Maschinelles Lernen und Datenanalyse

In der Werkstoff- und Prüftechnik

Prof. Dr. Stefan Bosse

Universität Koblenz - FB Informatik - Praktische Informatik

Universität Siegen - FB Maschinenbau / LMW

Regressionsverfahren, SVM und KNN

Bisher wurden vor allem kategorische Zielvariablen betrachtet. Nun soll ein Schwerpunkt auf numerischen Variablen liegen.

Regressionsverfahren, SVM und KNN

Bisher wurden vor allem kategorische Zielvariablen betrachtet. Nun soll ein Schwerpunkt auf numerischen Variablen liegen.

Regressionsverfahren passen eine parametrisierte mathematische Funktion an Messdaten an.

Regressionsverfahren, SVM und KNN

Bisher wurden vor allem kategorische Zielvariablen betrachtet. Nun soll ein Schwerpunkt auf numerischen Variablen liegen.

Regressionsverfahren passen eine parametrisierte mathematische Funktion an Messdaten an.

Neben den "klassischen" Regressionsverfahren wie Least Square Fit gehören grundsätzlich auch Support Vector Machines (SVM) und Künstliche Neuronale Netzwerke dazu!

SVM gehören zu den Regressionsverfahren

SVM gehören zu den Regressionsverfahren

SVM nutzen aber bei der Parameteranpassung (Training) eine andere Fehlerfunktion (Loss) als bei anderen gängigen Regressionsverfahren (z.B. Least-Square Minimierung)

SVM gehören zu den Regressionsverfahren

SVM nutzen aber bei der Parameteranpassung (Training) eine andere Fehlerfunktion (Loss) als bei anderen gängigen Regressionsverfahren (z.B. Least-Square Minimierung)

SVM können primär kategorische und weniger numerische Zielvariablen abbilden

SVM gehören zu den Regressionsverfahren

SVM nutzen aber bei der Parameteranpassung (Training) eine andere Fehlerfunktion (Loss) als bei anderen gängigen Regressionsverfahren (z.B. Least-Square Minimierung)

SVM können primär kategorische und weniger numerische Zielvariablen abbilden

SVM sind aber (zunächst) lineare Klassifikatoren!

Regressionsverfahren

- Klassifikationsprobleme sind durch Booleschen Ausgabevariablen gekennzeichnet (Klasse c_i={true,false})

- Bei Regressionproblemen findet hingegen eine Ausgabe mit kontinuierlichen Variablen statt, idealerweise $y \in [0,1]$

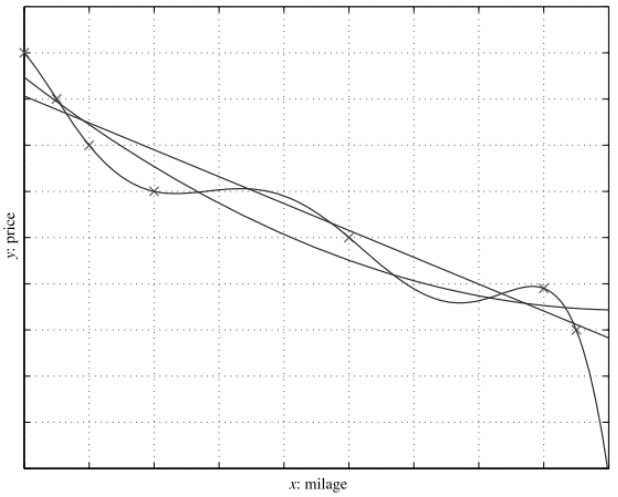
- D.h. es gibt Trainingsdaten mit:

$$D = X^t = \left\{ \vec{x}^t, r^t \right\}_{t=1}^{N}$$

wobei $r \in \mathbb{R}$ (kontinuierliche Zielvariable). Wenn Rauschen vernachlässigt wird handelt es sich um ein reines Interpolationsproblem.

- Das Ziel ist es nun, eine Funktion $f(\vec{x})$ zu finden, die die Trainingsdaten optimal repräsentiert (also die beste Hypothese g von f finden)

Beispiel



[19] Abb. 1. Lineare Gerade, Polynome zweiter Ordnung und sechster Ordnung werden an denselben Satz von Punkten angepasst. Die höchste Ordnung ergibt eine perfekte Passform, aber angesichts dieser vielen Daten ist es sehr unwahrscheinlich, dass die reale Kurve so geformt ist. Die zweite Ordnung scheint besser zu sein als die lineare Anpassung bei der Erfassung des Trends in den Trainingsdaten (Extrapolation).

