Maschinelles Lernen und Datenanalyse

In der Mess- und Prüftechnik

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

Klassifikation und Regression mit Künstlichen Neuronale Netze

Zielvariablen: Numerische Variablen

Eigenschaftsvariablen: Numerische Variablen

Modell: Gerichteter Graph (zyklisch oder azyklisch)

Training und Algorithmen: Backpropagation

Klasse: Überwachtes Lernen

Künstliche Neuronale Netze

- Ein Künstliches Neuronales Netz (KNN) ist ein gerichteter Graph bestehend aus einer Menge von Knoten $\boldsymbol{N}$ und Kanten $\boldsymbol{E}$ die die Knoten verbinden
  - Knoten: Neuron oder Perzeptron mit einem oder mehreren Eingängen $\boldsymbol{I}$ und einem Ausgang o; Berechnungsfunktion $g(\boldsymbol{I})$: $\boldsymbol{I} \rightarrow o$
  - Kanten: Gewichteter Datenfluss vom Ausgang eines Neurons zum Eingang eines anderen (oder des selben) Neurons

Ein KNN ist eine Komposition aus einer Vielzahl von Abbildungsfunktionen $\boldsymbol{G}=(g_1,g_2,..,g_m)$. Es gibt Parallelen zu Regressionsverfahren mit Funktionen.

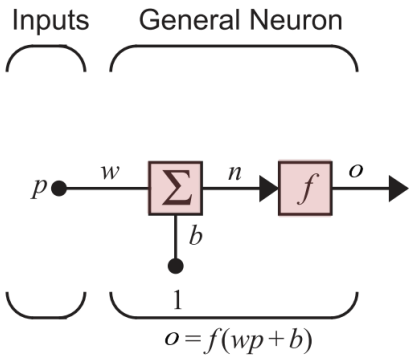
- Zusammengefasst ausgedrückt:

$$M(X) : X \rightarrow Y, X = \{x_i\}, Y = \{y_j\}$$
$$KNN = \langle N_x, N_d, N_y, E \rangle$$
$$N_x = \{n_i : n_i \leftrightarrow \{x_j\}\}, N_d = \{n_d\}, N_y = \{n_k : n_k \leftrightarrow y_k\}$$
$$n = g(\vec{p}, \vec{w}, b) : \vec{p} \rightarrow o = f(\sum_i w_i p_i + b)$$
$$E = \{e_{ij} : n_i \mapsto n_j w_{ij}\}$$

- *f* ist eine Transferfunktion die die akkumulierten Eingangswerte auf den Ausgangswert *o* abbildet, und *g* ist dann die gewichtete und akkumulative Transferfunktion

- Unterschied (künstliches) Neuron und Perzeptron:
  - Ein Neuron ist immer eine Elementarzelle
  - Ein Perzeptron kann ein einzelnes Neuron oder ein Netzwerk aus Neuronen beschreiben
- Daher gibt es:
  - Single Layer Perceptron (SLP) → Nur Eingangs- $\boldsymbol{N}_{\mathrm{x}}$ und Ausgangsneuronen $\boldsymbol{N}_{\mathrm{y}}$
  - Multi Layer Perceptron (MLP) → + Innere Neuronen $\boldsymbol{N}_{\mathrm{d}}$

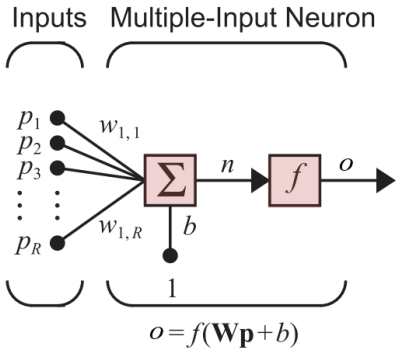
Das Neuron



[15]

Abb. 1. Ein einzelnes Neuron mit einem einzelnen Eingang p und einem Ausgang o. w ist ein Gewichtungsfaktor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset)

Das Mehreingangsneuron

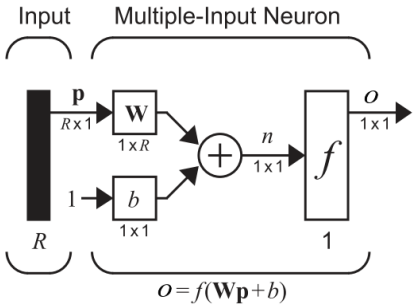


[15]

