# Neuronale Netze

Stand Oktober 2009



Fachhochschule Gelsenkirchen
Fachbereich Physikalische Technik
- Bärmann -

## Inhalt

# 1 Übersicht

## 1.1 Biologische Grundlagen

***„Neuronale Netze - Informationsverarbeitung nach dem Vorbild der Natur“*** ist die Motivation sich mit **künstlichen neuronalen Netzen** (NN) zu beschäftigen. Die Natur ist zu **spektakulären Leistungen** im Bereich der Informationsverarbeitung, Muster- und Bilderkennung und Bewegungskoordination in der Lage. Und damit ist noch lange nicht die menschliche Denkfähigkeit gemeint. Selbst die einfache Stubenfliege, die kollisionsfrei durch das Zimmer fliegt und mit den Füßen voran an der Decke landet, ist in dieser Hinsicht allen derzeit bestehenden technischen Systemen überlegen. Und dabei verfügt sie im wahrsten Sinne des Wortes nicht einmal über ein Spatzenhirn. Bevor wir also an die technische Implementation von Systemen der künstlichen Intelligenz gehen, sollten wir uns die Frage stellen: ***„Wie macht es die Natur?“*** – gemeint ist natürlich die Informationsverarbeitung.

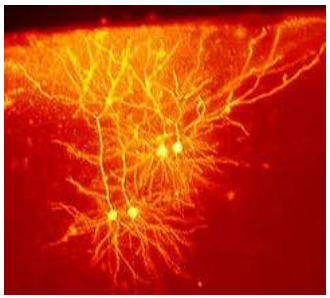


**Abb.1: Neuronales Netz mit mehreren Neuronen**

Untersucht man ein **biologisches neuronales Netz** (z.B. Fliegen- oder Mäusehirn) stellt man die folgenden Eigenschaften fest:

- Neuronale Netze bestehen aus **vielen Neuronen**, die untereinander stark vernetzt sind.
- Die Prozessoren (Neuronen) **arbeiten simultan** in einer massiv parallelen Architektur.
- Die Information steckt in der **Verbindungsstruktur**.
- Neuronale Netze sind **lernfähig**.

Die **Informationsverarbeitung** wird also von den unzähligen **Neuronen (Nervenzellen)** geleistet - beim Menschen ca. 10 Milliarden -, die in geeigneter Weise miteinander





verschaltet sind. Das einzelne Neuron gliedert sich grob in die **Dendriten** (das Dendritengeflecht), den **Zellkörper** und das **Axon** (die Nervenfaser).

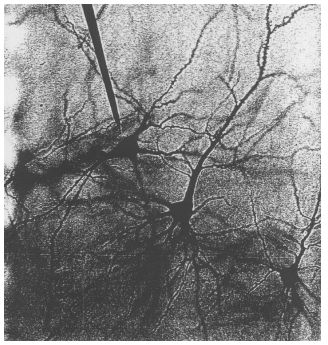


**Abb. 2:** 3 Neuronen, Dendritengeflecht (unten), Zellkörper (Verdickung) und Axon (Nervenfaser im oberen Bildteil) erkennbar.

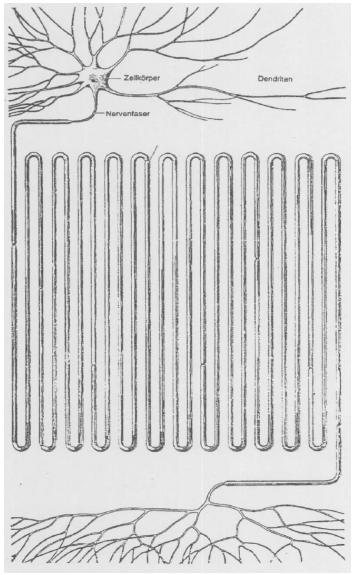



**Abb. 3:** Nervenzelle eines Wirbeltiers in 250-facher Vergrößerung (schematisch)

Über das Dendritengeflecht laufen in ein Neuron Signale der vorgeschalteten Neuronen ein. Diese werden im Zellkörper geeignet verrechnet und über das Axon wird das Ergebnis an nachgeschaltete Neuronen weitergegeben. Die Ankoppelung der vorgeschalteten Neuronen wird dabei über die so genannten **Synapsen** (Kontaktstellen) vermittelt. Davon gibt es prinzipiell 2 Typen, **erregende und hemmende Synapsen**.

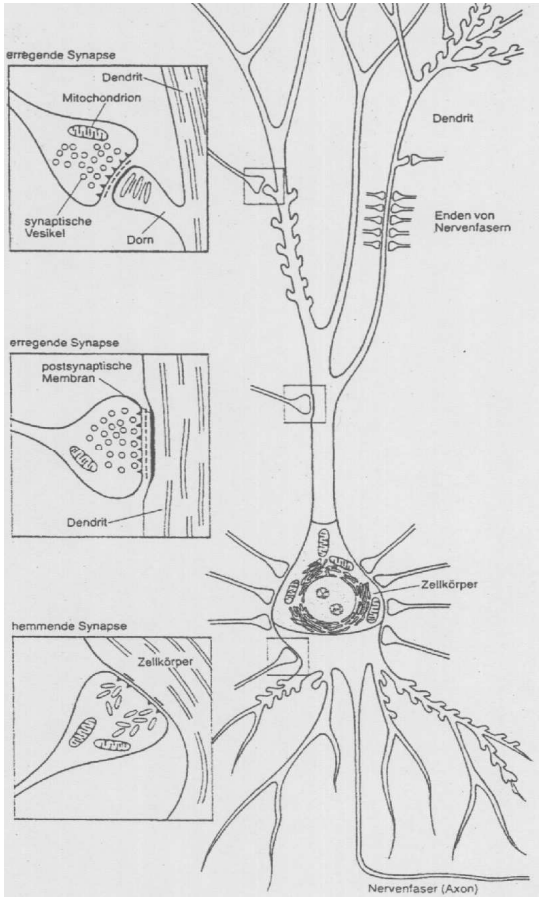



**Abb. 4:** Neuron mit Synapsen (schematisch)





Impulse, die ein Neuron über erregende Synapsen erreichen, motivieren es ebenfalls einen Impuls an die nachgeschalteten Neuronen zu übermitteln. Impulse über hemmende Synapsen unterdrücken diese Aktion. Darüber hinaus geht noch die **Kontaktstärke der Synapse** in die Berechnung ein. Das einfachste Modell der Informationsverarbeitung in biologischen neuronalen Netzen besteht in der folgenden **Modellvorstellung**:

- In ein Neuron laufen über die Synapsen Impulse der vorgeschalteten Neuronen ein. Die Impulsstärken werden mit der **Synapsenkopplungsstärke** (dem **synaptischen Gewicht**) multipliziert (gewichtet). Hemmenden Synapsen entspricht eine negative Kopplungsstärke.
- Sämtliche mit den Kopplungsstärken gewichteten Impulsstärken werden aufsummiert. Dies liefert den **Gesamtinput** in ein Neuron. Diese Verarbeitung entspricht der Bildung des **Skalarproduktes** des Vektors aller eingehenden Impulse mit dem Vektor der synaptischen Kopplungsstärken.
- Falls der Gesamtinput einen bestimmten **Schwellwert** überschreitet, dann liefert das Neuron selbst einen Impuls, der sich entlang des Axons ausbreitet und auf die Synapsen der nachgeschalteten Neuronen trifft. Das Neuron „feuert“.
- Ein Neuron erhält seinen Input typischerweise von 1000 bis 10000 vorgeschalteten Neuronen (oder Sinnesnervenzellen) und gibt sein Signal auch an entsprechend viele nachgeschaltete Neuronen weiter.
- Damit die Informationsverarbeitung zu sinnvollen Ergebnissen führt, ist also die **korrekte Verschaltung und Stärke der synaptischen Kontakte** entscheidend. Für die Einstellung der Verschaltung gibt es zwei Methoden:
  1. Die Verschaltung (insbesondere die synaptische Kopplung) kann durch **Lernvorgänge** an die Aufgabenstellung angepasst werden.
  2. Das Lebewesen kommt bereits **fest verdrahtet** zur Welt, d.h. die Verschaltung ist schon genetisch vorgegeben. Gewissermaßen hat dann die Evolution den Lernvorgang übernommen.
- Das einzelne Neuron hat dabei eine **„Taktfrequenz“** von ca. 1 kHz, d.h. es ist mindestens den Faktor $10^6$ langsamer als ein Intel-Prozessor. Dafür macht es aber auch in jedem Takt nicht nur eine Addition oder Multiplikation sondern ein langes Skalarprodukt von 1000 bis 10000 Werten. Damit ist es dann nur noch einen Faktor in der Größenordnung 1000 langsamer als ein Intelprozessor. Wenn man dann bedenkt, dass der Mensch über ca. $10^{10}$ Neuronen verfügt, ergibt sich eine Gesamtverarbeitungsleistung, die immer noch ca. sieben Zehnerpotenzen höher ist als beim Intelprozessor.

**Modellkritik:** Das Modell ist stark vereinfacht. Insbesondere gibt es nicht nur die Optionen „Impuls“ oder „kein Impuls“, sondern es findet noch eine Phasen- und Frequenzmodulation

statt. Darüber hinaus gibt es eine ganze Reihe unterschiedlicher Neuronentypen, die sich auch unterschiedlich verhalten. Dennoch liefert dieses Modell die wesentliche Handlungsanweisung zur Konstruktion künstlicher neuronaler Netze.

## 1.2 Künstliche Neuronale Netze

**Künstliche neuronale Netze** (KNN) sind spezielle **mathematische Modelle**, die sich an den Prinzipien der biologischen Informationsverarbeitung orientieren und die in besonderer Weise für Aufgaben des Anforderungstyps **"Mustererkennung"** geeignet sind. Dieser Begriff ist ganz allgemein zu verstehen, umfasst aber insbesondere das Erkennen von funktionalen Zusammenhängen zwischen physikalischen, chemischen, technischen, wirtschaftlichen etc. Größen. Wir werden im Folgenden noch sehen, dass sich sehr viele Fragestellungen der künstlichen Intelligenz auf die **Konstruktion geeigneter Funktionen** zurückführen lässt:
**Beispiele:**

- Bei der **automatischen Gesichtserkennung** muss einem Vektor von Bildinformationen eine Wahrscheinlichkeit zugeordnet werden, dass die gerade von der Kamera betrachtete Person mit dem Herrn Müller identisch ist.
- Bei einer **Autopilotanwendung** im Flugzeug muss dem Vektor der Sensor- und Kursinformationen der so genannte Stellgrößenvektor zugeordnet werden, d.h. der Vektor der die Stellung von Seiten-, Höhen-, Querruder, Schub der einzelnen Triebwerke, Klappenpositonen etc. angibt.

In beiden Fällen handelt es sich mathematisch gesprochen um eine **Funktion** des $R^m$ in den $R^n$, d.h. eine Funktion, die m-dimensionale Vektoren auf n-dimensionale Vektoren abbildet. Nun kann man versuchen, die für die aktuelle Aufgabe benötigte Funktion auf der Basis von physikalischen oder betriebswirtschaftlichen Kenntnissen zu konstruieren. Im Zusammenhang mit der künstlichen Intelligenz wird jedoch i. A. versucht, die benötigte Funktion anhand von **Messdaten (Beobachtungsdaten)** mittels eines **maschinellen Lernverfahrens** aus den Daten zu erzeugen. An dieser Stelle kommen jetzt die neuronalen Netze mit ihrer Lernfähigkeit ins Spiel, d.h. man versucht Prinzipien des Lernens von biologischen Systemen in die Technik zu übertragen bzw. als **Softwaresimulation** für technische Anwendungen nutzbar zu machen. Für die wichtigste Klasse künstlicher neuronaler Netze, die so genannten Feedforward-Netze (Kapitel 3.4) lassen sich wichtige theoretische Aussagen darüber treffen, dass diese Methoden des maschinellen Lernens in einem gewissen Sinne sogar optimal sind.
Die folgende Abbildung verdeutlicht die prinzipielle Vorgehensweise beim Einsatz von



Fachhochschule Gelsenkirchen
Fachbereich Physikalische Technik
- Bärmann -

**Methoden des maschinellen Lernens**:

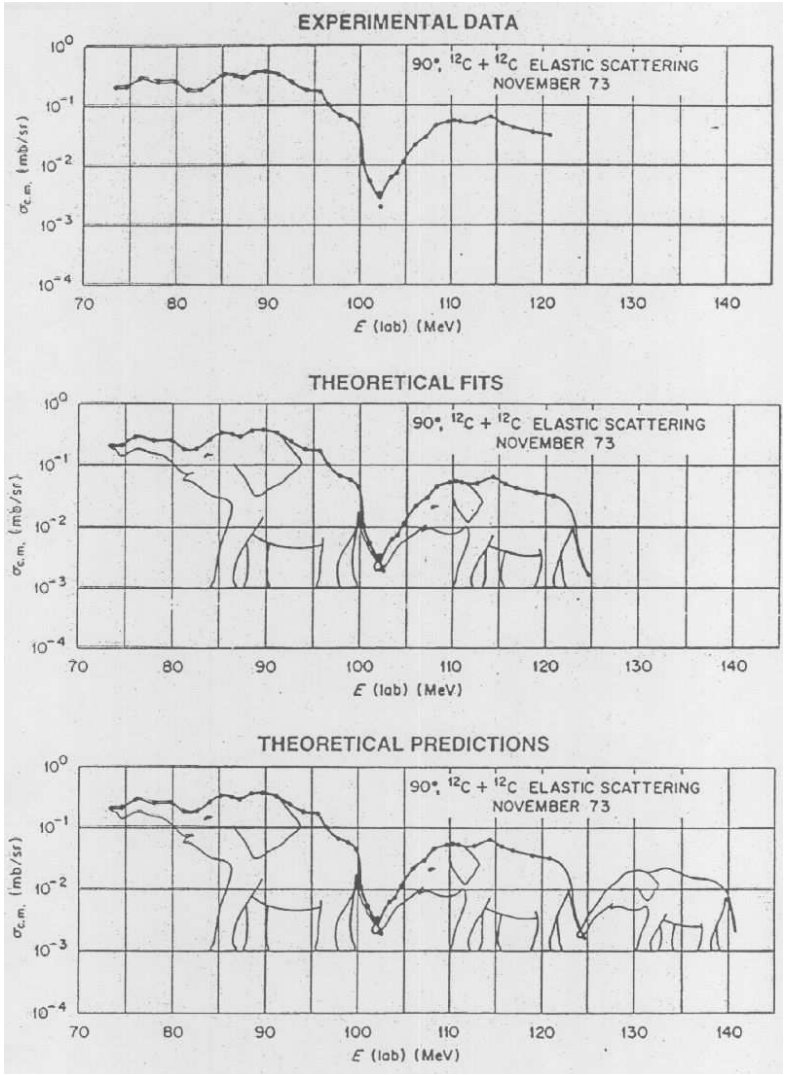



**Abb. 5**: Prinzip des maschinellen Lernens

1. Für die interessierende Fragestellung werden **Messdaten** („Experimental Data“) bereitgestellt.
2. Auf der Basis der Messdaten wird versucht, ein **Modell** für den durch die Daten repräsentierten physikalischen Zusammenhang zu erstellen. Man sagt auch, ein Modell wird an die Daten angepasst **(„gefittet“).**
3. Falls ein hinreichend genaues Modell erreicht werden konnte, wird dieses Modell für **Vorhersagen („Predictions“)** in neuen Anwendungsbereichen genutzt.

Die typischen Anwendungsbereiche sind dabei:

- **Bildverarbeitung**
- **Sprach- und Texterkennung**
- **Robotersteuerung**
- **Datenanalyse**, z.B. Aktienkurse
- **Prozesssteuerung**

**Beispiel:**


**Abb. 6:** Aktienkursprognose





**Beispiel: System zur Automatischen Sortierung von Abfallkomponenten**

**Ziel:** Erkennung von bestimmten Verpackungsobjekten (Joghurtbechern, Plastikflaschen etc.)

**Vorgehensweise:**

- Objekte werden von Kamera erfasst.
- Dem Bild werden bestimmte Merkmale entnommen
- Auf der Basis dieser Merkmale nimmt NN eine Klassifikation vor.

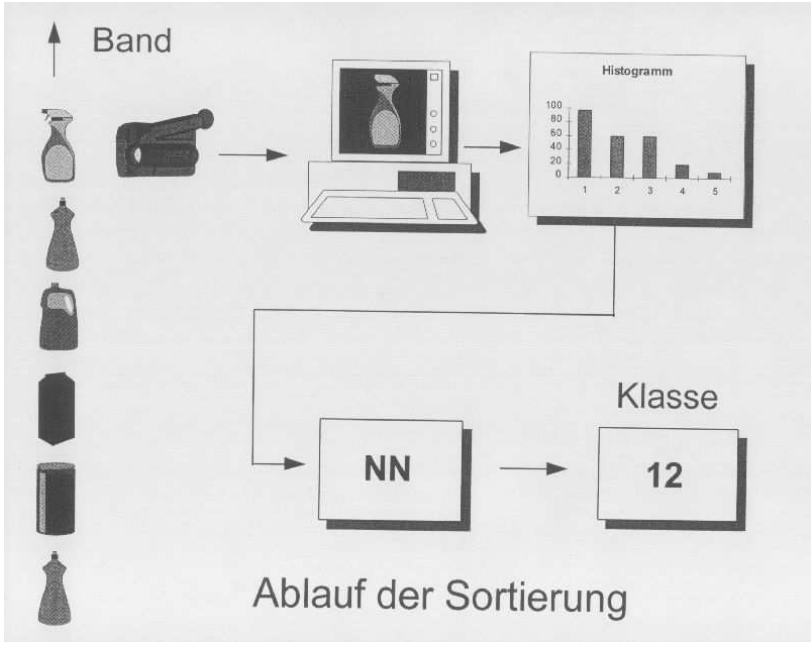



**Abb. 7:** Automatische Objekterkennung

# 2 Grundlagen

## 2.1 Entwicklung der Neuroinformatik - Geschichte der neuronalen Netze

**Frühe Anfänge (1942 - 1955)**

1943 W.Culloch, W.Pitts: „A logical calculus of ideas immanent in nervous activity“

1949 D.O. Hebb: „The Organisation of Behavoiur“
Klasssische Hebb’sche Lernregel

**Erste Blütezeit (1955 - 1969)**

1957 Mark I Perceptron wurde von Rosenblatt, Wigthman und Mitarbeitern am MIT entwickelt.
Perceptron wurde für Mustererkennungsprobleme eingesetzt und konnte einfache Ziffern erkennen.

1959 Frank Rosenblatt: „Principles of Neurodynamics“

1960 B. Widrow, M.E. Hoff: „Adaptive switching circuits“
Aufsatz enthielt Beschreibung des neuronalen Netzes Adaline.

1965 N. Nilson: „Learning Machines“
Überblicksartikel über die Arbeiten dieser Periode.

**Die stillen Jahre (1969 – 1982)**

1969 M.Minsky , S. Papert: „Perceptrons“
Mathematische Analyse des Perceptrons zeigte, dass dieses Modell wichtige Probleme gar nicht repräsentieren kann. ( Standardbeispiel : XOR - Funktion)

Trotzdem wurden in den nächsten 15 Jahren wichtige theoretische Grundlagen für eine spätere Renaissance der neuronalen Netze gelegt:





**Wichtige Aufsätze :**

1972 T. Kohonen: „Correlation matrix memories“
Modell eines linearen Assioziators

1974 P.Werbos: Dissertation über Backpropagation - Verfahren

1982 J. Hopfield: „Neuronal Networks and physical Systems“
Anwendung der neuronalen Netze in der Physik

1982 T. Kohonen: „Self-organized formation of topologically correct features“
Selbstorganisierende Karten

**Die Renaissance der neuronalen Netze (1985 – heute)**

Aufschwung nahmen die neuronalen Netze, da gezeigt werden konnte, dass die Beschränkungen der einstufigen Netze, wie des Perceptron, für mehrstufige, vorwärtsgerichtete Netze nicht galten.

