

# Maschinelles Lernen und Datenanalyse

*In der Mess- und Prüftechnik*

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

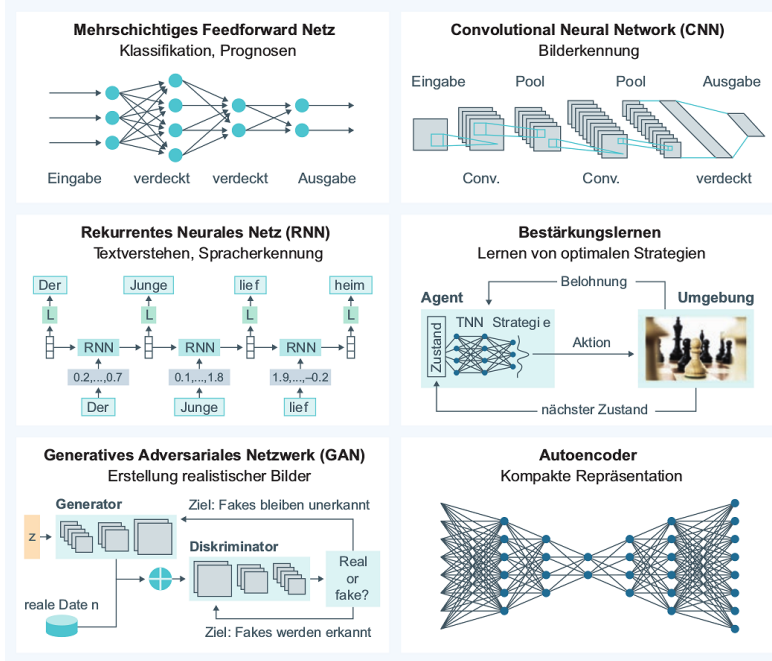


Abb. 1. Zusammenstellung von sechs unterschiedlichen Netzwerkarchitekturen. Der Name der Architektur steht über der Grafik, darunter folgt der hauptsächlich Anwendungsereich.

# Zustandsbasierte Netze

Bisher wurden nur "vorwärtsgerichtete" ANN betrachtet (Feed forward/ FF-ANN)



Die Ausgänge von FF-ANN hängen nur von aktuellen Eingängen ab!

Rückgekoppelte ANN (Recurrent / RNN) besitzen Zustand und Gedächtnis!

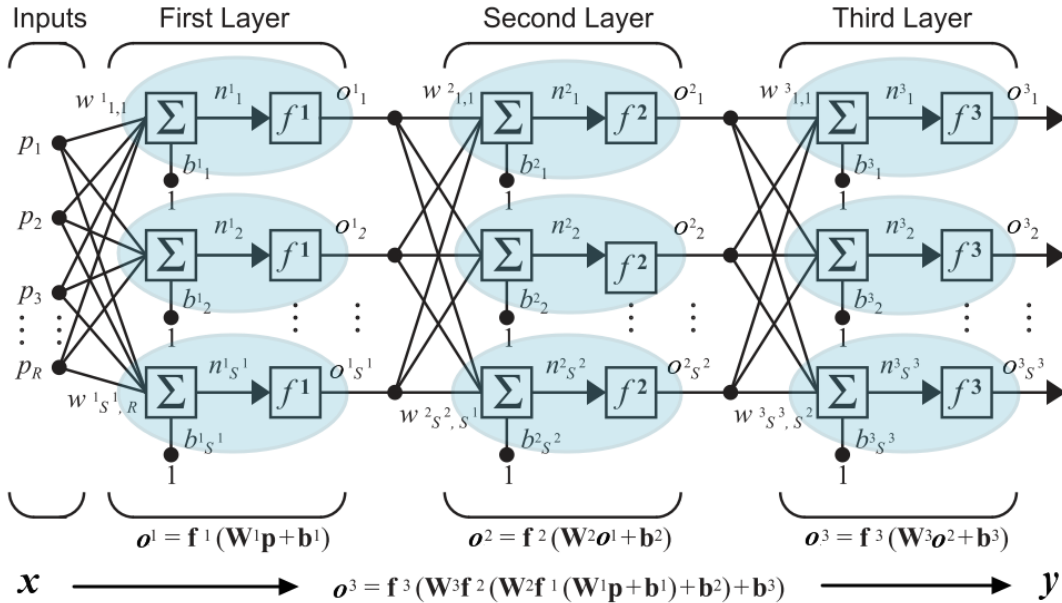
Eignung für Daten- und Zeitserienprädiktion!

## Wiederholung: FF-ANN

- Ein vorwärtsgekoppeltes ANN (Feed forward) besteht aus Neuronen die jeweils durch Funktionen  $f(\mathbf{i}): \mathbf{i} \rightarrow o$  repräsentiert werden können.
- Die Funktionen  $f$  werden als azyklischer gerichteter Graph dargestellt (also das NN), d.h., kein Neuron hat ein Eingangssignal von nachfolgenden Neuronen (die das Ausgangssignal dieses Neurons enthalten)



D.h: Das aktuelle Ausgangssignal  $\mathbf{y}$  eine FF-ANN hängt nur von den aktuellen Eingangssignalen  $\mathbf{x}$  ab!



