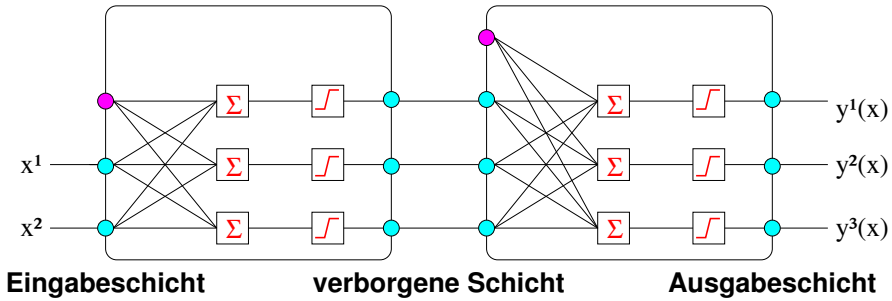


Beweis.

Durch genaues Hinschauen. □

Das Mehrschichtenperzeptron (MLP)

Beispiel: 2–3–3-Perzeptron (drei Schichten · zwei Stufen)



Konfiguration eines MLP als Klassifikator

1. D Eingabeneuronen und $(K-1)$ Ausgabeneuronen $D = 2$ und $K = 4$ im Beispiel
2. Wieviele verborgene Schichten? eine verborgene Schicht im Beispiel
3. Wieviele Neuronen in verborgenen Schichten? 3 verborgene Neuronen im Beispiel
4. Maschinelles Lernen der Gewichtmatrizen (3×3) und (4×3) im Beispiel

Optimierung der neuronalen Gewichtmatrix

Quadratischer Vorhersagefehler

zwischer *idealer* und *neuronaler* Prüfgröße:

$$\varepsilon(\omega|\boldsymbol{W}) \;=\; \sum_{\boldsymbol{x}\in\omega} \|\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x})\|^2$$

Stapelweise Auffrischung (*batch learning*)

$$\boldsymbol{W}' \;=\; \boldsymbol{W} - \eta\nabla_{\boldsymbol{W}}\left\{\varepsilon(\omega|\boldsymbol{W})\right\}\;, \qquad \nabla_{\boldsymbol{W}}\varepsilon = [\partial\varepsilon/\partial w_{jk}]_{j\prec k}$$

Additive Zerlegung des Gesamtfehlers

$$\frac{\partial\varepsilon(\omega|\boldsymbol{W})}{\partial w_{jk}} \;=\; \frac{\partial}{\partial w_{jk}}\sum_{\boldsymbol{x}\in\omega}\|\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x})\|^2 \;=\; \sum_{\boldsymbol{x}\in\omega}\frac{\partial\varepsilon(\boldsymbol{x}|\boldsymbol{W})}{\partial w_{jk}}$$

Error-Backpropagation-Algorithmus

Satz (Werbos 1972)

Gegeben sei ein nichtrekurrentes KNN mit Neuronen $\mathcal{N}$, Gewichtmatrix $\boldsymbol{W}$ und sigmoider Aktivierungsfunktion

$$f(\sigma) = 1/(1+e^{-\sigma})\;.$$

Dann gilt für die partiellen Ableitungen des quadratischen Fehlers

$$\varepsilon(\boldsymbol{W}|\boldsymbol{x}) \;=\; 1/2\cdot\|\boldsymbol{d} - \boldsymbol{x}_A\|^2$$

zwischen der Zielvorgabe $\boldsymbol{d}$ und der Netzausgabe $\boldsymbol{x}_A$ zur Netzeingabe $\boldsymbol{x}_E$ die Aussage

$$\partial\varepsilon/\partial w_{jk} \;=\; x_j\cdot x_k\cdot(1-x_k)\cdot\beta_k$$

mit den rekursiv berechenbaren Größen

$$\beta_j \;=\; \frac{\partial\varepsilon}{\partial x_j} \;=\; \begin{cases} (x_j - d_j) & j\in\mathcal{N}_A \\ \displaystyle\sum_{k|j\prec k}\beta_k x_k(1-x_k)w_{jk} & j\notin\mathcal{N}_A \end{cases}\;.$$

Beweis.