1985 J. Hopfield: „Neural Computation of Decisions in Optimization Problems“
Hopfield - Netze können schwierige Optimierungsaufgaben, wie das Traveling- Salesman - Problem lösen.

1986 Rumelhardt, Hilton und Williams: „Learning internal representations by error propagation“
Ausführliche Darstellung der Backpropagation - Methode, ein robustes Lernverfahren für vorwärtsgerichtete Netze

Seit 1986 hat sich das Gebiet nahezu explosionsartig entwickelt, die Aufsätze und Entwicklungen würden ein ganzes Buch füllen.

## 2.2 Methoden der künstlichen Intelligenz

Neuronale Netze stellen nur eine Methode aus einer Vielzahl von Strategien zur Lösung von Problemen der künstlichen Intelligenz (KI) dar. Darüber hinaus gibt es auch starke Bezüge zu Methoden der **klassischen Statistik**. Im Folgenden werden die wichtigsten KI-Verfahren kurz vorgestellt.

**A. Expertensysteme (klassische XPSe)**

Auf der Basis von Expertenbefragungen wird das vorliegende Wissen („Expertenwissen“) geeignet formalisiert („wenn – dann – Regeln“) und als Wissensbasis („knowledge-base“) gespeichert. Ein Algorithmus arbeitet die Wissensbasis ab um für eine konkrete Fragestellung eine Lösung zu finden.

**Vorteile:** Problemdomäne wird während der Wissenserhebung gründlich untersucht.

**Nachteile:** Extrem hoher Aufwand, Expertenwissen nur in wenigen Problemdomänen ausreichend vorhanden.

**Anwendungen:** Konfigurations- und Diagnosesysteme, medizinische Expertensysteme

**B. Induktive Systeme**

Expertensysteme bei denen die Regeln automatisch aus Messdaten erzeugt werden (ähnlich zu NNen). Bekanntester Vertreter ist der ID3-Algorithmus.

**Vorteile:** Aufwand gegenüber klassischen XPSen stark reduziert.

**Nachteile:** Weniger leistungsstark im Vergleich zu NNen bei ähnlichen Voraussetzungen.

**C. Fuzzy-Systeme**

XPSe auf der Basis „unscharfer“ Regeln („Fuzzy-Regeln“). Messwerte werden auf Zugehörigkeitsfunktionen abgebildet:





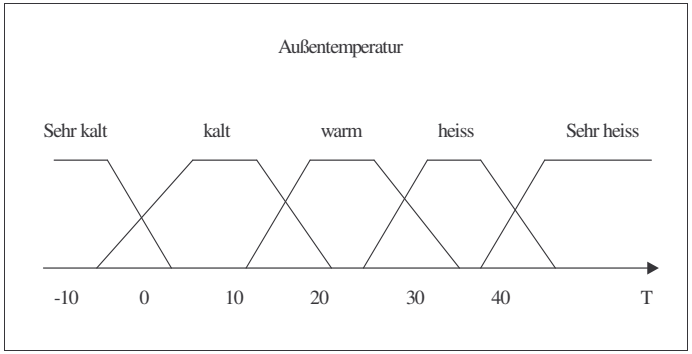



**Abb. 8:** Fuzzy-Zugehörigkeitsfunktionen für die Fuzzyfizierung der Außentemperatur

Für die Zugehörigkeitsfunktionen werden dann Regeln aufgestellt:
*„Wenn Außentemperatur sehr kalt, dann Heizung sehr hoch“*

**Vorteile:** Liegt der menschlichen Denkweise näher als klassische Expertensysteme.

**Nachteile:** Immer noch sehr aufwendig.

**Anwendungen:** Regelungstechnik, teilweise in entsprechenden Automatisierungs-systemen integriert (Siemens)

**D. Neuronale Netze**

Siehe Kapitel 3

**E. Neuro-Fuzzy-Systeme**

Kombination von Lernalgorithmen mit Fuzzy-Systemen. Auch als lernfähige Fuzzy-Systeme bezeichnet

**F. Weitere Ansätze**

- Genetische Algorithmen / Evolutionsstrategien
- Intelligente Agenten
- Artificial Life
- ... und unzählige weitere Methoden

## 2.3 Zielsetzungen der angewandten Neuroinformatik

### 2.3.1 Funktionsapproximation

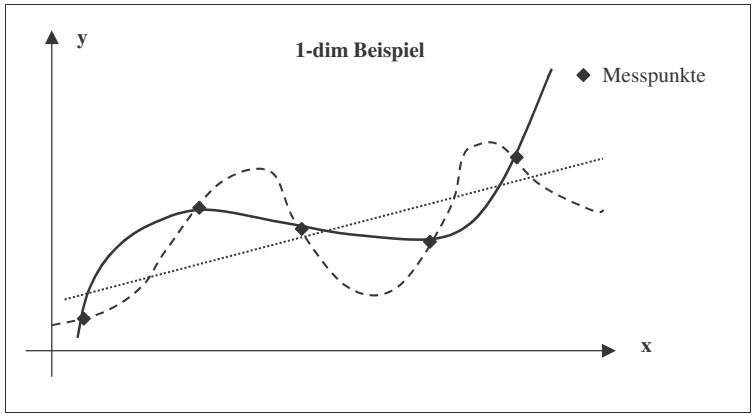



**Abb. 9:** Verschiedene Kurven durch 5 Messpunkte

**Aufgabe:** Finde **„optimale“ Kurve** durch die Messpunkte.

**Praxis:**
p Messpunkte $\underline{X}^i$, $\underline{Y}^i$ mit $\underline{X} \in R^m$, $\underline{Y} \in R^n$ , $i = 1...p$

d.h. finde n Funktionen mit

$$y_1 = f_1(x_1,...,x_m)$$
$$y_2 = f_2(x_1,...,x_m)$$
$$.$$
$$.$$
$$y_n = f_n(x_1,...,x_m)$$

die möglichst gut durch die Messpunkte gehen.

**Beispiele:**

- Aktienkursverläufe
- Modellierung technischer Anlagen
- Rezeptur- Eigenschaftsbeziehungen in den Materialwissenschaften
- Robotersteuerung





### 2.3.2 Klassifikation

**Aufgabe:** Ordne **Merkmalsvektor** $\underline{X}$ einer Klasse eines **Klassifikators** zu.

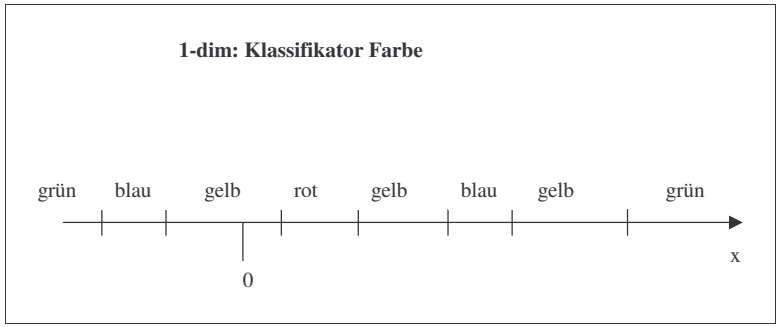



**Abb. 10:** Verschiedene Klasseninstanzen entlang eines 1-dim Parameters x

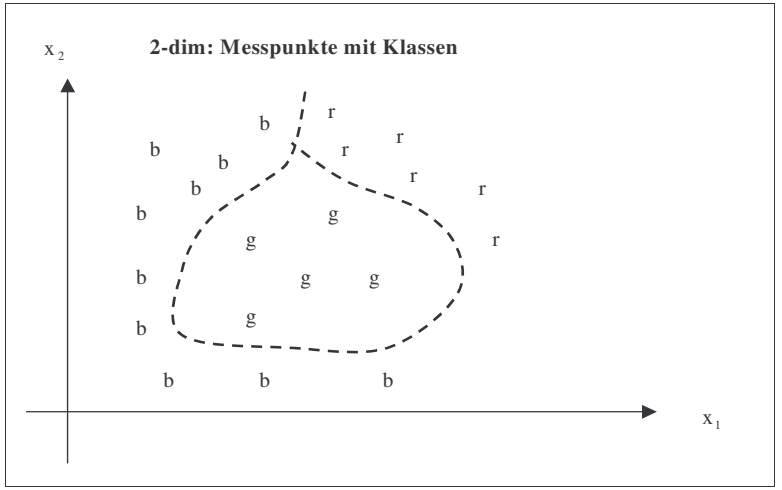



**Abb. 11:** Verschiedene Klasseninstanzen mit möglichem Verlauf der Klassengrenzen

**d.h. Aufgabe äquivalent zur Modellierung der Klassengrenzen.**
Aufgabe wird durch geeignete Kodierung auf Funktionsapproximation zurückgeführt (**Bayessche Kodierung**).

**Kodierungstabelle:**

| x1 | x2 | Farbe | | x1 | x2 | b | g | r |
|---|---|---|---|---|---|---|---|---|
| 0,1 | 0,5 | b | | 0,1 | 0,5 | 1 | 0 | 0 |
| 0,3 | 0,3 | g | ---> | 0,3 | 0,3 | 0 | 1 | 0 |
| 0,6 | 0,5 | r | | 0,6 | 0,5 | 0 | 0 | 1 |

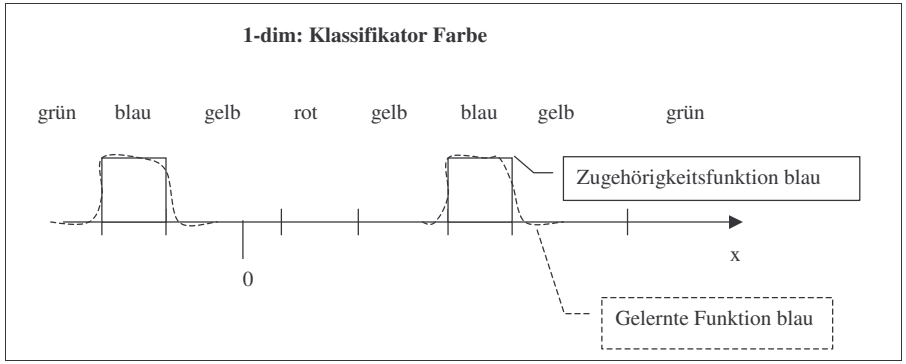



**Abb. 12:** Modellierung der Wahrscheinlichkeitsdichtefunktion

Die gelernte Funktion **(„Wahrscheinlichkeitsdichtefunktion“)** kann unmittelbar als Fuzzy-Zugehörigkeitsfunktion interpretiert werden. System schafft weiche Übergänge zwischen den Klassen.

**Beispiele:**

- Gesichtserkennung
- OCR („Optical Character Recognition“)
- Spracherkennung
- Erkennung unzulässiger Betriebszustände bei technischen Systemen (Airbag, Zündaussetzer...)

**Allgemein: Mustererkennungsaufgaben**





### 2.3.3 Weitere Anwendungen

- **Mustervervollständigung:** Ergänze unvollständiges oder verrauschtes (gestörtes) Muster zu Originalmuster. Vor allem für Bilder. „Hopfield-Netze“

- **Optimierung:** Allgemeines Optimierungsproblem wird durch rückgekoppeltes Netz iterativ gelöst. „Boltzmann-Netze“

- **Data Reconciliation**: Bestimme zu einem Messdatenvektor $\underline{X}$ einen konsistenten Messdatenvektor $\underline{\tilde{X}}$ , so dass $\underline{\tilde{X}}$ den bestehenden physikalischen Beziehungen genügt. „Autoassoziative Netze“

- **Topologische Karten**: Extrahierung wesentlicher Merkmale aus Datenvektoren. „Kohonen-Netze“

**Anmerkungen:**

1. Viele der genannten Anwendungen lassen sich auch ohne neuronale Netze und zum Teil auch deutlich effizienter mit Methoden der Statistik oder der numerischen Mathematik lösen. Der Versuch jedes Problem mittels eines NN zu bearbeiten, erinnert an den Handwerker, der als einziges Werkzeug nur den Hammer kennt. Dann sieht natürlich auch jedes Problem wie ein Nagel aus.

2. Rückgekoppelte Netze, bei denen die Information nicht nur in eine Richtung fließt, sind numerisch außerordentlich aufwendig, d.h. man muss mit extrem langen Rechenzeiten (Stunden bis Wochen) rechnen.

# 3 Netztypen und Lernverfahren

## 3.1 Lineare Regression

Die lineare Regression ist **das Standardverfahren der Statistik** um aus Messdaten auf den zugrunde liegenden funktionalen Zusammenhang zu schließen. Es handelt sich um ein bestens etabliertes Verfahren um Funktionen zu modellieren. Im Gegensatz zu einem weit verbreiteten Missverständnis lassen sich mit der linearen Regression (LR) keineswegs nur lineare Zusammenhänge (lineare Funktionen) modellieren. Im Gegenteil kann damit jede beliebige stetige Funktion (zumindest prinzipiell) modelliert werden. Jedes maschinelle Lernverfahren muss sich an der linearen Regression messen lassen. In diesem Sinne stellt das Verfahren die **Referenz („den Goldstandard“) für Lernverfahren** dar.

Zunächst diskutieren wir nur den **eindimensionalen Fall**, d.h. es liegt nur eine unabhängige Variable x (oder t) vor:

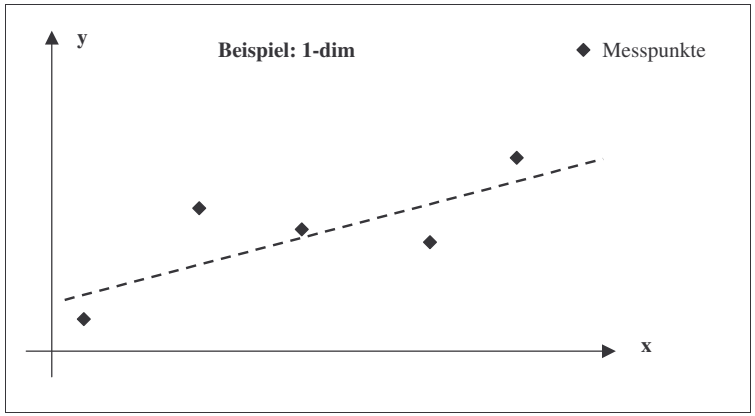



**Abb. 13:** Ausgleichsgerade $y = a_0 + a_1 x$

Im Beispiel ist die modellierte Funktion als Geradengleichung selbst eine lineare Funktion. Das muss aber keineswegs so sein, denn auch die folgenden 1-dimensionalen Beispiele sind lineare Regressionen:





- **Polynom:** $y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$
  Dabei ist x die unabhängige Variable und $a_0 ... a_4$ sind die gesuchten Regressionskoeffizienten. Die Koeffizienten sind anhand der Messdaten zu bestimmen.

- **Fourierreihe:** $y = a_0 + \sum_{k=1}^{n} \left( a_k \cos(k\,\omega t) + b_k \sin(k\,\omega t) \right)$
  mit der unabhängigen Variablen t und einer festen Grundfrequenz $\omega$. Die a- und b-Koeffizienten sind aus den Daten zu bestimmen.

- **Allgemeiner LR-Ansatz:** $y = a_1 \phi_1(x) + a_2 \phi_2(x) + .... + a_n \phi_n(x)$
  mit der unabhängigen Variablen x, den LR-Koeffizienten $a_i$ und beliebigen (auch nichtlinearen) Ansatzfunktionen $\phi_i$.

Dagegen ist das folgende Beispiel **keine LR:**

- **Nichtlineare Regression**: $y = a_0 + a_1 e^{b_1 x} + a_2 e^{b_2 x} + a_3 e^{b_3 x}$
  da in diesem Beispiel zumindest ein Teil der gesuchten Koeffizienten (nämlich die b-Koeffizienten) unter der nichtlinearen e-Funktion steht.

Charakteristisch für eine lineare Regression ist also, dass die gesuchten Regressionskoeffizienten nur in der ersten Potenz und nicht unter nichtlinearen Funktionen vorkommen. Auch Produkte der Koeffizienten sind nicht erlaubt.
**Lineare Regression: Der Ansatz muss linear in den gesuchten Koeffizienten sein!**

Aus einem LR-Ansatz lässt sich ein (im Allgemeinen überbestimmtes) **lineares Gleichungssystem zur Bestimmung der Regressionskoeffizienten** herleiten. Für jede der Ansatzfunktionen erhält man eine Spalte einer Matrix mit den entsprechenden Messwerten aus den p-vielen Datenpunkten $(x_i, y_i)$. Im Polynombeispiel ergibt sich so (1. Zeile nur als Spaltenbeschriftung):

$$\begin{array}{ccccc} 1 & x & x^2 & x^3 & x^4 \end{array} \quad \text{(nur Spaltenbeschriftung)}$$

$$\begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ 1 & & & & \\ 1 & & & & \\ . & & & & \\ . & & & & \\ 1 & x_p & x_p^2 & x_p^3 & x_p^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ . \\ y_p \end{pmatrix}$$

**Abb. 14:** (Überbestimmtes) lineares Gleichungssystem zur Bestimmung der Koeffizienten

Wie man sieht, erhält man von jedem Messpunkt eine Gleichung. Falls in diesem Beispiel p größer als 5 ist (d.h. dass man mehr als 5 Messpunkte hat (Standardfall)), ist das Gleichungssystem **überbestimmt**, d.h. es hat **mehr Gleichungen als Variablen** und damit i. A. keine Lösung.
**Beispiel:** Bei 5 Messpunkten und nur der Geradengleichung $y = a_0 + a_1 x$ als Ansatz, geht die Gerade eben i. A. nicht durch alle fünf Punkte (vgl. Abb. 13).

Das oben angegebene überbestimmte Gleichungssystem lässt sich jedoch mit Methoden der linearen Algebra auf ein exakt bestimmtes lineares Gleichungssystem transformieren. Dieses System, das so genannte **„lineare Ausgleichsproblem“**, hat dann genau so viele Gleichungen, wie es gesuchte Koeffizienten gibt. In ganz analoger Weise funktioniert diese Strategie auch bei **linearen Regressionsverfahren im Mehrdimensionalen**, d.h. bei mehreren unabhängigen Variablen. Diese Vorgehensweise soll dennoch hier aus den folgenden Gründen nicht weiter verfolgt werden:

1. Das Gleichungssystem ist in der Regel **schlecht konditioniert**, d.h. numerisch heikel (insbesondere bei einer großen Zahl von Regressionskoeffizienten). Es muss mit enormen Rundungsfehlereffekten gerechnet werden. Aus diesem Grund scheiden Standardverfahren, wie das Gauß-Eliminationsverfahren, aus.
2. Diese Vorgehensweise funktioniert nur bei linearen Regressionsverfahren. Die noch zu behandelnden neuronalen Netze sind vom Typ her jedoch nichtlineare Regressionsverfahren, für die ohnehin eine andere Strategie benötigt würde.

Wir benötigen also eine alternative Strategie zur Berechnung der Regressionskoeffizienten, die nicht unbedingt davon Gebrauch macht, dass der Ansatz linear in den gesuchten Koeffizienten ist. Dies führt auf das Gradientenverfahren. Zuvor soll jedoch der allgemeine n-dimensionale Fall der linearen Regression definiert werden:





**n-dimensionale lineare Regression:**
Gesucht ist eine Funktionen y von n unabhängigen Variablen $x_1,...,x_n$, d.h. $y = (x_1,...,x_n)$,
die möglichst gut zu den Messwerten $(y_i, \underline{X}_i)$ für $i = 1,...,p$ passt.

