

# Modellierung und Regelung einer mechanischen Presse mithilfe von Methoden des maschinellen Lernens

Tajinder Singh Dhaliwal

Master-Thesis 28. April 2015

Betreuer: Florian Hoppe



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT







TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Fachgebiet für Produktionstechnik  
und Umformmaschinen



Prof. Dr.-Ing. Dipl.-Wirtsch.-Ing.  
Peter Groche

## Master-Thesis

Für

Herrn B. Sc. Tajinder Singh Dhaliwal

**Thema:** Modellierung und Regelung einer mechanischen Presse mithilfe von Methoden des maschinellen Lernens

Modeling and Control of a Mechanical Forming Press using Machine Learning Methods

Für die Überwachung und Regelung von Pressen sind präzise Modelle notwendig. Im Gegensatz zu klassischen White-Box Modellierungsansätzen, in dem a priori alle Modellparameter und Einflüsse genau ermittelt und beschrieben werden, soll im Rahmen dieser Arbeit ein Ansatz für die lernende Modellierung der 3D-Servo-Presse verfolgt werden. Hierzu sollen Methoden aus dem maschinellen Lernen genutzt werden, um ein dynamisches Modell anhand gemessener Ein- und Ausgangsdaten der Presse zu erstellen. Hiermit sollen zum einen das Maschinenverhalten überwacht und zum anderen ein inverses Modell für die Regelung bereitgestellt werden.

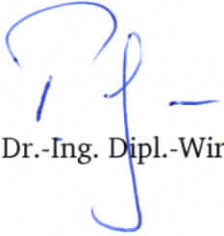
Im Einzelnen sind folgende Tätigkeiten durchzuführen:

- Recherche zum Stand der Technik über den Einsatz von maschinellem Lernen bei der Modellierung von Maschinen
- Auswahl und Implementierung von Verfahren in Matlab/Python
- Untersuchung der Verfahren anhand eines Simulationsmodells der 3D-Servo-Presse
- Experimentelle Untersuchung und Bewertung der Verfahren am Prototypen
- Zusammenstellen der Ergebnisse in einer schriftlichen Ausarbeitung

Beginn: 10.09.2018

Abgabe: 11.03.2019

Betreuer: M. Sc. Florian Hoppe



Prof. Dr.-Ing. Dipl.-Wirtsch.-Ing. P. Groche



---

## **Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt**

---

Hiermit versichere ich, Tajinder Singh Dhaliwal, die vorliegende Master-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 ABP überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum:

Unterschrift:

\_\_\_\_\_

\_\_\_\_\_

---

## Kurzfassung

---

Diese Arbeit befasst sich mit der Entwicklung eines Verfahrens zur Beherrschung von Unsicherheiten in Drei-Punkt-Richtprozessen. Beim Richten ist sowohl die Prozessgeschwindigkeit als auch Prozessgenauigkeit die genaue Prädiktion der Rückfederung des Bauteils von entscheidender Rolle. Dabei stellen oftmals schwankende Bauteileigenschaften eine Herausforderung dar, die eine Regelstrategie benötigt, die diese Schwankungen erkennen und kompensieren kann. Hierzu wird in dieser Arbeit ein Verfahren entwickelt, welches es erlaubt, parallel zu einem Biegeprozess in Echtzeit alle relevanten Bauteilinformationen aus der Kraft-Weg-Messung des Bauteils zu identifizieren und damit zu jedem Zeitpunkt die Rückfederung zu prädizieren.

Anhand der Ergebnisse an einer Drei-Punkt-Richtmaschine wird gezeigt, dass mit diesem Verfahren auch ohne Kenntnis der Materialeigenschaften mit nur einem Richthub eine hohe Genauigkeit erzielt werden kann. Darüber hinaus werden in Versuchen auch die Grenzen der Robustheit gegenüber Schwankungen in der Bauteilgeometrie getestet. Als Ausblick zu diesem Verfahren wird ein Lösungsansatz geliefert, mit dem ein höheres Maß an vertrauenswürdiger Information gewonnen werden kann und durch eine stochastische Modellierung der Unsicherheiten eine weitere Optimierung ist.

**Schlüsselwörter:** Drei-Punkt-Richten, Unsicherheit, Unwissen, Materialeigenschaft, Bauteileigenschaften, Robustheit

---

## Abstract

---

This thesis deals with the development of a method to control uncertainties in three-point straightening processes. Speed and accuracy in straightening processes are determined by its quality of springback prediction. Alternating material and part properties are a challenging task for springback prediction and require a control strategy which is able to detect and compensate those uncertainties. Therefore this thesis presents a method which is able to extract all essential information from the online force-displacement curvature during the straightening process and provides a real-time springback prediction.

Results from real processes on a three-point straightening machine have shown that this method is able to handle unknown uncertainties in material properties and achieve a high accuracy within one stroke. Additional results show the robustness of this method and its limits regarding uncertainties in part properties. A further solution is provided which gives an outlook on how to increase the amount of available, reliable information and therefore optimize the method with a stochastic uncertainty.

**Keywords:** three-point straightening, stochastic uncertainty, unknown uncertainty, material properties, part properties, robustness

---

# Inhaltsverzeichnis

<b>Symbole und Abkungen</b>	<b>vii</b>
<b>Abbildungsverzeichnis</b>	<b>ix</b>
<b>Tabellenverzeichnis</b>	<b>xi</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Abgrenzung zu vorherigen Arbeiten . . . . .	2
1.2 Vorgehensweise . . . . .	3
<b>2 Stand der Technik</b>	<b>5</b>
2.1 Modellierungsansätze . . . . .	5
2.2 Allgemeine Regelung . . . . .	7
2.3 Zustandsüberwachung . . . . .	9
2.4 Maschinelles Lernen . . . . .	11
2.5 Modellbildung durch künstliche neuronale Netzwerke . . . . .	12
2.5.1 Multi-Layer-Perceptron (statischer Funktionsapproximator) . . . . .	13
2.5.1.1 Herausforderungen bei der Auslegung neuronaler Netze . . . . .	16
2.5.2 Netze mit überlagerten Basisfunktionen (statische Funktionsapproxima-	
toren) . . . . .	19
2.5.3 Rekurrente neuronale Netze (dynamische Funktionsapproximatoren) . . .	22
2.5.3.1 Einfache rekurrente Netze (dynamische Funktionsapproximato-	
ren) . . . . .	23
2.5.3.2 Herausforderungen bei rekurrenten Netzen . . . . .	25
2.5.3.3 Moderne rekurrente Netzarchitekturen (dynamische Funktions-	
approximatoren) . . . . .	26
2.5.3.4 Klassifikation rekurrenter Netze . . . . .	28
2.6 Regelungskonzepte basierend auf Methoden des maschinellen Lernens . . . . .	28
2.6.1 Überwachtes Lernen . . . . .	28
2.6.1.1 Direkte inverse Regelung . . . . .	29
2.6.1.2 Indirekte neuronale Regelung mit Referenzmodell . . . . .	30
2.6.1.3 Internal Model Control (IMC) . . . . .	31
2.6.1.4 Feedback Linearisierung mit neuronalen Netzen . . . . .	32
2.6.1.5 Neuronale prädiktive Regelung . . . . .	33
2.6.1.6 Regelungssysteme mit neuronalen Netzen als Kompensatoren . .	34



2.6.2 Bestärkendes Lernen . . . . .	34
<b>3 Ausblick</b>	<b>37</b>
<b>Literaturverzeichnis</b>	<b>39</b>



---

# Symbole und Abkzungen

## Operatoren und Funktionen

Symbol	Beschreibung
*	Faltung
$\Theta(\cdot)$	Heaviside-/Sprung-Funktion

## Lateinische Symbole und Formelzeichen

Symbol	Beschreibung	Einheit
$b$	Integrationskonstante	N

## Griechische Symbole und Formelzeichen

Symbol	Beschreibung	Einheit
$\epsilon$	Rauschprozesses der Kraftmessung	N
$\varphi$	Umformgrad	

---

## Abkürzungen

---

Kel	vollstige Bezeichnung
-----	-----------------------

---

AI	Analog Input
----	--------------

AO	Analog Output
----	---------------

---

# Abbildungsverzeichnis

2.1	Mathematische Modelle von White-Box- bis zu Black-Box-Modellen [31] . . . . .	7
2.2	Erweiterte Grundstruktur des Regelkreises [42] . . . . .	8
2.3	Fehlererkennung mit Signal- und Prozessmodellen (a) signalmodellbasiert, (b) prozessmodellbasiert. [30] . . . . .	10
2.4	Übersicht der Fehlererkennungsmethoden [30] . . . . .	10
2.5	Gängige Lernverfahren und ihre Modelle [13] . . . . .	12
2.6	Multi-Layer-Perceptron (MLP) [65] . . . . .	13
2.7	Multi-Layer-Perceptron [64] . . . . .	15
2.8	Beispiele lokaler Basisfunktionen [60] . . . . .	20
2.9	RBF-Netz mit $p$ Stützstellen [60] . . . . .	20
2.10	Rekurrentes Netz [40] . . . . .	23
2.11	Jordan-Netz [40] . . . . .	24
2.12	Elman-Netz [40] . . . . .	25
2.13	Long Short-Term Memory-Zelle [40] . . . . .	27
2.14	Direkte inverse Regelung mit offline-Netztraining [62] . . . . .	29
2.15	Direkte inverse Regelung mit online-Netztraining [62] . . . . .	29
2.16	Indirekte neuronale Regelung mit Referenzmodell [62] . . . . .	30
2.17	Internal Model Control-Regelkreis [62] . . . . .	31
2.18	Feedback Linearisierung mit neuronalen Netzen [62] . . . . .	32
2.19	Neuronale prädiktive Regelung [62] . . . . .	33



---

# Tabellenverzeichnis

2.1	Eigenschaften theoretischer und experimenteller Modellierungsansätze [31] . . . .	6
-----	---	---



---

# 1 Einführung

Die Planung und Einführung von Fertigungssystemen geht mit Unsicherheiten einher. Dies ist bedingt durch die limitierte Gültigkeit von Annahmen in Bezug auf zukünftige Ereignisse während der Auswahl- und Entwicklungsphase von Fertigungstechnologien und Werkzeugmaschinen [19]. Nach [18] gibt es vier Arten von Unsicherheiten: die Marktakzeptanz von bestimmten Produkten, die Länge der Produktlebensphasen, spezifische Produkteigenschaften und die aggregierte Produktnachfrage. Ein mögliches Lösungskonzept zur Bewältigung dieser Unsicherheiten besteht darin, die Flexibilität von Fertigungssystemen zu erhöhen. Daraus ergeben sich nach [63] folgende Vorteile: durch die Erhöhung dieser können erstens eine höhere Anzahl an Produkten und Produktvariationen (Werkzeugflexibilität) in den Fertigungsprozess integriert werden, zweitens erhöht sich die Adaptionsfähigkeit des Fertigungsprozesses auf eine Veränderung der Produktpalette (Produktflexibilität), drittens erhöht sich bei flexiblen Fertigungsprozessen die Adaptionsfähigkeit auf Veränderungen des Prozesses, z.B. durch technologische Entwicklungen (Prozessflexibilität) und viertens kann durch solche Fertigungssysteme flexibler auf Nachfrageschwankungen reagiert werden (Nachfrageflexibilität).