Unsere gewählte Aktivierungsfunktion besitzt eine besonders schöne erste Ableitung:

$$f'(\sigma) \;=\; f(\sigma)\cdot(1-f(\sigma))$$

Für die gesuchte Gewichtableitung ergibt die Kettenregel

$$\frac{\partial\varepsilon}{\partial w_{ij}} \;=\; \frac{\partial\varepsilon}{\partial x_j}\cdot\frac{\partial x_j}{\partial w_{ij}}$$

Der **rechte Faktor** wird wie folgt zerlegt:

$$\frac{\partial x_j}{\partial w_{ij}} \;=\; \frac{\partial x_j}{\partial\sigma_j}\cdot\frac{\partial\sigma_j}{\partial w_{ij}}$$

Wegen $\dfrac{\partial x_j}{\partial\sigma_j} \;=\; \dfrac{\partial f(\sigma_j)}{\partial\sigma_j} \;=\; f'(\sigma_j) \;=\; f(\sigma_j)\cdot(1-f(\sigma_j)) \;=\; x_j(1-x_j)$

und

$$\frac{\partial\sigma_j}{\partial w_{ij}} \;=\; \frac{\partial}{\partial w_{ij}}\sum_{k|k\prec j} x_k w_{kj} \;=\; x_i$$

folgt unmittelbar

$$\frac{\partial x_j}{\partial w_{ij}} \;=\; f'(\sigma_j)\cdot x_i \;=\; x_j(1-x_j)\cdot x_i$$

□

Error-Backpropagation-Algorithmus

⟨Algorithmus⟩

1. Initialisiere Gewichtmatrix $\boldsymbol{W}$ und setze $\nu = 0$.
2. Initialisiere Akkumulatoren $\Delta^{(\nu)}w_{ij} = 0$.
3. Für alle Klassen $\kappa = 1,\ldots,K$ und für alle Muster $\boldsymbol{x}_E \in \omega_\kappa$:
   a. Berechne Eingangserregungen $\boldsymbol{\sigma}(\boldsymbol{x}_E)$.
      Berechne Ausgangserregungen $\boldsymbol{x}(\boldsymbol{x}_E)$.
   b. Für alle Ausgabeneuronen $j \in \mathcal{N}_A$:
   $$\beta_j \;=\; x_j - \begin{cases} 1 & \text{Neuron } j \text{ codiert } \Omega_\kappa \\ 0 & \text{sonst} \end{cases}$$
   c. Für alle anderen Neuronen $j \in \mathcal{N}\backslash\mathcal{N}_A$:
   $$\beta_j \;=\; \sum_{k|j\prec k} \beta_k\cdot f'(\sigma_k)\cdot w_{jk}$$
   d. Für alle Neuronen $i,j \in \mathcal{N}$ mit $i \prec j$:
   $$\Delta^{(\nu)}w_{ij} \leftarrow \Delta^{(\nu)}w_{ij} + \beta_j\cdot f'(\sigma_j)\cdot x_i$$
4. Berechnen der neuen Gewichte $\boldsymbol{W} \leftarrow \boldsymbol{W} + \eta\cdot\Delta^{(\nu)}\boldsymbol{W}$
5. Abbruch? Sonst $\nu\leftarrow\nu+1$ und weiter bei $\rightsquigarrow$ 2.

⟨zumAlgorithmus⟩

Beweis.

Der **linke Faktor** beschreibt die Fehlerkomponente des j-ten Neurons. Für Ausgabeneuronen gilt einfach

$$\beta_j \;=\; \frac{\partial\varepsilon}{\partial x_j} \;=\; \frac{\partial}{\partial x_j}\frac{1}{2}\cdot\sum_{k\in\mathcal{N}_A}(d_k-x_k)^2 \;=\; (x_j-d_j)$$

Für alle anderen Neuronen gilt die Rekursionsformel

$$\begin{aligned}\beta_j \;=\; \frac{\partial\varepsilon}{\partial x_j} &= \sum_{k|j\prec k}\frac{\partial\varepsilon}{\partial x_k}\cdot\frac{\partial x_k}{\partial x_j} \\ &= \sum_{k|j\prec k}\frac{\partial\varepsilon}{\partial x_k}\cdot\frac{\partial x_k}{\partial\sigma_k}\cdot\frac{\partial\sigma_k}{\partial x_j} \\ &= \sum_{k|j\prec k}\beta_k\cdot f'(\sigma_k)\cdot w_{jk}\end{aligned}$$

Insgesamt ergibt sich mit diesen Größen der Ausdruck

$$\frac{\partial\varepsilon}{\partial w_{ij}} \;=\; \beta_j\cdot f'(\sigma_j)\cdot x_i$$

für die partiellen Gewichtableitungen. Die Behauptung des Satzes folgt duch Einsetzen der Sigmoidfunktion. □