**LR-Ansatz:**

$$y = a_0 + a_1 x_1 + ... + a_n x_n$$
$$\left(+ a_{11} x_1^2 + a_{12} x_1 x_2 + ... + a_{nn} x_n^2 + ...\right)$$

die Terme in der Klammer stellen **nichtlineare Ansatzfunktionen im Mehrdimensionalen** dar, aber **alle Terme sind linear in den gesuchten Koeffizienten** $a_0, a_1, .., a_{ij}, ...$

**Berechnung der Koeffizienten:**
Die Vorgehensweise wird hier für den Fall von Koeffizienten $a_0$ bis $a_n$ diskutiert. Bei Vorliegen weiterer Koeffizienten ist die Vorgehensweise völlig analog. Es bezeichne $(y_i, \underline{X}_i)$ den **i-ten Datensatz (das i-te Muster, die i-te Messung)** und $x_{ij}$ die j-te Komponente des Vektors $\underline{X}_i$. Wir setzen voraus, dass die gesuchten Koeffizienten $a_0,...,a_n$ mit irgendwelchen (Anfangs-) Werten belegt sind und definieren damit den durch das Modell vorhergesagten Wert:

$$y_i^{pr} = a_0 + \sum_{j=1}^{n} a_j x_{ij} \qquad \textbf{vorhergesagter Wert} \text{ (pr = predicted)}$$

Für irgendwelche Werte der Koeffizienten wird $y_i^{pr}$ wohl kaum mit dem zu $\underline{X}_i$ gehörenden Messwert $y_i$ übereinstimmen. Daher definieren wir als Fehler des i-ten Musters:

$$e_i = y_i - y_i^{pr} \qquad \textbf{Fehler des i-ten Musters}$$

**Gesamtfehlerfunktion** für alle p-vielen Muster:

$$Z(a_0, a_1, ..., a_n) = \sum_{i=1}^{p} e_i^2$$

Häufig wird auch der **mittlere quadratische Gesamtfehler** verwendet:

$$Z^M(a_0, a_1, ..., a_n) = \frac{1}{p} \sum_{i=1}^{p} e_i^2$$

der sich von Z aber nur durch einen konstanten Faktor unterscheidet.

Die Bestimmung der gesuchten Koeffizienten ist also damit äquivalent zu der **Minimierungsaufgabe:**

Minimiere $Z(a_0, a_1, ..., a_n)$ oder $Z^M(a_0, a_1, ..., a_n)$ !

Hierfür wird im nächsten Abschnitt ein Verfahren (Gradientenverfahren) vorgestellt.

**Bemerkung:**
Falls der Ansatz y linear in den gesuchten Koeffizienten ist, kann aus der Forderung „Minimiere Z" wieder ein Lineares Gleichungssystem hergeleitet werden, nämlich das bereits erwähnte „Lineare Ausgleichsproblem".

**Zusammenfassung lineare Regression**
Im Zusammenhang mit der Modellierung von Funktionen mehrerer Veränderlicher sind die **Standardansätze der linearen Regression**:

- **Linearer Ansatz**
  $y = a_0 + a_1x_1 + ... + a_nx_n$

- **Wechselwirkungsansatz (WW)**
  $y = a_0 + a_1x_1 + ... + a_nx_n + a_{12}x_1x_2 + a_{13}x_1x_3 + ... + a_{1n}x_1x_n + a_{23}x_2x_3 + ... + a_{n-1n}x_{n-1}x_n$
  neben den linearen Termen tauchen die Kombinationen der Parameter **(„Wechselwirkungen")** auf. Dieser Ansatz wird sehr häufig im Zusammenhang mit **statistischer Versuchsplanung** eingesetzt. Dabei ist n aber in der Regel deutlich kleiner als 10 (typischerweise 2-6).

- **Quadratischer Ansatz** (vollständiger quadratischer Ansatz, **Ansatz zweiter Ordnung**)
  $y = a_0 + a_1x_1 + ... + a_nx_n + a_{11}x_1^2 + a_{12}x_1x_2 + a_{13}x_1x_3 + ... + a_{n-1n}x_{n-1}x_n + a_{nn}x_n^2$
  enthält also alle Terme des WW-ansatzes zzgl. der quadratischen Terme $a_{kk}x_k^2$.

- **Höhere Ansätze (z.B. kubischer Ansatz)**
  Theoretisch können jetzt beliebige weitere Potenzterme hinzugenommen werden. Die Terme dritter Ordnung (wie z.B. $x_1^3, x_2^2x_3, x_1x_2x_3$) werden als **kubische Terme** bezeichnet. Durch Hinzunahme von Termen hinreichend hoher Ordnung ist es theoretisch möglich **jede beliebige stetige Funktion zu approximieren** (zu modellieren). In der Praxis scheitert die Verwendung von Ansätzen höherer als zweiter Ordnung jedoch an der **kombinatorischen Explosion der Terme**.





| n | Linear | WW | Quad. | Kubisch |
|---|---|---|---|---|
| 1 | 2 | 2 | 3 | 4 |
| 2 | 3 | 4 | 6 | 10 |
| 3 | 4 | 7 | 10 | 20 |
| 4 | 5 | 11 | 15 | 35 |
| 5 | 6 | 16 | 21 | 56 |
| 10 | 11 | 56 | 66 | 286 |
| 20 | 21 | 211 | 231 | 1771 |
| 50 | 51 | 1276 | 1326 | 23426 |
| 100 | 101 | 5051 | 5151 | 176851 |
| 200 | 201 | 20101 | 20301 | 1373701 |
| 500 | 501 | 125251 | 125751 | 21084251 |
| 1000 | 1001 | 500501 | 501501 | 167668501 |

**Abb. 15:** Anzahl benötigter Terme für die verschiedenen linearen Regressionsansätze

**Typische Eigenschaften der linearen Regression** sind:

- **Aufwandsabschätzung:** Für die Anzahl der Terme findet man bei großen Werten von n näherungsweise:
  - Linearer Ansatz: $\cong n$
  - WW und quadratischer Ansatz: $\cong \frac{1}{2}n^2$
  - KubischerAnsatz: $\cong \frac{1}{6}n^3$

  Um das Verfahren sinnvoll durchzuführen zu können, müssen (deutlich) **mehr Datensätze als Koeffizienten** vorhanden sein.

- Für kleine n können die Koeffizienten direkt aus einem linearen Gleichungssystem berechnet werden. Dies geht bei nichtlinearen Regressionsverfahren (neuronalen Netzen) eben gerade nicht.

- Es können natürlich nur die Funktionen exakt modelliert werden, die vom Typ des jeweiligen Ansatzes sind.

- Falls n groß ist (größer ungefähr 10) wird üblicherweise nur der lineare Ansatz verwendet. In diesem Fall funktioniert die lineare Regression nur dann gut, wenn die Zusammenhänge auch im Wesentlichen linear sind.

- Tatsächlich kann die lineare Regression als das **einfachste denkbare NN** aufgefasst werden. Komplexere NNe (Feedforwardnetze) weisen jedoch eine wesentlich effizientere Modellierungsfähigkeit (selbst im Vergleich zur linearen Regression mit nichtlinearen Ansatzfunktionen) auf.

### 3.2 Gradientenverfahren

Möglichkeiten zur Minimierung einer Funktion, die auch für nichtlineare Regressionen noch funktionieren, sind iterative Verfahren. Wichtigstes Verfahren dieser Art ist das so genannte **Gradientenverfahren** zur Minimierung einer Funktion $y = f(x_1,...,x_n)$. Dieses Verfahren wird auch als **Gradientenabstieg** oder **Abstiegsverfahren** bezeichnet.

Wir setzen im Folgenden zur Definition des Gradientenverfahrens (wie in der Mathematik üblich) das Vorliegen einer Funktion $y = f(x_1,...,x_n)$ voraus, für die ein Minimum bestimmt werden soll, d.h. die $x_i$ sind die unabhängigen Variablen. In der späteren Anwendung auf Regressionsverfahren sind dann natürlich die gesuchten Regressionsparameter die unabhängigen Variablen.
Es sei $\underline{X}^0$ eine Näherung (**Startvektor, Anfangsnäherung**) für die gesuchten optimalen Werte mit den Komponenten $x_1^0, x_2^0,..., x_n^0$. Berechne neue Näherung nach folgender Vorschrift:

$$\underline{X}^{i+1} = \underline{X}^i - \eta \nabla f(\underline{X}^i) \quad \eta > 0 \quad i = 0,1,2,...$$

mit $$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ . \\ . \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

d.h. der **Gradient ist der Vektor der partiellen Ableitungen** der Funktion f nach den unabhängigen Variablen $x_i$. Diese partiellen Ableitungen sind i. A. selbst wieder Funktionen sämtlicher $x_i$. Aus diesem Grund muss der Gradient dann auch noch an der Stelle $\underline{X}^i$ ausgewertet werden. Eine Iteration, d.h. die Berechnung von $\underline{X}^{i+1}$ aus $\underline{X}^i$ wird als **Gradientenschritt** bezeichnet. Die Größe $\eta$ (eta) heißt **Schrittweite** des Gradientenschritts. Die Motivation hinter diesem Verfahren beruht auf der Überlegung, dass der **Gradient** einer Funktion in die **Richtung des steilsten Anstiegs** der Funktion zeigt. Diese Richtung kann natürlich von Punkt zu Punkt verschieden sein, deshalb auch die Auswertung des Gradienten an der Stelle $\underline{X}^i$. Dann sollte aber die negative Gradientenrichtung in die gewünschte Richtung des steilsten Abstiegs zeigen. Man geht also von einem Punkt $\underline{X}^i$ in die **Richtung des steilsten Abstiegs (daher auch Abstiegsverfahren)** und erhält so einen neuen Punkt $\underline{X}^{i+1}$. Dabei stellt sich natürlich die Frage, wie weit man in einem Schritt in die gerade berechnete negative Gradientenrichtung gehen soll. Dies wird gerade über die Schrittweite $\eta$ geregelt. Bei hinreichend kleiner Schrittweite ergibt sich an der neuen Stelle $\underline{X}^{i+1}$ ein geringerer Funktionswert als an der Ausgangsstelle $\underline{X}^i$, es sei denn die





Ausgangsstelle war bereits ein lokales Minimum. Die Verwendung einer zu geringen Schrittweite führt allerdings zu einem hohen Rechenaufwand, da pro Schritt nur geringe Fortschritte erzielt werden. Eine sehr große Schrittweite birgt die Gefahr, gewissermaßen über die Talsohle hinauszuschießen (man mache sich den Ablauf des Gradientenverfahrens im Gebirge klar), d.h. der neue Wert $\underline{X}^{i+1}$ hat dann u. U. sogar einen höheren Funktionswert als der Ausgangspunkt $\underline{X}^i$.

In Komponenten geschrieben, lautet ein Schritt des Gradientenverfahrens:

$$x_j^{i+1} = x_j^i - \eta \frac{\partial f}{\partial x_j}(\underline{X}^i) \quad j = 1,...,n$$

In der folgenden Abbildung ist das Prinzip des Gradientenverfahrens für den eindimensionalen Fall (d.h. nur eine unabhängige Variable x) dargestellt. In diesem Fall stimmt die part. Ableitung natürlich mit der gewöhnlichen Ableitung überein.

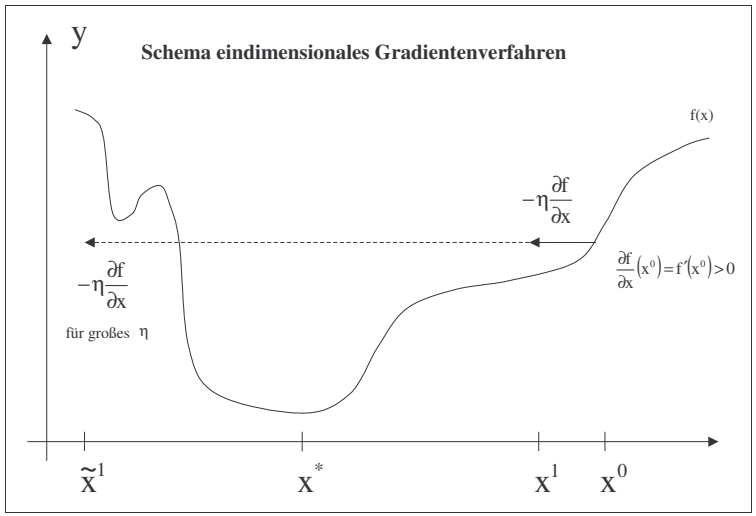



**Abb. 16:** Eindimensionales Gradientenverfahren zur Bestimmung von $x^*$

Verfahren startet bei $x^0$. An dieser Stelle ist der Gradient (die Ableitung) der Funktion f positiv, zeigt also nach rechts. Entsprechend zeigt $-\eta\frac{\partial f}{\partial x}(x^0)$ nach links.

- $x^1$ ist der neue Näherungswert für das gesuchte Minimum für kleines $\eta$. Der Punkt liegt zumindest näher an der gesuchten Lösung $x^*$.

- $\tilde{x}^1$ ist der neue Näherungswert für das gesuchte Minimum für großes $\eta$. Dieser Wert ist aber unbrauchbar, da $f(\tilde{x}^1) > f(x^0)$ gilt. *„Über das Tal hinausgeschossen."*

**Problem:** Falls $\eta$ zu groß ist, dann ist der Funktionswert an der neuen Stelle größer als an der alten Stelle. Falls $\eta$ zu klein ist, kommt man nicht von der Stelle.
**Lösung: Schrittweitensteuerung** während des Verfahrensablaufs.

Im Folgenden ist das Gradientenverfahren im 2-dimensionalen skizziert. Bei der uns interessierenden Anwendung auf lineare oder nichtlineare Regressionsverfahren ist das Problem häufig sehr hochdimensional (u. U. Hunderte oder Tausende von Parametern).

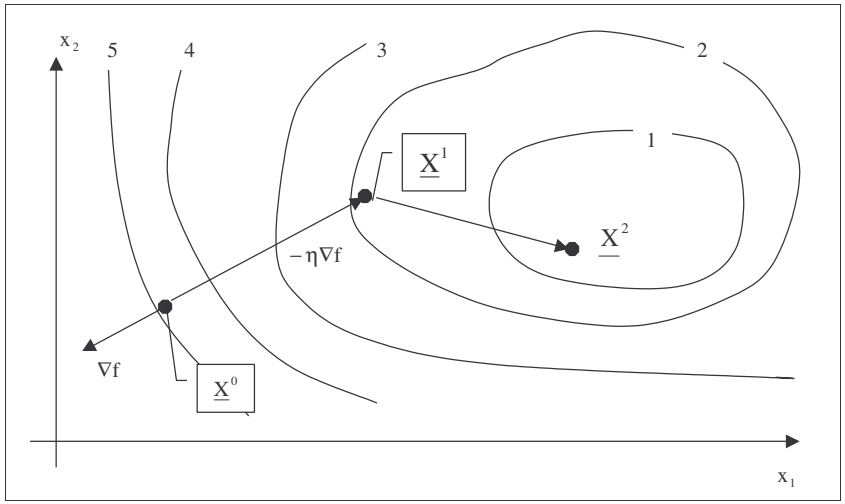



**Abb. 17:** Höhenlinien von f und zweidimensionales Gradientenverfahren

Zum Gradientenverfahren gibt es unzählige Varianten (z.B. Conjugierte Gradienten, Newtonverfahren), die in der Regel schneller zum Ziel führen sollen.





**Beispiel Newtonverfahren (NV):**

- hier werden auch die 2. partiellen Ableitungen von f benutzt, um weniger Iterationsschritte bis zum Ziel zu benötigen. Verfahren zweiter Ordnung.
- Aufwand pro Iterationsschritt
  - proportional zu $n^2$ beim NV
  - proportional zu n beim Gradientenverfahren
  - Bei n = 1000 kann ich für einen Schritt des NV („Newtonschritt") auch 1000 Gradientenschritte machen
  - Verfahren höherer Ordnung (d.h. größer 1) aus diesem Grund in der Praxis meist nicht brauchbar (nur für kleine Probleme $n <\approx 20$)

**Anwendung des Gradientenverfahrens zur Regression**
Die Anwendungen des Gradientenverfahrens auf die Minimierung der Funktion $Z(a_0,...,a_n)$ liefert also ein iteratives Verfahren („**Lernverfahren**") zur Ermittelung der Regressionskoeffizienten. Das Verfahren ist **selbstkorrigierend** (auftretende Rundungsfehler können in nachfolgenden Iterationsschritten wieder korrigiert werden), d.h. numerisch stabil. In der Praxis treten lange Rechenzeiten, bedingt durch Tausende von Iterationsschritten auf. In jedem Schritt müssen n-Ableitungen bestimmt werden. Dazu sind jedes Mal Fehlersummen mit p-vielen Summanden auszuwerten.
Für die notwendige Berechnung der partiellen Ableitungen können aus den Regressionsansätzen analytische Formeln hergeleitet werden. Bei vielen Anwendungen des Gradientenverfahrens werden die partiellen Ableitungen jedoch nach einer der folgenden Formeln berechnet:
**Vorwärtsgenommener Differenzenquotient** an der Stelle $\underline{X}$

$$\frac{\partial f}{\partial x_i}(\underline{X}) \cong \frac{f(x_1,...,x_{i-1},x_i+\varepsilon,x_{i+1},...x_n)-f(x_1,...,x_n)}{\varepsilon}$$

**Zentraler Differenzenquotient**

$$\frac{\partial f}{\partial x_i}(\underline{X}) \cong \frac{f(x_1,...,x_{i-1},x_i+\varepsilon,x_{i+1},...x_n)-f(x_1,...,x_{i-1},x_i-\varepsilon,x_{i+1},...x_n)}{2\varepsilon}$$

Dabei ist $\varepsilon > 0$ eine kleine Zahl, jedoch aus Gründen der numerischen Stabilität deutlich oberhalb der Genauigkeit des verwendeten numerischen Datentypen. Der exakte Wert der partiellen Ableitung wäre ja der jeweilige Limes für $\varepsilon$ gegen 0. In diesem Sinne sind die genannten Differenzenquotienten numerische Näherungen der partiellen Ableitungen. Der

zentrale Differenzenquotient bietet eine höhere Genauigkeit, ist aber mit doppelt so vielen Auswertungen der Funktion f verbunden.

**Beispiel:** Der Excel-Solver bietet beide Optionen und auch die Wahl zwischen Newton- und Gradientenverfahren (siehe Optionen beim Solver)





## 3.3 Adaline und Perceptron

Einstufige Netze, binäre Inputs und Outputs

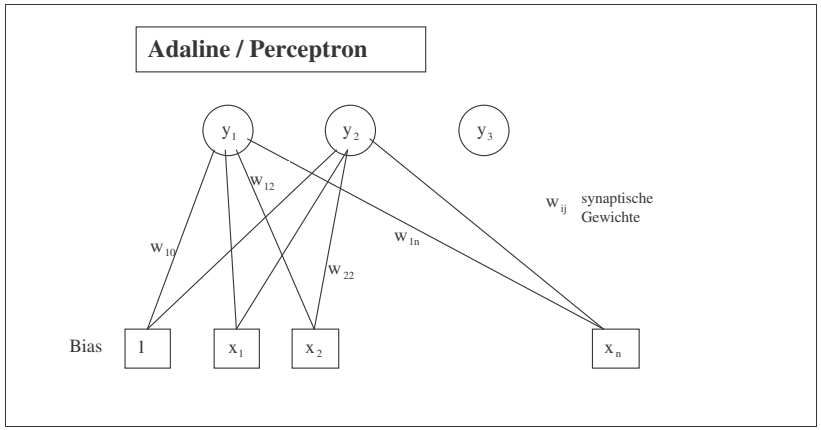



**Abb. 18:** Verschaltung einstufiger Netze mit synaptischen Gewichten.

Der **Output am Ausgangsknoten** $y_i$ ist definiert durch:

$$y_i = f\left(\sum_{j=0}^{n} w_{ij} x_j\right) \qquad x_0 = 1 \text{ Bias}$$

Adaline und Perceptron unterscheiden sich nur durch die **Aktivierungsfunktion** f:

$$f(x) = \begin{cases} 1 & \text{für } x \geq 0 \\ -1 & \text{für } x < 0 \end{cases} \qquad \text{Adaline}$$

$$f(x) = \begin{cases} 1 & \text{für } x \geq 0 \\ 0 & \text{für } x < 0 \end{cases} \qquad \text{Perceptron}$$

Gewichte werden anhand von Trainingsdaten mit Gradientenverfahren („**Delta-Regel**“) angepasst.