Bisher kommen Umformmaschinen vor allem bei großen Stückzahlen und ausgewählten Umformmethoden oder kleinen Stückzahlen und vorher speziell festgelegten Werkzeugverfahren zum Einsatz [19]. Dadurch ist die Adaptionsfähigkeit auf Nachfrageschwankungen sehr eingeschränkt [59]. Dagegen bietet die Integration von Servomotoren in Pressen neue Möglichkeiten der Flexibilisierung mit [20]. Um dem Anspruch einer höheren Flexibilität für Pressen gerecht zu werden, entwickelte das Institut für Produktionstechnik und Umformmaschinen (PtU) die neuartige 3D-Servo-Presse. Diese verfügt über drei Antriebssysteme. Diese erlauben es der 3D-Servo-Presse, zuzüglich zur translatorischen Stößelbewegung eine Verkipfung orthogonal zur Translationsbewegung durchzuführen. Dadurch ergeben sich insgesamt drei Freiheitsgrade: eine translatorische Hubbewegung und zwei rotatorische Kippbewegungen. Dadurch ist die Herstellung neuartiger Produktgeometrien und das Einbringen bestimmter Materialeigenschaften in den umgeformten Produkten durch definierte Werkzeugbewegungen denkbar. Beispielsweise lässt sich durch den Einsatz der 3D-Servo-Presse die Anzahl der Prozessschritte bei der Herstellung von Bauteilen mit sehr hohen Umformgraden durch die gezielte Steuerung des Materialflusses reduzieren [61]. Durch den Einsatz der 3D-Servo-Presse soll damit dem Anspruch nach höherer Flexibilisierung und der damit einhergehenden höheren Wirtschaftlichkeit gerecht werden.

---

Zusätzlich zum Thema Flexibilisierung von Fertigungssystemen hat sich das Thema Industrie 4.0 als weiterer Forschungsgegenstand in der Literatur etabliert. Nach [5] stellt die Integration von Sensoren und die damit ermöglichte Zustandsüberwachung einer Werkzeugmaschine ein Teilaspekt der Industrie 4.0 dar. Durch die Zustandsüberwachung ist eine frühzeitige Detektion von Ausfällen möglich. Darüber hinaus können durch die Erfassung des Betriebszustandes Prognosen zur zukünftigen Funktionsfähigkeit der Werkzeugmaschine gemacht werden. Diese erlauben im weiteren Verlauf das Initiieren von Maßnahmen zur Behebung von möglich auftretenden Ausfällen, Defekten etc. [5]

Sowohl für die Regelung der 3D-Servo-Presse während des normalen Betriebsfalles als auch für die Zustandsüberwachung der 3D-Servo-Presse ist eine Modellbildung dieser notwendig. Im Gegensatz zu vergangenen Arbeiten kommt in dieser Arbeit kein White-Box-Ansatz, in dem a priori alle Modellierungsparameter und Einflüsse genau ermittelt werden, sondern ein Blackbox-Ansatz zur Anwendung. Für die Parametrisierung des Black-Box-Modells kommen Methoden des maschinellen Lernens zum Einsatz, um anhand gemessener Eingangs- und Ausgangsgrößen ein dynamisches Modell zu erstellen. Auf Basis dieses Modells sollen Konzepte zur Zustandsüberwachung als auch zur Regelung der 3D-Servo-Presse entwickelt werden. Als Grundlage dieser Arbeit dient der Prototyp der 3D-Servo-Presse. Diese steht als Forschungsobjekt am PtU an der Technischen Universität Darmstadt zur Verfügung.

---

## 1.1 Abgrenzung zu vorherigen Arbeiten

---

Wie bereits erwähnt verfügt die 3D-Servo-Presse über drei Servo-Motoren, welche die Antriebsmomente liefern. Mit der Hilfe von ungleich übersetzenden Koppelgetrieben werden nicht nur die Drehmomente in die auf den Stößel wirkende Zustellkraft übersetzt, sondern auch eine Kippbewegung des Stößels in zwei rotatorische Freiheitsgraden ermöglicht. Für diese Presse sind in mehreren Vorarbeiten bereits Pressenmodelle entwickelt worden.

[55] entwickelte in seiner Arbeit ein mechanisches Mehrkörpermodell des Getriebes der 3D-Servo-Presse. Bei diesem Pressenmodell berücksichtigt er sowohl die Nachgiebigkeiten als auch die Massenträgheiten aller Getriebeglieder, um damit dynamische Vorgänge, wie z.B. Schwingungen und das Verfahren der Getriebestellung, zu untersuchen. [55] kommt zum Ergebnis, dass das Mehrkörpermodell steifer ist als das reale Getriebe. Nach einer erneuten Parameteridentifikation zeigte sich in den Messdaten eine Hysterese und mechanisches Spiel. Nach [55] konnten diese nicht durch das Modell abgebildet werden. Weiterhin kommt [55] zum Schluss, dass die aus der Identifikation entspringenden Parameter einer Unsicherheit unterliegen. Daraufhin untersuchte [55] die Auswirkungen der Parameterunsicherheit und wählte aus den unsicheren Parametern zwei Parametersätze, aus welchen zwei Grenzmodelle hervorgingen, ein weiches und steifes Pressenmodell. Mit diesen konnte [55] letztlich die Zustandsüberwachung



---

durchführen.

In dieser Arbeit soll nun im Gegensatz zu [55] ein Black-Box-Modell an Stelle eines White-Box-Modells entwickelt werden. Die Parametrisierung des Modells findet dabei über Methoden des maschinellen Lernens statt. Das Ziel besteht darin, bisher in White-Box-Modellen nicht abgebildete Effekte, wie z.b. die nicht lineare Reibung, Fertigungstoleranzen, statische Dehnungen, Lagerreibung, Abnutzungserscheinungen, etc. durch Methoden des maschinellen Lernens abzubilden.

---

## 1.2 Vorgehensweise

---



---

## 2 Stand der Technik

Das Ziel dieser Arbeit besteht darin, ein Ersatzmodell für den Prototypen der 3D-Servo-Pressen zu entwickeln. Die Parametrisierung erfolgt über Methoden des maschinellen Lernens. Auf Basis des Ersatzmodells folgt im weiteren Verlauf die Entwicklung von Konzepten zur Zustandsüberwachung und Regelung der 3D-Servo-Pressen. Zunächst soll jedoch auf die Grundlagen der Modellbildung eingegangen werden.

---

### 2.1 Modellierungsansätze

---

Nach [31] gibt es mehrere Möglichkeiten der Modellbildung, die theoretische, die experimentelle und theoretisch-experimentelle Mischformen. Zunächst erfolgt die Beschreibung der theoretischen Modellbildung.

Bei der theoretischen Modellierung findet die Beschreibung des Systems entweder über partielle oder gewöhnliche Differentialgleichungen statt. Da reale Systeme in vielen Fällen zu komplex und/oder kompliziert sind, ist es üblich, das Modell zu vereinfachen, z.B. durch das Ignorieren bestimmter real auftretender Effekte (z.B. nichtlineare Reibungseffekte). In der Regel ist es nach [31] möglich, das Modell durch vier Arten von Gleichungen zu beschreiben:

- *Bilanzgleichungen*: Dazu gehören die Massen-, Energie- und Impulserhaltung.
- *Physikalische oder chemische Zustandsgleichungen*: Diese sogenannten konstitutiven Gleichungen beschreiben reversible Vorgänge, wie z.B. das 2. Newtonsche Gesetz.
- *Phänomenologische Gleichungen*: Diese beschreiben irreversible Vorgänge, wie z.B. Reibung.
- *Verbindungsgleichungen*: Dazu gehören die Kirchhoffschen Regeln, Momentengleichgewichte, etc. [31]

Nach [31] setzt sich die theoretische Modellbeschreibung des Systems aus einer Mehrzahl von Gleichungen zusammen. Die Lösung des Gleichungssystems ist zwar explizit nicht immer möglich, aber trotzdem können individuelle Gleichungen wichtige Anhaltspunkte bezüglich der Modellstruktur liefern. [31]

Die experimentelle Modellbildung basiert auf Messdaten, welche im Zuge von Versuchen aufgenommen werden. Die Messdaten teilen sich auf in Eingangs- und Ausgangsgrößen des Systems. Als Eingangsgrößen können entweder reale Eingangsstellgrößen oder künstlich erzeugte Testsignale mit bestimmten Eigenschaften fungieren. Im Fall der 3D-Servo-Pressen seien der Exzenteranschub und die beiden Spindelanschübe die Eingangsgrößen des Systems, da sie die Höhenverstellung des Stößels als Ausgangsgröße festlegen. Das Ziel der theoretischen Modellbildung

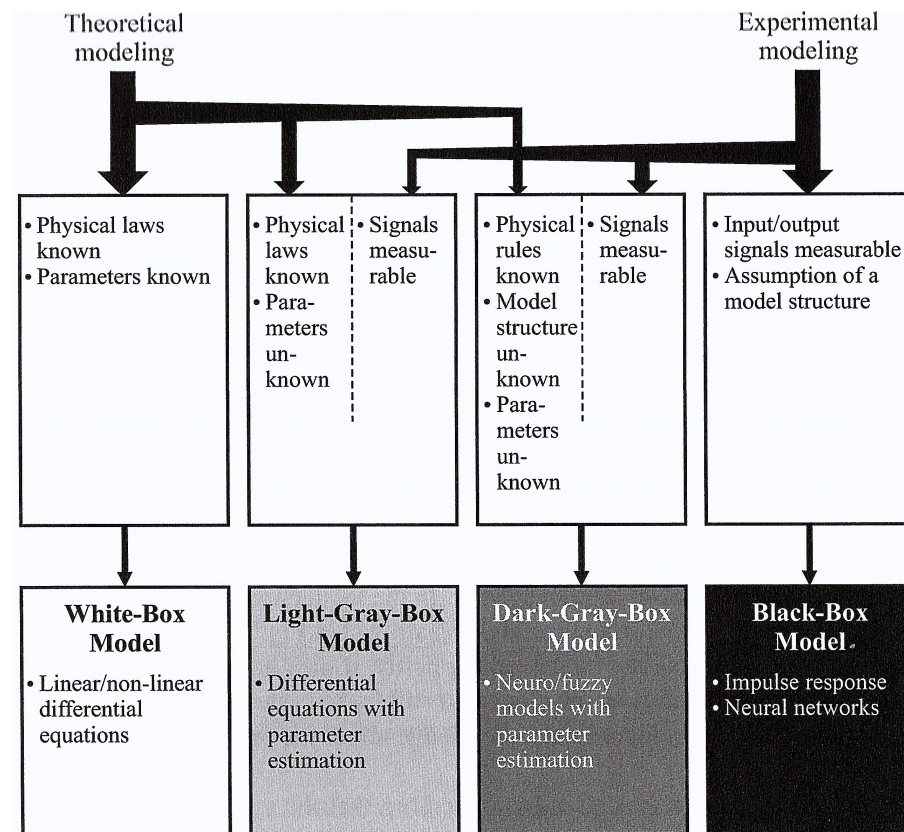
besteht nun darin, für eine ausgewählte Modellstruktur die Parameter so zu identifizieren, dass ein möglichst guter Zusammenhang zwischen Eingangs- und Ausgangsgrößen hergestellt wird. Der dafür in der Literatur standardmäßig verwendete Begriff ist Systemidentifikation. [31]

Die Bezeichnung der Modellstruktur, welche aus der theoretischen Modellbildung hervorgeht, lautet White-Box-Modell. Das Gegenstück dazu ist das Black-Box-Modell, welches aus der experimentellen Modellbildung hervorgeht. Nach [31] haben White-Box- und Black-Box-Ansätze verschiedene Vor- und Nachteile, aber dennoch sind die komplementär zueinander. Um Vorteile beider Modellierungsansätze zu vereinen, sind ebenfalls sogenannte Gray-Box-Modell-Ansätze in verschiedenen Abstufungen (Light-Gray-Box-Modell, Dark-Gray-Box-Modell) entstanden. Eine Übersicht über alle Modellierungsansätze liefert Abbildung 2.1. Die wesentlichen Eigenschaften beider Modellierungsansätze sind in Tabelle 2.1 zu finden.