Neuronale Architektur Feed-Forward SLP MLP Anwendung Rekurrente Netzwerke

Error-Backpropagation-Algorithmus

Mächtiges Lernverfahren mit mangelhaften Konvergenzeigenschaften

Anwendbarkeit

Alle nichtrekurrenten Netzwerke (incl. MLPs und TDNNs)
Alle differenzierbaren Aktivierungsfunktionen
Alle differenzierbaren Eingabekombinationen (incl. $\boldsymbol{w}_k^\top\boldsymbol{x}$ und $\|\boldsymbol{w}_k-\boldsymbol{x}\|$)

Konvergenzverhalten

Gradientenabstiegsverfahren — findet nur lokale Fehlerminima
Heuristische Schrittweite (Lernrate η) — ⇧ Oszillation · ⇩ Konvergenzrate

Konvergenzbeschleunigung

Der Korrekturterm wird durch **Momentumterm** erweitert:

$$\Delta^{(\nu)}w_{ij} \;=\; \eta\cdot\frac{\partial\varepsilon}{\partial w_{ij}} \;+\; \alpha\cdot\Delta^{(\nu-1)}w_{ij}$$

Resilienter EBP-Algorithmus, Gauß-Newton, Levenberg-Marquardt, ...

Mehrschichtenperzeptron

Serielle Notation mit abstrakten Verarbeitungsstufen

$1 + L$ Variablenschichten

zu je D_ℓ Komponenten:

$$\Omega_\ell \hat{=} \mathbb{R}^{D_\ell}$$

L Verarbeitungsstufen

zu je D_ℓ Einzelfunktionen:

$$f_{\ell,j} : \Omega_{\ell-1} \to \mathbb{R}$$

für $j = 1, \ldots, D_\ell$

Beispielverarbeitungsstufen

Lineargewichtungsstufe

$$f_\ell : \left\{ \begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R}^m \\ x & \mapsto & A^\top \cdot x + b \end{array} \right.$$

Kompandierungsstufe (z.B. Sigmoid)

$$f_\ell : \left\{ \begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R}^n \\ x & \mapsto & (\varphi(x_1), \ldots, \varphi(x_n))^\top \end{array} \right.$$

Softmaxstufe o.ä.

$$f_\ell : \left\{ \begin{array}{ccc} \mathbb{R}^n & \to & \mathbb{R}^k \\ x & \mapsto & (e^{x_j} / \sum_i e^{x_i})_{j=1..k} \end{array} \right.$$

Berechnung der Vorwärtsvariablen

Schichtenparallel von Stufe zu Stufe

Eingabe$\rightsquigarrow$Ausgabe

$$\begin{array}{lcl} x_0 & := & x \in \mathbb{R}^{D_0} \\ x_1 & := & f_1(x_0) \\ x_2 & := & f_2(x_1) \\ \cdots & & \cdots \\ x_\ell & := & f_\ell(x_{\ell-1}) \\ \cdots & & \cdots \\ x_L & := & f_L(x_{L-1}) \end{array}$$

Zweistufen-MLP zur Klassifikation

$$\begin{array}{lcll} x_{0,i} & = & x_i & \text{für } i = 1..N \\ x_{1,j} & = & W_{0j} + \sum_i W_{ij} \cdot x_{0,i} & \text{für } j = 1..M \\ x_{2,j} & = & (1 + e^{-x_{1,j}})^{-1} & \text{für } j = 1..M \\ x_{3,k} & = & V_{0k} + \sum_j V_{jk} \cdot x_{2,j} & \text{für } k = 1..K \\ x_{4,k} & = & (1 + e^{-x_{3,k}})^{-1} & \text{für } k = 1..K \\ \hat{y}_k & = & x_{4,k} & \end{array}$$

$$f_{1:L}(x) = (f_1 \circ f_2 \circ \ldots \circ f_L)(x) = f_L(f_{L-1}(\ldots(f_2(f_1(x)))\ldots))$$

Kosten, Lerndaten und Gütefunktion

Optimierung der freien MLP-Parameter θ

Kostenfunktional

$\varepsilon(\hat{y}, y^*)$ für
Wunschergebnis y^* und
MLP-Ausgabe $\hat{y} = f_{1:L}(x)$

Beispiel für Regression/Klassifikation

$\varepsilon(\hat{y}, y^*) = \frac{1}{2}\|\hat{y} - y^*\|^2$ (OLS)
$\varepsilon(\hat{y}, k^*) = -\log(\hat{y}_{k^*})$ (MAP)

Stichprobe $\omega = (x^{(t)}, y^{(t)})_{t=1..T}$ für überwachtes Lernen

Näherungsziel: $\mathcal{E}[\varepsilon(f_\theta(\mathbb{X}), \mathbb{Y})] \approx \sum_{t=1}^{T} \varepsilon(f_\theta(x^{(t)}), y^{(t)})$ (LLN)