**Beispiel:**

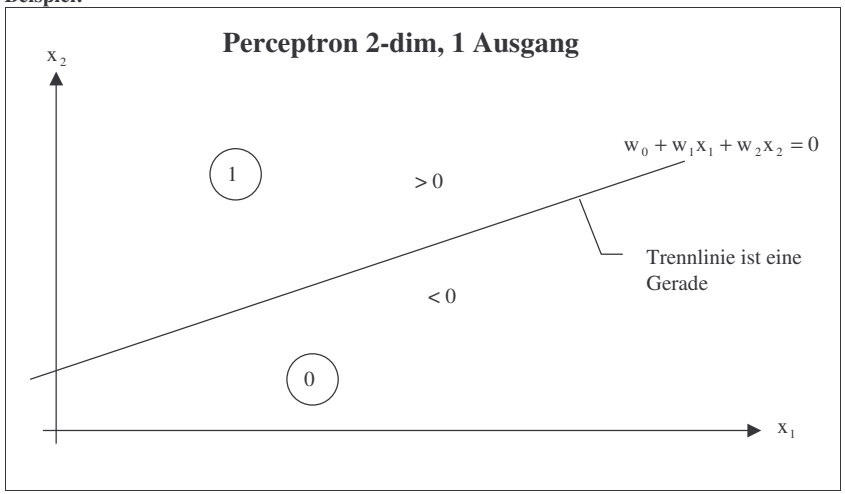



**Abb. 19:** Aktivierungsfunktion schaltet entlang einer Geraden („Trennlinie“) um.

Wie die Abbildung zeigt, zerlegt das Perceptron den Eingangsraum $(x_1, x_2)$ in zwei Bereiche entlang einer Geraden. Die Gerade ist dadurch definiert, dass das Argument der Aktivierungsfunktion gleich Null ist. In höherdimensionalen Fällen wird der Bereich durch eine (affine Hyper-) Ebene zerlegt.

### Beispiel: XOR-Problem

Die XOR-Funktion ist durch die folgende Tabelle definiert:

| $x_1$ | $x_2$ | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |





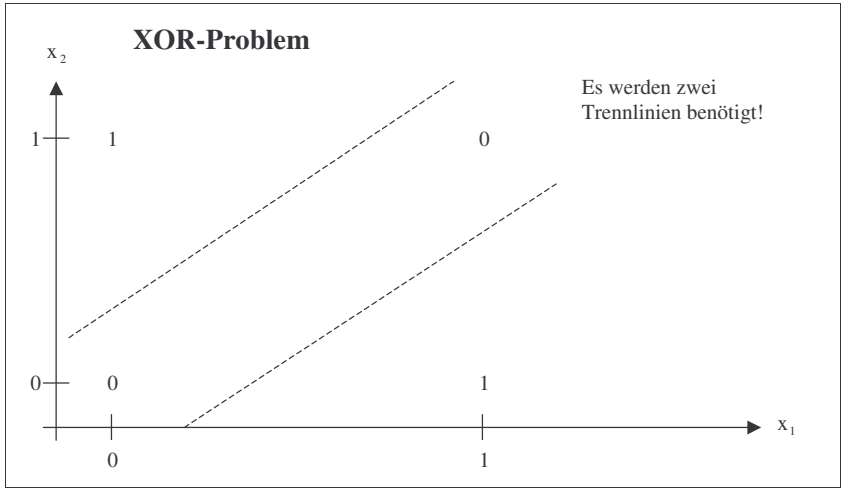



**Abb. 20:** Perceptron kann XOR-Problem nicht lösen.

**Adaline und Perceptron sind einfach zu simpel!**

Sie entsprechen einer einzelnen Nervenzelle und nicht einem NN. Sie machen letztlich nicht mehr als lineare Regression (mit Abschneiden der Funktionswerte). Die Erkenntnis (1969 Minsky, Papert), dass das Perceptron das XOR-Problem nicht lösen kann, hat die ganze Entwicklung um 15 Jahre blockiert.

### 3.4 Feedforward-Netzwerke

Die weitaus meisten praxisrelevanten Anwendungen künstlicher neuronaler Netze wurden und werden mit Feedforwardnetzwerken realisiert. Sie ermöglichen die effiziente Modellierung beliebiger stetiger Funktionen und bilden aus diesem Grund das „Arbeitspferd“ der angewandten Neuroinformatik.

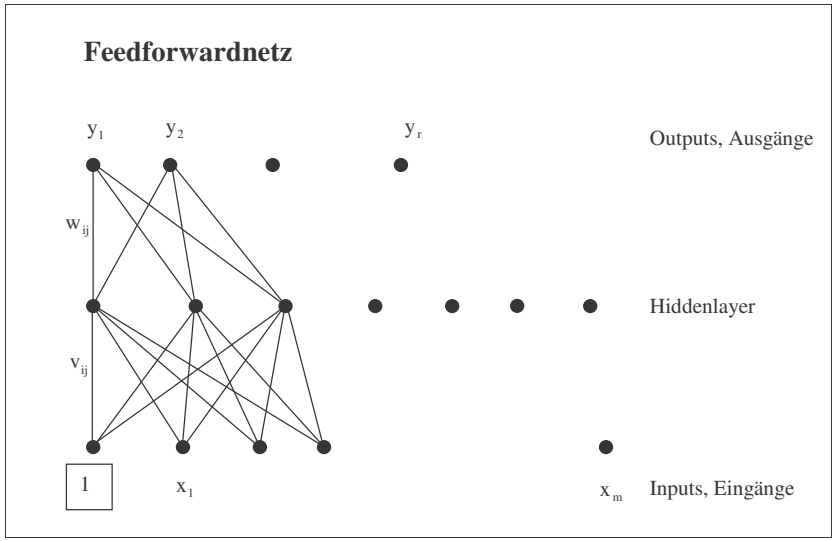



**Abb. 21:** Struktur eines Feedforwardnetzes mit einem Hiddenlayer

**Eigenschaften von Feedforwardnetzen:**

- (Mindestens) ein **Hiddenlayer** mit n Neuronen.
- Aufeinanderfolgende Schichten sind **voll vernetzt**.
- Im j-ten **Hiddenlayerknoten (Hiddenlayerneuron)** wird zunächst der Gesamteingang $net_j$ berechnet:

$$net_j = \sum_{i=0}^{m} v_{ji} x_i \qquad x_0 = 1, \text{ Bias}$$





- Der Ausgabewert des j-ten Hiddenlayerneuron ist $\sigma(net_j)$ mit der so genannten **Sigmoidfunktion:**

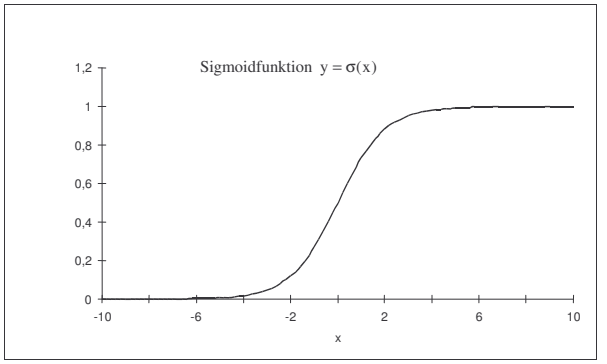



**Abb. 22:** Beispiel einer Sigmoidfunktion (hier der Verlauf der Fermifunktion)

Eine **Sigmoidfunktion ist** eine monoton wachsende und beschränkte stetige Funktion, d.h. die Funktion hat sowohl bei $-\infty$ als auch bei $\infty$ einen endlichen Grenzwert. Damit handelt es sich insbesondere um **eine nichtlineare Funktion**. Die Fermifunktion der Abbildung ist nur eine mögliche Sigmoidfunktion. Schematisch lässt sich die Verarbeitung der Signale in einem Hiddenlayerknoten wie folgt darstellen:

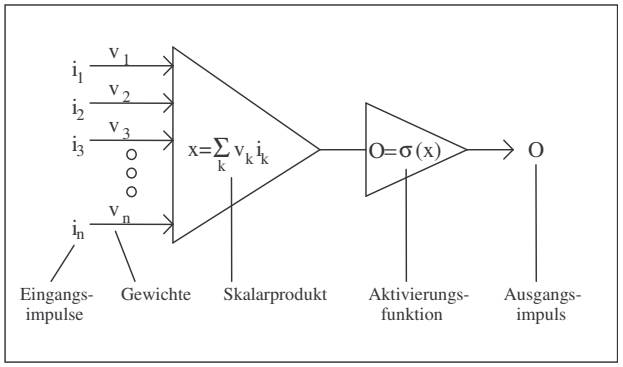



**Abb. 23:** Verarbeitung der Inputsignale in einem Hiddenlayerknoten (Knotenindex weggelassen), x entspricht $net_j$.

- Der **Ausgang** $y_k$ berechnet sich dann analog zu:

  $$y_k = g\left(\sum_{j=1}^{n} w_{kj}\,\sigma(net_j)\right)$$

  mit $g(x) = \sigma(x)$ oder $g(x) = x$

- Üblicherweise werden alle Inputs- und Outputs auf ein einheitliches Intervall skaliert. Meist wird eines der folgenden Intervalle gewählt: $[0,1],[0.1,0.9],[-1,1],[-\frac{\pi}{2},\frac{\pi}{2}]$. Mit der folgenden Funktion kann beispielsweise auf das Intervall [0,1] skaliert werden:

  $$x_s = \frac{x - x_{min}}{x_{max} - x_{min}}$$ **„Lineare Skalierung“**

  x ist unskalierter Input- oder Outputwert, $x_s$ ist skalierter Wert.

  Eine Skalierung ist für die Outputs unbedingt erforderlich, falls $g(x) = \sigma(x)$ gewählt wird.

- Für $\sigma(x)$ wird normalerweise die **Fermifunktion** verwendet:

  $$\sigma(x) = \frac{1}{1 + e^{-x}}$$

  Alternativen sind $\sigma(x) = \arctan(x)$ oder $\sigma(x) = \tanh(x)$. Dann sind natürlich dazu passende Skalierungsintervalle zu wählen.

- Auf die Verwendung einer Sigmoidfunktion (entspricht der Aktivierungsfunktion) bei den Hiddenlayerknoten kann nicht verzichtet werden. Ohne diese Funktion kollabiert ein Feedforwardnetzwerk sofort wieder zu einer linearen Regression. Diese Funktion sorgt dafür, dass die Gewichte $v_{ij}$ als nichtlineare Regressionsparameter in die Modellierung eingehen. Damit entspricht ein Feedforwardnetz mathematisch einem **nichtlinearen Regressionsverfahren.**

- **Netztraining/Lernverfahren**: Die **synaptischen Gewichte** (Netzgewichte) $v_{ij}, w_{ij}$, die natürlich nichts anderes darstellen als die Regressionsparameter, werden durch ein iteratives Verfahren (**„Lernalgorithmus**“) anhand von Beispieldatensätzen (**„Lernset“**) an die zu modellierende Funktion angepasst. Bekanntestes Verfahren dieser Art ist **„Backpropagation (of errors)“**. Daher werden Feedforwardnetze gelegentlich auch als „Backpropagationnetzwerke“ bezeichnet. Diese Bezeichnung ist jedoch irreführend, da auf einer Feedforwardstruktur unzählige verschiedene Lernalgorithmen ablaufen können (derzeit sind Hunderte von Algorithmen für Feedforwardnetze veröffentlicht). Backpropagation ist nichts anderes als die **Anwendung des Gradientenverfahrens** auf die Bestimmung der Netzgewichte, allerdings unter Ausnutzung effizienter (analytischer)





Formeln zur Berechnung des Gradienten. Dabei wird einfach der quadratische Fehler minimiert. Bei mehr als einem Output wird der Fehler des i-ten Musters (d.h. des i-ten Beispieldatensatzes):

$$E_i = \sum_{k=1}^{r}\left(y_{ik} - y_{ik}^{pr}\right)^2$$

Bei der Korrektur der Gewichte gibt es nun zwei Möglichkeiten

a) Erst alle Fehler $E_i$ zum Gesamtfehler $E = \sum_{i=1}^{p} E_i$ aufsummieren und dann $-\eta\nabla E$ zur Korrektur der Netzgewichte benutzen. **„Batch-Backpropagation (BP)“.**

b) Nach jedem Muster $-\eta\nabla E_i$ zur Korrektur verwenden. **„Single-Step BP“.** Obwohl bei dieser Variante von vorne herein gar nicht klar ist, dass sie überhaupt konvergiert *(„Warum nicht?“),* zeigt sich, dass sie bei richtiger Anwendung (hinreichend kleine Schrittweiten) der Variante a) sogar überlegen ist. Insbesondere bleibt sie nicht so häufig in lokalen Minima der Fehlerfunktion hängen.

Backpropagation ist als Lernverfahren in der Regel sehr langsam. In der Praxis werden Hunderte bis Tausende von Epochen (d.h. Anwendungen der gesamten Lernmenge) benötigt um den Modellierungsfehler hinreichend klein zu bekommen. Aus diesem Grund ist eine **Vielzahl von wesentlich effizienteren Lernverfahren bekannt**. Ab Anfang der 90er Jahre wurden für neuronale Netze **Spezialalgorithmen („special solver“)** entwickelt, die ausschließlich für das Training von Feedforwardnetzen konzipiert sind und bei denen die spezielle Netzstruktur ausgenutzt wird. Dagegen ist das Gradientenverfahren ein ganz allgemeines Optimierungsverfahren **(„general solver“)**, das bei Optimierungsaufgaben jeder Art zum Einsatz kommt. Special Solver erreichen Geschwindigkeitsvorteile gegenüber BP um Faktoren größer als 1000, d.h. eine Lernaufgabe, die mittels BP viele Stunden dauert, reduziert sich auf einige Sekunden. Da diese Verfahren jedoch in der Regel wesentlich komplizierter sind, werden sie in kommerziellen Tools oft nicht verwendet. Die Software NN-Tool in den Praktikumsaufgaben setzt jedoch einen solchen Lernalgorithmus ein.

- **Theoretische Aussagen:** Wie bereits zu Anfang erwähnt, stellen Feedforwardnetze sehr effiziente Methoden für die praktische Anwendung dar. Dazu lassen sich jedoch auch theoretische Aussagen bezüglich der Modellierungsfähigkeit herleiten:

**Satz 1**: **Satz von Hornik, Stinchcombe, White**
Feedforwardnetze mit mindestens einem Hiddenlayer können jede stetige Funktion von $R^m \rightarrow R^n$ approximieren.

**Satz 2**: **Barronsches Theorem**
Feedforwardnetze sind die effizientesten bekannten Approximationsverfahren.

- **Anmerkungen** zu den theoretischen Aussagen:
  Der Satz 1 stellt gewissermaßen eine **Minimalforderung (notwendige Bedingung)** für ein leistungsstarkes Approximationsverfahren dar. Dieser Satz gilt entsprechend auch für eine Reihe weiterer Verfahren (u. A. für die lineare Regression) und ist für sich genommen noch kein Argument für die Nutzung von Feedforwardnetzen.
  Erst Satz 2 liefert ein **hinreichendes Argument** für die Nutzung von Feedforwardnetzen. Die Effizienz ist dabei wie folgt definiert:
  ***„Ein Approximationsverfahren A ist effizienter als ein Verfahren B, wenn Verfahren A aus einer vorgegebenen Menge von Datensätzen ein Modell mit geringerem Modellfehler (genaueres Modell) als Verfahren B erzeugt bzw. wenn Verfahren A zur Erreichung einer vorgegebenen Modellgenauigkeit weniger Beispieldatensätze benötigt als Verfahren B.“***
  In diesem Sinne ist ein Verfahren also besonders effizient, wenn es mit **möglichst wenig Beispieldatensätzen** auskommt. Dahinter steckt die Begründung, dass Beispieldatensätze in der Praxis nicht in beliebiger Menge zur Verfügung stehen, da sie in der Regel aufwendig zu beschaffen sind (häufig nur durch kostspielige Versuchsreihen). Dagegen spielt die Effizienz des Lernvorgangs (d.h. die benötigte Rechenzeit zur Modellerzeugung) nur eine untergeordnete Rolle. In dieser Hinsicht ist die lineare Regression ohnehin kaum zu übertreffen.

- **Prinzip der Modellierung** (anschaulich): Im folgenden 1-dim Diagramm ist dargestellt, wie mittels zweier Sigmoidfunktionen ein „Hügel“, definiert durch Datenpunkte, modelliert werden kann. Wenn man sich nun klarmacht, dass jede stetige 1-dim Funktion als Überlagerung von hinreichend vielen Hügeln dargestellt werden kann, erhält man einen anschaulichen Hinweis wie die Modellierung im Eindimensionalen funktioniert. Analoge Überlegungen gelten dann auch für den mehrdimensionalen Fall.





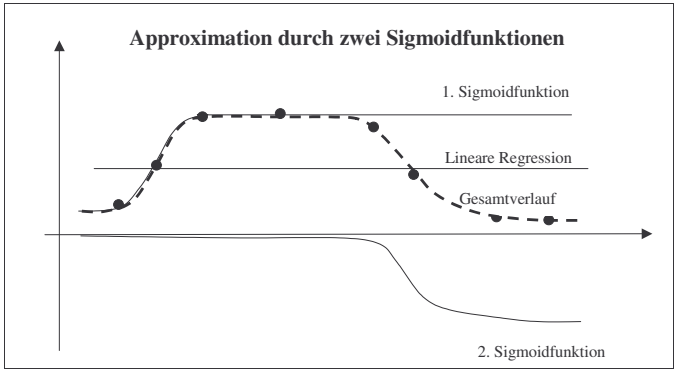



**Abb. 24:** Approximation eines durch Punkte vorgegebenen Funktionsverlaufs mit zwei Sigmoidfunktionen.

- Auch die XOR-Funktion stellt für ein Feedforwardnetz mit zwei Hiddenlayerknoten jetzt kein Problem mehr dar:

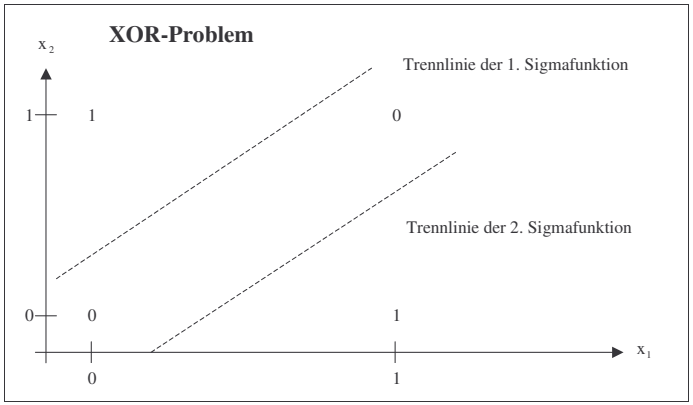



**Abb. 25:** XOR-Problem mit zwei Sigmoidfunktionen.

- **Offene Fragen:**
  a) Wie viele innere Knoten?
  b) Wie viele Lernschritte?
  c) Wann ist ein gutes Modell erreicht?

### 3.5 Spezielle Netztypen (Exoten)

**A) Radiale Basisfunktionen (RBF-Netze)**
Ebenfalls vom Feedforwardtyp, aber mit anderer Übertragungsfunktion. Für hochdimensionale Probleme weniger geeignet.

**B) Hopfield-Netze**
Rückgekoppelte Netze zur Mustererkennung. Zu gestörtem Bild kann ungestörtes Bild gefunden werden.

**C) Boltzmann-Netze**
Rückgekoppelte Netze für Optimierungsaufgaben.

**D) Recurrente Netze**
Rückgekoppelte Netze für Zeitreihenprognosen (Beispiel Aktienkurs). Sehr rechenaufwendige Lernalgorithmen.

**E) Kohonen-Karten, Feature Maps**
Automatische Extraktion von Features, Simulation der Informationsrepräsentation im Gehirn.

...

**Zusammenfassung:**
- Alle diese Typen haben nur geringe praktische Bedeutung.
- Lernaufwand ist bei fast allen rückgekoppelten Netzen immens hoch.
- Viele Aufgaben lassen sich anders besser lösen.