**Tabelle 2.1:** Eigenschaften theoretischer und experimenteller Modellierungsansätze [31]

<b>Theoretische Modellbildung</b>	<b>Experimentelle Modellbildung (Systemidentifikation)</b>
Modellstruktur folgt Naturgesetzen	Modellstruktur muss angenommen werden
Modellierung des Übertragungsverhaltens und innerer Systemvorgänge	lediglich Identifizierung des Übertragungsverhaltens
Gültigkeit des Modells für eine Reihe von Prozessen ähnlichen Types und unterschiedlicher Randbedingungen	Modell ist nur für das untersuchte System innerhalb der Betriebsgrenzen gültig
Modellkoeffizienten sind nicht exakt bekannt	exaktere Bestimmung der Modellkoeffizienten für das gegebene System innerhalb der Betriebsgrenzen
Modell kann für nicht existente Systeme entwickelt werden	Modell kann nur für ein bereits existierendes System entwickelt werden
das interne Systemverhalten muss bekannt und mathematisch beschreibbar sein	Identifikationsmethoden sind unabhängig vom untersuchten System und können auf andere Systeme angewendet werden
üblich langsamer Prozess mit hohem Zeitaufwand	schneller Prozess bei bereits bekannten Identifikationsmethoden
Modelle können sehr komplex und detailliert ausfallen	Modellgröße kann auf den Anwendungsfall des Modells angepasst werden

Wie in Tabelle 2.1 zu erkennen ist, umgehen Black-Box-Modelle einen wesentlichen Nachteil von White-Box-Modellen, nämlich die exakte Kenntnis der internen Systemzusammenhänge, das Aufstellen von mathematischen Gleichungen zur Beschreibung dieser und die explizite



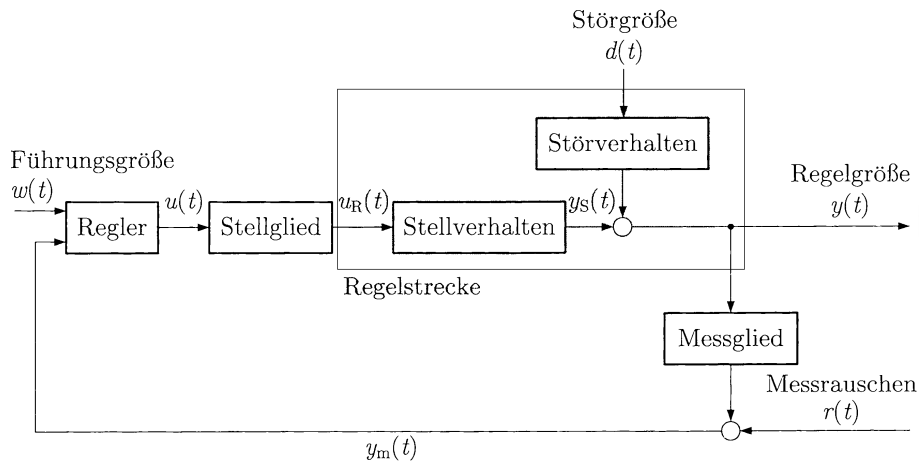
**Abbildung 2.1:** Mathematische Modelle von White-Box- bis zu Black-Box-Modellen [31]

te Lösung des Gleichungssystems. Für komplexe Systeme wie der 3D-Servo-Pressen stößt der White-Box-Ansatz auf viele Probleme, siehe Abschnitt 1.1. Deshalb fungiert das Black-Box-Modell als Grundlage für die Modellbildung des Prototypen der 3D-Servo-Pressen. Wie in Abbildung 2.1 zu erkennen ist, können Neuronale Netze als Methode zur Parameteridentifikation von Black-Box-Modellen dienen. Neuronale Netze stellen ein Teilgebiet des Maschinellen Lernens dar. Die Grundlagen der Neuronalen Netze werden in Kapitel QUELLE beschrieben. Das Black-Box-Modell dient als Grundlage für die Regelung und die Zustandsüberwachung der 3D-Servo-Pressen. Zunächst sollen im nächsten Abschnitt die Grundlagen der Regelung vermittelt werden.

## 2.2 Allgemeine Regelung

Die in Abschnitt 2.1 vorgestellten Modellierungsansätze haben den Zweck, ein reales System, wie z.B. die 3D-Servo-Pressen, in ein Ersatzmodell zu überführen. Dieses hat den Zweck, einen möglichst guten mathematischen Zusammenhang zwischen den Eingangs- und Ausgangsgrößen des realen Systems zu liefern.

Nach [42] wirken Eingangsgrößen (z.B. Spindel- und Exzentervorschübe) auf das System ein und verursachen zeitliche Veränderungen des Systems. Die Ausgangsgrößen (z.B. die Position des Stößels) dagegen beschreiben das Systemverhalten als Reaktion auf die Eingangsgrößen.



**Abbildung 2.2:** Erweiterte Grundstruktur des Regelkreises [42]

Da sich dabei Kenngrößen des Systems zeitlich verändern, ergibt sich für solchen der Begriff *Dynamisches System*. Nach [42] besteht die Aufgabe der Regelungstechnik nun darin, für ein solches dynamisches System die beeinflussbare Größe  $u(t)$  (Stellgröße bzw. Eingangsgröße) so anzupassen, dass ein Regelungsziel  $w(t)$  (Führungsgröße bzw. Sollwert) erreicht wird, siehe Abbildung 2.2.

Dabei erfüllt die Regeleinrichtung die Aufgabe, unter Nutzung der gemessenen Werte für die Regelgröße (Ausgangsgröße)  $y(t)$  die Stellgröße (Eingangsgröße)  $u(t)$  so vorzugeben, dass die Differenz zwischen der gemessenen Regelgröße (Ausgangsgröße)  $y(t)$  und Führungsgröße  $w(t)$  minimal ist. Die Bezeichnung dieser Differenz lautet Regelabweichung  $e(t)$ , anhand derer die Regeleinrichtung die Stellgröße  $u(t)$  zweckmäßig vorgibt. [42]

$$e(t) = w(t) - y(t) \quad (2.1)$$

Laut [42] gibt es häufig eine Differenz zwischen der Regelgröße  $y(t)$  und der gemessenen Regelgröße  $y_m(t)$ , da das Messglied selbst über dynamische nichtlineare Eigenschaften verfügt. Dasselbe gilt für Stellglieder (z.B. ein Servomotor, welcher eine Spindel antreibt), welche darüber hinaus sich häufig durch dynamisches Verhalten auszeichnen. Als Resultat ergibt sich daraus eine Differenz zwischen der vorgegebenen Stellgröße  $u(t)$  und der für den Prozess wirksamen Stellgröße  $u_R(t)$ . Diese verursacht eine zeitliche Veränderung des Systems (z.B. Stoßelbewegung) und erzeugt somit ein Stellverhalten, welches sich mit einem Störverhalten, hervorgerufen durch die unbekannte Störgröße  $d(t)$ , überlagert. [42]

Die Auslegung der Regelstrecke muss nun derartig gestaltet sein, dass sie trotz des dynamischen Verhaltens von Messgliedern und Stellgliedern eine minimal mögliche Differenz zwischen der Regelgröße  $y(t)$  und Führungsgröße  $w(t)$  einstellt. Dafür ist die Auslegung des Reglers entscheidend.

Für die Auslegung des Reglers ist der Einsatz von Ersatzmodellen notwendig, welche das Systemverhalten des realen Systems so gut wie möglich abbilden. Wie bereits in Abschnitt 1.1

---

diskutiert, ist der Einsatz von White-Box-Modellen als Ersatzmodell mit Problemen verbunden. Diese versuchen durch Modellvereinfachungen das Systemverhalten mathematisch zu beschreiben. Damit können sie aber maximal das Stellverhalten des Systems, aber nicht das durch die unbekannte Störgröße  $d(t)$  hervorgerufene Störverhalten abbilden. Black-Box-Modelle dagegen, welche mit den Methoden des maschinellen Lernens parametrisiert sind, können auch das Störverhalten mit abbilden, da die Parametrisierung über gemessene Daten stattfindet. Ein Black-Box-Modell, welches das reale Systemverhalten genau genug beschreibt, kann neben der Auslegung von Reglern oder als Hilfsglied im Regelkreis auch im Bereich der Zustandsüberwachung zum Einsatz kommen. Die Grundlagen der Zustandsüberwachung sind im nächsten Abschnitt erläutert.

---

## 2.3 Zustandsüberwachung

---

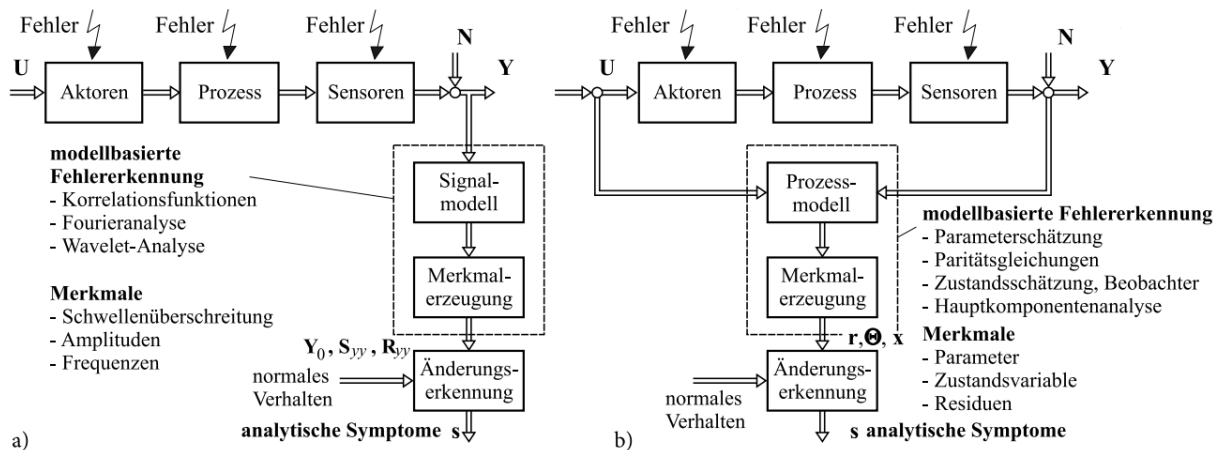
Neben dem Anwendungsbereich der Regelungstechnik ist der Einsatz von Black-Box-Modellen ebenfalls im Bereich der Zustandsüberwachung denkbar.

Nach [30] ergeben sich für die Zustandsüberwachung neben der Anzeige des gegenwärtigen Prozesszustandes ebenfalls die Meldung unerlaubter Betriebszustände und die Einleitung notwendiger Maßnahmen zur Einhaltung des Betriebs und zur Unfallvermeidung als Aufgabengebiete. [30] unterscheidet dabei drei Arten der Überwachung:

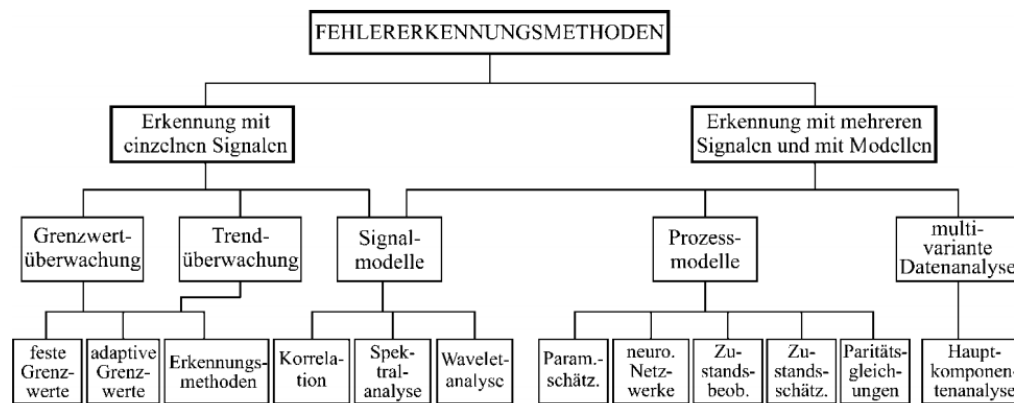
- *Grenzwertüberwachung*: Nach Überschreiten von festgelegten Toleranzen durch gemessene Größen erzeugt das System Alarmmeldungen.
- *Automatischer Schutz*: Bei der Entstehung von gefährlichen Prozesszuständen leitet die Grenzwertüberwachung Gegenmaßnahmen zur Überführung in einen sicheren Zustand ein.
- *Überwachung mit Fehlerdiagnose*: Merkmale werden aus messbaren Größen abgeleitet, daraus Symptome erzeugt, Fehlerdiagnosen durchgeführt und Gegenmaßnahmen eingeleitet. [30]

Nach [30] eignen sich die ersten beiden Methoden vor allem für stationäre Prozesse, da bei diesen die Definition eines eng anliegenden Toleranzbandes um die Prozessgröße herum möglich ist. Diese zeichnen sich durch eine hohe Zuverlässigkeit und Einfachheit aus. [30]

Mit den ersten beiden Methoden ist es jedoch kaum möglich, Prozesse in dynamischen Betriebszuständen zu überwachen, da dabei große Veränderungen der Prozessgrößen entstehen. Beispielsweise würde dabei ein für einen stationären Prozess ausgelegtes Toleranzband überschritten werden, was zu einem Fehlalarm führen könnte. Zudem eignen sich die ersten beiden Methoden nicht zum frühzeitigen Erkennen kleiner Fehler, da dabei das Toleranzband nicht überschritten würde. Auch bleiben der Fehlerort, die Fehlergröße und die Fehlerursache unberücksichtigt. Aus diesen Gründen ist die Weiterentwicklung der dritten Methode, die Überwachung mit integrierter Fehlerdiagnose, notwendig [30].