Es wird immer einen Fehler ε geben (Rauschen in den "Trainingsdaten" $\boldsymbol{r}$):

$$y\left(\vec{x}^t\right) = r^t = f\left(\vec{x}^t\right) + \epsilon$$

- Ziel der Regression ist es diesen Fehler über eine Verlustfunktion zu minimieren in dem Parameter $\vec{\Theta}$ der Funktion *f* angepasst werden:

$$\arg \min_{\theta} \quad \epsilon \rightarrow E(g|X) = \frac{1}{N} \sum \left(r^t - g\left(\vec{x}^t\right)\right)^2$$

Lineare Multivariate Regression

- Es wird angenommen dass die Funktion $f(\vec{x})$ durch eine lineare Funktion über $\vec{x}$ abgebildet werden kann.
- Dann gilt:

$$y_p(\vec{x}) = g(\vec{x}) = w_1 x_1 + w_2 x_2 + .. + w_d x_d + w_0 = \sum_{j=1}^{d} w_d x_d + w_0$$

Schon dieses Problem kann unterbestimmt sein, d.h., es kann unendlich viele Hypothesen g von der unbekannten Funktion f geben!

- Bei nichtlinearen Zusammenhängen wird die Regressionsfunktion noch komplexer!

Nichtlineare Univariate Regression

- Es gibt nur eine Eingabevariable und die Hypothesenfunktion wird durch ein Polynom *k*-ter Ordnung approximiert:

$$g(x) = w_0 + w_1 x + w_2 x^2 + .. + w_k x^k = \sum_{j=1}^{k} w_j x^j + w_0$$

- Wird von einem Polynom ersten Grades ausgegangen (Gerade) dann gilt es folgende Gleichung zu bestimmen und das Minimierungsproblem zu lösen:

$$g(x) = w_1 x + w_0$$

$$E\left(w_1, w_0 | X^t\right) = \frac{1}{N} \sum_{t=1}^{N} \left(r^t - \left(w_1 x^t + w_0\right)\right)^2$$

- Den Minimumspunkt dieser Fehlergleichung findet man durch **Gradientenbildung** der Parameter, das bedeutet dann:

$$w_1 = \frac{\sum_t x^t r^t - \overline{x}\overline{r}N}{\sum_t \left(x^t\right)^2 - N\overline{x}^2}$$

$$w_0 = \overline{r} - w_1\overline{x}$$

$$\overline{x} = \sum_t \frac{x^t}{N}, \overline{r} = \sum_t \frac{r^t}{N}$$

Nichtlineare Multivariate Regression

- Sehr hochdimensionales Problem! Und i.A. völlig unterbestimmt (d.h. kein eindeutigen Zusammenhänge zwischen $\vec{x}$ und y)

$$g(\vec{x}) = \sum_{i=1}^{d}\sum_{j=1}^{k} w_{i,j}x_i^j + w_0$$

Es können auch exponentielle, logarithmische, und sinusoidale Terme hinzukommen!

- Ein numerisches Lösen ist meist nicht mehr möglich; daher Verwendung nichtlinearer Randbedingungslöser sowie statistische Verfahren wie randomiserte Monte Carlo Simulation und Simmuliertes Abkühlen (Evolutionäre Algorithmen?)

Warum können hochdimensionale Polynome nicht mehr mit gradientenbasierten Verfahren numerisch auf einem Computer (gut oder überhaupt) lösbar sein?
Hinweis: Wie entwickeln sich Gradienten bei Polynomen sehr hoher Ordnung oder gar Exponentialterme wie b^n?

???

Warum können hochdimensionale Polynome nicht mehr mit gradientenbasierten Verfahren numerisch auf einem Computer (gut oder überhaupt) lösbar sein?
Hinweis: Wie entwickeln sich Gradienten bei Polynomen sehr hoher Ordnung oder gar Exponentialterme wie b^n?

???

Nicht begrenzter und steigender Gradient!