# 4 Praktischer Einsatz von Feedforward Netzwerken

Die in diesem Kapitel vorgestellten Methoden und Überlegungen beziehen sich zwar auf den Einsatz bei neuronalen Netzen, gelten aber in der Regel auch bei anderen Modellierungsverfahren (insbesondere bei der linearen Regression).

### 4.1 Parameterauswahl und Kodierung

**Wichtigster Aspekt überhaupt:**
Was hier falsch gemacht wird, kann durch ein noch so gutes NN nicht wieder aufgeholt werden.

#### 4.1.1 Wirkungsrichtung
In welcher Richtung liegt bei einem Problem überhaupt ein funktionaler Zusammenhang vor, d.h. was sind die Inputs, was die Outputs?

**Beispiele:**
1. **Rezepturoptimierung**
   Gesucht ist die **optimale Rezeptur (Zusammensetzung) für Gummimischungen** von Autoreifen. Ziel ist eine Rezeptur, die optimale Haftung mit minimalem Abrieb vereinigt.

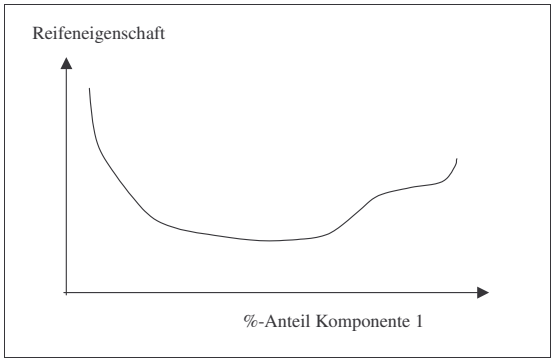



**Abb. 26:** Reifeneigenschaften sind Funktionen der Rezeptur aber nicht umgekehrt!

D.h. einem Netz kann ich nur die Komponenten vorgeben und es prognostiziert die Eigenschaften:

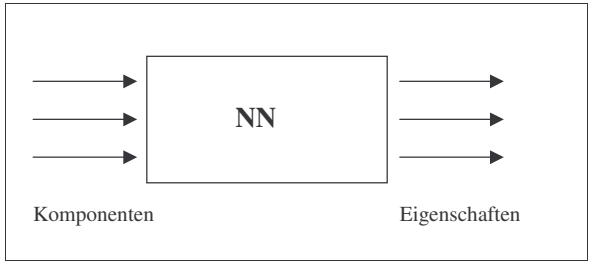



**Abb. 27:** Modellierung von Reifeneigenschaften als Funktionen der Rezeptur mit einem Neuronalen Netz.

**Inverse Fragestellung:** Die eigentlich gewünschte Vorgabe des Eigenschaftsprofils und die Prognose der dazu passenden Rezeptur ist direkt i. A. nicht möglich, da in dieser Richtung **kein funktionaler Zusammenhang** vorliegt. Diese Fragestellung kann also von einem NN nicht direkt erledigt werden. Stattdessen muss diese Aufgabe als Optimierungsaufgabe formuliert und gelöst werden, in der das NN die zur Optimierung benötigte Funktion liefert. D.h. die inverse Fragestellung führt auf ein Optimierungsproblem, bei dem (bei fertig trainierten Netz) die Eingangsparameter variiert werden.

**2. Strukturaufklärung in der chemischen Analyse:**
Vermessen wird ein Spektrum (z.B. NMR-Spektrum), das Netz soll die zugehörige chemische Struktur liefern. Dennoch: Zu jeder chemischen Struktur gehört eindeutig ein Spektrum, nicht umgekehrt. Also:

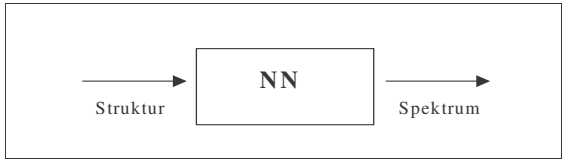



**Abb. 28:** Modellierung eines Spektrums als Funktionen der chemischen Struktur.

Auch hier muss die eigentlich interessierende Fragestellung wieder als inverse Fragestellung formuliert werden.





**Satz:**
Neuronale Netze können nur Funktionen lernen, d.h. die Ausgangsgrößen müssen Funktionen (im mathematischen Sinne) der Eingänge sein.

**Hinweise:**

- Kausalität, d.h. Ursache-Wirkung berücksichtigen.
- „An welchen Parametern kann ich frei drehen?“ $\rightarrow$ Inputs.

### 4.1.2 Auswahl der richtigen Ein- und Ausgangsgrößen

Wie sind die Inputs und Outputs zu kodieren, so dass überhaupt eine Chance für eine hinreichend gute Modellierung besteht?

**Beispiele:**

1) **Prognose des Schlaganfallrisikos**

   Aus dem Strömungsgeräusch des Blutes soll die Gefährdung eines Patienten für Schlaganfall prognostiziert werden.

   Ausgangsgröße Gefährdung: $\begin{cases} 0 & \text{keine Gefährdung} \\ 1 & \text{hohe Gefährdung} \end{cases}$

   Eingangsgrößen ?

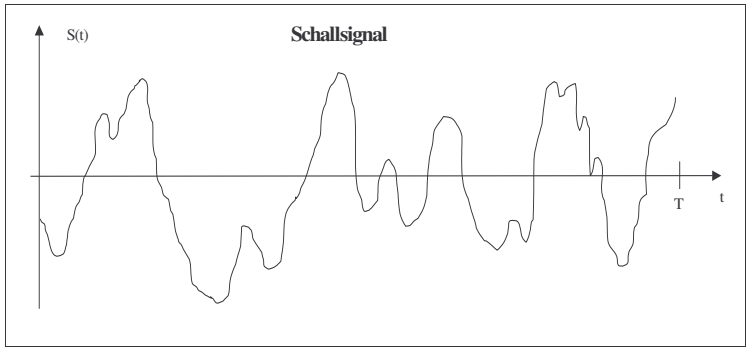



**Abb. 29:** Verlauf des Schallsignalpegels (schematisch).

Schallsignal wird über eine Zeitspanne T aufgenommen. Signal wird nach Frequenzen zerlegt (**„Fourieranalyse“**), d.h.:

$$S(t) = a_0 + \sum_{k=1}^{\infty} \left( a_k \cos(k\,\omega t) + b_k \sin(k\,\omega t) \right) \qquad \omega = \frac{2\pi}{T}$$

dabei sind $a_k$, $b_k$ die Amplituden der verschiedenen Oberschwingungen der Grundfrequenz $\omega$.

Alternativ kann auch nach Amplitude und Phase zerlegt werden:

$$S(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\,\omega t - \delta_k) \qquad \omega = \frac{2\pi}{T}$$

mit $a_k$ Amplitude, $\delta_k$ Phase.





Mögliche Eingangsparameter ergeben sich aus den Amplituden und Phasen durch Begrenzung der unendlichen Reihe bei einer Maximalfrequenz $N_{max}$ (z.B. $N_{max} = 100$):

$a_k$, $b_k$ für $k \le N_{max}$

$a_k$, $\delta_k$ für $k \le N_{max}$

Damit ergibt sich die folgende Netzstruktur:

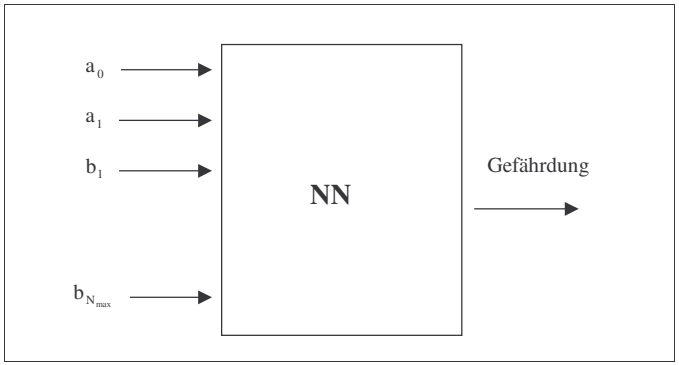



**Abb. 30:** Prognose der Schlaganfallgefährdung, d.h. ca. 200 Inputs, ein Output.

2) **Bildanalyse**

   Erkennung von Objekten (insbesondere Personen, Gesichter) anhand von Kameraaufnahmen. In der folgenden Abbildung ist ein System des Zentrums für Neuroinformatik der Uni Bochum („ZN“) zur Gesichtserkennung dargestellt. Bei Anwendungen dieser Art wird zwischen zwei Betriebsarten unterschieden:

   - **Erkennung:** System überwacht bestimmte Bereiche (z.B. Zugänge zu Bahnhöfen, Flughäfen) an denen viele Personen vorbeikommen und überprüft, ob eine der Personen von „besonderem Interesse“ ist, d.h. auf einer Fahndungsliste steht.
   - **Identifikation:** Das System soll nur bestätigen, dass eine Person mit einer bestimmten vorgegebenen Person übereinstimmt. Typisch: Mitarbeiter möchte Zugang zu einem sicherheitskritischen Bereich, steckt ID-Karte in ein Lesegerät und das System prüft, ob die über die Karte identifizierte Person mit der Person, die gerade in die Kamera blickt, übereinstimmt.

   Die Identifikation ist deutlich einfacher als die Erkennung, insbesondere da die interessierenden Personen bei der Erkennung oft daran gar kein gesteigertes Interesse haben.

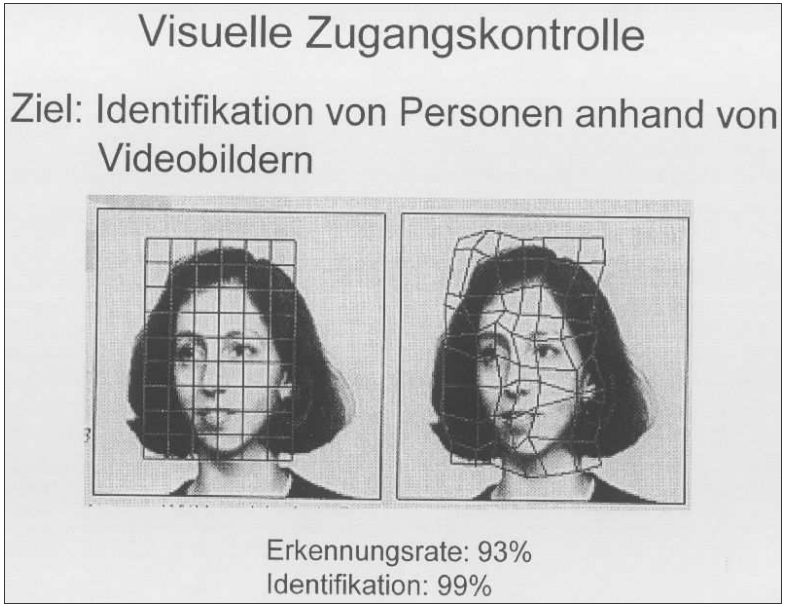



**Abb. 31:** Gesichtserkennungssystem des Zentrums für Neuroinformatik.

Die **typische Situation bei der Bildanalyse** ist die Folgende:
Eine Kamera nimmt **1000*1000 Bildpunkte** (Richtwert) auf. An jedem Punkt gibt es 3 Farbwerte (rot, grün, blau) oder einen Grauwert (Schwärzung), d.h. **ein Bild ≙ 3 Mio. reeller Zahlen** für ein Farbbild, bzw. einer Million für ein SW-Bild (genauer sind es in der Regel durch jeweils ein Byte kodierte Integerwerte). Daraus folgt, dass die **einzelne Bildpunktinformation völlig ungeeignet** ist. Der einzelne Bildpunkt enthält praktisch keine problemrelevante Information. Für eine Bilderkennung werden also Eingangsparameter benötigt, in denen substanziell Information steckt.
Typischerweise wird ein Bild (oder einzelne Bildteilbereiche) in **„Features“** zerlegt, z.B. in so genannte **„Eigenmoden“:**

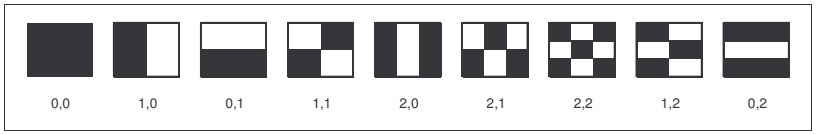



**Abb. 32:** Die ersten 9 Bildeigenmoden für SW-Bilder (schematisch).





Bei Farbbildern wird eine entsprechende Zerlegung für jeden der drei Farbauszüge vorgenommen. **Zu jeder Eigenmode gehört dann ein Amplitudenwert**, der angibt in welchem Maße die Mode im aktuellen Bild vertreten ist. Bei negativen Amplitudenwerten drehen sich schwarz und weiß gerade um. Die Indices entsprechen der Anzahl der Farbwechsel in x- bzw. y-Richtung. Die Grundmode 0,0 gibt beispielsweise die mittlere Schwärzung des Bildes an. Je höher der entsprechende Amplitudenwert, desto dunkler ist das Bild insgesamt. Die Mode 0,1 enthält Informationen, ob die obere Bildhälfte im Mittel heller als die untere Hälfte ist. Wenn man alle Moden bis 1000,1000 mitnimmt, lässt sich das ursprüngliche Bild exakt rekonstruieren. Jedes Bild kann somit in eine Reihe nach diesen Eigenmoden entwickelt werden (**„FFT – Fast Fourier Transformation“**). Eine solche Zerlegung nach Eigenmoden kann für das Gesamtbild oder für einzelne lokale Bildbereiche vorgenommen werden (**„Wavelets“**). Beim System der ZN wird dies lokal in kreisförmigen Umgebungen der Bildgitterpunkte durchgeführt. Tatsächlich will man zur Bildanalyse natürlich nicht sämtliche Eigenmoden, die zur exakten Bildrekonstruktion nötig sind, berücksichtigen. Die Überlegung besteht darin, dass die höheren Eigenmoden immer feinere Details des Bildes kodieren *(„bis hin zum einzelnen Haar“)*, dass diese Details für die Bilderkennung aber unnötig sind. D .h. auch hier schneidet man die Reihe ab und berücksichtigt nur die ersten $N_{max}$ Moden (analog zum vorhergehenden Beispiel). Deren Amplituden sind dann die Inputparameter (ca. 1000). Beim System der ZN werden beispielsweise an jedem Gitterpunkt ein Dutzend Parameter erfasst.

**3) Spektrenprognose für chemische Verbindungen**

Für die Aufklärung der Struktur organischer chemischer Verbindungen möchte man das NMR-Spektrum der Verbindung vorhersagen. Das Spektrum besteht aus einer Reihe von Peaks, die von den einzelnen Kohlenstoffatomen im Molekül erzeugt werden. Genauer erzeugen nur $C_{13}$ – Kohlenstoffatome diese Spektren; sie kommen aber auf allen Positionen im Molekül häufig genug vor. Die Lage dieser Peaks, d.h. die zugehörige Frequenz wird durch das lokale Magnetfeld am Ort des C-Atoms bestimmt. Dieses Magnetfeld wird selbst wieder von den Umgebungsatomen, d.h. den Atomen in der näheren und weiteren Nachbarschaft des gerade betrachteten C-Atoms, beeinflusst. Damit enthält die Frequenz Informationen über die räumliche Struktur des Moleküls.

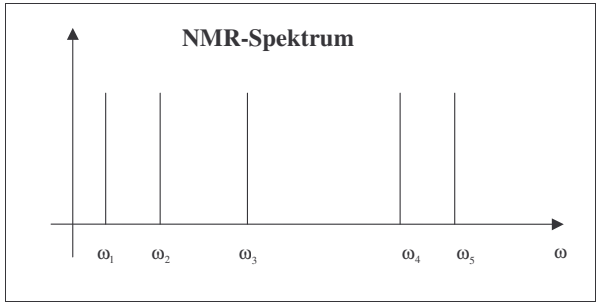



**Abb. 33:** NMR-Spektrum (schematisch) mit 5 Peaks, d.h. 5 C-Atomen

Während die Lage der Peaks, d.h. die Frequenz in nahe liegender Weise die Ausgangsparameter definieren, muss für die Eingangsparameter die chemische Struktur kodiert werden.

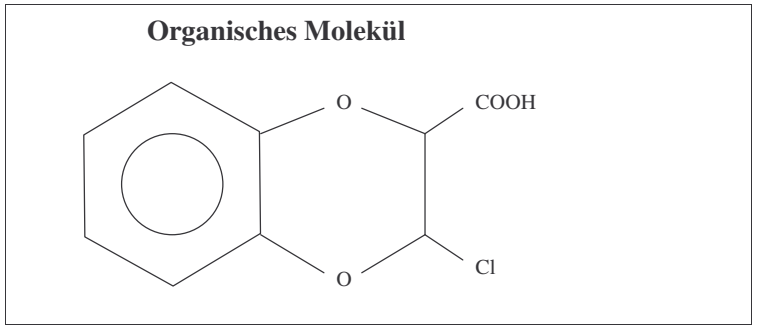



**Abb. 34:** Strukturformel eines organischen Moleküls.

Die Aufgabe besteht also darin, **zweidimensionale Graphen** von Molekülen (bei einigen Molekülen spielt sogar die 3D-Struktur eine Rolle) auf eine fest vorgegebene Anzahl von Parametern umzukodieren. Eine solche Kodierung kann nicht eindeutig sein! Aus diesem Grund existiert eine ganze Reihe von Kodierungsmethoden für chemische Strukturen, die in den einzelnen Anwendungen unterschiedlich gut funktionieren:

**A) Inkrementenmethode:**
Eine Methode ist die so genannte **Inkrementen- bzw. Substrukturmethode**. Man definiert für organische Verbindungen eine Anzahl von typischen Substrukturen (z.B. 50





verschiedene) und zählt dann das Auftreten der einzelnen Substruktur im vorgegebenen Molekül. Im dargestellten Molekül tauchen beispielsweise die folgenden Substrukturen (Inkremente) auf:

| Substruktur | Anzahl |
|---|---|
| CH | 2 |
| C aromatisch | 6 |
| COOH | 1 |
| O | 2 |
| Cl | 1 |

**Abb. 35:** Inkrementzahlen zum Molekül der Abb. 34

Problematisch bei dieser Art der Kodierung ist, dass zwar die einzelnen Substrukturen erfasst werden, jedoch ihre räumliche Beziehung untereinander nicht kodiert wird. So können durchaus verschiedene Moleküle die gleichen Inkrementzahlen aufweisen, sich chemisch jedoch unterschiedlich verhalten und auch ein anderes NMR-Spektrum aufweisen. Aus diesem Grund werden zum Teil wesentlich mehr (z.B. 1000 verschiedene) und wesentlich kompliziertere Substrukturen verwendet, in die dann auch noch die nächsten Nachbaratome mit eingehen.

**B) Kodierung nach physikalischen Parametern**
Das gesamte Molekül wird durch einzelne physikalische Parameter beschrieben, z.B.:

- Molmasse
- Lineare Abmessungen (Länge, Breite, Höhe)
- Dipol- und Quadrupolmoment
- Polarisierbarkeit

...
Auch diese Kodierung ist selbstverständlich nicht eindeutig. Darüber hinaus setzt die Bestimmung von vielen dieser Parameter aufwendige quantenchemische Berechnungen voraus.

**4) Technischer Prozess mit einer Vielzahl von Messstellen**
Im folgenden Bild ist ein technischer Produktionsprozess zur Kunststoffherstellung dargestellt. Das Edukt wird in Form von Granulat oben in den Turm eingefüllt und durchläuft den Turm während es durch Stickstoff beheizt wird (ca. 200 Grad Celsius). Dabei findet eine Polymerisationsreaktion statt und die Viskosität des Materials steigt deutlich an.