**Abbildung 2.3:** Fehlererkennung mit Signal- und Prozessmodellen (a) signalmodellbasiert, (b) prozessmodellbasiert. [30]



**Abbildung 2.4:** Übersicht der Fehlererkennungsmethoden [30]

Nach [30] ist für die Fehlerdiagnose das Erkennen von sogenannten Symptomen entscheidend. Ein Symptom entspricht der Differenz zwischen dem beobachteten und dem normalen Merkmal. Je nachdem, ob es sich um ein Signal- oder um ein Prozessmodell handelt, können Merkmale entweder Schwellenwertüberschreitungen, Amplituden und Frequenzen (bei Signalmodellen) oder Parameter, Zustandsvariablen und Residuen (bei Prozessmodellen) sein, siehe Abbildung 2.3. [30]

In beiden Fällen findet ein Vergleich der Merkmale mit den normalen Merkmalen des fehlerlosen Prozesses statt. Mit Hilfe von Methoden zur Erkennung signifikanter Änderungen (z.B. Parameterabschätzung, Paritätsgleichungen, Zustandsgrößenbeobachter, neuronale Netzwerke, etc.) können so die *analytischen Symptome* ermittelt werden, siehe Abbildung 2.4. [30]

Im weiteren Verlauf findet die Ermittlung des Zusammenhangs zwischen den analytischen Symptomen und den Fehlern entweder durch Klassifikationsmethoden (statistisch, geometrisch, Neuronale Netze, Fuzzy Cluster) oder durch Interferenzmethoden (Kausales Netz, Fehler-Symptom-Baum) statt. [30]



---

Damit lässt sich sagen, dass Neuronale Netze im Bereich der Zustandsüberwachung zwei Anwendungsgebiete einnehmen. Zum einen können neuronale Netze als Prozessmodell zwecks Fehlererkennung fungieren. Zum Beispiel kann das neuronale Netz mit einem realen Prozess-eingangssignal beaufschlagt werden. Bei einem fehlerhaften Prozess würde das vom neuronalen Netz herausgegebene Ausgangssignal sich vom Ausgangssignal eines normalen Prozesses unterscheiden. Weiterhin kann das neuronale Netz als Klassifikationsmethode genutzt werden, um bereits ermittelte *analytische Symptome* mit Fehlern in einen Zusammenhang zu bringen.

---

## 2.4 Maschinelles Lernen

---

Wie bereits erwähnt, soll die Parametrisierung des in dieser Arbeit entwickelten Black-Box-Modells mit Methoden des maschinellen Lernens stattfinden. Da der Begriff des maschinellen Lernens ein weites Spektrum an theoretischen Grundlagen und praktischen Anwendungen umfasst, enthält der nächste Abschnitt grundlegende Erläuterungen zu diesem Begriff.

Der Begriff des maschinellen Lernens ist ein Oberbegriff, welcher eine Reihe von Methoden umfasst, welche den Zweck haben, Modelle auf Basis von gesammelten bzw. gemessenen Daten zu generieren. Je umfangreicher dabei die Datengrundlage ist, desto besser ist das Modell in der Lage, das Verhalten des realen Systems abzubilden. Darüber hinaus ist das erzeugte Modell fähig, das Systemverhalten über die verwendete Datengrundlage hinweg zu verallgemeinern. Damit ist die Anwendung des Modells auf bisher unbekannte Datensätze derselben Art möglich, um daraus z.B. Vorhersagen über das zukünftige Systemverhalten zu machen. Der entscheidende Unterschied zu klassischen Systemidentifikationsmethoden ist, dass die Parametrisierung des Modells automatisiert durch Algorithmen und nicht durch explizites Festlegen stattfindet. [15] Gemeinhin wird in der Literatur die Parametrisierung bzw. Adaptierung der Modelle durch Daten als *Training* bezeichnet. Nach [13] gibt es verschiedenen Lernstile. Einen Überblick über gängige Lernverfahren, ihre Lernziele und Modelle gibt Abbildung 2.5.

Der Begriff des *überwachten Lernens* inkludiert, dass während des Trainings bekannte (z.B. gemessene) Ausgangsgrößen dem System vorgegeben werden, um eine bestimmte Lernaufgabe (Klassifikation oder Regression) zu lösen. Nach dem Training des Netzes kann dann der Lernerfolg anhand bekannter richtiger Ergebnisse validiert und überwacht werden.

Dagegen findet beim *bestärkenden Lernen* keine direkte Übergabe der erwünschten Ausgangsgrößen an das System statt. Stattdessen soll das System autonom eine Strategie erlernen, um die Anzahl der erhaltenen positiven Belohnungen zu maximieren. Dafür empfängt das System zu verschiedenen Zeitpunkten auf Basis seiner Aktionen eine positive oder negative Belohnung. Daraus ermittelt er eine Nutzenfunktion, welche seine Aktionen bewertet. Diese Nutzenfunktion entspricht üblicherweise seiner Strategie, die es zu verbessern gilt.

Beim *unüberwachten Lernen* findet weder eine Übergabe der gewünschten Ausgangsgrößen noch eine Übergabe von Belohnungsanreizen an das System statt. Dieser Lernstil kommt somit vor allem bei Dimensionsreduktions- und Clustering-Anwendungen zum Einsatz.

Lernstil	Lernaufgabe	Lernverfahren	Modell
Überwacht	Regression	Lineare Regression	Regressionsgerade
		Klassifikations- und Regressionsbaumverfahren (CART)	Regressionsbaum
	Klassifikation	Logistische Regression	Trennlinie
		Iterative Dichotomizer (ID3)	Entscheidungsbaum
		Stützvektormaschine (SVM)	Hyperebene
		Bayessche Inferenz	Bayessche Modelle
Unüberwacht	Clustering	K-Means	Clustermittelpunkte
	Dimensionsreduktion	Kernel Principal Component Analysis (PCA)	Zusammengesetzte Merkmale
Bestärkend	Sequentielles Entscheiden	Q-Lernen	Strategien
Verschiedene	Verschiedene	Rückwärtspropagierung	Künstliche Neuronale Netze

**Abbildung 2.5:** Gängige Lernverfahren und ihre Modelle [13]

Interessanterweise kommen neuronale Netze sowohl bei *bestärkenden* (das Identifizieren einer Systemstrategie, um die Anzahl der empfangenen positiven Belohnungen zu maximieren) als auch bei *überwachten* Lernstilen (zur Bildung eines nichtlinearen dynamischen Modells für ein System) zur Anwendung. Die Regelung der 3D-Servo-Pressen kann prinzipiell durch das Anwenden beider Lernstile erfolgen, entweder durch das autonome Identifizieren einer Betriebsstrategie (das Einstellen der Spindel- und Exzentervorschübe), um die maximale Anzahl an positiven Belohnungen (das richtige Einstellen der Stößelposition) zu erreichen oder durch das Entwickeln eines nichtlinearen dynamischen Black-Box-Modells, welches als Hilfsglied in einem Regelkreis agiert oder in Kombination mit einem anderen Hilfsglied (welches ebenfalls ein neuronales Netz sein kann) selbst als Regler agiert.

## 2.5 Modellbildung durch künstliche neuronale Netzwerke

Wie in Abschnitt 2.4 erläutert, bieten sich für die Modellierung des dynamischen Verhaltens der 3D-Servo-Pressen neuronale Netze an. Nach [40] sind diese in der Lage, jeden stetigen nichtlinearen Zusammenhang zwischen Eingangs- und Ausgangsgrößen beliebig genau abzubilden. Neuronale Netze lassen sich grob in statische und dynamische Funktionsapproximatoren aufteilen. Die statischen Funktionsapproximatoren wiederum teilen sich auf in Netze mit überlagerten Basisfunktionen und Multi-Layer-Perceptron (MLP)-Netze. Zunächst sei eine Einführung in MLP-Netze gegeben.

### 2.5.1 Multi-Layer-Perceptron (statischer Funktionsapproximator)

Die grundsätzliche Funktionsweise von neuronalen Netzen besteht darin, einen mehrdimensionalen Eingabevektor  $x = (x_1, x_2, \dots, x_n)$  in ein Ausgabesignal  $\hat{y} = N(x|w)$  zu transformieren. Dabei bezeichnet  $w = (w_1, \dots, w_n)$  die Netzwerkgewichte. Diese nehmen vor der Parametrisierung bzw. vor dem Training des Netzes zunächst zufällige Werte an. Das Multi-Layer-Perceptron-Netz ist ein weitverbreiteter Netzwerktyp, siehe Abbildung 2.6.

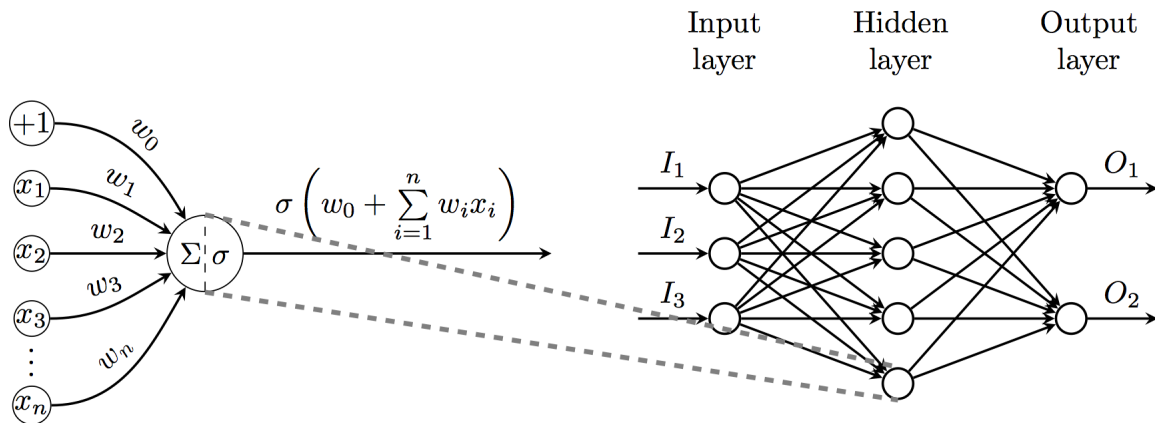


Abbildung 2.6: Multi-Layer-Perceptron (MLP) [65]

Bei diesem Netzwerktyp entspricht der Wert eines Neurons einer linearen Überlagerung  $n$  sigmoidaler Funktionen, ausgedrückt durch

$$\sigma\left(w_0 + \sum_{i=1}^n w_i x_i\right) \quad (2.2)$$

Die lineare Überlagerung der Netzwerkgewichte  $w_1, w_2, \dots, w_n$  multipliziert mit den Eingangsgrößen  $x_1, x_2, \dots, x_n$  und verschoben um den Wert  $w_0$  wird nochmals durch eine Aktivierungsfunktion transformiert. Mögliche Aktivierungsfunktionen sind die *logistische Sigmoid-Funktion*  $\sigma(z) = \frac{1}{1+e^{-z}}$ , die *tangentiale hyperbolische Sigmoid-Funktion*  $\phi(z) = \frac{2}{1+e^{-2z}} - 1$  und die *lineare Funktion*  $a(z) = z$ .