## Recurrent ANN: Die Rückkopplung

- Ausgänge von Ausgangs- oder inneren Neuron werden auf Eingänge von vorherigen Neuronen gekoppelt
- Dadurch werden die Netzwerke zustandsbasiert, d.h., die aktuellen Ausgänge hängen neben den aktuellen Werten der Eingänge von der Historie der Werte von Ein- und Ausgängen ab!
- Recurrent ANN (RNN) sind durch hohe "Instabilität" schwierig mit gradientenbasierten Verfahren zu trainieren!
- Daher im Laufe der Zeit verbesserte Architekturen wie LSTM oder GRU Netzwerke

# LSTM Netzwerke

- Long-short-term Memory (LSTM) Netzwerke bilden eine bekannte Architektur die:
  - Für die Daten- und Zeitserienprädiktion verwendbar sind
  - Mit gradientenbasierten Trainingsverfahren einigermaßen stabil (konvergent) trainiert werden können

## Daten- und Zeitserien

- Es sei  $\{x_n\}^t$  eine Serie von Eingabedaten (z.B. zeitaufgelöstes Sensorsignal einer Ultraschallmessung), d.h.  $\{x_n\} = \{x_1, x_2, \dots, x_t\}$
- Die einzelnen Werte (skalar oder vektoriell) sind aufeinander folgend → **Ordnungsrelation**

$$f_{\delta}(\{s_i\}_i^n, n) : s_{n-m}, s_{n-m+1}, \dots, s_n \rightarrow s_{n+\delta}$$

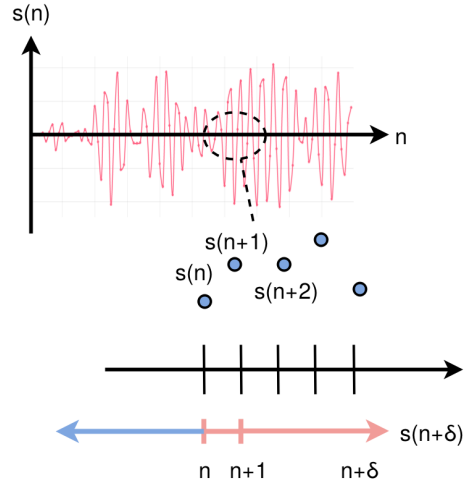


Abb. 2. Datenserie und Prädiktion  $s(n+\delta)$



- Neben der Prädiktion von "zukünftigen" Entwicklungen einer Variable  $x_i$  kann auch eine zukünftige Prognostik einer anderen Zielvariable  $y_i$  erfolgen:

$$f_{\delta}(\{x_i\}_i^n, n) : \{x_{n-m}, x_{n-m+1}, \dots, x_n\} \rightarrow y_{n+\delta}$$

- Etwaige Zielvariablen könnten aus dem Signal abgeleitete Eigenschaften (Merkmale) sein:
  - Schadensinformation
  - Änderung einer zweiten Sensorvariable
  - Zustandsvariable
  - Veränderung von Betriebsbedingungen
  - Prozessparameter

# Netzwerkarchitekturen

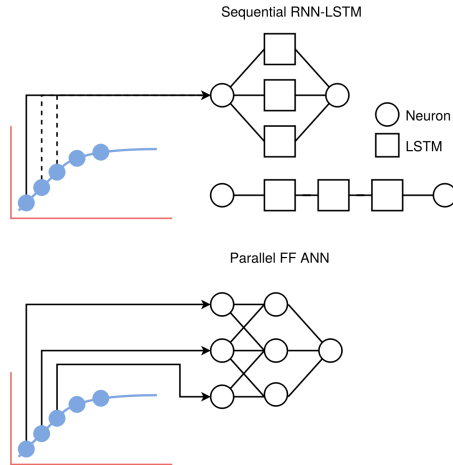


Abb. 3. (Oben) Die Datenserie kann sequentiell (in ein RNN/LSTM) oder auch (Unten) parallel in ein FNN eingegeben werden

# LSTM Zelle

- Eine LSTM Zelle besteht aus:
  - Einer "Speicherzelle"
  - Mehreren Gattern die aktiv Verbindungen (Kanten) zwischen Neuronen steuern, d.h., effektiv das Kantengewicht in Abhängigkeit von Ausgangswerten von Neuronen verändern /Ventile/
- Eine wichtige Rolle bei der Speicherung vergangener Daten (Gedächtnis) ist das Vergessen gesteuert durch das "Forget Gate"