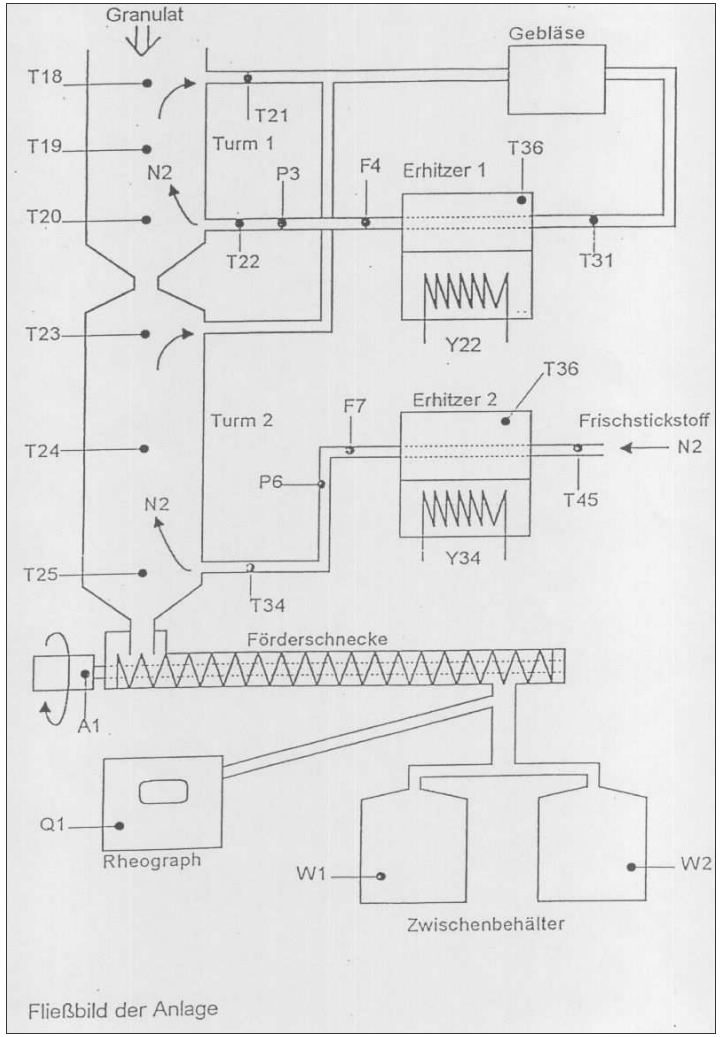



**Abb. 36:** Technischer Produktionsprozess mit Messstellen

Diese Endviskosität ist auch die Qualitätsgröße (Ausgangsparameter). Sie wird bei der Abfüllung des Produktes mittels des Rheographen (stichprobenartig) erfasst. Von Interesse ist es hier, die zu erwartende Produktqualität schon während des





Produktionsprozesses auf der Basis der Prozessparameter vorherzusagen. Kodierung:
**Eingangsgrößen sind die Prozessparameter** *(„Welche? Alle?“)*
**Ausgangsgrößen sind die Produkteigenschaften** (hier die resultierende Viskosität)

Dabei stellt sich die Frage, wie die Eingangsgrößen auszuwählen sind.
**Problem:**

- Eingangsgrößen sind untereinander stark korreliert. Benachbarte Temperaturmessstellen zeigen vergleichbare Werte an.
- Einflussbewertung schwierig.
- Viele Parameter.

**Lösung:**
**Beschränkung auf die wichtigsten Eingangsparameter**. *„Wie sind die auszuwählen?“*

**Zusammenfassung:**
**Auswahl von Eingangsparametern**
Gute Eingangsparameter sollten folgende Eigenschaften aufweisen:

- Alle wesentlichen Einflüsse auf die Ausgangsgrößen berücksichtigen.
- Möglichst wenige. (So wenig wie möglich, so viel wie nötig!)
- Möglichst oft und möglichst genau gemessen.
- Hohe Korrelation zu den Ausgängen, geringe Korrelation untereinander.
- Gleichmäßige Häufigkeitsverteilung.

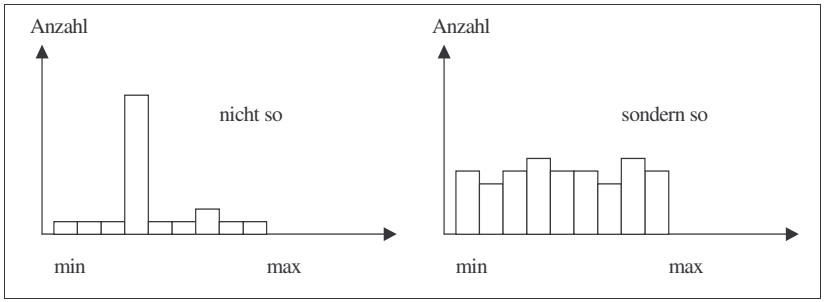



**Abb. 37:** Häufigkeitsverteilungen (Histogramme).

**Auswahl von Ausgangsparametern**

Gute Ausgangsparameter sollten folgende Eigenschaften aufweisen:

- Nur die problemrelevanten, d.h. möglichst wenige.
- Gleichmäßige Häufigkeitsverteilung, ggf. Transformieren. In vielen Fällen lässt sich eine ungleichmäßige Verteilung eines Parameters durch Anwendung einer nichtlinearen Skalierungstransformation (z.B. logarithmische Transformation) in eine gleichmäßigere Verteilung überführen.

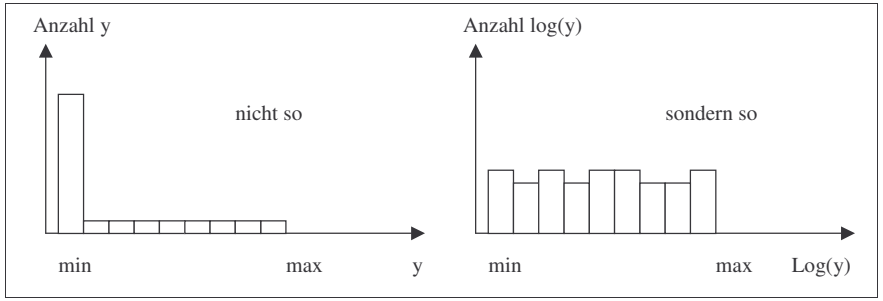



**Abb. 38:** Logarithmische Transformation bei Konzentration bei kleinen Werten.





### 4.1.3 Behandlung unvollständiger Datensätze

1. **Eingangsparameter unvollständig**
   Datensätze mit unvollständigen Inputs sind unbrauchbar. In manchen Fällen wird versucht, den Input zu interpolierenden. Dabei wird aber entweder keine neue Information bereitgestellt *(„Interpolation passt“)* oder es werden sogar Fehlinformationen in das Modell eingetragen. Aus diesem Grund sind Datensätze mit unvollständigen Inputs aus der Lerndatenmenge zu entfernen.

2. **Ausgangsparameter unvollständig**
   In diesem Fall kann für jeden Ausgang einzeln ein NN trainiert werden, so dass die Datensätze jeweils in allen Inputs und dem betrachteten Output vollständig sind. Das hat darüber hinaus den Vorteil, dass die Netzstruktur auch für jeden Output einzeln optimiert werden kann und nicht nur eine Lösung gefunden wird, die im Mittel über alle Outputs gut ist (die kann dann nämlich für einen einzelnen Output auch relativ schlecht sein). Die jeweiligen Teilnetze für die einzelnen Outputs lassen sich dann wie folgt zu einem Gesamtnetz zusammenfügen:

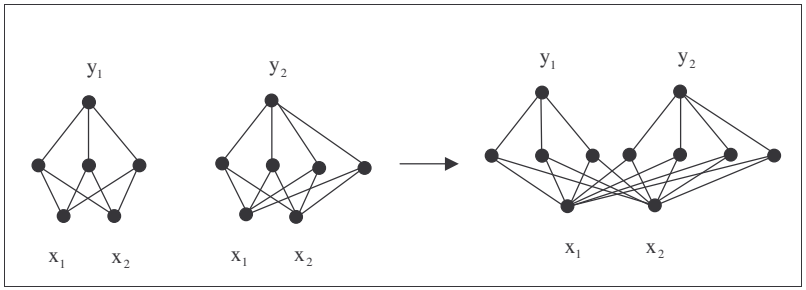



**Abb. 39:** Vereinigung zweier Teilnetze zu einem Gesamtnetz.

Die dargestellte Methodik wird von der im Praktikum eingesetzten Software NN-Tool automatisch ausgeführt.

### 4.1.4 Behandlung von Klassifikatoren

Klassifikatoren sind nichtnumerische Parameter, die einen bestimmten Wert (Klasse, Instanz) aus einer vorgegebenen Menge von Werten (Klassen) annehmen, z. B. Klassifikator „Farbe“ mit den Instanzen „rot“, „gelb“, „grün“, „blau“. Diese Werte müssen in genau so viele numerische Parameter umkodiert werden, wie es Klasseninstanzen gibt („Bayes’sche Kodierung“).

| Farbe | | rot | gelb | grün | blau |
|---|---|---|---|---|---|
| rot | | 1 | 0 | 0 | 0 |
| gelb | -> | 0 | 1 | 0 | 0 |
| grün | | 0 | 0 | 1 | 0 |
| blau | | 0 | 0 | 0 | 1 |

**Abb. 40:** Umkodierung eines Klassifikators

Die dargestellte Methodik wird von der Software NN-Tool automatisch ausgeführt.





## 4.2 Zeitreihen

**Ziel:** Modellierung des dynamischen, d.h. zeitabhängigen Verhaltens eines Prozesses (z.B. Aktienkurs, Chemiereaktor, Industrieroboter,...).

**Häufig zitierte, jedoch falsche Aussage:** *„Neuronale Netze können keine dynamischen Systeme modellieren, da sie nur Funktionen approximieren können. Die Dynamik von Prozessen wird jedoch durch Differenzialgleichungen beschrieben.“*

Die Aussage ist völlig falsch. **Das Verhalten dynamischer Systeme wird durch zeitabhängige Funktionen beschrieben** und kann daher problemlos mittels neuronaler Netze modelliert werden. Die genannten Differenzialgleichungen (samt der zugehörigen Anfangsbedingungen) dienen ja nur dazu, aus der unendlichen Vielfalt von Funktionen gerade die richtigen auszuwählen. In diesem Sinne gehen Lernalgorithmen ähnlich vor. Sie benutzen die Messdaten um aus der Vielfalt der möglichen Funktionen die richtigen auszuwählen.

**Voraussetzung / Annahme:**

- Der zu modellierende Prozess kann prinzipiell durch ein System gewöhnlicher Differenzialgleichungen beschrieben werden. Ggf. kommen noch Totzeiten hinzu.

**Bemerkung:**

- Die Differenzialgleichungen sind selbstverständlich nicht bekannt, sonst könnte der Prozess ja unmittelbar mit den DGLs modelliert werden. Wir setzen nur voraus, dass es ein solches System gibt. Dadurch folgt z.B., dass der betrachtete Prozess überhaupt eine eindeutig definierte Dynamik besitzt.
- Die Annahme - sie stellt die denkbar einfachste mathematische Formulierung dynamischer Systeme dar - kann weiter abgeschwächt werden (z.B. auf partielle DGLs) und gilt auch keineswegs für alle Anwendungen. Diese Fälle sollen hier jedoch nicht diskutiert werden.

Unter der gemachten Annahme gilt für die **Prozessdynamik:**

| | |
|---|---|
| $\dot{\underline{X}}(t) = f\left(\underline{X}(t), \underline{U}(t)\right)$ | f ist nicht bekannt! |
| $\underline{X} = (x_1, \ldots x_N)$ | Systemzustand (Zustandsvektor) |
| $\underline{U} = (u_1, \ldots u_m)$ | Steuervektor (Stellgrößen, Einflussmöglichkeiten) |

Die **Dimension N des Zustandsvektors** ist unter Umständen auch nicht bekannt (z.B. bei der Aktienkursprognose). Bei technischen Anwendungen steht sie jedoch in vielen Fällen fest.

**Beispiel:**
Bei einer Autopilotanwendung für ein Flugzeug enthält der Zustandsvektor die Position, Geschwindigkeit und Orientierung (hoffentlich Nase voraus, Kabinenboden unten) des Flugzeugs. Der Steuervektor enthält die Stellung aller Ruder (Quer-, Seiten-, Höhenruder), den Schub der Triebwerke sowie die Stellung der Klappen (z.B. Landeklappen), d.h. der Steuervektor umfasst sämtliche Möglichkeiten den Prozess zu beeinflussen. Alle genannten Größen sind Funktionen der Zeit. **Die DGL sagt dann nichts anderes, als dass die Änderung des Zustands und damit die zukünftige Entwicklung vom aktuellen Zustand und den aktuellen Stellgrößen abhängen.**

**Zeitdiskretisierung:**
Wir setzen voraus, dass das System nur zu diskreten Zeittakten $t_n$ abgefragt wird (die Physik im Hintergrund soll aber zeitkontinuierlich bleiben). Dies ist bei technischen Systemen fast immer der Fall. Beispiel: der Autopilot im Flugzeug fragt alle 10ms die Messfühler ab.
Damit definieren wir den Systemzustand zum Zeitpunkt $t_n$:

$$\underline{X}_n := \underline{X}(t_n)$$

**Aufgabe:**
Prognostiziere $\underline{X}_{n+1}$ auf der Basis von $\underline{X}_n$ und dem Verlauf von $\underline{U}$ im Intervall $[t_n, t_{n+1})$.

$\underline{X}_{n+1}$ ist als Lösung der Differenzialgleichung eindeutig durch $\underline{X}_n$ und dem Verlauf von $\underline{U}$ bestimmt. An dieser Stelle machen wir also Gebrauch von unserer Voraussetzung, dass es im Hintergrund eine DGL gibt, die wir nur nicht kennen. Falls $\underline{U}$ im Intervall zwischen zwei Takten konstant ist, d.h. falls $\underline{U}(t) = \underline{U}(t_n) := \underline{U}_n$ im Intervall $[t_n, t_{n+1})$ gilt, lässt sich die Lösung wie folgt modellieren:

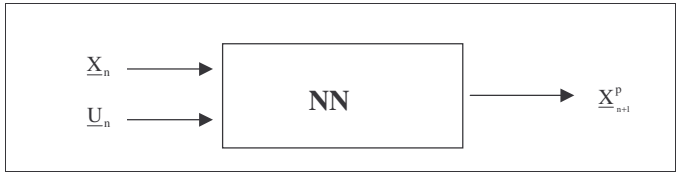



**Abb. 41:** Prognose um einen Zeittakt in die Zukunft.





Die Zeitreihe (Dynamik) wird also auf eine Funktion zurückgeführt. Man beachte, dass auf der rechten Seite der prognostizierte Wert $\underline{X}^p_{n+1}$ anstelle von $\underline{X}_{n+1}$ steht. Die beiden Werte weichen in der Regel durch den Modellfehler (aber auch durch Messfehler) von einander ab.

Falls sich $\underline{U}$ im Intervall $[t_n, t_{n+1})$ ändert, müssen ggf. zusätzliche Eingänge für $\underline{U}(t)$ an Zwischenwerten $(t_{n_0} = t_n, t_{n_1} = t_n + \Delta t, t_{n_2} = t_n + 2\Delta t, ...)$ eingeführt werden. Im Folgenden wird dieser Fall nicht weiter diskutiert, da er keine grundsätzlichen technischen Schwierigkeiten bereitet, aber den Schreibaufwand unnötig erhöht.

Das Netz liefert also eine Prognose $\underline{X}^p_{n+1}$ für den Systemzustand $\underline{X}_{n+1}$. Mit dieser Prognose *könnte* dann $\underline{X}^p_{n+2}$ berechnet werden:

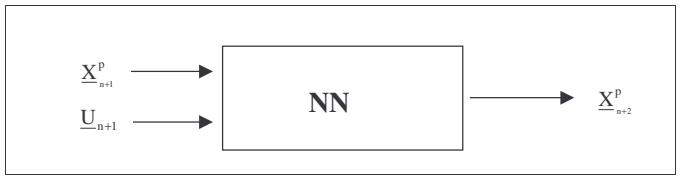



**Abb. 42:** Prognose um einen weiteren Zeittakt durch rekursives Einsetzen.

**Verfahren ist jedoch im Allgemeinen instabil!**
Bedingt durch die Modellfehler laufen die Prognosen $\underline{X}^p_{n+k}$ und die gemessenen Werte $\underline{X}_{n+k}$ mit zunehmenden k immer weiter auseinander (dies gilt im Übrigen auch für die Lösung von Differenzialgleichungen). Besser ist es, abzuwarten bis $\underline{X}_{n+1}$ verfügbar ist und dann:

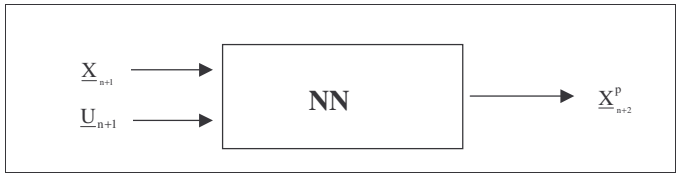



**Abb. 43:** Stabile Prognose um einen weiteren Zeittakt.

In manchen Anwendungen benötigt man jedoch bereits zum Zeitpunkt $t_n$ eine Prognose für $\underline{X}_{n+2}$ (oder noch weiter in der Zukunft liegende Werte). In diesem Fall bietet sich die folgende Modellierung an:

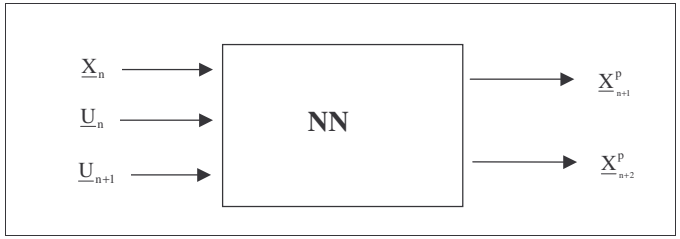



**Abb. 44:** Stabile Prognose um zwei Zeittakte.

Im Gegensatz zur Modellierung durch rekursives Einsetzen (nach Abb. 42) liefert diese Methode eine Prognose für $\underline{X}_{n+2}$, bei welcher der Modellfehler nicht anwächst. Im Allgemeinen ist der Modellfehler für $\underline{X}_{n+2}$ bei dieser Methode vergleichbar mit dem Fehler für $\underline{X}_{n+1}$. Entsprechendes gilt für Prognosen über weitere Schritte.

**Häufige Situation in der praktischen Anwendung:**

System wird durch N-dimensionalen Zustandsvektor $\underline{X}$ beschrieben. Es wird jedoch nur eine Komponente y (z.B. $y = x_1$) gemessen. Analoge Betrachtungen gelten auch für den Fall, dass nur eine Teilmenge der Komponenten des Zustandsvektors gemessen wird. Wir beschränken uns aus Gründen der Übersichtlichkeit auf den Fall einer gemessen Komponente. In diesem Fall ist der Steuervektor $\underline{U}$ dann häufig auch nur eindimensional.

Da y nur eine Komponente des Zustandsvektors umfasst, ist die zukünftige Entwicklung von y natürlich nicht nur durch den Momentanwert von y und die Steuergröße u bestimmt. Um eine Modellierung zu ermöglichen, müssen hinreichend viele Werte von y aus der Vergangenheit mitgenommen werden.

**Ansatz:**

$$y_{n+1} = f\left(y_n, y_{n-1}, y_{n-2}, ..., y_{n-k}, u_n\right) \qquad \text{mit } k = N-1$$

Ein solcher Ansatz wird in der Literatur auch als „NARMAX“ bzw. „Nichtlinearer ARMAX-Ansatz“ bezeichnet.