Die Auswahl der Aktivierungsfunktion ist hierbei vom konkreten Anwendungsfall abhängig. Das Training bzw. die Parametrisierung des Netzes erfolgt nun in zwei Schritten. Im ersten Schritt findet eine Übergabe der Eingangsgrößen  $x_1, x_2, \dots, x_n$  an das Netz statt. Weiterhin nehmen die Netzwerkgewichte  $w$  anfangs zufällige Werte an. Da die Eingangsgrößen und die Netzwerkgewichte bekannt sind, kann jeder Neuronenwert nach Gleichung 2.2 berechnet werden. Die nachfolgenden Neuronenwerte der nächsten Schicht lassen sich somit sukzessive als Funktion der vorherigen Schicht berechnen. Auf diese Weise des Vorwärtsrechnens findet eine Transformation der Eingangsgrößen  $x_1, x_2, \dots, x_n$  in ein Ausgangssignal  $\hat{y}$  statt (in Abbildung 2.6 als  $O_1$  und  $O_2$  bezeichnet).

Der zweite Schritt umfasst nun die Adaptierung der Netzwerkgewichte, sodass die vom Netz herausgegebenen Ausgabegrößen  $\hat{y}$  mit den gemessenen Ausgabegrößen  $y_i$  möglichst übereinstimmen. Dabei ist es zweckmäßig, die Abweichung zwischen diesen Größen in Form der Fehlerfunktion

$$E(w) = (y_i - N(x|w))^2 \quad (2.3)$$

zu beschreiben. Die Adaptierung der Netzwerkgewichte beschränkt sich nun darauf, die Fehlerfunktion zu minimieren. Eine der gängigsten Methoden ist die Methode des Gradientenabstieges. Diese gibt als Adaptionregel für die Netzwerkgewichte

$$\Delta w_i = \epsilon(-\nabla E_i) = \epsilon * -\frac{\partial E}{\partial w} \Big|_{w_i} \quad (2.4)$$

mit  $\epsilon$  als Lernrate bzw. Adaptionsschrittweite vor. Das Ziel besteht darin, für die Fehlerfunktion  $E(w)$  die partiellen Ableitungen  $\frac{\partial E}{\partial w} \Big|_{w_i}$  zu identifizieren und gleich null zu setzen. Damit würde die Fehlerfunktion  $E(w)$  ein lokales Minima erreichen. Dies geschieht durch die lokale Suche nach dem Minimum der Fehlerfunktion  $E(w)$  in Richtung des steilsten Abstiegs. Die Richtung des steilsten Abstiegs entspricht dabei dem negativen Gradienten von  $E$  nach den Netzwerkgewichten  $w_i$ .

Die Adaption findet für jedes vorhandene Datenpaar  $((x_1, x_2, \dots, x_n)|(y_1))$  statt. Durch das Wiederholen des Adaptionvorgang passt sich das Netz an das zu modellierende Systemverhalten an.

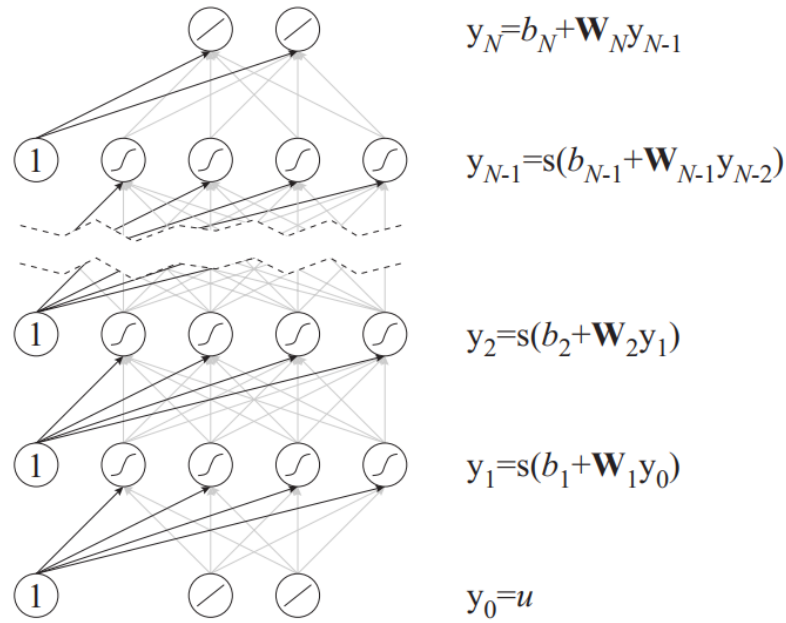
Wie bereits erwähnt, bietet sich die Methode des Gradientenabstieges als ein mögliches Verfahren zur Minimierung der Fehlerfunktion an. Es gibt darüber hinaus in der Literatur eine Vielzahl von Verfahren, die sich nach [62] in folgende Kategorien einteilen lassen:

- Gradientenverfahren erster Ordnung
- Gradientenverfahren zweiter Ordnung
- stochastische Optimierungsverfahren
- Verfahren der globalen Optimierung

### Backpropagation-Algorithmus

Den Verfahren der ersten Kategorie ist gemein, dass die Berechnung des Gradienten des Gütefunktional

$$\nabla E_i = \frac{\partial E}{\partial w} \Big|_{w_i} \quad (2.5)$$



**Abbildung 2.7:** Multi-Layer-Perceptron [64]

in jeder Iteration problematisch ist, da sich die Gradienten nicht in einem Schritt berechnen lassen. In der Regel kommt für die Lösung dieses Problems der *Backpropagation*-Algorithmus zum Einsatz. Dieser wendet die Kettenregel an, um die Gradienten in mehrere Faktoren, bestehend aus partiellen Ableitungen, zu zerlegen. Auf diese Weise können die partiellen Ableitungen der letzten Schicht, welche in der Regel bekannt sind, benutzt werden können, um die partiellen Ableitungen der vorangehenden Schichten zu bestimmen. Dies ist gleichbedeutend mit dem Belegen der *Jacobi-Matrix*.

Die Ermittlung der notwendigen partiellen Ableitungen ergibt sich nach [64] durch folgende Rekursionsformeln

$$\begin{aligned}
 Y'_{i-1} &= Y'_i \cdot S'_i \cdot W_i \\
 \frac{\partial y_N}{\partial (W_i)_{jk}} &= Y'_i \cdot S'_i \cdot \begin{pmatrix} (y_{i-1})_k \\ \dots \\ \dots \\ (y_{i-1})_k \end{pmatrix} \\
 \frac{\partial y_N}{\partial \vec{b}_i} &= Y'_i \cdot S'_i
 \end{aligned} \tag{2.6}$$

Dabei enthält die Matrix  $Y'_i$  die partiellen Ableitungen der Ausgabegrößen der letzten, also der  $N$ ten Schicht, nach der  $i$ ten Schicht. Am Anfang der Rekursion entspricht diese der Ein-

---

heitsmatrix. Die Matrix  $S'_i$  enthält die Ableitungen der Aktivierungsfunktionen der  $i$ -ten Schicht. [64]

Die Ermittlung der partiellen Ableitungen findet also von der letzten bis zur ersten Schicht, also rückwärts, statt. Nach Ermittlung der partiellen Ableitungen aller Schichten, kann die Adaption der Netzwerkgewichte gemäß einer Adaptionregel, siehe 2.4, erfolgen. Während des Adaptionprozesses nimmt der Gesamtfehler ab. Sinkt der Gesamtfehler über alle Netzausgänge  $y_1, \dots, y_m$  unter eine festgelegte Grenze, bricht das Training des Netzes ab.

Es bleibt festzuhalten, dass die Anpassungsfähigkeit des Netzes an das nichtlineare Verhalten dynamischer Systeme von folgenden Faktoren abhängt:

- Anzahl der Neuronen
- Anzahl der Schichten
- Wahl des Algorithmus zur Minimierung der Fehlerfunktion
- Auswahl der Aktivierungsfunktionen
- Initialisierung der Gewichte
- Repräsentationsfähigkeit der Trainingsdaten

Die richtige Auslegung der Parameter für das neuronale Netz stellt eine Herausforderung dar. In Abschnitt 2.5.1.1 ist eine ausführliche Diskussion über die Auswahl der Netzparameter, vor allem über den Algorithmus und die Anzahl der Neuronen, enthalten.

---

### 2.5.1.1 Herausforderungen bei der Auslegung neuronaler Netze

---

#### Anzahl der Neuronen und Gewichte

Für die Auslegung neuronaler Netze ist die Abschätzung der Neuronenanzahl relevant. Dabei ist die Festlegung der optimalen Neuronenanzahl eine Abwägung zwischen der Trainingszeit, der Generalisierungs- und der Anpassungsfähigkeit an die Nichtlinearitäten des dynamischen Systems. Beispielsweise erhöht nach [17] die Anzahl der Neuronen in der verdeckten Schicht die Güte des neuronalen Netzes bei einem Anstieg nichtlinearer Effekte.

Nach [1] können ebenfalls heuristische Aussagen über eine minimale und maximale Neuronenanzahl getroffen werden. Diese bieten allerdings nur Abschätzungen an. Weiterhin gibt es Vorschläge, die Neuronenanzahl während des Trainings zu adaptieren.

Dafür lassen sich die Verfahren in zwei Gruppen aufteilen, einmal in konstruktive und reduzierende Verfahren.

Bei den reduzierenden Verfahren ist die Ausgangsbasis ein bereits trainiertes Netz, welches ein System ausreichend genau modelliert.

Nach [21] sind viele neuronale Netze über-parametrisiert, d.h. sie verfügen über redundante

---

Gewichte. Eine Möglichkeit der Reduktion besteht nun darin, sämtliche Gewichte, deren Wert unter einer vorher definierten Grenze liegt, zu entfernen. Es findet somit eine Unterscheidung zwischen bedeutenden und unbedeutenden Gewichte statt. Nach einer erneuten Trainingsphase findet eine erneute Adaptierung der verbliebenen Netzwerkgewichte statt, sodass diese die erzeugte Abweichung durch das Entfernen der unbedeutenden Gewichte kompensieren können. [21]

Der Einsatz reduzierender Verfahren ist mit Nachteilen verbunden. Zum einen muss ein bereits über-parametrisiertes Netz vorliegen. Die Trainingszeit bei über-parametrisierten Netzen ist in Regel höher also bei nicht über-parametrisierten Netzen. Der Reduktionsvorgang der neuronalen Netze stellt somit einen weiteren Arbeitsschritt dar, dessen zeitlicher Aufwand sich mit dem bereits erhöhten zeitlichen Aufwand für das Training des über-parametrisierten Netzes überlagert.

Konstruktive Ansätze dagegen verfolgen den Ansatz, durch Algorithmen ein anfangs mit einer minimalen Topologie ausgestattetes Netz inkrementell mit einer höheren Anzahl an Neuronen und Gewichten auszustatten, wobei jede Modifikation auf ihre Güte untersucht wird. Nach [49] kann dabei durch die Auswahl des Algorithmus erzwungen werden, dass jede weitere Netztopologie einen geringeren Fehler aufweist als die Vorgängertopologie.

Der Vorteil bei konstruktiven Ansätzen liegt darin, dass das Entstehen über-parametrisierter Netze von Anfang an unterbunden wird. Über-parametrisierte Netze neigen zum sogenannten *Overfitting*. Beim *Overfitting* sind Netze nicht in der Lage, die Zusammenhänge zwischen Eingangs- und Ausgangsgrößen ausreichend zu verallgemeinern bzw. zu generalisieren. Sie sind zwar in der Lage, den Trainingsdatensatz gut abzubilden, aber nicht in der Lage, für einen unbekannten Datensatz die Ausgangsgröße gut genug zu approximieren. Das gleiche Problem tritt beim *Underfitting* auf. Dabei ist das Netz unter-parametrisiert, d.h. es verfügt nicht über genug Parameter (Netzwerkgewichte), um die Zusammenhänge zwischen Eingangs- und Ausgangsgrößen gut genug abzubilden.