Beispiel für eine "gedämpfte" Gedächtnisfunktion: Ein Tiefpassfilter 1. Ordnung

$$f(x, n) = f(n - 1) \cdot \alpha + \beta \cdot x$$

- Der Parameter  $\beta=(0,1)$  bestimmt den Einfluss neuer Werte  $x$  und der Parameter  $\alpha=(0,1)$  bestimmt den Einfluss alter Werte (Vergessen); i.A.  $\alpha=1-\beta$

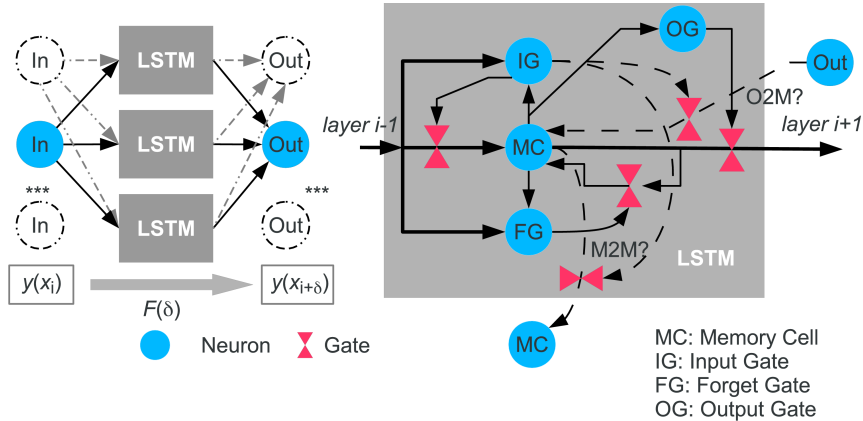


Abb. 4. Aufbau einer LSTM Zelle mit zentraler Speicherzelle (Neuron) und den Gattern

## LSTM Demo

- Lernen einer Prädiktorfunktion für Datenserien nach der Sinusfunktion mit LSTM Netzwerk:

$$f_{\delta}(x_n) : x_n \rightarrow x_{n+\delta}$$

- Für jedes  $\delta = \{1, 2, \dots, m\}$  muss eine eigene Prädiktorfunktion  $f_{\delta}$  erzeugt und trainiert werden
- Konfiguration des Netzwerks:  $[1, l, 1]$ , wobei  $l = \{1, 2, \dots\}$  die Anzahl der LSTM Zellen in der inneren Schicht ist.

# Web WorkShell Live

**CLEAR** **LOAD** **+** **-** **data** **model** **train** **tes**

# Zusammenfassung

- Diskrete Daten- und Zeitserien  $x(i)$  kommen in der Mess- und Prüftechnik als Sensorsignal häufig vor, aber auch bei
  - Wirtschaftsdaten (Aktienkurse), Wetterdaten, Klimadaten, usw.
- Die Vorhersage von unbekanntem zukünftigen Datenwerten aus historischen bekannten Datenwerten erfordert **zustandsbasierte Prädiktorfunktionen** sofern es
  - keinen ausgewiesenen Start- und Endpunkt gibt (also **sequentielle Aktivierung** von  $M$ ).
  - Gibt es ausgewiesene Start- und Endpunkte der Serie kann auch eine zustandslose Prädiktorfunktion verwendet werden (**parallele Aktivierung** einer Auswahl von Datenpunkten)

- Die Long Short Term Memory Zelle ist eine bekannte Architektur für eine RNN PF mit Zustand (Speicher)
- Problem beim Trainieren von RNN ist Instabilität und der Zustandsspeicher (wirkt sich auf Fehlerberechnung ungünstig aus)



# Faltungsnetzwerke (CNN)

Faltungsnetzwerke vereinigen KNN mit mathematischen Faltungsoperationen die typischerweise auf Bilddaten angewendet werden

Bisher waren die Eingangsvariablen von Prädiktormodellen unmittelbar mit Merkmalen verknüpft (Ausnahme: Datenserien!)

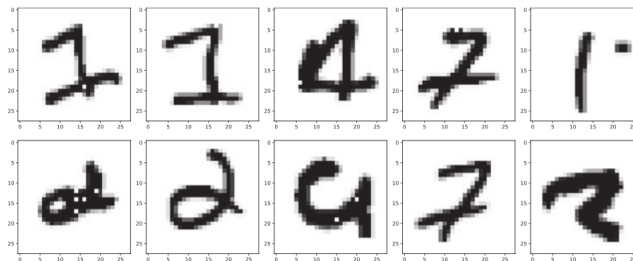
Bei der Bildererkennung möchte man häufig Objekte (also geometrische Strukturen) in Bildern finden. Diese können aber an beliebigen Stellen im Bild vorhanden sein

Die Merkmale sind nicht mehr einzelnen Variablen zuzuordnen.

Daher werden kleine Faltungsoperationen auf das gesamte Bild angewendet, die erst die Merkmale extrahieren.