Direkte Lösungsverfahren

Single Value Decomposition (SVD)

- Erweiterung der Eigenwertanalyse und verwendet Matrixalgebra und Inversionsmethoden [4]
- Ansatz: Dekomposition einer (nichtlinearen) Funktion $f(\vec{x},\vec{\Theta})$ mit b Basisfunktionen ϕ, z.B. *sin* oder ähnlich, mit Parametersatz Θ:

$$f_\Theta(\vec{x}) = \sum_{j=1}^{b} \Theta_j \phi_j(x)$$

$$f_\Theta(\vec{x}) = \vec{\Theta}^T \vec{\phi}(x)$$

- Z.B. $\vec{\phi}(x) = (1, x, x^2, .., x^{b-1})^T$, oder $\vec{\phi}(x) = (1, \sin(x), \cos(x), \sin(2x), ...)^T$

- Die gesamte Trainingstabelle wird in Matrixform repräsentiert, und somit erhält man eine Parametermatrix $\hat{\Phi}$ mit den Basisfunktionstermen für die anzupassenden Funktion $f(\vec{x})_\Theta$ (Design Matrix):

$$\hat{\Phi} = \begin{pmatrix} \phi_1(\vec{x}_1) & .. & \phi_b(\vec{x}_1) \\ \dots & \dots & \dots \\ \phi_1(\vec{x}_n) & .. & \phi_b(\vec{x}_n) \end{pmatrix}$$

Die Größe der Design Matrix als Ausgangspunkt für SVD/LS Verfahren wächst quadratisch mit der Anzahl der Trainingsdateninstanzen!

- Die Lösung (der Funktionsparameter Θ) geschieht dann doch Matrixinversion der Designmatrix Φ:

$$\hat{\Theta}_{LS} = \left(\Phi^T \Phi\right)^{-1} \Phi^T y$$
$$\hat{\Theta}_{LS} = \Phi^G y$$

mit Φ^G als generalisierte Inverse der Matrix Φ

Die generalisierte Inverse wird dann mit dem SVD Verfahren mit sogenannten links- und rechtssingulären Vektoren bestimmt.

Vertiefung: M. Sugiyama, Introduction to Statistical Machine Learning. 2016, Kapitel 22.2

Support Vector Machines (SVM)

Obwohl die SVM zu den linearen (oder nichtlinearen) Regressionsverfahren gehören, wird die SVM primär für die binäre Klassifikation eingesetzt!

- SVM Verfahren gehören zu den "Maximum Margin" Methoden

Binärer Klassifikator

- Ein binärer Klassifikator soll durch eine **lineare Funktion** repräsentiert werden (*y*= {-1,+1}, linearer Kernel):

$$f(\vec{x}) : \vec{x} \rightarrow y = \vec{w}^T \vec{x} + \gamma$$

Dabei sind $\boldsymbol{w}$ und γ die Parameter des Modells die durch das Training an das Problem angepasst werden müssen.

- $\boldsymbol{w}$ ist ein Normalenvektor der bei einem binären Klassifikationsproblem die beiden Instanzklassen trennt
- *y* kann tatsächlich auch außerhalb der Grenzen [-1,1] liegen (klar: Polynome haben keine Grenzen)!

[4]

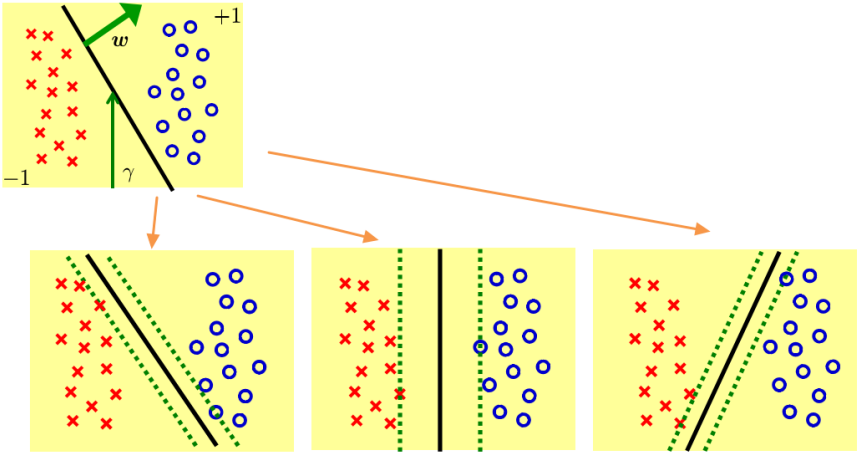


Abb. 2. (Oben) ***w*** ist der Normalenvektor und γ die Verschiebung der Trennungsgrenze für zwei Klasseninstanzen (Unten) Verschiedene ***w***/γ Varianten der Trennungsgrenze mit unterschiedlichen Rändern (Sicherheitsbereichen)

Training

- Das Lernen von $\boldsymbol{w}$ und γ erfordert die Berechnung des Abstandes von allen Dateninstanzen $(\boldsymbol{x},y)_\mathrm{i}$ von der Trenngrenze. Die Abstände müssen positiv sein:

$$f(\vec{x}_i)y_i = (\vec{w}^T\vec{x}_i + \gamma)y_i > 0, \forall i$$

für alle Dateninstanzen $\boldsymbol{D}=\{(\boldsymbol{x}_\mathrm{i},y_\mathrm{i})\}^\mathrm{n}$.