### **Trainingsalgorithmen**

Wie in Abschnitt 2.5.1 erwähnt, ist die Methode des Gradientenabstieges eine Möglichkeit der Gewichtsadaption. Jedoch ist diese Methode im Bereich der Gewichtsadaption für neuronale Netze zum größten Teil durch fortschrittlichere Methoden abgelöst worden. Die Limitierungen der Methode des Gradientenabstieges und alternative Algorithmen (z.B. konjugierte Gradienten-Methode, David-Fletcher-Powell-Algorithmus) sind in [58], [43] und [9] sehr gut dokumentiert. Eine weitere Alternative stellt der Levenberg-Marquadt-Algorithmus dar, dessen Implementierung in [4] diskutiert ist. Weitere Trainingsverfahren sind genetische Algorithmen, deren Einführung in [24] erörtert worden ist. Diese kommen nicht nur für die Adaptierung der Gewichte, sondern auch für die Adaptierung der Netzwerktopologie (Anzahl der Schichten, Anzahl der Neuronen, Auswahl der Aktivierungsfunktionen, etc.) zum Einsatz, siehe [7] und [23]. Weitere Möglichkeiten sind stochastische Methoden des Gradientenabstieges (stochastic gradient descent oder SGD), welche vor allem das Ziel haben, den Trainingsvorgang zu beschleunigen. [14] und [70] verändern dafür die Adaptions- bzw. Trainingsrate während der Trainings-

---

phase. Dies führt zu schnelleren bei konvexen, aber zu geringeren Konvergenzgeschwindigkeiten bei nicht-konvexen Fehlerfunktionen.

Bei der Auswahl von Algorithmen spielt außerdem die Anzahl der Netzwerkgewichte eine Rolle. Nach [58] eignet sich der Levenberg-Marquardt-Algorithmus bei mehreren Zehnten, der Davidon-Fletcher-Power-Algorithmus bei mehreren Hunderten und der konjugierte Gradienten-Algorithmus bei mehreren Tausenden von Netzwerkgewichten.

### Aktivierungsfunktionen

Wie in Abschnitt 2.5.1 erwähnt, gibt es für die Auswahl der Aktivierungsfunktion mehrere Möglichkeiten (u.a. die *logistische Sigmoid*-, die *tangentiale hyperbolische Sigmoidfunktion* und die *lineare Funktion*). Während die *lineare Funktion* die Eingangsgröße linear in die Ausgangsgröße transformiert, besteht die Aufgabe der *logistischen Sigmoid*- und der *tangentiale hyperbolischen Sigmoid*-Funktion darin, Nichtlinearitäten in das Netz zu induzieren. So transformiert die *logistische Sigmoid*-Funktion die Eingangsgröße in einen Wertebereich zwischen 0 und 1 und die *tangentiale hyperbolische Sigmoid*-Funktion die Eingangsgröße in einen Wertebereich zwischen -1 und 1. Generell eignet sich jedoch jede differenzierbare Funktion als Aktivierungsfunktion. Die Auswahl der Aktivierungsfunktion der letzten Schicht sollte jedoch auf den Wertebereich der Ausgangsgröße abgestimmt sein. Befindet sich der Wertebereich der Ausgangsgrößen nicht nur in einem Bereich zwischen -1 und 1, kommen dafür in der Regel *lineare* Aktivierungsfunktionen in der letzten Schicht, auch Ausgabeschicht genannt, zum Einsatz. Unterscheidungsmerkmal ist nicht nur der Wertebereich der Ausgangsgröße, sondern auch der Typ der Ausgangsgröße, welcher numerisch, kategorisch oder binär sein kann. Weiterhin haben die Aktivierungsfunktionen einen Einfluss auf die Konvergenzgeschwindigkeit bzw. Trainingsgeschwindigkeit des Netzes. Nach [9] weisen Netze mit *tangentiale hyperbolischen Sigmoid*-Funktionen höhere Konvergenzzeiten auf als Netze mit *logistischen Sigmoid*-Funktionen. In [36] dagegen sind die Vorzüge der *tangentiale hyperbolischen Sigmoid*-Funktion gegenüber anderen Aktivierungsfunktionen in Bezug auf mathematischen Eigenschaften während der Trainingsphase diskutiert.

### Initialisierung der Gewichte

Wie in Abschnitt 2.5.1 beschrieben, nehmen die Gewichte am Anfang der Trainingsphase zufällige Werte an. Erst im Laufe des Trainings findet die Adaption der Gewichte durch das Minimieren der Fehlerfunktion statt. Dieses findet durch das Anwenden verschiedener Algorithmen (Methode des Gradientenabstieges, Levenberg-Marquardt-Algorithmus etc.) statt. Diese sind allerdings nicht in der Lage, das globale Minimum der Fehlerfunktion zu identifizieren. In der Regel finden die Algorithmen lediglich lokale Minima der Fehlerfunktion. Somit können zwei neuronale Netze, welche mit unterschiedlichen Gewichten initialisiert wurden, aber deren Gewichte mit dem gleichen Algorithmus adaptiert wurden, unterschiedlichen Güten aufweisen, da der Trainingsprozess bei unterschiedlichen lokalen Minima abbricht. In der Literatur finden sich heuristische Ansätze, die initialen Werte für die Gewichte festzulegen. Zum Beispiel wählt [52] die initialen Gewichte zwischen der Eingabeschicht und der verdeckten Schicht auf Basis einer Kompen-



---

tenanalyse und die initialen Gewichte zwischen der verdeckten Schicht und der letzten Schicht auf Basis einer multiplen linearen Regression aus.

### **Anzahl der Schichten**

Die Netzwerktopologie ist ebenfalls durch die Anzahl der Schichten bestimmt. Nach [27] ist jedoch ein Netz mit nur einer verdeckten Schicht in der Lage, fast jede beliebige Funktion zu approximieren. Die Notwendigkeit einer zweiten Schicht kann sich dann ergeben, wenn stückweise stetige Funktionen zu approximieren sind.

### **Eignung für Regelungsaufgaben**

MLP-Netze gehören zur Gruppe der konnektionistischen Darstellung. Diese sind zwar in der Lage, Funktionen höherer Ordnung abzubilden, allerdings ist eine physikalische Deutung der Parameter in der Regel nicht möglich. Nach [60] sind MLP-Netze nur bedingt für regelungstechnische Aufgaben geeignet, da eine Veränderung der Stützstelle sich global auswirkt und die Gestaltung einer optimalen Netzwerktopologie aufwändig ist.

### **Schlussfolgerung**

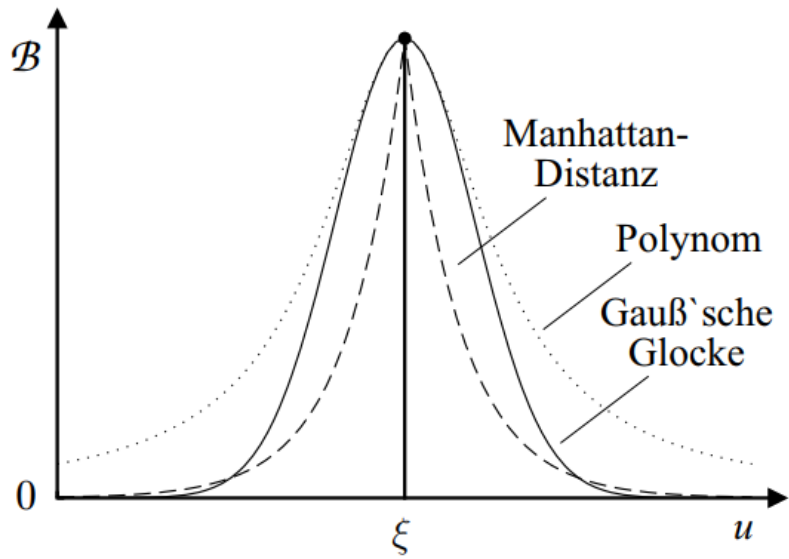
Das bisher vorgestellte *MLP-Netz* ist vor allem dazu geeignet, den statischen Zusammenhang zwischen Eingangs- und Ausgangsgrößen abzubilden. *Feedforward-Netze* sind rein vorwärts gerichtete Netze und definieren sich lediglich über die Eingabegrößen und die Parametrisierung der Gewichte. Somit sind *Feedforward-Netze* zustandsfrei. Eine weitere Gruppe statischer Funktionsapproximatoren stellen Netze mit überlagerten Basisfunktionen dar.

---

## **2.5.2 Netze mit überlagerten Basisfunktionen (statische Funktionsapproximatoren)**

---

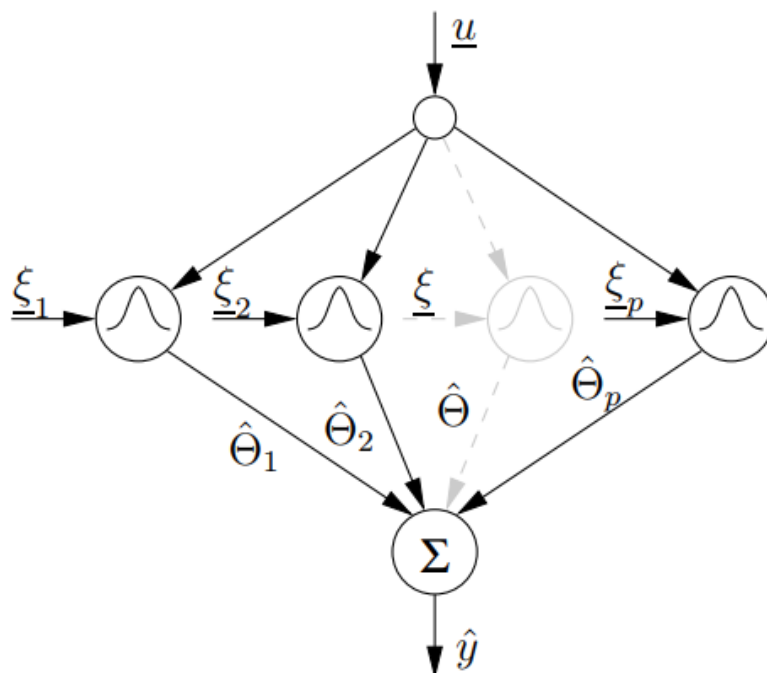
Nach [60] eignen sich vor allem Netze, welche nach der Methode der stützwertbasierten Approximation arbeiten, für regelungstechnische Aufgaben. Diese zeichnen sich vor allem durch kurze Adaptionszeiten und nachweisbare Stabilität aus. Im Gegensatz zu MLP-Netzen, welche zur Gruppe der konnektionistischen Darstellung gehören, verwenden solche Netze lokale Basisfunktionen. Beispiele solcher Basisfunktionen sind die Gaußsche Glockenkurve, Polynome, ein Dreiecksfenster oder die Manhattan-Distanz, siehe Abbildung 2.8.



**Abbildung 2.8:** Beispiele lokaler Basisfunktionen [60]

Zu den Netzstrukturen mit überlagerten Basisfunktionen gehören das Radial-Basis-Funktion (RBF)-, das General-Regression-Neural-Network (GRNN)-, das harmonisch aktivierte neuronale Netz (HANN) und das Local Linear Model Tree (LOLIMOT)-Netz.

In allen vier Netzstrukturen werden die lokalen Basisfunktionen als Aktivierungsfunktionen der Neuronen eingesetzt, siehe beispielsweise das RBF-Netz in Abbildung 2.9.