Gradientenabstieg

$$\frac{\partial \mathcal{E}[\varepsilon(f_\theta(\mathbb{X}), \mathbb{Y})]}{\partial \theta} \approx \sum_{t=1}^{T} \frac{\partial \varepsilon(f_\theta(x^{(t)}), y^{(t)})}{\partial \theta}$$

Ableitung nach den Vorwärtsvariablen

Fehlerrückführung vom MLP-Ausgang zum MLP-Eingang

$$\xi_{\ell,j} \stackrel{\text{def}}{=} \partial\varepsilon / \partial x_{\ell,j} \quad \text{für alle } \left\{ \begin{array}{l} \ell = 0..L \\ j = 1..D_\ell \end{array} \right\}$$

Rekursionsbeginn

$$\xi_{L,j} = \frac{\partial}{\partial x_{L,j}} \varepsilon(x_L, y^*)$$

OLS-Regression

$$\begin{array}{rcl} \xi_{L,j} & = & \frac{\partial}{\partial x_{L,j}} \frac{1}{2} \sum_i (x_{L,i} - y_i^*)^2 \\ & = & x_{L,i} - y_i^* \end{array}$$

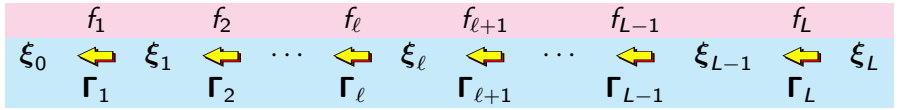
Rekursionsschritt

$$\begin{array}{rcl} \xi_{\ell-1,j} & = & \frac{\partial \varepsilon}{\partial x_{\ell-1,j}} \\ & = & \sum_k \frac{\partial \varepsilon}{\partial x_{\ell,k}} \cdot \frac{\partial x_{\ell,k}}{\partial x_{\ell-1,j}} \\ & = & \sum_k \xi_{\ell,k} \cdot \gamma_{\ell,j,k} \end{array}$$

Lokale Ableitungen

zwischen den Variablen zweier benachbarter MLP-Stufen

$$\gamma_{\ell,j,k} \stackrel{\text{def}}{=} \partial x_{\ell,k} \,/\, \partial x_{\ell-1,j} \qquad \text{für alle } \begin{Bmatrix} \ell = 1..L \\ j = 1..D_{\ell-1} \\ k = 1..D_\ell \end{Bmatrix}$$



Aktivierungsstufe

$$\gamma_{\ell,j,k} = \frac{\partial}{\partial x_{\ell-1,j}} \varphi(x_{\ell-1,k}) = \begin{cases} \varphi'(x_{\ell-1,k}) & j = k \\ 0 & j \neq k \end{cases}$$

Linearkombination

$$\gamma_{\ell,j,k} = \frac{\partial}{\partial x_{\ell-1,j}} \left(b_{\ell,k} + \sum_i A_{\ell,i,k} \cdot x_{\ell-1,i} \right) = A_{\ell,j,k}$$

Softmaxstufe

$$\gamma_{\ell,j,k} = \frac{\partial}{\partial x_{\ell-1,j}} \frac{e^{x_{\ell-1,k}}}{\sum_i e^{x_{\ell-1,i}}} = \begin{cases} -x_{\ell,j} \cdot x_{\ell,k} + x_{\ell,k} & j = k \\ -x_{\ell,j} \cdot x_{\ell,k} + 0 & j \neq k \end{cases}$$

Kompandierungsfunktionen und ihre Ableitungen

nichtlineare, monotone Skalentransformationen für Aktivierungsstufen

Name		Funktion	Ableitung
Identität[1]	$\mathrm{id}(x)$	x	1
Stufe[2]	$\mathrm{sign}(x)$	$2 \cdot \mathbf{I}_{x>0}(x) - 1$	0 für $x \neq 0$
Sigmoid[3]	$\sigma(x)$	$1 \,/\, (1 + e^{-x})$	$\sigma(x) \cdot (1 - \sigma(x))$
Tangens[4]	$\tanh(x)$	$(e^{2x} - 1) \,/\, (e^{2x} + 1)$	$1 - \tanh^2(x)$
Ellbogen[5]	$\varphi_+(x)$	$\max(0, x)$	$\mathbf{I}_{x>0}(x)$
μ-Gesetz[1]	$\phi_\mu(x)$	$\mathrm{sign}(x) \cdot \log(1 + \mu\lvert x\rvert)$	$(\lvert x\rvert + 1/\mu)^{-1}$
μ-invers[1]	$\varphi_\mu(x)$	$\mathrm{sign}(x) \cdot (e^{\lvert x\rvert} - 1)/\mu$	$e^{\lvert x\rvert}/\mu$