# Beispiele

- Handschriftenerkennung
- Schadensdiagnostik (Bildaufnahmen, Röntgentomographie, usw.)
- Zeitaufgelöste Frequenzspektren (Datenexpansion von Zeitsignalen!)
- Bestimmung von Region of Interest Bereichen



[20]

---

Abb. 5. Beispiel aus dem MNIST Datensatz von handgeschriebene Ziffern und Buchstaben. Beispiele bei denen die Ziffernbilder nur schwer zu klassifizieren sind.

# Daten

- Man unterscheidet:
  - Vektoren (1D)
  - Matrizen (2D)
  - Tensoren (3d), wobei Tensoren mathematisch mehr sind als Vektoren von Matrizen!

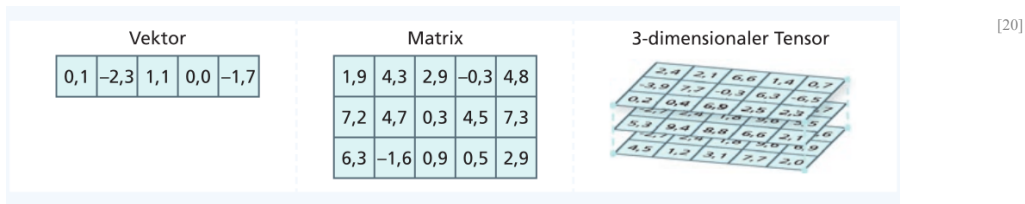


Abb. 6. Vektor, Matrix und Tensor mit beliebigen reellen Zahlen. Die Eingaben für tiefe neuronale Netze werden meist in dieser Form kodiert. Man kann auch die Dimension zu der Bezeichnung hinzufügen. Im Bild haben wir einen 5-Vektor, eine  $3 \times 5$ -Matrix und einen  $3 \times 3 \times 5$ -Tensor.

# Klassifikationsproblem

- Auch bei der Objekterkennung in Bildern handelt es sich häufig um diskrete Klassifikationsprobleme.
- Sinnvoll ist aber neben einer binären Ausgabe (Klasse ja/nein) die Ausgabe eines kontinuierlichen Wertes im Bereich  $[0,1]$ , als eine Art Bewertungsskala (Wahrscheinlichkeitsmaß)

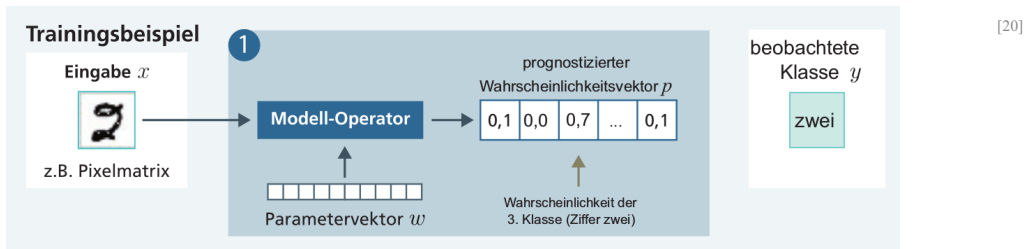
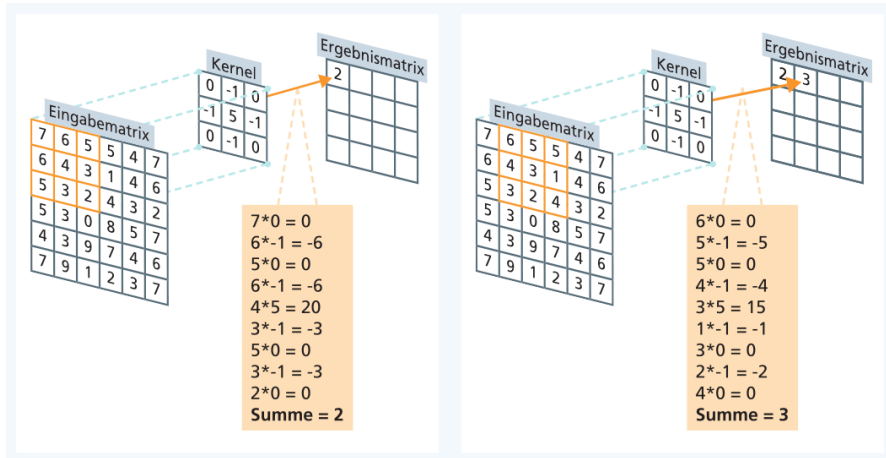


Abb. 7. Die Prognose berechnet mit dem Modelloperator für ein Trainingsbeispiel  $(x,y)(i)$  aus der Eingabe  $x$  einen Wahrscheinlichkeitsvektor  $p$  mit den Wahrscheinlichkeiten der verschiedenen Klassen (Ziffern). Ziel ist es, für die beobachtete Klasse  $y$  (z.B. „zwei“) eine möglichst hohe Wahrscheinlichkeit zu prognostizieren. Die Prognose des Modells hängt von einem Parametervektor  $w$  ab.