Abb. 2. Ein einzelnes Neuron mit einem Eingangsvektor $\boldsymbol{p}$ und einem skalaren Ausgang o. $\boldsymbol{w}$ ist ein Gewichtungsfaktorvektor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset)

Neuronale Netze und Matrizen

- Neuronale Netze werden durch eine Graphenstruktur (statische Parameter) und mathematisch durch Matrizen (dynamische Parameter) beschrieben:

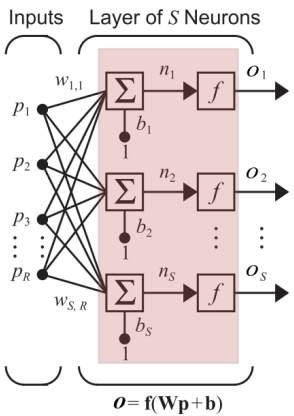


[15]

Abb. 3. Ein einzelnes Neuron mit einem Eingangsvektor $\boldsymbol{p}$ und einem skalaren Ausgang o. $\boldsymbol{w}$ ist ein Gewichtungsfaktorvektor (ein Gewicht für eingehendes p) und b ist ein Bias (Offset); jetzt in Matrizenform (Annotation)

Schichten von Neuronalen Netzen

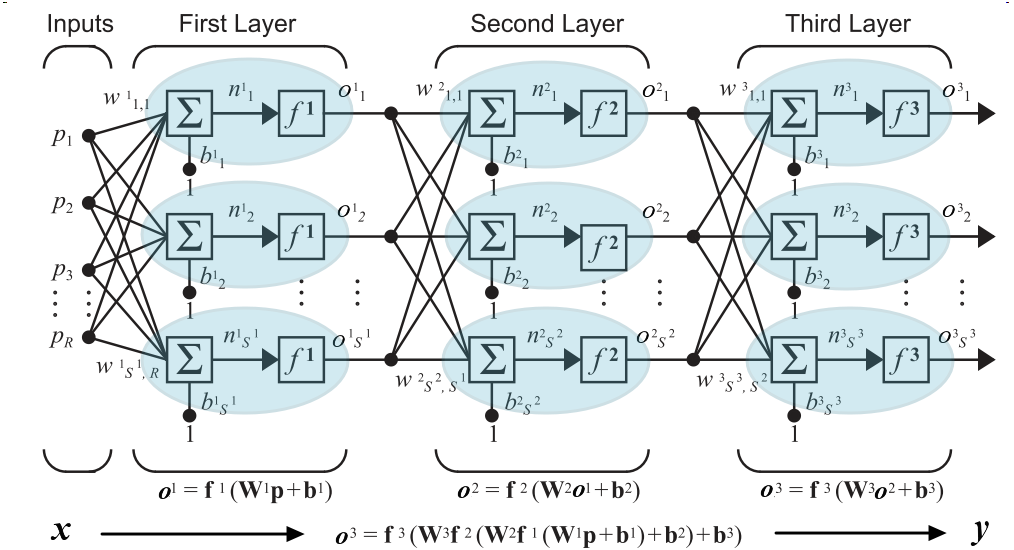
- I.A. werden Neuronen von neuronalen Netzen in Schichten (Layer) angeordnet und gruppiert
  - Günstig für Matrixalgebra
  - Aber nicht notwendig!



[15]

Abb. 4. Neuronales Netzwerk mit Neuronen in einer Schicht angeordnet

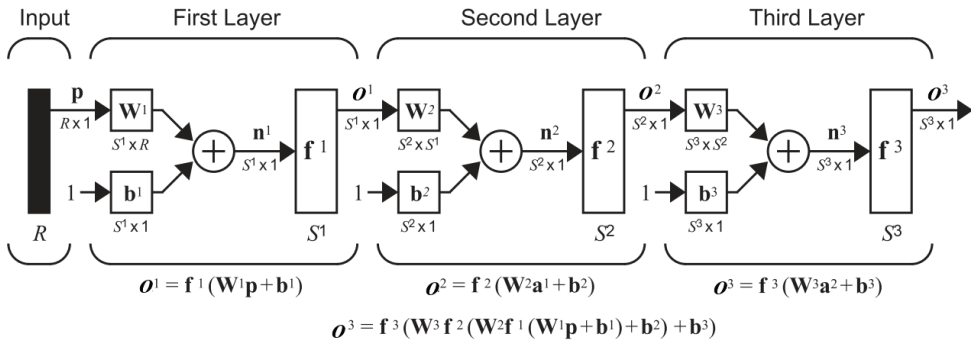
Struktur eines KNN



[15]