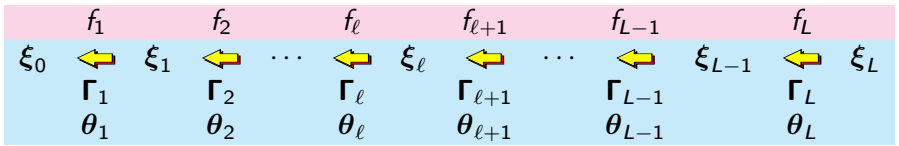
Bemerkung

Wertebereiche: [1] $\mathbb{R}$ [2] $\{-1, 0, 1\}$ [3] $(0, 1)$ [4] $(-1, 1)$ [5] $\mathbb{R}_+$

Ableitung nach den MLP-Parametern

„Synaptische Gewichte“ des Neuronennetzes · Kompandierungsparameter

$$\frac{\partial \varepsilon}{\partial \theta_{\ell,i}} = \sum_k \frac{\partial \varepsilon}{\partial x_{\ell,k}} \cdot \frac{\partial x_{\ell,k}}{\partial \theta_{\ell,i}} = \sum_k \xi_{\ell,k} \cdot \frac{\partial}{\partial \theta_{\ell,i}} f_{\ell,k}(\boldsymbol{x}_{\ell-1}) \qquad \text{für alle } \begin{Bmatrix} \ell = 1..L \\ i = 1..I_\ell \end{Bmatrix}$$



Linearkoeffizienten

$$\frac{\partial \varepsilon}{\partial A_{\ell,j,k}} = \frac{\partial \varepsilon}{\partial x_{\ell,k}} \cdot \frac{\partial x_{\ell,k}}{\partial A_{\ell,j,k}} = \xi_{\ell,k} \cdot \frac{\partial}{\partial A_{\ell,j,k}} \left(b_{\ell,k} + \sum_i A_{\ell,i,k} \cdot x_{\ell-1,i} \right) = \xi_{\ell,k} \cdot x_{\ell-1,j}$$

Aktivierungschwellen

$$\frac{\partial \varepsilon}{\partial b_{\ell,k}} = \frac{\partial \varepsilon}{\partial x_{\ell,k}} \cdot \frac{\partial x_{\ell,k}}{\partial b_{\ell,k}} = \xi_{\ell,k} \cdot \frac{\partial}{\partial b_{\ell,k}} \left(b_{\ell,k} + \sum_i A_{\ell,i,k} \cdot x_{\ell-1,i} \right) = \xi_{\ell,k} \cdot 1$$