# Faltungsoperation



[20]

Abb. 8. Erster Berechnungsschritt (links) und zweiter Berechnungsschritt (rechts) in der Faltungsschicht für jeweils einen verschobenen kleinen Bereich der Eingabematrix. Dabei wird der Kernel sukzessive über die gesamte Eingabematrix „geschoben“ und die Ergebnismatrix gefüllt

# Convolutional Neural Networks (CNN)

- Ein CNN ist aus verschiedenen Schichten (Ebenen) zusammengesetzt:
  - Faltungsebenen
  - Zusammenführungs (pooling) Ebenen
  - Ausgabe durch Klassifikationsebenen (softmax)

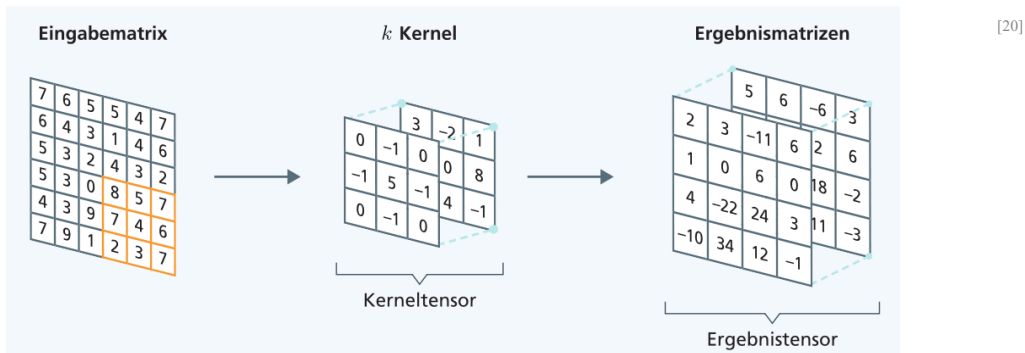


Abb. 9. Eine Faltungsschicht enthält  $k$  Kernel und  $k$  Ergebnismatrizen, welche jeweils zu Tensoren zusammengefasst werden

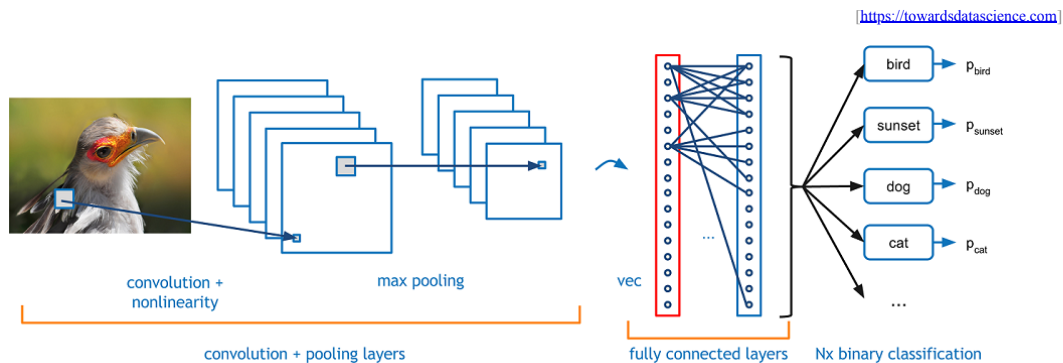


Abb. 10. Allgemeiner Aufbau eines CNN mit wechselnden Schichten aus Faltungen, Zusammenführung, und schließlich binärer Klassifikation

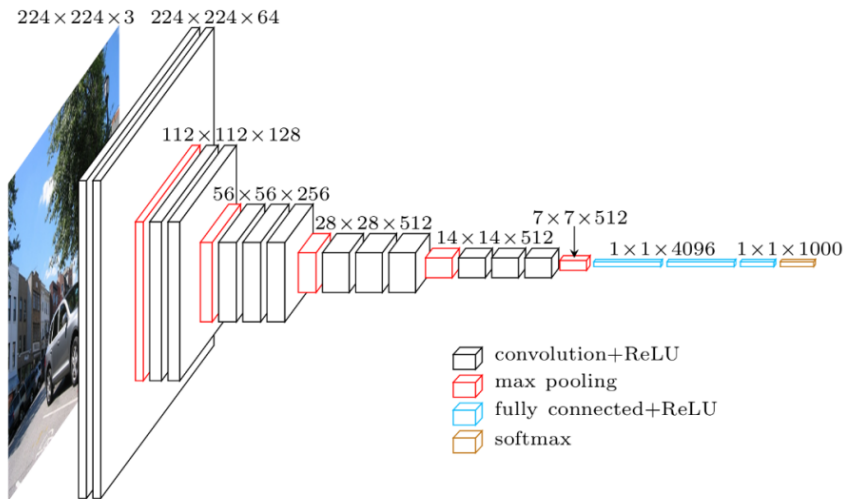


Abb. 11. Je nach Anzahl der Strukturmerkmale kann es eine große Anzahl folgender Faltungs- und Zusammenführungsschichten geben