Abb. 5. Grundlegende Struktur eines KNN mit Matrizen (blaue Ellipse=1 Neuron)

Vereinfachte Form eines KNN



[15]

Abb. 6. Vereinfachte Struktur eines KNN mit Matrizen

Klassen von KNN

Vorwärtsgekoppelte Netzwerke

Azyklischer gerichteter Graph, d.h. es gibt nur eine Vorwärtspropagation von einer Schicht zur nächsten (keine Rückkopplung).

- Diese Netzwerke können rein funktional beschrieben und berechnet werden.
- Es gibt keinen Zustand!
- D.h. die aktuellen Ausgangswerte hängen nur von den aktuellen Eingangswerten ab!

Rückgekoppelte Netzwerke

Zyklischer gerichteter Graph, d.h. es gibt Rückkopplungen (Ausgang eines Neurons geht in Eingänge der aktuellen oder vorherigen Schichten).

- Diese Netzwerke können nicht rein funktional beschrieben und berechnet werden!
- Sie besitzen einen Zustand, d.h. der Ausgangswert hängt von der Historie vergangener Eingabewerte und Berechnungen ab!

Rückgekoppelte Netzwerke

- Geeignet für Prädiktion auf zeit- und Datenserien $D(t)=d_0,d_1,...,d_t$

[15]

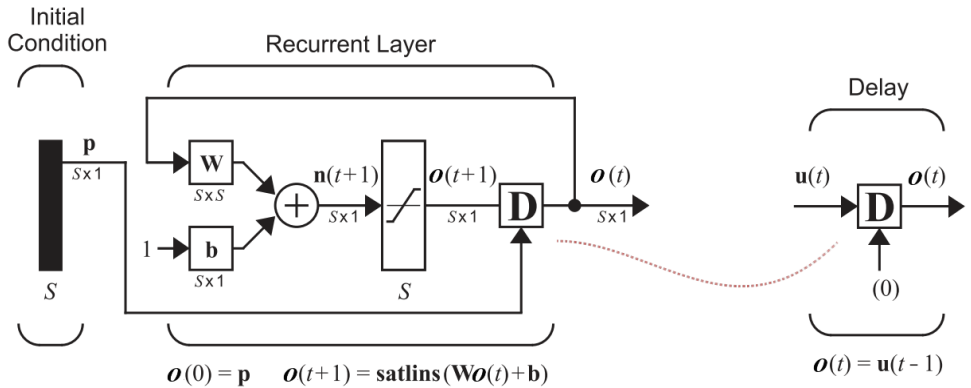


Abb. 7. Rückgekoppeltes und zustandsbehaftetes KNN mit einer Verzögerungsfunktion (Speicher)

Transferfunktion

- Auch Aktivierungsfunktion genannt (in Anlehnung an biologische Vorbild mit stark nichtlinearer Übertragungskennline)
  - Biologisch: Häufig eine Schwellwertfunktion
  - Künstlich / ML: Auch lineare Übertragunsfunktionen!
- Es gibt eine Vielzahl verschiedener Funktionen
  - Die einfachste wäre (wenn auch wenig in Gebrauch): $f(a) = a$

Warum ist eine solche Übertragungsfunktion ungeeignet bzw. problematisch?

- Welche mathematischen **Eigenschaften** (Übertragungskurve) sollte wohl eine Transferfunktion besitzen?
  - Zur Erinnerung: Wir nehmen an dass der Wertebereich von einem $x \approx [-1,1]$ ist. Ebenso für ein $y \approx [-1,1]$.