**Abbildung 2.9:** RBF-Netz mit  $p$  Stützstellen [60]

Zudem enthält jede lokale Basisfunktion  $\mathbf{B}$  einen Vektor  $\underline{\xi}_i$ , welcher die Lage des Zentrums, also das Maximum, der lokalen Basisfunktion angibt. Es findet nun eine Transformation der (mehrdimensionalen) Eingangsgrößen  $\underline{u}$  durch jede Basisfunktion statt, ausgedrückt durch die gesamte Aktivierungsfunktion  $\underline{\mathbf{A}}(\underline{u})$

$$\underline{\mathbf{A}}(\underline{u}) = [\mathbf{B}(\underline{u}, \underline{\xi}_1) \quad \mathbf{B}(\underline{u}, \underline{\xi}_2) \quad \dots \quad \mathbf{B}(\underline{u}, \underline{\xi}_p)]^T \quad (2.7)$$

Weiterhin existiert ein gleichlanger Gewichtsvektor  $\underline{\theta}$ , ausgedrückt durch

$$\underline{\theta} = [\theta_1 \quad \theta_2 \quad \dots \quad \theta_p]^T \quad (2.8)$$

Am Anfang des Trainings nehmen die Gewichte zufällige Werte an. Die Ausgangsgröße des untrainierten Netzes ergibt sich nun als Skalarprodukt aus dem Aktivierungs- und Gewichtsvektor

$$y(\underline{u}) = \underline{\theta}^T \underline{\mathbf{A}}(\underline{u}) \quad (2.9)$$

Es wird davon ausgegangen, dass eine trainierte Netzstruktur existiert, deren Approximationsfehler (Differenz zwischen der Ausgangsgröße des Netzes und der zu approximierenden Ausgangsgröße) unter einer beliebig kleinen Schranke liegt. Die Ausgangsgröße  $\hat{y}$  eines trainierten Netzes ist durch

$$\hat{y}(\underline{u}) = \hat{\underline{\theta}}^T \underline{\mathbf{A}}(\underline{u}) \quad (2.10)$$

beschrieben. Nun kann der Adaptionfehler zwischen der untrainierten und der trainierten Netzstruktur durch

$$e(\underline{u}) = \hat{y}(\underline{u}) - y(\underline{u}) = (\hat{\underline{\theta}}^T - \underline{\theta}^T) \underline{\mathbf{A}}(\underline{u}) \quad (2.11)$$

beschrieben werden. Somit kann wie bei den MLP-Netzen die Aufgabe der Adaption bei allen vier Netzstrukturen (RBF-, GRNN-, HANN- und LOLIMOT-Netz) auf die Anpassung der Gewichte reduziert werden. Jede Netzstruktur verwendet jedoch eine unterschiedliche Aktivierungsfunktion, aus der sich für jede Netzstruktur verschiedene Vor- und Nachteile ergeben.

Nach [60] weist das RBF-Netz, welches eine Gaußsche Glockenkurve als lokale Basisfunktion verwendet, mangelndes Monotonieverhalten zwischen Stützwerten und mangelndes Extrapolationsverhalten auf. Es kommt z.B. dazu, dass die Ausgangsgröße der Netzstruktur außerhalb

---

des Wertebereiches der Trainingsdaten gegen Null sinkt. Bei regelungstechnischen Anwendungen ist allerdings das Einhalten der Monotonie zwischen Stützwerten und ein nicht gegen null konvergierendes Extrapolationsverhalten erwünscht. [60]

Diese beiden Nachteile des RBF-Netzes umgeht das GRNN-Netz, da es normierte Gaußsche Glockenkurve als lokale Aktivierungsfunktion verwendet. Durch die Normierung ergeben sich zwei Vorteile gegenüber dem RBF-Netz, erstens wird ein monotoner Verlauf zwischen den Stützwerten sicher gestellt. Der zweite Vorteil besteht darin, dass die Ausgangsgröße des Netzes außerhalb des Wertebereiches der Trainingsdaten den nächstgelegenen Stützwert asymptotisch zustrebt und nicht gegen null geht. Dadurch ergibt sich ein besseres Extrapolationsverhalten für regelungstechnische Anwendungen. Das GRNN-Netz weist jedoch den Nachteil auf, dass für eine hohe Approximationsgenauigkeit eine große Anzahl an Stützstellen und ein somit hoher Rechenaufwand notwendig sind. Insbesondere für die Darstellung periodischer Signale kann dies problematisch sein. [60]

Aus diesem Grund kommen für diesen Zweck häufig HANN-Netze zum Einsatz, welche periodische Funktionen als Basisfunktion einsetzen. Nach dem Trainingsvorgang entsprechen die Netzwerkgewichte den Fourierkoeffizienten der reellen frequenzdiskreten Fourierreihe. Durch das Lernergebnis kann somit eine Aussage über die spektrale Zusammensetzung des identifizierten Signals gemacht werden, was für Überwachungs- und Diagnosezwecke interessant ist. HANN-Netze können aber auch erweitert werden, sodass nicht nur periodische, sondern auch nicht-periodische Signale berücksichtigt werden können. [60]

Eine neuere Netzwerkstruktur stellt das *LOLIMOT*-Netz dar, welches eine komplexe, nicht lineare Funktion durch einfachere lokale lineare Modelle mittels Zugehörigkeitsfunktionen überlagert. Ein entscheidendes Unterscheidungsmerkmal zu anderen Netzstrukturen ist, dass nicht nur eine Adaptierung der Gewichte, sondern schrittweise eine Netzstruktur aufgebaut wird. Dabei findet sowohl eine Optimierung in Bezug auf die Anzahl der Neuronen als auch in Bezug auf deren Gültigkeitsbereiches im Eingangsbereich statt. Entscheidende Vorteile des HANN-Netzes sind die schnellere Parameterberechnung und der relativ geringe Anstieg des Parameteranzahl bei steigender Eingangsdimension des Netzes. [60]

Sowohl MLP-Netze als auch Netze mit überlagerten Basisfunktionen stellen statische Funktionsapproximatoren auf. Eine weitere Klasse neuronaler Netze sind dynamische Funktionsapproximatoren, deren Grundlagen im nächsten Abschnitt erläutert sind.

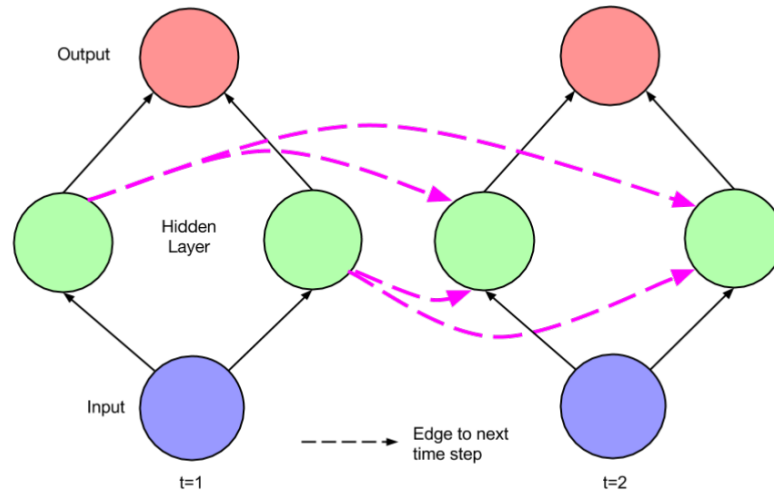
---

### 2.5.3 Rekurrente neuronale Netze (dynamische Funktionsapproximatoren)

---

Bei *rekurrenten Netzen* findet - im Gegensatz zu statischen Funktionsapproximatoren - eine Berücksichtigung des zeitlichen Verlaufs der Eingabe- und Ausgabegrößen statt. Dies wird durch den Einsatz von *rekurrenten Kanten* ermöglicht, welche zwei neuronale Netze in aufeinanderfolgenden Zeitschritten miteinander verbinden, siehe Abbildung 2.10.

Die *rekurrenten Kanten* erlauben es, dass die Netzwerkneuronen zum Zeitpunkt  $t$  nicht nur die Eingangsgröße  $x(t)$ , sondern auch die Ausgangsgröße  $h^{(t-1)}$  der verdeckten Neuronen des



**Abbildung 2.10:** Rekurrentes Netz [40]

vorherigen Zeitschrittes zum Eingang haben. Der Ausgang des Netzes  $\hat{y}^{(t)}$  zum Zeitpunkt  $t$  ergibt sich aus dem Ausgang  $h^{(t)}$  aller verdeckten Neuronen zum Zeitpunkt  $t$ . Auf diese Weise beeinflusst die Eingangsgröße  $x^{(t-1)}$  zum Zeitpunkt  $t-1$  die Ausgangsgröße  $\hat{y}^{(t)}$  zum Zeitpunkt  $t$ . Der Ausgang der verdeckten Neuronen  $h^{(t)}$  zum Zeitpunkt  $t$  kann nach [40] durch

$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{hh}h^{(t-1)} + b_h) \quad (2.12)$$

beschrieben werden. Dabei besteht die Matrix  $W^{hx}$  aus den konventionellen Gewichten, welche sich zwischen den Eingangsgrößen und den verdeckten Neuronen befinden. Die Matrix  $W^{hh}$  dagegen besteht aus *rekurrenten Gewichten*, welche sich zwischen den verdeckten Neuronen des aktuellen und des vorangegangenen Zeitschrittes befinden. Für die Adaptierung der Netzwerkgewichte kommt bei *rekurrenten Netzen* üblicherweise der *Backpropagation-Through-Time (BBTT)*- oder der *Real-Time Recurrent Learning (RTRL)*-Algorithmus zum Einsatz. Der Einsatz dieser Algorithmen ist jedoch mit Problemen verbunden, siehe Abschnitt 2.5.3.2.

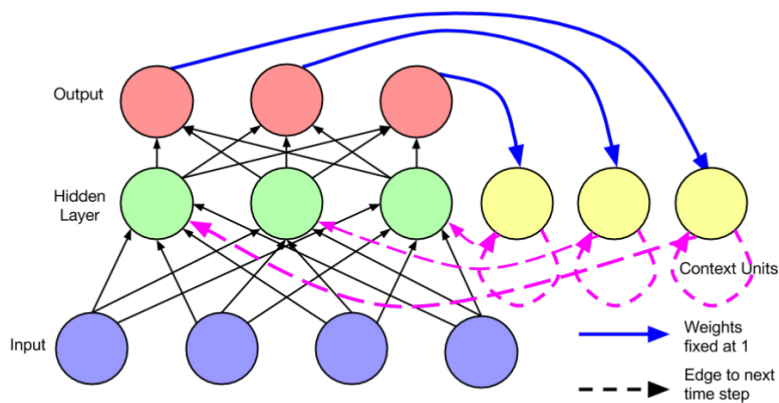
Im Laufe der Zeit haben sich unterschiedliche Varianten *rekurrenter Netze* entwickelt. Zunächst seien frühere Architekturen *rekurrenter Netze* erläutert.

### 2.5.3.1 Einfache rekurrente Netze (dynamische Funktionsapproximatoren)

#### Jordan-Netz

Das *Jordan-Netz* stellt eine der frühen rekurrenten Netzarchitekturen dar, siehe Abbildung 2.11.