- Da $\boldsymbol{w}$ und γ beliebig gewählt werden können, kann die Randbedingung auch mit $(..)y_\mathrm{i} \geq 1$ gewählt werden.
- Weiterhin kann es sinnvoll sein alle Dateninstanzen um den Ursprung des Koordinatensystems zu **zentrieren**

Wichtig: Die Werte für *y* liegen im Intervall [-1,1]!

- Alle Probleme die (..)$y_i \geq 1$ erfüllen mit einem ($\boldsymbol{w}$,γ) sind linear separierbar.
- Es gibt unendlich viele Lösungen (also Entscheidungsgrenzen)
- Man wählt das ($\boldsymbol{w}$,γ) aus bei der alle Dateninstanzen die größte Trennung besitzen (breitester Trennbereich, siehe Abb.)
- Der Abstand der Dateninstanzen $\boldsymbol{D}$ ist definiert als das Minimum des normalisierten Abstandes:

$$m_i = (\vec{w}^T \vec{x}_i + \gamma) y_i / ||\vec{w}||$$

- D.h. SVM zu trainieren ist das Minimierungsproblem zu lösen:

$$\min_i \frac{(\vec{w}^T \vec{x}_i + \gamma) y_i}{||w||} = \frac{1}{||w||}$$

Vertiefung: M. Sugiyama, Introduction to Statistical Machine Learning. 2016., Kapitel 27

- Jede Dateninstanz die nicht in den Trennbereich "eindringt" ist ein Supportvektor!

Harter Trennungsbereich (Hard SVM)

- Bei einer "harten" Trennung einer SVM gilt dann:

$$\min_i 1/2||w||^2, (\vec{w}^T\vec{x}_i + \gamma)y_i \geqslant 1, \forall i$$

Hier wird aber keine Lösung für $\boldsymbol{w}$ und γ gefunden wenn das Problem nicht strikt linear separierbar ist (also keine einzige Gerade die Klassen trennen kann)

Weicher Trennungsbereich (Soft SVM)

- Die SVM mit harten Trennungsbereich erfordert lineare Separierbarkeit der Dateninstanzen
  - Ist in der Realität aber nicht immer oder eher selten gegeben
- Die weiche Trennung durch eine SVM führt einen Fehlerparametervektor $\xi=\{\xi_i\}^n$ für die Bestimmung des Trennbereichs ein:

$$\min_{\forall i: w,\xi,\gamma} \left[1/2||w||^2 + C\sum_i \xi_i \right],$$
$$(\vec{w}^T\vec{x}_i + \gamma)y_i \geqslant 1-\xi_i, \xi_i \geqslant 0, \forall i$$

- Sie lässt einzelne nicht (linear) separierbare Datenpunkte zu (zur Erinnerung: *i* ist der i-te Datenpunkt, d.h. die i-te Dateninstanz).

[4]

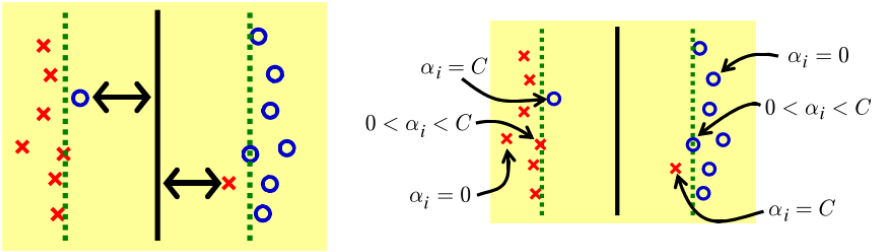


Abb. 3. Weicher Trennbereich einer SVM (Soft margin SVM). Durch ξ werden kleine Klassifikationsfehlerbereiche erlaubt.

Die Ausreißer können durch Rauschen und Messunsicherheit (random. und systematischer Fehler) aber auch aufgrund eines nichtlinear separierbaren Problems entstehen!