Transferfunktionen besitzen häufig begrenzende Eigenschaften (Sättigungsverhalten), und nicht lineares Übertragungsverhalten

[15]

Name	Input/Output Relation	Icon	Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins
Log-Sigmoid	$a = \frac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

Abb. 8. Verschiedene gebräuchliche Transferfunktionen $f(a)$

Ein einfaches Neuron - Funktional

$$f_{sigmoid}(a) = \frac{1}{1+e^{-a}}$$

$$g(x_1, x_2, x_3) = f_{sigmoid}(b + \sum w_i x_i)$$

Web WorkShell Live

CLEAR LOAD + - Neuron

Parametersatz des KNN

Statische Parameter

- Anzahl der Eingangsneuronen (verbunden mit **x**), abhängig von der Anzahl der Eingabevariablen |**x**| und der Kodierung (numerisch vs. kategorisch)
- Anzahl der Ausgangsneuronen (abhängig von der Kodierung). Bei numerischen Zielvariablen **y** gilt also: $|N_y|=|\mathbf{y}|$
- Anzahl der inneren verdeckten Neuronen $|N_d|$ und deren Anordnung in Schichten
- D.h. die **Konfiguration** des Netwerks ist $[c_1,c_2,..,c_m]$ bei m Schichten und c_i Neuronen pro Schicht

- Bei vollständig verbundenen Schichten ist keine Angabe der Vernetzung notwendig

Dynamische Parameter

- Im wesentlichen die Gewichtungsmatrix $\mathbf{W}_\mathrm{i}$ (Schicht *i*):

$$W_i = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}, B_i = \begin{bmatrix} b_1 \\ \vdots \\ b_S \end{bmatrix}$$

Mit *S*: Anzahl der Neuronen in der Schicht, *R*: Anzahl der Eingangsvariablen (oder Neuronen der vorherigen Schicht)

- Der Ausgangswert eines Neurons n_j ist dann gegeben durch einen Wert aus B und die j-te Zeile von $\boldsymbol{W}$:

$$o(\vec{p}) = f({}_jW^T\vec{p} + b_i)$$

- Bei mehrschichtigen Netzwerken hat man eine Menge von Gewichtematrizen, die zu einem Tensor zusammengefasst werden können.

Training von KNN

- Wie bei allen überwachten Lernproblemen gilt es eine Fehlerfunktion zu minimieren:

$$M(\vec{x}) : \vec{x} \rightarrow \vec{y}$$
$$\underset{W}{\operatorname{argmin}} \; err(M) = |y(\vec{x}) - y_0(\vec{x})|, \forall (x, y_0) \in D$$

> Ziel ist die Minimierung des Fehlers unserer Modellhypothese *M*(**x**) durch Anpassung der Gewichtematrix **W** und evtl. (wenn vorhanden) des Offsetvektors **B**

Fehler

LS1

$$err = y - y_0$$

$$err = |y - y_0|$$

LS2

$$err = (y - y_0)^2$$

Es ist leicht zu erkennen dass das Training einen hochdimensionalen Parametersatz anpassen muss. Es ist nicht unmittelbar klar wie ein optimales $\boldsymbol{W}$ abgeleitet werden kann!

Erklärbarkeit

- Der Zusammenhang von y und x ($x \rightarrow y$) ist schon bei einem einschichtigen Netzwerk nur noch schwer nachvollziehbar!
- Eine Invertierung (inverses Problem $y \rightarrow x$) ist ebenso nur schwer möglich
- Eigentlich ist nur ein einzelnes Neuron erklärbar und verständlich
  - Dort ist die Anpassung (des Gewichtungsvektors **w**) noch durch multivariate Regression möglich

Beispiel

- Trainingsverfahren: Einfache Fehlerückpropagation
- Problem: $\mathbf{x}=(a,b)$, y
- Netzwerk: Ein Neuron, Sigmoid Transferfunktion

Web WorkShell Live

CLEAR LOAD + - Neuron

Nichtlineare Probleme

SLP können nur lineare Probleme separieren.

[15]