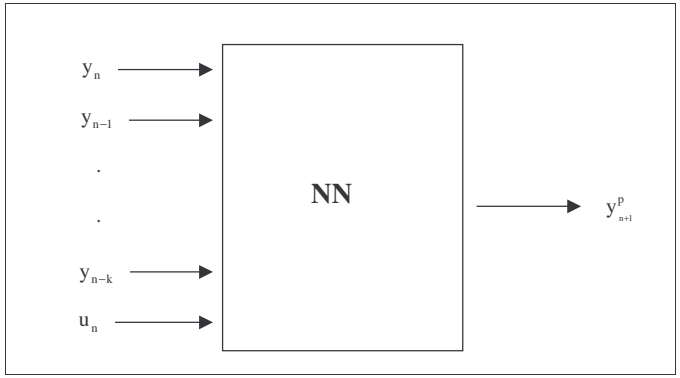
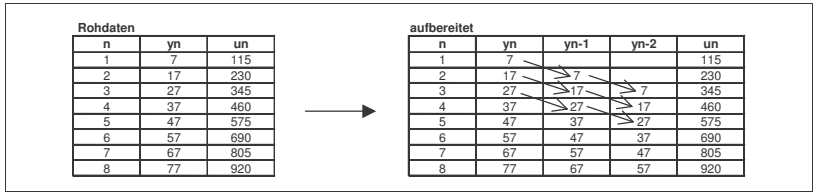



**Abb. 45:** Neuronales Netz als NARMAX-Modell

Zum Netztraining müssen die Datensätze entsprechend aufbereitet werden:



Rohdaten

| n | yn | un |
|---|---|---|
| 1 | 7 | 115 |
| 2 | 17 | 230 |
| 3 | 27 | 345 |
| 4 | 37 | 460 |
| 5 | 47 | 575 |
| 6 | 57 | 690 |
| 7 | 67 | 805 |
| 8 | 77 | 920 |

aufbereitet

| n | yn | yn-1 | yn-2 | un |
|---|---|---|---|---|
| 1 | 7 | | | 115 |
| 2 | 17 | 7 | | 230 |
| 3 | 27 | 17 | 7 | 345 |
| 4 | 37 | 27 | 17 | 460 |
| 5 | 47 | 37 | 27 | 575 |
| 6 | 57 | 47 | 37 | 690 |
| 7 | 67 | 57 | 47 | 805 |
| 8 | 77 | 67 | 57 | 920 |

**Abb. 46:** Datenaufbereitung zum Training von Netzen für Zeitreihenprognosen.

**Totzeiten:**

Bei bestimmten Prozessen können noch zusätzliche Totzeiten auftreten. In vielen technischen Anwendungen hängt z.B. die Qualitätsgröße (Output) zum Zeitpunkt t von den Prozessbedingungen zu einer früheren Zeit ab:

$$Q(t) = f\left(\underline{X}(t - T_t), \underline{U}(t - T_t)\right) \qquad T_t = \text{Totzeit}$$

**Beispiel (siehe 4.1.2):**

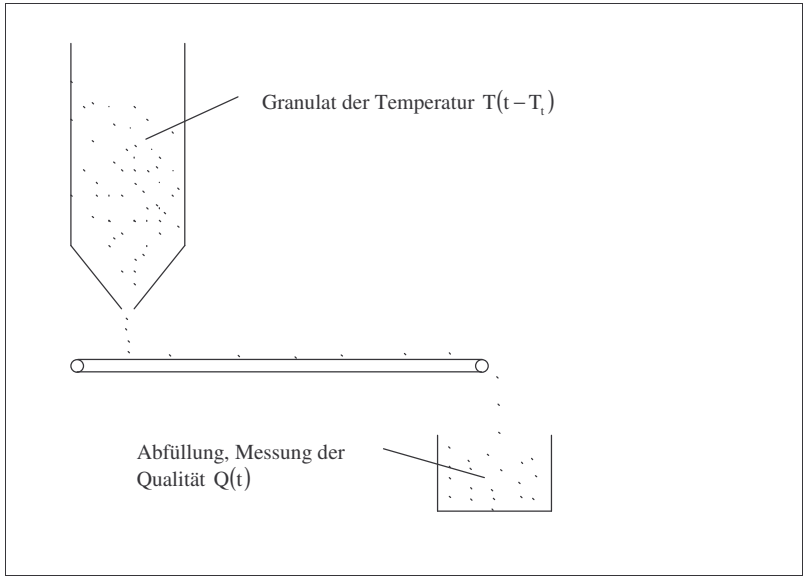



**Abb. 47:** Totzeitbehafteter Prozess.

Im Beispiel hängt die Qualität, die erst bei Abfüllung gemessen wird, von den Prozessparametern (insbesondere der Temperatur) ab, die in der Anlage herrschten, als die Granulatkörnchen durch die reaktive Zone liefen. Bei diesem Prozess liegt die Totzeit im Bereich von 5 Stunden.

Eine Totzeit kann konstant sein. Dann entspricht die **Datenaufbereitung** der oben in Abb. 46 bereits beschriebenen Vorgehensweise. Eine Totzeit kann sich jedoch auch dynamisch ändern (**„dynamische Totzeiten“**). Z.B. kann ein Förderband unterschiedlich schnell laufen. Zum Netztraining müssen dann die zusammenpassenden Daten für die Ein- und Ausgänge in Abhängigkeit der Drehzahl zusammengesucht werden.





## 4.3 Kriterien zur Beurteilung der Prognosegüte

Wenn wir die Ein- und Ausgangsparameter richtig aufbereitet haben und damit dann ein Neuronales Netz trainieren wollen, benötigen wir Kriterien dafür, wie gut die Modellierung beim konkreten Problem eigentlich funktioniert. Die **Modellgüte** hängt ja bekanntlich auch noch davon ab, über wie viele innere Knoten ein Netz verfügt oder wie viele Lernschritte durchgeführt wurden. Man möchte also das bezüglich der Modellgüte (sprich Modellgenauigkeit) **optimale Netz** finden. Darüber hinaus muss man sich klarmachen, dass das beste Netz (bzgl. der Modellgüte) im Allgemeinen keineswegs dasjenige ist, das die Lerndaten exakt wiedergibt. In der Regel weisen die Lerndaten **Messfehler** auf. Ein exaktes Lernen der Daten würde also bedeuten, dass auch alle Messfehler auf den Daten mitgelernt wurden. Wünschenswert ist dagegen, dass ein Netz die durch die Daten repräsentierten funktionalen Beziehungen *(„die Physik hinter den Daten“)* lernt und für neue Situationen, d.h. Situationen, die nicht in den Lerndaten repräsentiert sind, möglichst gute Prognosen abgibt. Wenn es nur darum ginge, die Lerndaten exakt wiederzugeben, könnten wir sie gleich in einer Datenbank abspeichern. Aus den genannten Gründen ist es notwendig, die Gesamtmenge der zur Verfügung stehenden Datensätze in drei disjunkte Teilmengen aufzuteilen:

- **Lernset** (Trainingsdaten, Lerndaten): Datensätze werden zur Einstellung der Netzgewichte verwendet.
- **Testset (Testdaten):** Datensätze werden nicht zum Training, aber zur Auswahl der optimalen Netzstruktur verwendet.
- **Validationset:** Datensätze werden weder zum Training noch zur Auswahl der Netzstruktur herangezogen. Sie dienen der Abschätzung der Prognosegüte auf unbekannten Datensätzen.

**Bemerkungen:**

- Die Bezeichnungen Testset und Validationset werden in der Literatur manchmal auch umgekehrt verwendet.
- In vielen Anwendungen wird auf Grund der geringen Gesamtmenge an verfügbaren Datensätzen auf den Validationset verzichtet und der Testset zur Abschätzung der Modellgüte herangezogen, was eigentlich statistisch nicht ganz sauber ist.

Nach der Aufteilung in die Sets, wird dann in der Regel nicht ein einzelnes NN trainiert, sondern es müssen **verschiedene Kombinationen** bezüglich der Zahl der inneren Knoten (Hiddenlayerknoten) sowie der Zahl der Lernschritte (Iterationsschritte) durchgeführt

werden. Dabei ergibt sich schematisch folgendes Bild:

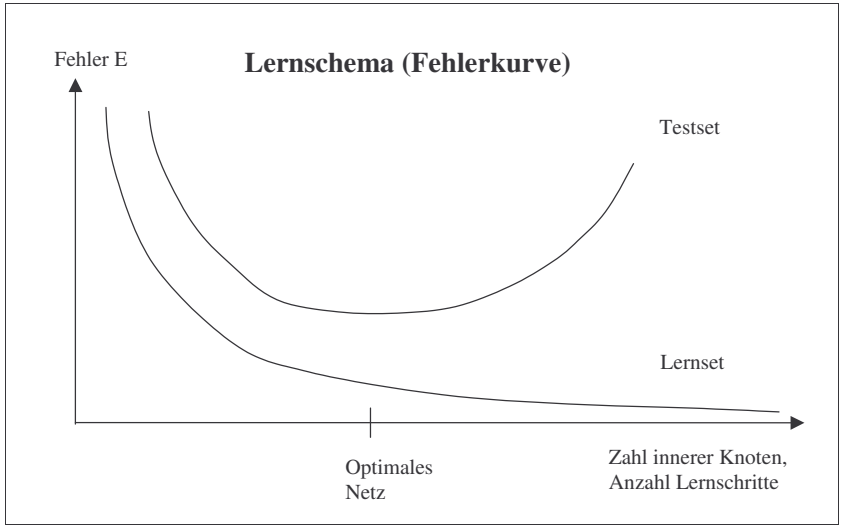



**Abb. 48:** Fehlerverlauf auf Lern- und Testset in Abhängigkeit von der Zahl innerer Knoten und der Zahl der Lernschritte (eigentlich ein 3D-Diagramm).

Wie man sieht fällt der Fehler, d.h. die Abweichung zwischen gemessenen und prognostizierten Werten (genaue Definition folgt noch) auf dem Lernset monoton gegen Null. Falls ein Netz über genau so viele innere Knoten verfügt, wie es Datensätze in der Lernmenge gibt, kann die Lernmenge exakt gelernt werden (d.h. Fehler gleich Null). Auf dem Testset fällt der Fehler zunächst ebenso, erreicht dann ein Minimum und steigt danach wieder an, weil das Netz nun beginnt, die Lerndaten mit allen ihren Messfehlern auswendig zu lernen (so genanntes **„Overfitting“**). Durch den Overfittingeffekt steigt der Fehler (fällt die Modellgüte) auf Datensätzen, die nicht zur Einstellung der Netzgewichte verwendet worden sind. Man sagt auch, das Netz kann dann weniger gut verallgemeinern, die so genannte **„Generalisierungsfähigkeit“** ist schlecht. Das kann natürlich nicht in unserem Sinne sein. Wir suchen also das Netz, das auf den Testdaten den geringsten Fehler (**„optimales Netz“**) aufweist. Dieser Fehler hängt also mindestens von den folgenden beiden Parametern ab:

- **Zahl der inneren Knoten** (Hiddenlayerknoten)
- **Zahl der Lernschritte** (Iterationsschritte)





In der Praxis hängt er in vielen Fällen auch noch von der genauen Einteilung der verfügbaren Datensätze auf Lern-, Test- und Validationset ab.

Zur **Gütebewertung** benötigen wir also **Fehlermaße**. Die folgenden Maße werden in der Praxis am häufigsten verwendet:

**A) Mittlerer quadratischer Fehler des i-ten Outputs:**

$$E_i = \frac{1}{p}\sum_{j=1}^{p}\left(y_{ji} - y_{ji}^{p}\right)^2 \quad \rightarrow \quad 0$$

p = Anzahl der Datensätze des betreffenden Sets

**B) Mittlerer quadratischer Gesamtfehler:**

$$E = \sum_{i=1}^{r} E_i = \frac{1}{p}\sum_{i=1}^{r}\sum_{j=1}^{p}\left(y_{ji} - y_{ji}^{p}\right)^2 \quad \rightarrow \quad 0$$

**C) Mittlerer absoluter Fehler des i-ten Outputs:**

$$E_i = \frac{1}{p}\sum_{j=1}^{p}\left|y_{ji} - y_{ji}^{p}\right| \quad \rightarrow \quad 0$$

**D) Mittlerer absoluter Fehler des i-ten Outputs in % der Skalenlänge (relativer Fehler in %):**

$$E_i = 100\% \frac{\frac{1}{p}\sum_{j=1}^{p}\left|y_{ji} - y_{ji}^{p}\right|}{\max_j\left(y_{ji}\right) - \min_j\left(y_{ji}\right)} \quad \rightarrow \quad 0$$

**E) Korrelationskoeffizient des i-ten Outputs**

$$R_i = \frac{\sum_{j=1}^{p}\left(y_{ji} - \bar{y}_i\right)\left(y_{ji}^{p} - \bar{y}_i\right)}{\sqrt{\sum_{j=1}^{p}\left(y_{ji} - \bar{y}_i\right)^2 \sum_{j=1}^{p}\left(y_{ji}^{p} - \bar{y}_i\right)^2}} \quad \rightarrow \quad 1$$

und viele weitere Fehlermaße.
Die Wahl des richtigen Fehlermaßes ist Thema nicht endender Diskussionen. NN-Tool benutzt u. A. die Fehlermaße D und E. Prinzipiell gilt:

**Fehlermaße sind statistische Größen!**
$\Rightarrow$ Testset und Validationset sollten nicht zu klein sein um statistisch signifikante Aussagen

zu ermöglichen. Andererseits sollte die Lernmenge (der Lernset) möglichst groß sein um überhaupt gute Modelle zu ermöglichen.

**Praxis:**
**Daten sind teuer (selten)!**

⇒ Übliche Vorgehensweise: Kein Validationset, 80% der Daten als Lernset, 20% Testset.

⇒ Ggf. Verwendung von Spezialmethoden („NN-Tool"), insbesondere „Crossvalidation" mit dynamischer Aufteilung von Lern- und Testset.

**Graphische Beurteilung der Modellgüte:**
Neben der Beurteilung der Netzgüte mittels Fehlermaßen, sind auch graphische Darstellungen zur Beurteilung von Bedeutung. Die wichtigste derartige Darstellung ist der so genannte **Scatterplot:**

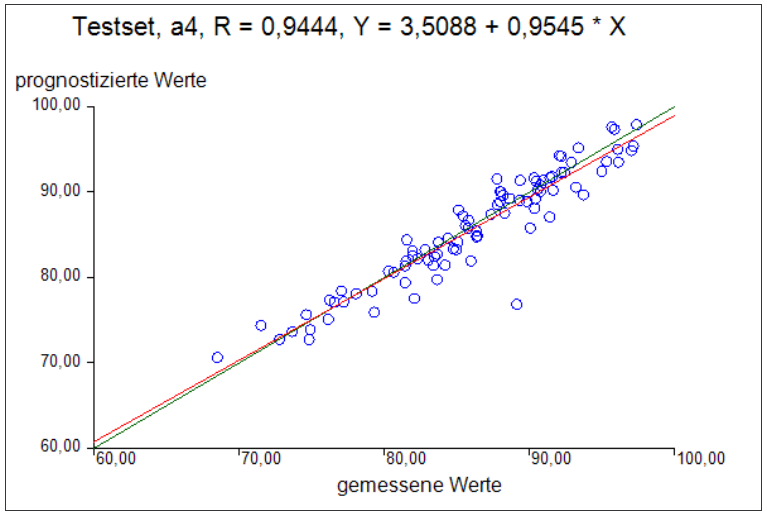



**Abb. 49:** Scatterplot für den Ausgangsparameter „a4", ermittelt auf den Daten des Testset.

Für jeden Datensatz des Testsets ist hier der berechnete (prognostizierte) gegen den vorgegebenen (gemessenen) Wert aufgetragen. Darüber hinaus ist die **"Winkelhalbierende"** (blaue Linie) sowie die **Ausgleichsgerade** (rote Linie) durch die Punktewolke eingetragen.





Zusätzlich sind der **Korrelationskoeffizient** (hier 0,9444) sowie die Gleichung der Ausgleichsgeraden in der Form $y = a_0 + a_1 x$ angegeben. Der a1-Koeffizient (hier 0,9545) muss für ein gutes Modell in der Nähe von 1 liegen. In diesem Sinne ist der **a1-Koeffizeint ebenfalls ein Gütemaß**. Dagegen muss der a0-Koeffizient nur klein in Bezug auf die typischen Werte eines Ausgangsparameters sein. Insofern ist hier ein Wert von 3,5088 auch noch als unkritisch anzusehen. Idealerweise sollten alle Punkte auf der „Winkelhalbierenden" (d.h. auf der Geraden y = x) liegen und die Ausgleichsgerade sollte mit der Winkelhalbierenden übereinstimmen. Auf diese Weise ermöglicht diese Darstellung eine **schnelle visuelle Kontrolle der Güte des Netzes**, die bei mehr als 2 Eingangsgrößen durch keine andere graphische Darstellung vergleichbar leicht möglich ist.

Neben dem Scatterplot steht zur Beurteilung der Modellgenauigkeit auch noch der **Verlaufsplot** zur Verfügung:

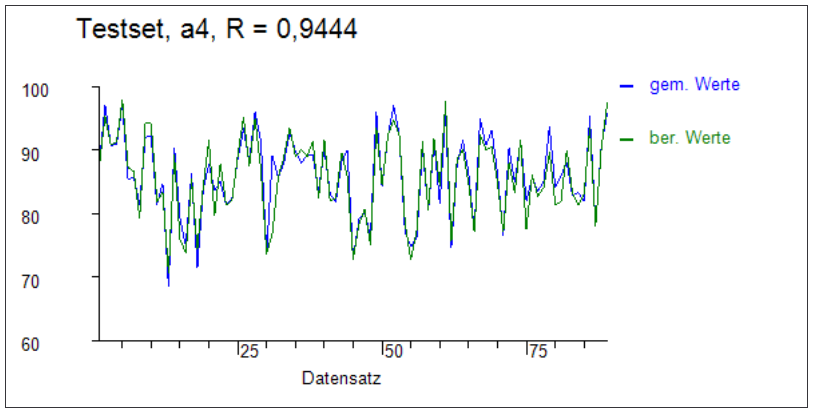



**Abb. 50:** Verlaufsplot für den Ausgangsparameter „a4", ermittelt auf den Daten des Testset.

Bei dieser Darstellung werden für die einzelnen Datensätze des Testsets (bzw. Lernsets) der prognostizierte und der gemessene Wert über der Nummer des Datensatzes aufgetragen. Diese Darstellung wird insbesondere für Zeitreihen benutzt. Im Allgemeinen ist jedoch der Scatterplot die aussagekräftigere Darstellung.

Für Ausgangsparameter vom Klassifikatortyp hat sich die so genannte **Confusionmatrix**-Darstellung durchgesetzt.

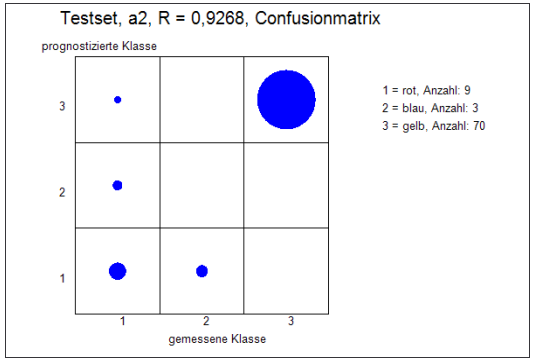



**Abb. 51:** Confusionmatrix für den Ausgangsparameter „a2“, ermittelt auf den Daten des Testset.

Die Größe (genauer die Fläche) der farbigen Kreise entspricht der Anzahl der Datensätze des jeweiligen Sets mit den entsprechenden gemessenen und prognostizierten Klassen. Idealerweise sollte die Confusionmatrix natürlich wie folgt aussehen:

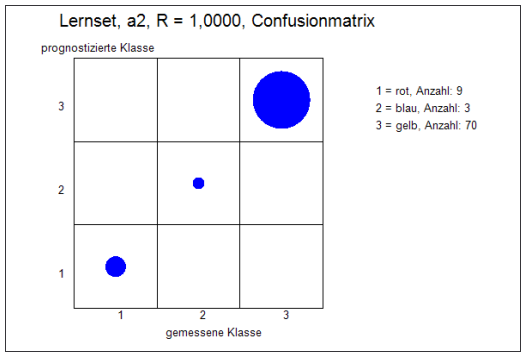



**Abb. 52:** Confusionmatrix für den Ausgangsparameter „a2“, ermittelt auf den Daten des Lernset.





## 4.4 Optimierung der Netzstruktur

Nachdem wir im vorangegangenen Kapitel Kriterien für die Modellgüte definiert haben, können wir nun die Aufgabe angehen, eine in den genannten Fehlermaßen optimale Netzstruktur zu finden. Im Allgemeinen ist das optimale Netz bezüglich des quadratischen Fehlers nicht unbedingt das beste Netz bezüglich des entsprechenden absoluten Fehlers („Betragsfehler“). Z.B. werden durch das Quadrieren einzelne Datensätze mit großem Fehler („**Ausreißer**“) stärker gewichtet als beim absoluten Fehler. Da es sich bei diesen aber häufig um Messfehler handelt, ist dieser Effekt meistens unerwünscht. Aus diesem Grund ist der absolute Fehler vorzuziehen. In der Regel ist ein Netz, das bzgl. eines der genannten Fehlermaße gut ist, jedoch auch in allen anderen Fehlermaßen gut.