Dabei ist C ein einstellbarer Parameter der den Fehler steuert und für den gilt:

$$C = \alpha_i + \beta_i$$

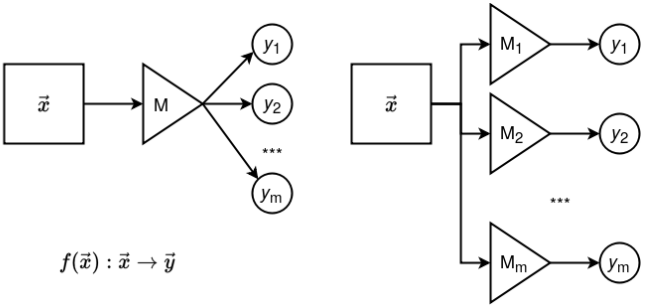
Größere C Werte machen den Abstandsfehler kleiner und für große C geht die weiche in eine harte SVM über

- $\alpha_i = 0$ impliziert $m_i \geq 1$: die i-te Trainingsinstanz x_i ist auf der Margengrenze oder innerhalb der Marge und ist korrekt klassifiziert.
- $\alpha_i = C$ impliziert $m_i \leq 1$: x_i ist auf der Margengrenze oder außerhalb der Marge. Wenn $\xi_i > 1$, $m_i < 0$ dann ist x_i falsch klassifiziert.
- $0 < \alpha_i < C$ impliziert $m_i = 1$: x_i ist auf der Margengrenze und ist korrekt klassifiziert.
- $m_i > 1$ impliziert $\alpha_i = 0$: wenn x_i innerhalb der Marge ist, $\alpha_i = 0$.
- $m_i < 1$ impliziert $\alpha_i = C$: wenn x_i außerhalb der Marge ist, $\alpha_i = C$.

Multiklassen SVM

Jedes Multiklassenproblem mit *m* verschiedenen (diskreten) Klassenwerten kann auf *m* binäre Klassifikationsprobleme transformiert werden

- Anders als bei ANN ist bei SVMs aber nur eine One-hot Kodierung möglich (ggfs. mit Softmaxfunktion).



R SVM (custom)

```
model <- svm(
    x = data.frame,
    y? = vector, # scaled to [-1,1]?
    formula? = y ~ x,
    # threshold function on output? highest value of multi-svms is winner
    threshold = boolean,
    # default : 1.0. C in SVM.
    C = number,
    # default : 1e-4. Higher tolerance --> Higher precision
    tol = number,
    # default : 20. Higher max_passes --> Higher precision
    max_passes = number,
    # default : 1e-5. Higher alpha_tolerance --> Higher precision
    alpha_tol = number,
    kernel = string|list, # linear, rbf, .. list(type= 'rbf', sigma= 0.5) ..
    # list( type = "polynomial", c =  1, d = 5=
)
```

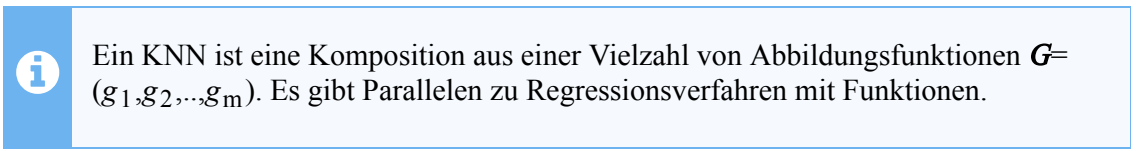
Beispiel

Web WorkShell Live

CLEAR LOAD + - R data prepro train test

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.

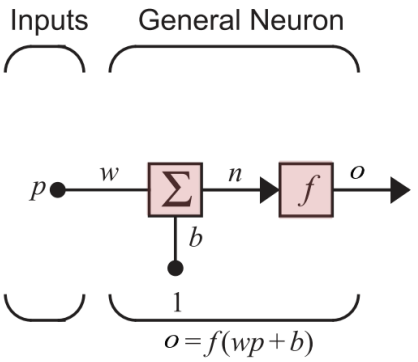
- Zusammengefasst ausgedrückt:

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

- *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 $\boldsymbol{N}_{\mathrm{d}}$

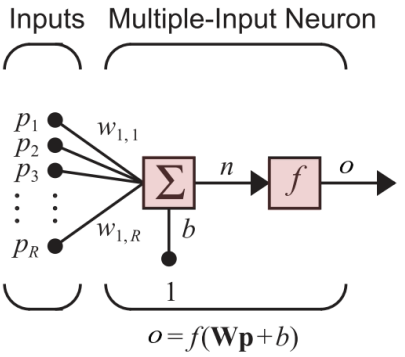
Das Neuron



[15]

Abb. 4. 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

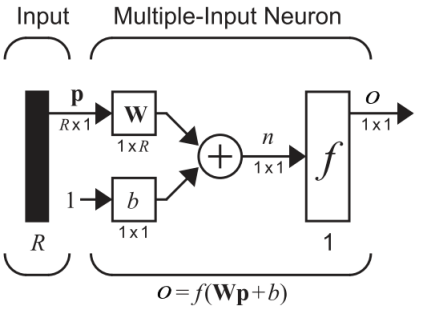


[15]