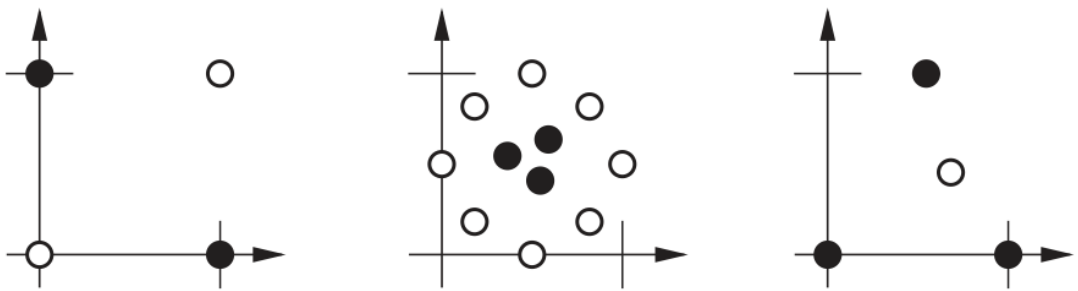


Abb. 9. Nichtlinear separierbare Probleme - nur mit MLP klassifizierbar

Web WorkShell Live

CLEAR LOAD + - Neuron

Error Backpropagation Verfahren

- Bekanntes und gängiges Verfahren

https://hmkcode.com/ai/backpropagation-step-by-step

Gradientenverfahren

- Baut auf dem Minimierungsansatz "Gradient Descent" (GD) auf (Absteigender Gradient)
- Beim GD Verfahren wird eine Funktion, z.B. $f(x,w)$: $x \rightarrow y$ derart über den Parameter w angepasst so dass der Fehler $err=|y-y_0|$ minimal wird

- Es wird nun die Änderung des Fehlers beobachtet Δerr und der (oder später die) Parameter w mit der Ableitung des Fehlerwerts $\partial err/\partial w$ zu der Änderung des Parameters korrigiert:

$$w' = w - \alpha \frac{\partial err}{\partial w}$$

Zur Berechnung des Fehlergradientens wird die Ableitung der Transferfunktion benötigt.

- Vereinfacht gilt aber (grobe Näherung):

$$\frac{\partial err}{\partial w} \sim x(y - y_0)$$

- Jetzt wird ein neuronales Netzwerk betrachtet, wo die Neuronen ebenfalls Funktionen mit Eingangsvariablen und Ausgangsvariablen sind

- Bei zusammengesetzten Funktionen (z.B. auch Neuronen in inneren Schichten) müssen die Gewichte schrittweise von hinten nach vorne angepasst werden

[hmkcode.com/ai/backpropagation-step-by-step]

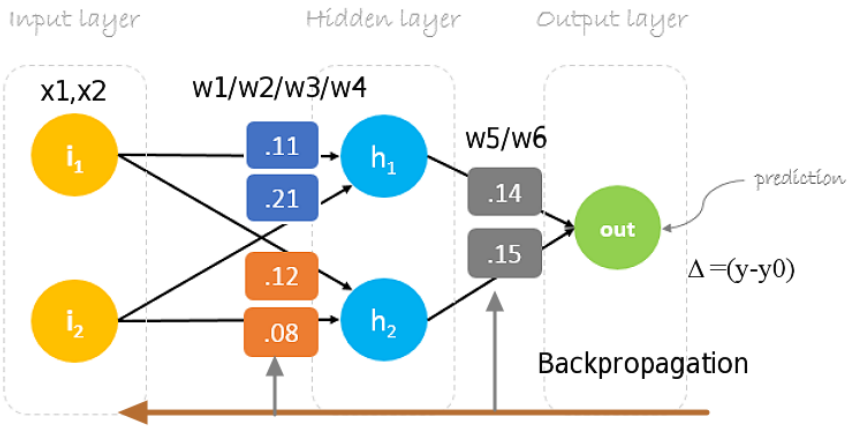


Abb. 10. Beispiel eines ANN mit Kantengewichten und dem Ansatz der Backpropagation

- Die Gewichte werden nun Schicht für Schicht unter Einbeziehung der gewichteten Fehlerpropagation gleichermaßen angepasst

[hmkcode.com/ai/backpropagation-step-by-step]

$$
\begin{aligned}
{}^{*}w_6 &= w_6 - \alpha\,(h_2 \,.\, \boldsymbol{\Delta}) \\
{}^{*}w_5 &= w_5 - \alpha\,(h_1 \,.\, \boldsymbol{\Delta}) \\
{}^{*}w_4 &= w_4 - \alpha\,(i_2 \,.\, \boldsymbol{\Delta} w_6) \\
{}^{*}w_3 &= w_3 - \alpha\,(i_1 \,.\, \boldsymbol{\Delta} w_6) \\
{}^{*}w_2 &= w_2 - \alpha\,(i_2 \,.\, \boldsymbol{\Delta} w_5) \\
{}^{*}w_1 &= w_1 - \alpha\,(i_1 \,.\, \boldsymbol{\Delta} w_5)
\end{aligned}
\qquad
\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \alpha\boldsymbol{\Delta}\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} \alpha h_1 \boldsymbol{\Delta} \\ \alpha h_2 \boldsymbol{\Delta} \end{bmatrix}
$$

$$
\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \alpha\boldsymbol{\Delta}\begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} \alpha i_1 \boldsymbol{\Delta} w_5 & \alpha i_1 \boldsymbol{\Delta} w_6 \\ \alpha i_2 \boldsymbol{\Delta} w_5 & \alpha i_2 \boldsymbol{\Delta} w_6 \end{bmatrix}
$$