Fehlerrückführung in Matrixschreibweise

L-Stufen-MLP · Regressionsaufgabe · Methode der kleinsten Quadrate

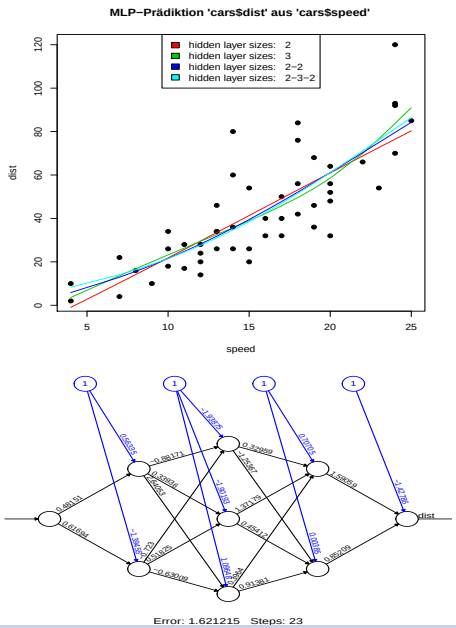
⟨Algorithmus⟩

1. Initialisiere die Parameterfelder $\boldsymbol{A}_\ell$ und $\boldsymbol{b}_\ell$ für $\ell = 1..L$
2. Initialisiere Akkumulatorfelder $\boldsymbol{\Delta}_\ell^A$ und $\boldsymbol{\Delta}_\ell^b$ zu Null
3. Für alle Lernstichprobenpaare $(\boldsymbol{x}, \boldsymbol{y}) \in \omega$ berechne ...
   - a Vorwärtsvariable $\boldsymbol{x}_\ell = \begin{cases} \boldsymbol{x} & \ell = 0 \\ \boldsymbol{A}_\ell^\top \cdot \boldsymbol{x}_{\ell-1} + \boldsymbol{b}_\ell & \text{linear} \\ \boldsymbol{\varphi}(\boldsymbol{x}_{\ell-1}) & \text{skalar} \end{cases}$
   - b Rückwärtsvariable $\boldsymbol{\xi}_\ell = \begin{cases} \boldsymbol{x}_L - \boldsymbol{y} & \ell = L \\ \boldsymbol{A}_{\ell+1} \cdot \boldsymbol{\xi}_{\ell+1} & \text{linear} \\ \mathrm{diag}(\boldsymbol{\varphi}(\boldsymbol{x}_\ell)) \cdot \boldsymbol{\xi}_{\ell+1} & \text{skalar} \end{cases}$
   - c Aktualisiere $\boldsymbol{\Delta}_\ell^A$ und $\boldsymbol{\Delta}_\ell^b$ um $\boldsymbol{x}_\ell \cdot \boldsymbol{\xi}_\ell^\top$ bzw. $\boldsymbol{\xi}_\ell$
4. Verschiebe die alten Parameter in Gradientenrichtung
5. Abbruch oder weiter bei ⇝ 2

⟨zumdingIA⟩

Mehrschichtenperzeptron (MLP)

`neuralnet::neuralnet (formula, data, hidden=1, algorithm='rprop+', ...)`



Funktionsargumente

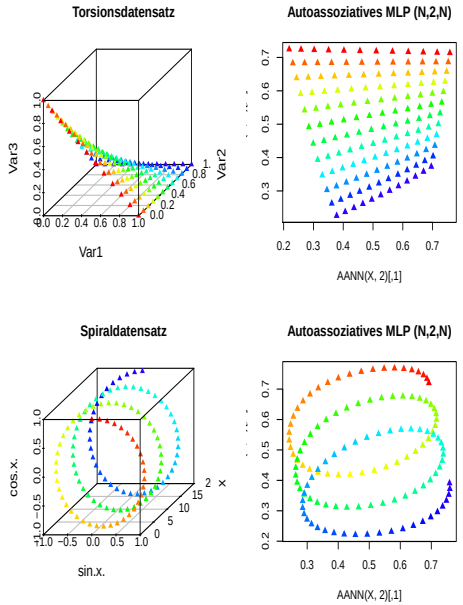
`formula` Output/Input-Variablen
`data` Datensatz
`hidden` Zwischenschichtengrößen
`algorithm` {backprop, sag, slr, rprop+, rprop-}
`return` Objekt der Klasse `nn`

R-Programmcode

```
require (neuralnet)
s <- 100
H <- list (2, 3, c(2,2), c(2,3,2))
plot (cars, pch=19)
for (j in seq_along(H)) {
   o <- neuralnet (
      dist~speed, cars/s, H[[j]])
   y <- compute(o,cars$speed/s)$net
   lines (cars$speed, y*s, col=1+j)
   }
plot (o, rep="best")
```


Autoassoziatives Mehrschichtenperzeptron

`neuralnet::neuralnet (formula, data, hidden=1, algorithm='rprop+', ...)`



Funktionsargumente

`formula` Output/Input-Variablen
`data` Datensatz
`hidden` Zwischenschichtengrößen
`algorithm` {backprop, sag, slr, rprop+, rprop-}
`return` Objekt der Klasse `nn`

R-Programmcode

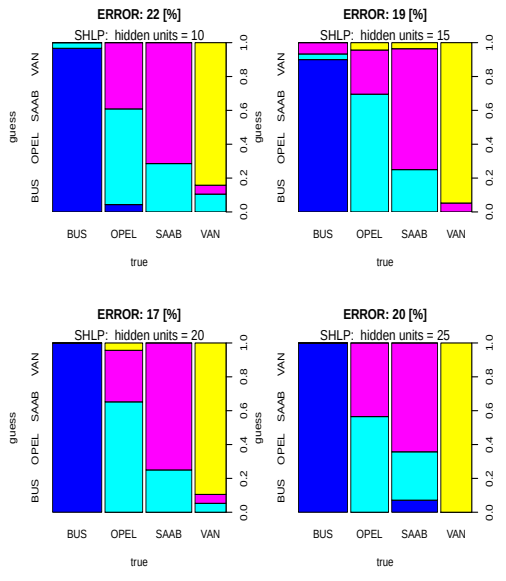
```
AANN <-function (X, k=2, sd=1/3, ...)
   {
   require (neuralnet)
   X <- scale(X) * sd

   vn <- colnames(X)
   iv <- paste (vn, collapse="+")
   ov <- paste (paste (
       vn, 1, sep="."), collapse="+")
   fo <- formula (paste (iv, "~", ov))

   o <- neuralnet (formula=fo,
       data.frame(X,X), hid=k, ...)
   compute(o,X)$neurons[[2]][,-1]
   }
```