Abb. 5. Ein einzelnes Neuron mit einem Eingangsvektor ***p*** und einem skalaren Ausgang *o*. ***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:

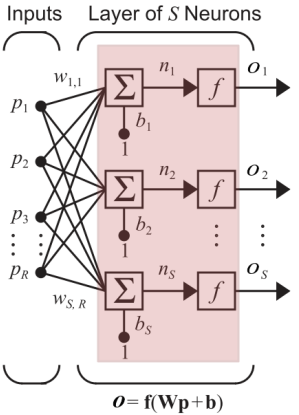


[15]

Abb. 6. Ein einzelnes Neuron mit einem Eingangsvektor ***p*** und einem skalaren Ausgang *o*. ***w*** ist ein Gewichtungsfaktorvektor (ein Gewicht für eingehendes *p*) und *b* ist ein Bias (Offset); jetzt in Matrizenform (Annotation)

Schichten von Neuronalen Netzen

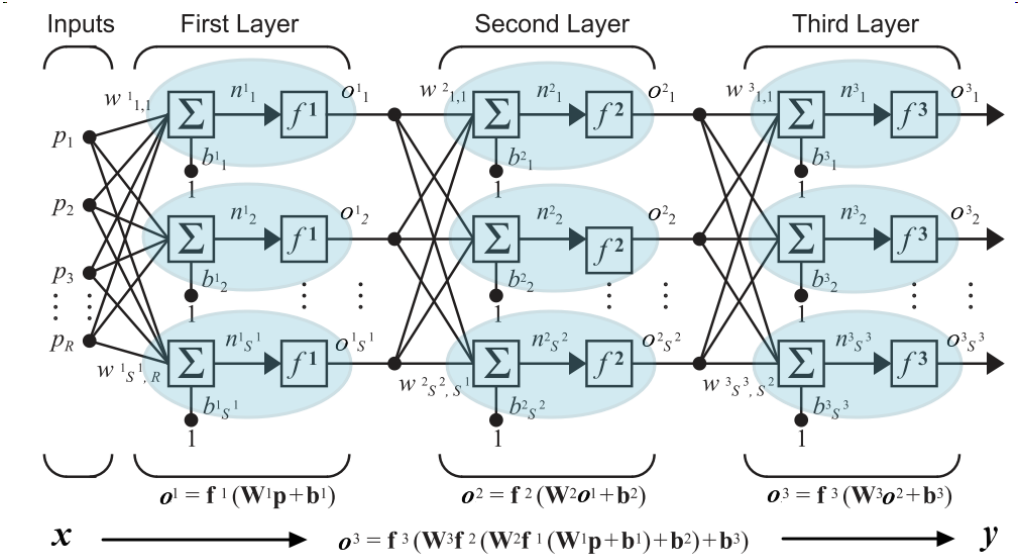
- I.A. werden Neuronen von neuronalen Netzen in Schichten (Layer) angeordnet und gruppiert
  - Günstig für Matrixalgebra
  - Aber nicht notwendig!



[15]

Abb. 7. Neuronales Netzwerk mit Neuronen in einer Schicht angeordnet

Struktur eines KNN



[15]

Abb. 8. Grundlegende Struktur eines KNN mit Matrizen (blaue Ellipse=1 Neuron)

Vereinfachte Form eines KNN

[15]

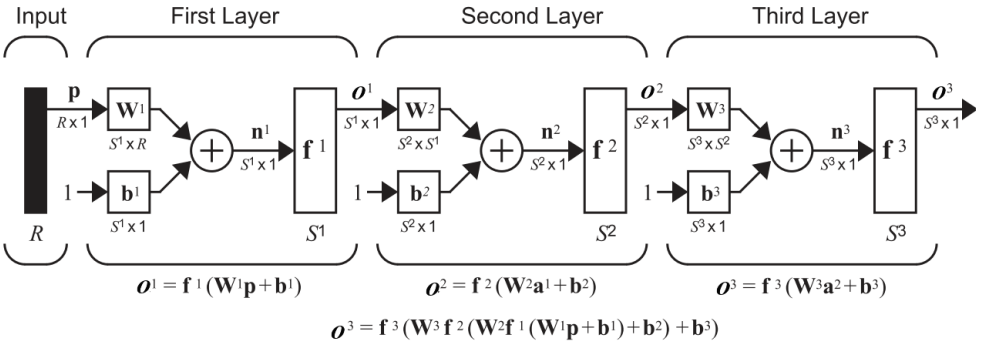


Abb. 9. Vereinfachte Struktur eines KNN mit Matrizen

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]