Abb. 11. Backpropagation des Fehlers zu den Eingängen des Beispielnetzwerkes

Der Einfluss des Ausgabefehlers bei der Rüchpropagation nimmt von Schicht zu Schicht ab. Daher sind mehrschichtige Netzwerke zunächst schwer/langsam (bis gar nicht) trainierbar.

- Bei Transferfunktionen mit Sättigung (Clipping) kann es zu "toten" Netzwerknoten kommen,
  - d.h. weder eine kleine Änderung am Eingang eines Neurons noch eine kleine Korrektur der Gewichte/des Bias führen zu einer Änderung des Ausgangswertes kommen (gesättigte Netzwerkknoten)
  - Eine weitere Fehlerpropagation wird dadurch verhindert
- Ausweg: Randomisiertes Drop-out (Abschalten von Neuronen) und Suche nach gesättigten Neuronen mit anschließender Parameterkorrektur so dass der Ausgang der Transferfunktion in den Arbeitsbereich verlegt wird!

Kategorische Multiklassen Probleme

- Wenn die Ergbnisvariable vom kategorischen Typ ist dann gibt es zwei Möglichkeiten:

One-Hot Kodierung

Jedes Klassensymbol (also ein diskreter Wert v_i der Zielvariable y) wird durch ein Ausgangsneuron repräsentiert

Multi-level Kodierung

Jedes Klassensymbol wird durch einen Wert aus dem Wertebereich eines Ausgangsneurons repräsentiert

- Problem: Nicht lineare Transferfunktion und Sättigungsverhalten
- Die gleichen Verfahren sind auch auf kategorische Eingabevariablen anwendbar

Numerische Prädiktorfunktionen

- Neben der Klassifikation lassen sich mit ANN auch numerische (kontinuierliche) Funktionen lernen
- Damit wird **Funktionsapproximation** wie bei den Regressionsverfahren möglich
  - Unterschied: Bei der Regression ist die funktionale Struktur von $f(x)$: $x \rightarrow y$ bereits fest und muss vorgegeben sein
  - Die Verwendung eines ANN bietet da auch noch indirekt das Lernen der funktionalen Strukturen neben der Anpassung der Parameter

- Es können auch mehrdimensionalen Vektorfunktionen (also mit mehreren Ausgabevariablen) approximiert werden durch:
  1. Mehrere Ausgangsneuronen (gekoppeltes Netzwerk)
  2. Mehrere Netzwerke mit jeweils einem Ausgangsneuron (entkoppelte Netzwerke)

Die Wahl der Transferfunktion muss sorgfältig geschehen. Nichtlinearitäten der Transferfunktionen in den Randbereichen des Übertragungsbereichs muss berücksichtigt oder genutzt werden.

- Die Sigmoid (Log Rregression) Funktion ist abschnittsweise linear - ähnlich einem elektronischen Operationsverstärker ⇒ Analoge Rechner!!
- Begrenzung/Sättigung schränkt den Lösungsraum ein (gewollt!)

Literatur zur Vertiefung

[1] M. T. Hagan, howard B. Demuth, M. H. Beale, and O. D. Jesus, *Neural Network Design*. https://hagan.okstate.edu/nnd.html

Zusammenfassung

- Neuronale Netze bestehen aus Neuronen
- Neuronen sind Funktionen
- Die Kanten verbinden Ausgänge von Neuronen mit den Eingängen nachfolgender Neuronen mit einer Multiplikation eines Gewichtfaktors
- Alle Eingänge eines Neurons werden summiert, das Ergebnis einer Transfer/Aktivierungsfunktion übergeben (reduktion eines Vektors auf Skalar)
- Training ist ein Minimierungsproblem und bedeutet Anpassung der Gewichte um den Ausgangsfehler zu minimieren
    - Gängiges Verfahren: Fehlerrückpropagation und Fehlergradient