# Zusammenfassung

CNN sind geeignet um ortsunabhängig verschiedene Strukturmerkmale in den Daten zu klassifizieren

Ein CNN beruht auf Matrixalgebra mit Faltungsoperationen

Zum Nachlesen: <https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529?gi=521747216671>; G. Paaß, Künstliche Intelligenz, Was steckt hinter der Technologie der Zukunft?, Springer

Bekanntes Softwareframework für den Browser: convnet.js  
<https://cs.stanford.edu/people/karpathy/convnetjs>

# Inverse Modellierung

Meistens kann ein Modell  $M(X): X \rightarrow Y$  empirisch bestimmt werden

Häufig ist das inverse Modell von Bedeutung:  $M^{-1}Y: Y \rightarrow X!$

ML bietet Möglichkeit "Prädiktives Modellieren"

Aber wie kann man  $M^{-1}$  aus  $M$  ableiten?

# Inverse Funktionen: Analytische und numerische Ableitung

- Gegeben sei eine Funktion  $f(x): \mathbb{R} \rightarrow \mathbb{R}: x \rightarrow y$ , z.B.
  - $f(x)_1: y = x+a$
  - $f(x)_2: y = x^2+a$
  - $f(x)_3: y = \sin(x)$
- Die Bestimmung der inversen Funktion kann häufig durch einfache algebraische Umformung generell und exakt berechnet werden:
  - $f^{-1}(y)_1: x = y-a$
  - $f^{-1}(y)_2: x = \{ \sqrt{y-a}, -\sqrt{y-a} \}$
  - $f^{-1}(y)_3: x = \arcsin(y)$

- Schon bei der zweiten Funktion gibt es mehr als eine Lösung, und die inverse Sinusfunktion kann nicht exakt analytisch berechnet werden sondern benötigt eine **Approximation** durch eine **geometrische Reihe**:

$$\arcsin(z) = \sum_{n=0}^{\infty} \frac{(2n-1)!! z^{2n+1}}{(2n)!! (2n+1)}$$



Inverse Probleme sind nicht trivial (zu lösen)!



Wie sieht es bei multivariaten Funktionen aus?

# Multivariate Funktionen

- Eine Funktion  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  stelle eine Informationskompression dar;
  - Aber i.A. als irreversible Reduktion (Informationsverlust)!
- Die Inversion einer Funktion  $f: \mathbb{R}^n \rightarrow \mathbb{R}$  ergibt eine große Menge an Lösungen da:
  - $f^{-1}(y): \mathbb{R} \rightarrow \mathbb{R}^n$  (Informationsexpansion bzw. Dekompression)
  - Die Lösungsmenge kann unendlich groß sein!
- Beispiel:  $f(x_1, x_2): y = x_1 + x_2$ 
  - (Unendlich viele) Lösungen für  $y=0$ :  $\mathbf{x} = \{ (0,0), (-1,1), (-2,2), (-3,3), \dots \}$

## Einschränkung des Eingabe- und Lösungsraums

1. **Intervallarithmetik** → D.h. eine Variable  $x$  wird nur in einem Intervall  $[a,b]$  betrachtet (und  $f$ )
2. **Diskretisierung** des Intervalls;  $[a,b] \rightarrow \{ a, a+\delta, a+2\delta, \dots, b \}$

## Randbedingungs lösen

- Wenn für alle Eingabevariablen und ebenso für die Zielvariable diskrete Werte in einem endlichen Bereich liegen könnten man das Inversionsproblem durch einen Randbedingungs löser (Constraint Solving Problem) lösen
  - Aber auch dieser Ansatz liefert entweder viele oder nur eine Auswahl an Lösungen

- Beispiel eines Erfüllbarkeitsproblems über relationale Ausdrücke:

**Problem**

$$f(x_1, x_2) = y = x_1 + x_2$$

$$x_1 = \{ 1, 2, 3 \}$$

$$x_2 = \{ 1, 2, 3 \}$$

$$y = \{ 2, 3, \dots, 6 \}$$

**Randbedingungen**

$$x_1 \geq 1 \wedge x_1 \leq 3$$

$$x_2 \geq 1 \wedge x_2 \leq 3$$

$$x_1 = 1 \vee x_1 = 2 \vee x_1 = 3$$

$$x_2 = 1 \vee x_2 = 2 \vee x_2 = 3$$

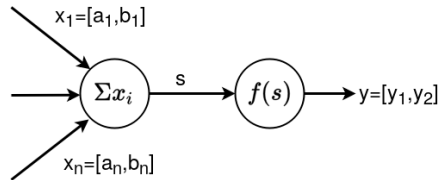
$$y = x_1 + x_2$$

# Web WorkShell Live

**CLEAR** **LOAD** **+** **-** **csp**



# Das Single Layer Perceptron



---

Abb. 12. Ein SLP (künstliches Neuron) besteht aus zwei verbundenen Funktionsblöcken (Summierer und Aktivierungsfunktion)