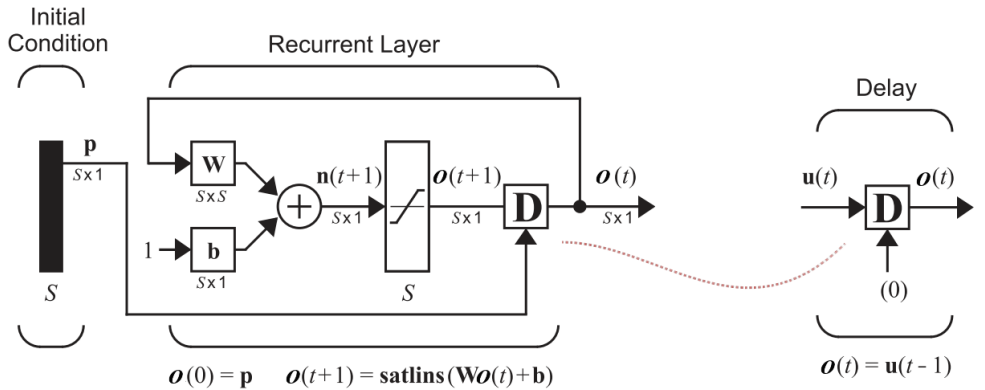


Abb. 10. Rückgekoppeltes und zustandsbehaftetes KNN mit einer Verzögerungsfunktion (Speicher)

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): $f(a) = a$

- 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	C	compet

Abb. 11. 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 + - R 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 $\mathbf{w}$) 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 + - R Neuron

Nichtlineare Probleme

SLP können nur lineare Probleme separieren.

[15]

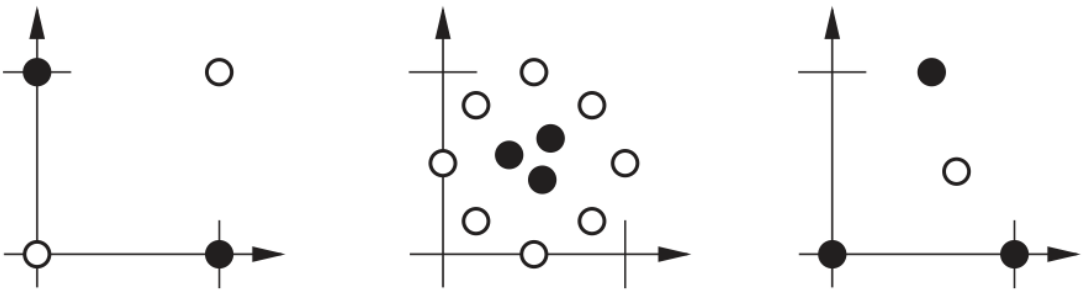


Abb. 12. Nichtlinear separierbare Probleme - nur mit MLP klassifizierbar

Web WorkShell Live

CLEAR LOAD + - R 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. $f(x,\boldsymbol{w})$: $x \rightarrow y$ derart über den Parameter w angepasst so dass der Fehler $err=|y-y_0|$ minimal wird

- Es wird nun die Änderung des Fehlers *∂err* beobachtet und der (oder später die) Parameter *w* mit der Ableitung des Fehlerwerts *∂err*/*∂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 für die analytische Ableitung aber (grobe Näherung), d.h. die numerische Ableitung:

$$\frac{\partial err}{\partial w_i} \sim \frac{\Delta err}{\Delta w_i} = \frac{\Delta(y - y_0)}{\Delta w_i} = \frac{\Delta(f(x,\vec{w}) - y_0)}{\Delta w_i} = \frac{(f(x, w_i + \epsilon) - f(x, w_i) - y_0)}{\epsilon}$$

- 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]