Dreischichtenperzeptron (SHLP)

`nnet::nnet (x, y, size, Wts, mask, linout, entropy, softmax, censored, skip=FALSE, MaxNWts=1000, ...)`



Funktionsargumente

`x, y` Matrix/Dataframe für E/A
`formula, data` *(Formelinterface)*
`size` verborgene Neuronen
`skip` Kurzschlusskanten?
`Wts, MaxNWts` Startwerte, $\#_{max}$
`return` Objekt der Klasse `nnet`

R-Programmcode

```
require (nnet)
for (h in 5*2:5) {
   o <- nnet (CLASS~., df[[1]],
      Wts=sin(1:nwts),
      MaxNWts=nwts,
      size=h)
   guess <- predict (o, df[[2]],
      type="class")
   plot (data.frame(true,guess),
      col=3+1:4)
}
```

CAM — Content-Addressable Memory

Assoziatives Gedächtnis: Zugriffsschlüssel $\subseteq$ Datensatz

- **Getaktete** Neuronen mit **diskreten** Zuständen

$$x_k(t) \in \{-1, +1\}\ , \qquad t \in \mathbb{N},\ k \in \mathcal{N}$$

- Nichtlineare Systemdynamik $\boldsymbol{x}_t \mapsto \boldsymbol{x}_{t+1}$

$$x_k(t+1) \ = \ \text{sign}\left(\sum_j w_{jk} \cdot x_j(t)\right)$$

- Berechnung eines **Gleichgewichtszustandes** (Äquilibrium) $\boldsymbol{u}$:

$$\boldsymbol{u}(\boldsymbol{x}_0) \stackrel{\text{def}}{=} \lim_{t\to\infty} \boldsymbol{x}(t)$$

- Zustandsraum $\{0,1\}^N$ zerfällt in **Attraktionsbassins**:

$$\boldsymbol{x} \cong \boldsymbol{y} \quad \Leftrightarrow \quad \boldsymbol{x} \stackrel{\infty}{\to} \boldsymbol{u} \stackrel{\infty}{\leftarrow} \boldsymbol{y}$$

Hopfield-Netze

Vollständig verschaltetes CAM mit symmetrischer Gewichtmatrix

Definition

Das dynamische System mit dem Zustandsübergang

$$x_k(t+1) \ = \ \text{sign}\left(\sum_{j=1}^{N} w_{jk} \cdot x_j(t)\right)\ , \qquad t \in \mathbb{N},\ k = 1, \ldots, N$$

mit der symmetrischen Gewichtmatrix $\boldsymbol{W}$ heißt **Hopfield-Netz**. Es definiert eine (partielle) Abbildung

$$\boldsymbol{u} : \left\{ \begin{array}{lcl} \{-1,+1\}^N & \to & \{-1,+1\}^N \\ \boldsymbol{z} & \rightharpoonup & \lim\limits_{t\to\infty} \boldsymbol{x}(t) \mid_{\boldsymbol{x}(0)=\boldsymbol{z}} \end{array} \right.$$

im Raum der N-dimensionalen Binärmuster.

Hebb'sche Lernregel

Lemma

Zur Speicherung von K „Prototypen" $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$ sind die Gewichte w_{jk} proportional zum Aktivierungsratenprodukt der vernetzten Neuronen zu wählen:

$$w_{jk} \ = \ \sum_{\lambda=1}^{K} y_{\lambda,j} \cdot y_{\lambda,k}$$

Unter der Voraussetzung

$$K \ \leq \ 0.138 \cdot N$$

*hinreichender **Kapazität** ist dann jeder Prototyp ein Äquilibrium des gelernten Netzwerks.*

Beweis.