# Inverses Problem ML: Naiver Lösungsansatz

## Datentabelle

- Problem: Die Inversion der Datentabelle liefert einzelne Dateninstanzen
- Ziel: Das Modell  $M$  soll repräsentativ und generalisierbar sein
- Daher: Auch  $M^{-1}$  sollte möglich nur repräsentative Eingabevektoren liefern
  - Mittelwertbildung der Dateninstanzen und deren Variablen wenig hilfreich!
  - Zwei Instanzen:  $x=\text{Sonnig}$ ,  $x=\text{Regen} \Rightarrow x=(\text{Sonnig}+\text{Regen})/2=\text{wolkig}???$
  - Majoritäten könnten repräsentative Variablenwerte ergeben!?

## Inverses Problem ML: Entscheidungsbaum

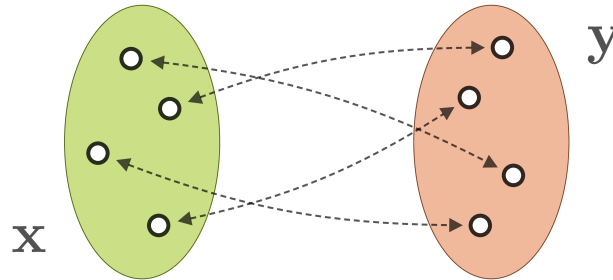
- Ein empirisch gelernter Entscheidungsbaum kann ein generalisiertes Modell sein  $M(x): x \rightarrow y$
- Die Invertierung geschieht durch Rückwärtsiteration startend bei allen Endknoten (Blättern) mit  $y_i = y$
- Es wird i.A. mehr als eine Lösung geben (Repräsentanz?)
- Die Frage ist die Ableitung der resultierenden Variable  $x$  aus den Knoten
  - Bei kategorischen Variablen triviales und eindeutiges Problem
  - Bei numerischen Variablen und einem relationalen Baum mit  $N(x) = \{ x < \epsilon, x \geq \epsilon \}$  ist gerade der Teilungswert  $\epsilon$  nicht repräsentativ (Rand!!)



# Invertierbare ANN

<https://hci.iwr.uni-heidelberg.de/vislearn/inverse-problems-invertible-neural-networks/>

Ausgangspunkt: Ein- und Ausgabedaten besitzen die gleiche Dimension!



## Invertierbare Netzwerkstruktur

- Es wird ein übergeordnetes reversibles Berechnungsnetzwerk eingeführt (Affine Kopplungsschicht)

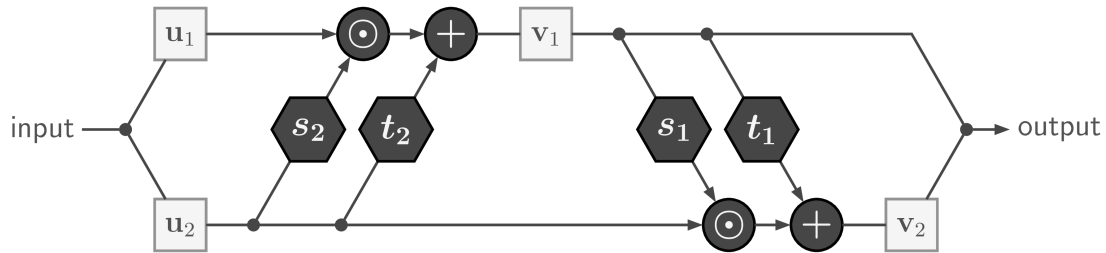


Abb. 14. Die Eingabedaten werden aufgespalten in  $[u_1, u_2]$  und durch die gelernten Funktionen  $s_i$  und  $t_i$  transformiert und in Wechsellage gekoppelt. Die Ausgabe ist die Verkettung der resultierenden Teile  $[v_1, v_2]$ .  $\odot$  - Elementweise Multiplikation

## Invertierung

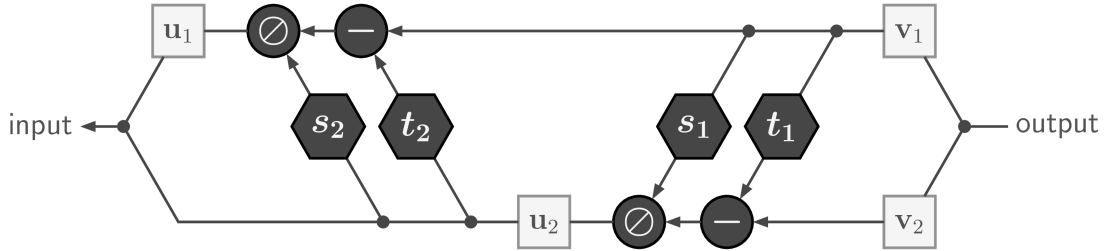


Abb. 15. Invertierung des Netzwerks. Mit einer Umschaltung können  $[u_1, u_2]$  aus  $[v_1, v_2]$  wiederhergestellt werden, um die Umkehrung der gesamten affinen Kopplungsschicht zu berechnen.  $\emptyset$  - Elementweise Division