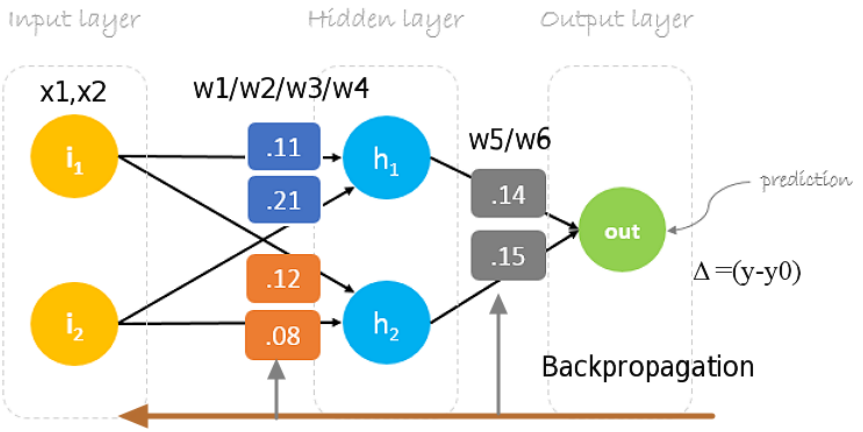


Abb. 13. Beispiel eines ANN mit Kantengewichten und dem Ansatz der Backpropagation

- Die Anpassung in der letzten Ausgabeschicht ist einfach, da der Fehler direkt verfügbar ist ($E=y_i-y_{i,0}$)

- Es wird für jedes Gewicht die Ableitung an der Stelle x_i, also der jweilige EIngangswert der Funktion, entweder analytisch ode numerisch approximiert mit einer kleine VAriation ε des Parameters w_i berechnet.

1. Berechnung aller Gradientenwerte (gesamtes Netzwerk)
2. Anpassung der Gewichte anhand der berechneten Gradientenwerte

- Die Anpassung in inneren/ vorderen Schrichten bedarf der Berechnung von rückwärts geleiteten Fehlerwerten, d.h. der Ausgangsfehlerwert wird zurück propagiert, und damit dann jeweils der Fehlergradient berechnet.

- 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. 14. Backpropagation des Fehlers zu den Eingängen des Beispielnetzwerkes

- Ansatz: Jeder Knoten *i* in der Schicht *n* liefert (bei FCANN) einen Beitrag über die gewichteten Kanten zu allen Konten *j* der Schicht *n*+1=*m*.
- Sei *n* die Aktuelle Schicht mit *I* Konten und *m* die nächste Schicht mit *J* Knoten, d.h. bei der Rückpropagation die vorherige. Dann gilt für den Fehler e_i des i-ten Knoten in *n* (vor *m*) die gewichtete Fehlersumme:

$$e_i = \sum_{j=1}^{J} e_j w_i$$

$$g_k = \frac{\partial f}{\partial w_k}$$

$$eg_k = e_i g_k$$

- Die Ableitung des Fehlers für das k-te Gewicht des i-ten Knotens wird in den Fehlerterm e_i (gesamter Knoten) und die reine Ableitung der Funktion nach dem Parameter w_k aufgespalten.
- Schließlich werden beide Werte multipliziert

Der Einfluss des Ausgabefehlers bei der Rüchpropagation nimmt von Schicht zu Schicht praktisch ab. Daher sind mehrschichtige Netzwerke zunächst schwerer/langsamer (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

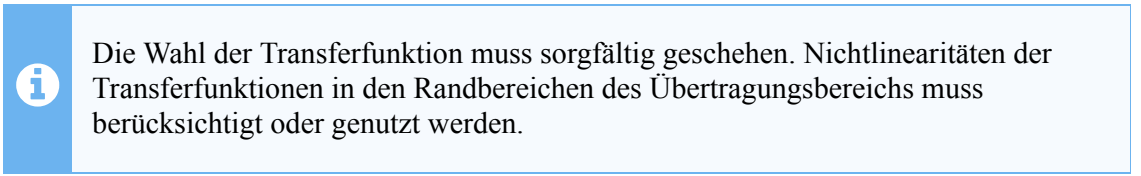
Jedes Klassensymbol wird durch einen Wert aus dem Wertebereich eines Ausgangsneurons repräsentiert

- 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 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

[Sugiyama, ItSML, pp 303]

Zusammenfassung

- Regressionsverfahren werden auf kontinuierliche Zielvariablen angewendet (der Hypothesenraum kann bei nichtlinearen Problemen sehr groß werden)
- Eine SVM wird als binärer Klassifikator verwendet und wird i.A. durch eine lineare Funktion (Kernel) repräsentiert
  - Das Problem sollte dann linear separierbar sein!
- Multiklassenprobleme werden auf Multi-SVMs zurückgeführt
  - Verwendung einer Softmax Funktion für eindeutige Klassentrennung
- Das Training einer SVM ist ein Minimierungsproblem dass den Trennbereich maximiert und den Fehler minimiert

Zusammenfassung

- Neuronale Netze bestehen aus Neuronen
- Neuronen sind zusammengesetzte Funktionen: Produktsummation und Transferfunktion
- 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