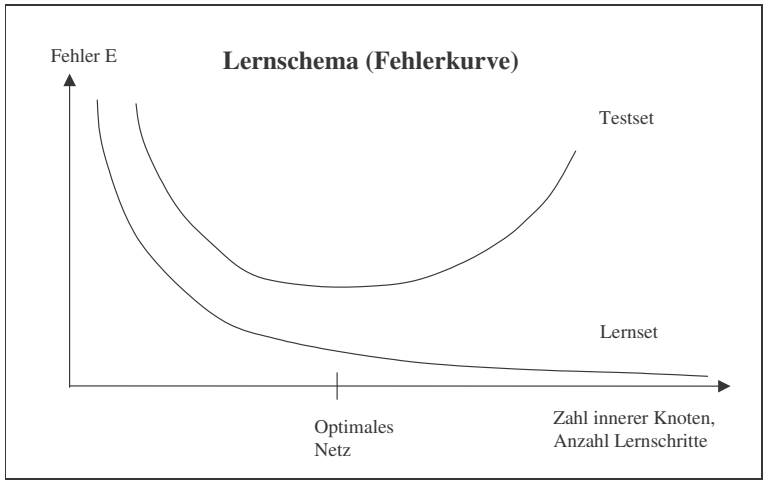



**Abb. 53:** Fehlerkurve

Wir suchen jetzt also Verfahren um dem optimalen Netz der Fehlerkurve, das durch die richtige Anzahl innerer Knoten sowie der entsprechenden Anzahl von Lernschritten gekennzeichnet ist, möglichst nahe zu kommen, bzw. diese beiden Parameter genau zu bestimmen. Dabei hat man dann noch die Wahl zwischen der so genannten **Gesamtfehleroptimierung** und der **Einzeloptimierung**. Bei der Gesamtfehleroptimierung wird ein Netz für alle zu modellierenden Ausgangsgrößen optimiert. Bei der Einzeloptimierung wird für jede Ausgangsgröße einzeln die optimale Netzstruktur bestimmt. Die entsprechenden Teilnetze werden zuletzt zu einem Gesamtnetz vereinigt (vgl. 4.1.3). Dies

führt erstens im Allgemeinen zu besseren Modellen und hat darüber hinaus den Vorzug, dass bei unvollständigen Datensätzen die Information besser ausgenutzt werden kann. Diese Methode ist allerdings auch entsprechend rechenintensiver. Wir diskutieren zunächst die Gesamtfehleroptimierung. Die entsprechenden Methoden für die Einzelfehleroptimierung ergeben sich dann unmittelbar, in dem man die Verfahren auf die Teilnetze, die dann eben jeweils nur eine Outputgröße haben, anwendet.

**A. Gesamtfehleroptimierung**

Der quadratische Gesamtfehler bei r-vielen Outputs und p-vielen Datensätzen im betreffenden Set ist definiert durch:

$$E = \sum_{i=1}^{r} E_i = \frac{1}{p} \sum_{i=1}^{r} \sum_{j=1}^{p} \left( y_{ji} - y_{ji}^{p} \right)^2$$ Gesamtfehler über alle Muster j und alle Outputs i

Der Fehler E ist eine Funktion der Zahl innerer Knoten n und der Zahl der Iterationsschritte (Lernschritte) $I_t$ :

$$E = E(n, I_t)$$

Dieser Zusammenhang gilt auch für alle anderen üblichen Fehlermaße, d.h. die Überlegungen hängen nicht von der speziellen Wahl des Fehlermaßes ab. Benötigt wird eine Strategie zur Bestimmung der optimalen Werte $n^*$, $I_t^*$ bei denen E am geringsten ist, d.h. wir suchen:

$$\min_{n, I_t} E(n, I_t)$$

Übliche Strategien sind:

1. **Daumenregeln für n, $I_t$ durchprobieren:**

   Bei p vielen Datensätzen, m Eingangs- und r Ausgangsgrößen gelten folgende Daumenregeln für n:

   R1) n sollte proportional zu r wachsen und $n \geq r$ .
   R2) $n < p$ Mussbedingung, besser ist $n < \frac{1}{3} p$ oder $n < \frac{1}{5} p$ .
   R3) $n \cdot (m+1) < p$
   R4) n muss bei komplexerem Verhalten erhöht werden.

   Zu einem gegebenen n wird $I_t$ als dann eindimensionales Minimum der Fehlerkurve bestimmt, d.h. man hört auf, wenn der Fehler auf dem Testset wieder signifikant ansteigt.





**Beispiel:** p = 400 Datensätze, m = 6 Eingangsparameter, r = 2 Ausgangsparameter.

R1) liefert $n \geq 2$ .
R2) $n < 133$ .
R3) $n < \frac{400}{7} \Rightarrow n < 57$

d.h. optimales Netz zwischen 2 und 56 inneren Knoten.

R4) Softwareprodukt Neuromodel: „Halten Sie die Modellierungsaufgabe für komplex?“

**Strategie:** Eher an unterer Grenze anfangen, z.B. n= 6.

2. **Wachsende Netze**

   Während des Lernvorgangs werden nach gewissen Kriterien innere Knoten eingefügt. Start mit kleinem Netz.

3. **Pruning-Verfahren**

   Start mit großem Netz (d.h. viele innere Knoten). Während des Lernvorgangs werden „unwichtige“ Knoten eliminiert.

4. **Rasterverfahren**

   Kombinationen für n und $I_t$ (alle) durchprobieren („NN-Tool“). Liefert bestmögliche Ergebnisse. Benötigt maximale Rechenzeit.

**B. Einzeloptimierung**

Für die verschiedenen Ausgänge wird jeweils einzeln ein optimales Netz erstellt, d.h. es wird nacheinander

$$E_i = \frac{1}{p} \sum_{j=1}^{p} \left( y_{ji} - y_{ji}^{p} \right)^2$$

minimiert. NN-Tool kombiniert diese Methode mit dem Rasterverfahren zur Bestimmung der jeweils optimalen Werte für n und $I_t$ **(„Automatische Netzstrukturoptimierung“).**

## Literatur


- Anderson, J. A.; Rosenfeld, E.: **Neurocomputing - Foundations of Research.** Cambridge: MIT Press, 1989

- Brause, Rüdiger: **Neuronale Netze. Eine Einführung in die Neuroinformatik.** Stuttgart: Teubner, 1999

- Cichocki, A.; Unbehauen, R.: **Neural Networks for Optimization and Signal Processing.** Stuttgart/Chichester: Teubner/ John Wiley & Sons, 1993

- S. Haykin
  **Neural Networks**
  A Comprehensive Foundation
  Prentice-Hall, Upper Saddle River, NJ, USA 1994

- D. Nauck, C. Borgelt, F. Klawonn und R. Kruse: **Neuro-Fuzzy-Systeme**
  Von den Grundlagen Neuronaler Netze zu modernen Fuzzy-Systemen
  Vieweg, Braunschweig/Wiesbaden 2003 (3. Auflage)

- Ritter, H.; Martinez, T.; Schwan, K.: **Neuronale Netze.** Addison-Wesley, 1991

- R. Rojas
  **Theorie der neuronalen Netze: Eine systematische Einführung**
  Springer, Berlin 1993

- Rumelhardt, D. E.; McClelland, J. L.: **Parallel Distributed Processing.** Cambridge: MIT Press, 1986 (zwei Bände plus Lernprogramm)

- Schöneburg, E.; Hansen, N.; Gawelczyk, A.: **Neuronale Netzwerke**. München: Markt und Technik Verlag, 1990

- A. Zell
  **Simulation neuronaler Netze**
  Addison-Wesley, Bonn 1994




Fachhochschule Gelsenkirchen
Fachbereich Physikalische Technik
- Bärmann -

- Link zu einer Bücherübersicht von Andreas Stuhlmüller:
  www.aiplayground.org/artikel/neuronale-netze-buecher/

# Anhang: Praktikumsaufgaben:
# Übung 01: Lineare Regression

## 1a) Polynomapproximation mit dem Excel-Solver

Man bestimme die Koeffizienten $a_0$,..., $a_7$ von Polynomen von 2., 3., 4., 5., 6. und 7. Grad, die durch die folgenden Punkte optimal hindurchgehen:

| x | y |
|---|---|
| -5 | -10 |
| -4 | 3 |
| -3 | 2 |
| -2 | -4 |
| -1 | 3 |
| 0 | 3 |
| 1 | 2 |
| 2 | 5 |
| 3 | 7 |
| 4 | 6 |
| 5 | 3 |

Hinweis: Man berechne zunächst bei vorgegebenen Werten der Koeffizienten in einer weiteren Spalte Schätzwerte $y_{pred}$ für y (pred steht für predicted = vorhergesagt), bilde dann die quadratischen Abweichungen und summiere diese auf. Die Summe ist dann mit dem Solver zu minimieren, dabei dürfen die Koeffizienten variiert werden:

| x | y | $y_{pred}$ | $(y-y_{pred})^2$ | Fehlersumme |
|---|---|---|---|---|
| -5 | -10 | -9,93912415 | 0,00370587 | 26,2081448 |
| -4 | 3 | 2,97862909 | 0,00045672 | |
| -3 | 2 | 0,95638974 | 1,08912238 | |
| -2 | -4 | -0,99095902 | 9,05432762 | |
| -1 | 3 | 0,16948134 | 8,01183589 | |
| 0 | 3 | 2,58823886 | 0,16954724 | |
| 1 | 2 | 4,1624683 | 4,67626916 | |
| 2 | 5 | 4,69433136 | 0,09343331 | |
| 3 | 7 | 5,57875987 | 2,01992351 | |
| 4 | 6 | 7,02060199 | 1,04162843 | |
| 5 | 3 | 2,78115151 | 0,04789466 | |



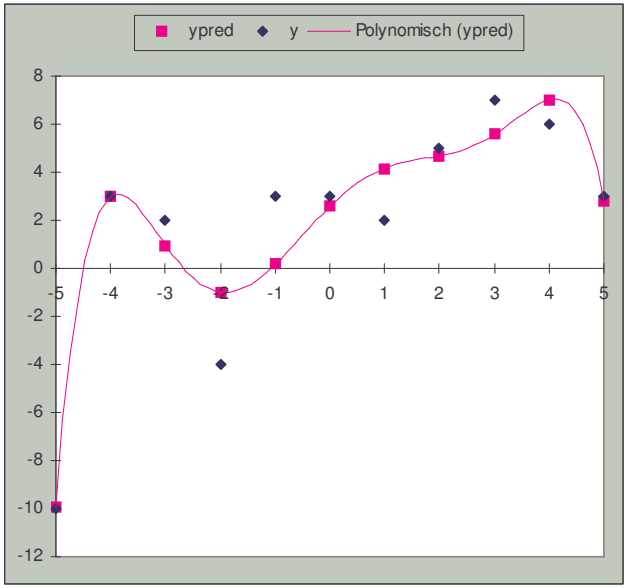


Die Polynomwerte sind dabei mittels des Hornerschemas zu bestimmen:

$$P(x) = a_0 + a_1 x + \ldots + a_7 x^7 = a_0 + x(a_1 + x(a_2 + x(\ldots + x(a_5 + x(a_6 + a_7 x)\ldots)$$

Man beginne mit dem Polynom 7. Grades, die niedrigeren Polynome können dann ermittelt werden, in dem die entsprechenden höheren Koeffizienten gleich 0 gesetzt und nicht mitoptimiert werden.

Die Ergebnisse sind grafisch auszuwerten. Dokumentation in Word. Schema:



### 1b) Modellierung eines physikalischen Zusammenhangs auf der Basis von Meßdaten

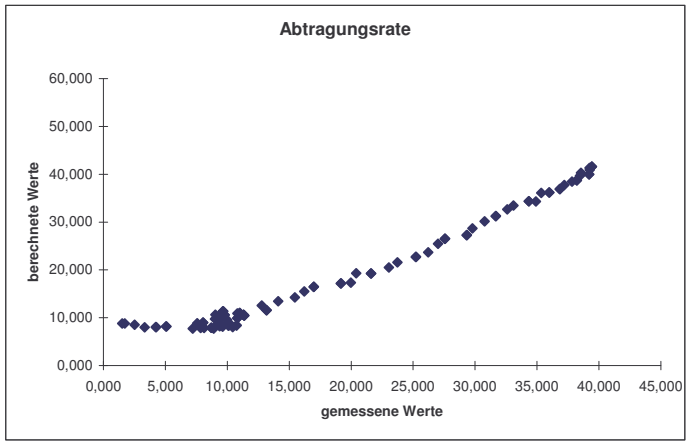
Ihnen stehen 100 Datensätze (`T:\Neuronale_Netze\Beispieldaten\Werkstoffdatensaetze.dat`) von Legierungen samt ihren Abtragungsraten in heißer Schwefelsäure zur Verfügung. Erstellen Sie ein Modell für den Zusammenhang zwischen Legierungszusammensetzung und Abtragungsrate. Benutzen Sie zunächst einen rein linearen Ansatz, d.h.:

$$Z = a_0 + a_1 \cdot Fe + a_2 \cdot Ni + ... + a_6 \cdot Mo$$

und berechnen Sie die Koeffizienten $a_0,..,a_6$ analog zur Polynomapproximation (vgl. Aufgabenteil 1a.).
Ergänzen Sie den Ansatz anschließend um ein quadratisches Glied in der Chromkonzentration d.h.:

$$Z = a_0 + a_1 \cdot Fe + a_2 \cdot Ni + ... + a_6 \cdot Mo + a_7 \cdot Cr^2$$

Stellen Sie die Ergebnisse jeweils grafisch in Form eines "Scatterplots" (=Streudiagramm) dar. Schema:




Fügen Sie anschließend die Ausgleichsgerade in das Diagramm ein und bestimmen Sie den Korrelationskoeffizienten als Wurzel aus dem Bestimmtheitsmaß.





# Übung 02: Perceptron und Feedforwardnetz

### 2a) Übertragungsfunktion neuronaler Netze

Definieren Sie die Sigmafunktion: $\sigma(x) = \frac{1}{1+e^{-x}}$ unter Excel-VBA:

1. Wechseln Sie in einer Excel-Mappe unter „Extras/Makro“ in den Visual Basic-Editor und fügen Sie ein Modul zum VBA-Projekt hinzu.
2. Definieren Sie die Funktion als VB-Function in diesem Modul.
3. Geben Sie die Funktion im Intervall -10 bis 10 als Excel-Diagramm aus (Schrittweite 0,2).
4. Berechnen Sie anschließend in einer Zelle den Wert $\sigma(-1000)$.
5. Korrigieren Sie die Funktion für große Argumentwerte.

### 2b) Perceptron

Konstruieren Sie in einer Excelmappe ein Perceptron mit der unter a) definierten Übertragungsfunktion für die Aufgaben „And“, „OR“ und „XOR“ (nacheinander, nicht parallel). Trainieren Sie das Perceptron mit dem Excel-Solver. Minimieren Sie dazu die quadratischen Abweichungen zwischen dem Output des Perceptrons und den durch die Funktionen vorgegebenen Werte.

### 2c) Feedforwardnetz

Konstruieren Sie in einer Excelmappe ein Feedforwardnetz mit 3 inneren Knoten mit der unter a) definierten Übertragungsfunktion für die Aufgaben „And“, „OR“ und „XOR“. Trainieren Sie das Feedforwardnetz mit dem Excel-Solver.
Hinweis: Das Feedforwardnetz neigt für diese Aufgaben dazu in lokale Minima zu laufen. Man muss ggf. öfter mit unterschiedlichen Anfangsgewichten optimieren. Insbesondere für XOR ist eine Lösung nur sehr schwer zu finden.

# Übung 03: Einführung NN-Tool

**3a) Einführung NN-Tool**

Arbeiten Sie die Kapitel 1. bis 4. im Handbuch zu NN-Tool durch (bis Seite 23). Kopieren Sie dazu die Beispielanwendung „TEST.xls“ im Verzeichnis „Beispieldaten“ in ein eigenes Verzeichnis.

**3b) NN-Tool, Klassifikatoren und Lernparameter**

Arbeiten Sie Kapitel 5. im Handbuch zu NN-Tool durch (Seiten 24 - 31). Kopieren Sie dazu die Beispielanwendung „class6.xls“ im Verzeichnis „Beispieldaten“ in ein eigenes Verzeichnis.

**3c) XOR**

Trainieren Sie die XOR-Aufgabe mit NN-Tool. Definieren Sie die Daten zunächst in einer Excel-Mappe. Wie umgehen Sie das Problem, dass NN-Tool einen Testset verlangt?
Beachten Sie beim Netztraining die Hinweise zum Thema „Gewichtsinitialisierung“ auf Seite 29 des Handbuchs.

**3d) Verständniskontrolle**

Beantworten Sie die folgenden Fragen bzw. erklären Sie die Begriffe (ohne Zuhilfenahme des Handbuchs):

- Datendatei, Parametername, Eingangsparameter, Ausgangsparameter, Kennung, aktive/passive Parameter, Eingangsknoten, Klassifikatoren
- Wann ist ein Datensatz vollständig?
- Was ist der Unterschied zwischen einem Eingangsparameter und einem Eingangsknoten?
- Lernset/Testset
- Anzahl innere Knoten, Modellierungsfähigkeit, Anzahl äußerer Iterationen
- Korrelation, rel. Fehler , Absoluter Fehler, Regressionskoeffizienten
- Scatterplot, Verlaufsplot
- Was sind die Steuerparameter für das Lernverfahren?
- Welche Möglichkeiten der Lernset/Testsetaufteilung gibt es?



Fachhochschule Gelsenkirchen
Fachbereich Physikalische Technik
- Bärmann -

# Übung 04: NN-Tool, Zeitreihen und spezielle Funktionen

**4a) NN-Tool, Zeitreihen**

Arbeiten Sie Kapitel 6. im Handbuch zu NN-Tool durch (Seiten 32-36). Kopieren Sie dazu die Beispielanwendung „Zeit.pat“ im Verzeichnis „Beispieldaten“ in ein eigenes Verzeichnis.

**4b) NN-Tool, Anwendungsmodule Mischpult und Optimierer**

Arbeiten Sie die Kapitel 7. und 8. im Handbuch zu NN-Tool durch (Seiten 37 - 50).

**4c) NN-Tool, spezielle Funktionen**

Schauen Sie sich im Handbuch den Anhang mit den Spezialfunktionen an. Lesen Sie insbesondere die folgenden Anhänge:

- Anhang 1, Versuchsplanung
- Anhang 2, automatische Dokumentation (*)
- Anhang 4, spezielle Datenanalysefunktionen
- Anhang 5, Ausreißerliste (*)
- Anhang 7, Nichtlineare Skalierungstransformationen
- Anhang 8, Input-Output-Zuordnung (*)
- Anhang 9, Crossvalidation (*)

Probieren Sie die mit (*) markierten Methoden am Testbeispiel „Class6“ aus.

**4d) Verständniskontrolle**

Beantworten Sie die folgenden Fragen bzw. erklären Sie die Begriffe (ohne Zuhilfenahme des Handbuchs):

- Zeitreihen, statische und dynamische Totzeiten, Datenunterbrechungen
- Komplexe Optimierung, Rezepturnebenbedingungen
- Wie funktioniert der Optimierer und welche Steuerparameter (numerische Parameter) gibt es?
- Versuchsplanung, automatische Dokumentation
- Gleichverteilung, Normalverteilung, Nichtlineare Skalierungstransformation
- Input-Output-Zuordnung
- Crossvalidation
- Welche Optionen gibt es für Crossvalidation?