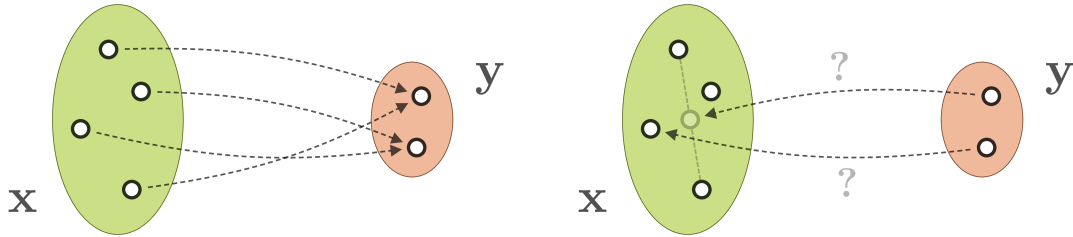
Entscheidend ist, dass die Transformationen  $s_i$  und  $t_i$  selbst **nicht invertierbar** sein müssen und durch beliebige neuronale Netze dargestellt werden können, die durch standardmäßige Backpropagation entlang des Berechnungsgraphen trainiert werden.

## Mehrdeutige Abbildungen

D.h. die Eingabedimension ist wie üblich viel größer als die Ausgabedimension!

- Die Inversion erzeugt Mehrdeutigkeit bei der Abbildung  $y \rightarrow x$ .





Dimensionsreduzierende Abbildung  $x \rightarrow y$  Mehrdeutigkeit zwischen  $y$  und  $x$

## Transformation in Bijektive Abbildungen

Eine zusätzliche latente Variable  $z$  wird eingeführt, die die Information erfasst, die sonst im Forward-Prozess verloren gehen würde. Folglich,  $x \leftrightarrow [y,z]$  wird eine bijektive Zuordnung.

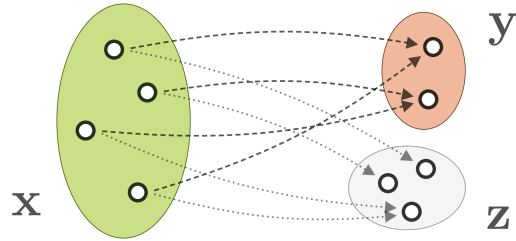


Abb. 16. Durch zusätzliche latente Variable  $z$  wird die inverse Abbildung "vervollständigt"

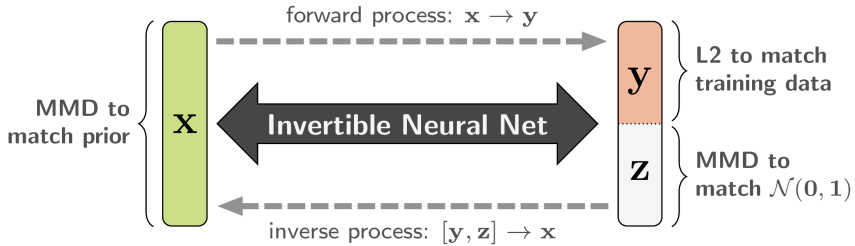


Abb. 17.  $z$  muss unabhängig von  $y$  sein und muss einer einfachen Stichprobenverteilung  $\mathcal{N}(0, 1)$  folgen.

- Beide Bedingungen können mit einem maximalen mittleren Diskrepanzverlust (MMD) erreicht werden, die mit zwei Verteilungen übereinstimmt durch Vergleich von Stichproben.

Die Verteilung  $p(x|y)$  kann angenähert werden, indem einfach wiederholt  $z$  abgetastet wird und die rückgerichtete Berechnung des Netzwerks durchgeführt wird, d.h.  $[y,z] \rightarrow x$ .

- Aus  $p(x|y)$  wird in eine deterministische Funktion  $x=f(y,z)$  mit der "verrauschten" Variable  $z$ .

# Zusammenfassung

- Das Training von Vorwärtsmodellen ist ein Standardverfahren
- Häufig - gerade in der Soziologie - ist man an Rückwärtsmodellen interessiert, d.h. die Invertierung der aus empirischen Daten algorithmisch gelernten Modelle
- Die Inversion ist schwierig durch Mehrdeutigkeit der Abbildung
- Variablenintervalle und Wertdiskretisierung können das Inversionsproblem auf Randbedingungs lösen reduzieren und lösbar machen
- Inversion von ANN benötigt eine übergeordnete bidirektionale und umschaltbare Netzwerkstruktur