Für Gleichgewichtspunkte $\boldsymbol{x} = \operatorname{sign}(\boldsymbol{W}\boldsymbol{x})$ gilt:

$$\sum_j w_{jk} x_j \;=\; \sum_j \sum_\lambda y_{\lambda,j} y_{\lambda,k} x_j$$

Für einen Prototypen $\boldsymbol{x} = \boldsymbol{y}_\nu$ gilt insbesondere

$$\sum_j w_{jk} y_{\nu,j} \;=\; \sum_j \sum_\lambda y_{\lambda,j} y_{\lambda,k} y_{\nu,j} \;=\; N \cdot y_{\nu,k} + \sum_j \sum_{\lambda \neq \nu} y_{\lambda,j} y_{\lambda,k} y_{\nu,j}$$

$\boldsymbol{y}_\nu$ ist Gleichgewichtspunkt, wenn der rechte Term betragsmäßig kleiner als N ist. □

Hopfield-Netze — Beispielanwendung

(John S. Denker, 1986)

Telefonbucheinträge

(25 Zeichen/Zeile, 5 Bit/Zeichen)

John Stewart Denker	8128
Lawrence David Jackel	7773
Richard Edward Howard	5952
Wayne P. Hubbard	7707
Brian W. Straughn	3126
... ...	....
... ...	....
... ...	....
... ...	....
... ...	....
John Henry Scofield	8109

Erfolgreicher Zugriff

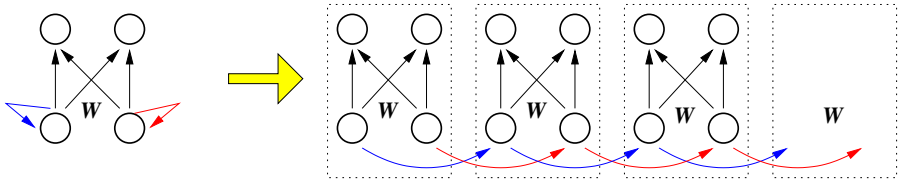
t	$E = -x^\top Wx$	CAM-Zustand	
0.0	-0.0	john s	
0.2	-0.0784	john sdewirubneoimv	8109
0.4	-0.8426	john sdewirtbnenimv	8129
0.6	-0.8451	john sdewirtbnenimv	8129
0.8	-0.8581	john sdewirt nenkmv	8128
1.0	-0.9099	john sdewart denker	8128
1.2	-0.9824	john stewart denker	8128

Erfolgloser Zugriff

t	$E = -x^\top Wx$	CAM-Zustand	
0.0	-0.0	garbage	
0.2	-0.0244	garbagee lafj naabd	5173
0.4	-0.6280	garbaged derjd naabd	7173
0.6	-0.6904	garbaged derjd naabd	7173
0.8	-0.7345	gasbafed derjd naabd	7173
1.0	-0.7595	gasbabed derjd naabd	7173
1.2	-0.7709	fasjebad derjd naabd	7173
1.4	-0.8276	fasjebad derjd naabd	7173
1.6	-0.8282	fasjeb d derjd naabd	7173

Zeitverzögerungsnetzwerke

TDNN — *time-delayed neural networks*



Simulation rekurrenter Netzwerke

Das Verhalten eines rekurrenten Netzwerks mit ausschließlich **einfachen Zyklen** $k \prec k$, $k \in \mathcal{N}$ kann im Rahmen eines T-**Takte-Gedächtnisses** durch ein Feed-Forward-Netz mit $T \cdot N$ Neuronen simuliert werden.

Zusammenfassung (8)

1. Das **Modellneuron** kombiniert seine **Eingangssignale** und berechnet daraus sein **Aktivierungsniveau**.
2. Die **Verschaltung** der Modellneuronen ist durch das Nullenmuster der **Gewichtmatrix** synaptischer Verbindungen definiert.
3. **Rekurrente** Netzwerke sind durch die Existenz **zyklischer Verbindungen** charakterisiert und beschreiben **dynamische Systeme**, deren Verhalten durch die **Attraktionsgebiete** ihrer Fixpunktzustände definiert ist.
4. **Nichtrekurrente** Netzwerke beschreiben pure Vorwärtsberechnungen und definieren durch die Zustände ihrer **Eingabe-** und **Ausgabeneuronen** eine **nichtlineare Vektorabbildung**.
5. Das **einstufige Perzeptron** ist nicht in der Lage, die **logische XOR-Funktion** zu lernen.
6. Ein **mehrstufiges Perzeptron** (MLP) mit mindestens vier Neuronenschichten gilt als (asymptotisch) **universeller Approximator**.
7. Jedes MLP mit **differenzierbarer** Aktivierungsfunktion kann mittels **Error-Backpropagation**-Algorithmus seine Gewichtmatrix im Sinne **kleinster Fehlerquadrate** auf eine Lerndatenprobe optimieren.
8. Für die **numerische Klassifikation** verwendet man MLPs mit D Eingabeneuronen, K Ausgabeneuronen und einer der anvisierten **Modellkapazität** angemessenen Anzahl **verborgener Neuronen**.