Das *Jordan-Netz* entspricht einem *Feedforward-Netz* mit einer nachgeschalteten Zustandschicht, bestehend aus Zustandsneuronen. Diese sind über feste Gewichte (in der Regel haben diese den Wert 1) mit den Neuronen der Ausgangsschicht verbunden. Im nächsten Zeitschritt übergeben diese ihre Ausgangsgröße an die Neuronen der verdeckten Schicht. Durch die festen



**Abbildung 2.11: Jordan-Netz [40]**

Gewichte der *rekurrenten* Kanten findet eine Speicherung der Ausgangsgrößen aus dem letzten Zeitschritt in den Zustandsneuronen statt. Auf diese Weise ist das Netz in der Lage Zustände zu speichern, wozu konventionelle *Feedforward-Netze* nicht in der Lage sind. Der Ausgang der verdeckten Schicht  $h^{(t)}$  zum Zeitpunkt  $t$  ergibt sich nach [33] durch

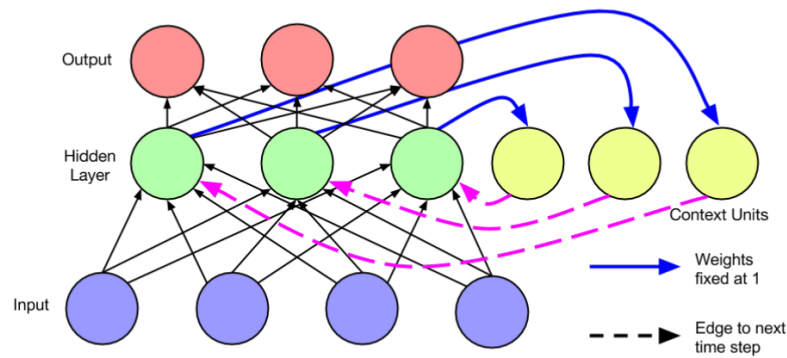
$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{hh}y^{(t-1)} + b^h) \quad (2.13)$$

Dabei enthält die Matrix  $W^{hx}$  die Gewichte zwischen den Eingangsgrößen und den Neuronen in der verdeckten Schicht. Die Matrix  $W^{hh}$  dagegen enthält die konstanten Gewichte der *rekurrenten Kanten*. Der Unterschied zu moderneren *rekurrenten Netzen* besteht darin, dass das Jordan-Netz wie ein *Feedforward-Netz* mittels des *Backpropagation*-Algorithmus trainiert wird, wobei die Gewichte der *rekurrenten Kanten* ignoriert bzw. nicht adaptiert werden. Nach [40] findet - im Gegensatz zu modernen *rekurrenten Netzen* - keine Rückführung des Fehlers über verschiedene Zeitschritte statt. Damit ist dieses Netz nur eingeschränkt für komplexere Dynamiken einsetzbar. [40] Ein weiterer Nachteil ist, dass die Anzahl der *rekurrenten Kanten* durch die Anzahl der Ausgabeneuronen, also durch die Problemstellung, festgelegt ist. Diesen Nachteil umgeht das *Elman-Netz*.

### Elman-Netz

Das *Elman-Netz* stellt ebenfalls eine der früheren *rekurrenten* Netzarchitekturen dar. Der entscheidende Unterschied zum *Jordan-Netz* ist, dass die *rekurrenten Kanten* nicht wie beim *Jordan-Netz* die Ausgabeneuronen mit den Zustandsneuronen, sondern die verdeckten Neuronen mit den Zustandsneuronen verbinden, siehe Abbildung 2.12.

Das *Elman-Netz* ist wie das *Jordan-Netz* in der Lage, Zustände zu speichern, indem die Zustandsneuronen den Zustand des vorherigen Zeitschrittes im nächsten Zeitschritt an die verdeckten



**Abbildung 2.12:** Elman-Netz [40]

Neuronen übergeben. Somit ergibt sich der Ausgang der verdeckten Schicht  $h^{(t)}$  zum Zeitpunkt  $t$  nach [16] durch

$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{hh}h^{(t-1)} + b^h) \quad (2.14)$$

Auch hier enthält die Matrix  $W^{(hx)}$  die Gewichte zwischen den Eingangsgrößen und den verdeckten Neuronen und die Matrix  $W^{(hh)}$  die konstanten Gewichte der *rekurrenten Kanten*. Wie bei den *Jordan-Netzen* findet das Training des Netzes über den *Backpropagation-Algorithmus* statt, wobei die konstanten Gewichte der *rekurrenten Kanten* ignoriert bzw. nicht adaptiert werden. Somit ergibt sich auch beim *Elman-Netz* das Problem, dass keine Rückführung des Fehlers über die Zeit stattfindet, sodass die Anwendbarkeit für komplexere Dynamiken beschränkt ist [40]. Beim *Elman-Netz* hängt jedoch die Anzahl der *rekurrenten Kanten* nicht von der Anzahl der Ausgabeneuronen, sondern von der Anzahl der verdeckten Neuronen ab. Da die Anzahl der Neuronen in der verdeckten Schicht frei ausgewählt werden kann, hängt die Anzahl der *rekurrenten Kanten* nicht von der Aufgabenstellung ab. Dies ist ein Vorteil gegenüber dem *Jordan-Netz*.

### 2.5.3.2 Herausforderungen bei rekurrenten Netzen

#### Verschwindende und explodierende Gradienten

*Rekurrente Netze* können - im Gegensatz zu *Feedforward-Netzen* - durch den Einsatz *rekurrenter Kanten* Informationen speichern. Jedoch ist die Anwendung bestimmter Algorithmen für das Netztraining mit Problemen verbunden.

Nach [25] ist der Einsatz von gradientenbasierten Trainingsalgorithmen, zu denen der *Backpropagation-Through-Time*- und der *Real-Time Recurrent*-Algorithmus gehören, beim Training *rekurrenter Netze* problematisch. Beim Netztraining können dabei zwei Effekte auftreten. Der erste Effekt umfasst das *Verschwinden der Gradienten*. Dabei nehmen die Gradienten verschwindend kleine Werte an. Dies führt dazu, dass die Adaptierung der Gewichte, also das

---

Training des Netzes, einen sehr hohen Zeitaufwand erfordert oder überhaupt nicht mehr stattfindet. Der zweite Effekt umfasst das *Explodieren der Gradienten*. Dabei nehmen die Gradienten sehr hohe Werte an, was zu oszillierenden Gewichten und somit zu einer schlechteren Robustheit des Netzes führt. [25]

[25] schlägt vor, das Problem der *verschwindenden Gradienten* durch eine neue Netzstruktur zu lösen. Bei dieser Netzstruktur kommen sogenannte *Long Short-Term Memory* (LSTM)-Zellen zum Einsatz, welche die konventionellen verdeckten Knoten ersetzen, siehe Abschnitt 2.5.3.3.

Die mathematischen Hintergründe und die Charakterisierung der Rahmenbedingungen, welche zu *verschwindenden* oder zu *explodierenden* Gradienten führt, sind in [51] erläutert. [51] schlägt vor, den Wertebereich der Gewichte durch Regularisierung einzuschränken. Der Einschränkung des Wertebereiches findet derart statt, dass eine zu große Veränderung des Gradienten unterbunden ist.

### Minimierung der Fehlerfunktion

Wie bei den *Feedforward-Netzen* tritt bei den *rekurrenten Netzen* das Problem auf, dass das Auffinden des globalen Minima der Fehlerfunktion kaum möglich ist.

Die Herausforderung besteht darin, durch die Adaption der Gewichte ein möglichst tiefes lokales Minima der Fehlerfunktion zu besetzen. Nach [11] sollte bei höher dimensionalen nicht-konvexen Fehlerfunktionen Sattelpunkten eine viel höhere Bedeutung zukommen als lokalen Minima, da die Anzahl der kritischen lokalen Minima bei größeren Netzen exponentiell sinkt. Demnach wird von [11] ein Sattelpunkt-freier Newton-Algorithmus vorgeschlagen. Diese Auslegung des Algorithmus unterscheidet sich von der konventionellen Newton-Methode darin, dass die Gewichtsadaption derart stattfindet, dass der Wert der Fehlerfunktion sich möglichst weit weg von den kritischeren Sattelpunkten befindet. Es konnten damit zwar Verbesserungen in Hinblick auf die Güte von *rekurrenten Netzen* erzielt werden, allerdings ist für die Anwendung des Sattelpunkt-freien Newton-Methode die rechenintensive Ermittlung der Hesse-Matrix notwendig. [11]

---

#### 2.5.3.3 Moderne rekurrente Netzarchitekturen (dynamische Funktionsapproximatoren)

---

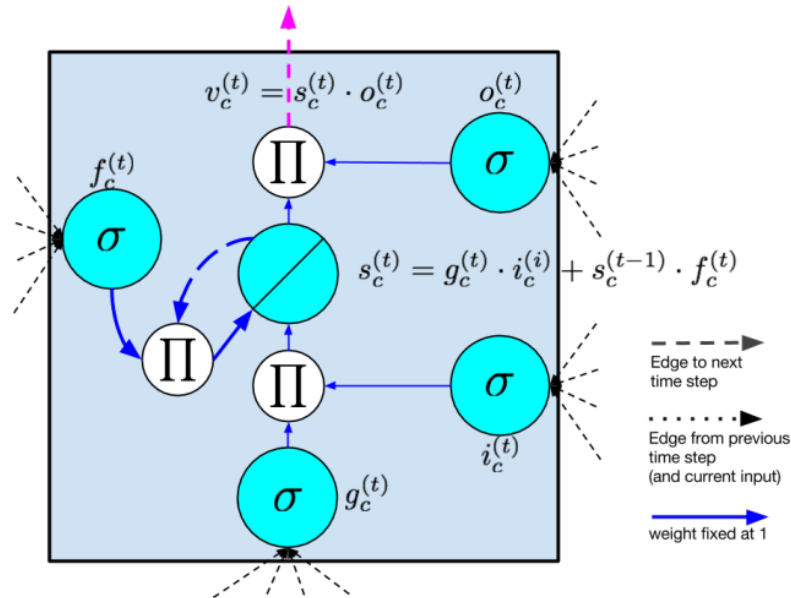
### Long Short-Term Memory-Netze

Wie in 2.5.3.2 erläutern, kann bei *rekurrenten Netzen* der Effekt *verschwindender Gradienten* vorkommen. Zu diesem Zweck führte [25] eine neue Netzarchitektur ein, bei dem die verdeckten Neuronen durch sogenannte *Long Short-Term Memory* (LSTM)-Zellen ersetzt sind, siehe Abbildung 2.13.

Nach [40] bestehen LSTM-Netzarchitekturen aus folgenden Komponenten:

- Eingangsneuron  $g_c^{(t)}$ : Das Eingangsneuron transformiert sowohl die Eingangsgrößen  $x^{(t)}$  des aktuellen Zeitpunktes  $t$  als auch die Ausgänge der verdeckten Schicht  $h^{(t-1)}$  aus dem





**Abbildung 2.13:** Long Short-Term Memory-Zelle [40]

letzten Zeitschritt durch eine Aktivierungsfunktionen. Mögliche Aktivierungsfunktionen sind die *tanh*- und die *sigmoid*-Funktion.

- Eingangsgate  $i_c^{(t)}$ : Das Eingangsgate enthält eine *sigmoid*-Aktivierungsfunktion. Das Gate hat wie das Eingangsneuron die aktuellen Eingangsgrößen  $x^{(t)}$  als auch die Ausgänge der verdeckten Schicht  $h^{(t-1)}$  aus dem vorherigen Zeitschritt zum Eingang. Die Ausgangsgröße des Gates  $i_c^{(t)}$  wird nun mit der Ausgangsgröße  $g_c^{(t)}$  des Eingangsneurons multipliziert. Damit steuert das Gate den Informationsfluss innerhalb der LSTM-Zelle. Ist die Ausgangsgröße des Gates  $i_c^{(t)} = 0$ , blockiert das Gate den Informationsfluss. Ist die Ausgangsgröße des Gates  $i_c^{(t)} = 1$ , gibt das Gate den Informationsfluss frei.
- Das Zustandsneuron  $s_c$  enthält eine lineare Aktivierungsfunktion und soll den internen Zustand der Zelle repräsentieren. Ein Merkmal ist die *rekurrente Kante* mit einem konstanten Gewicht, welche das Zustandsneuron mit sich selbst verbindet. Durch die *rekurrente Kante* können Fehler über mehrere Zeitschritte hinweg in das Zustandsneuron induziert werden. Durch den konstanten Fehlerfluss (in der Literatur oftmals als *constant error carousel* bezeichnet) wird das *Verschwinden* und *Explodieren* der Gradienten unterbunden.
- Löschungsgate  $f_c$ : Durch das Löschungsgate soll das Netz in die Lage versetzt werden, zu lernen, die Inhalte des Zustandsneurons zu löschen. Dies kann vor allem bei kontinuierlich laufenden Netzen hilfreich sein.
- Ausgangsgate: Die Ausgangsgröße  $v_c^{(t)}$  des Ausgangsgates berechnet sich aus der Multiplikation zwischen der Ausgangsgröße der Zustandsneurons  $s_c^{(t)}$  und der Ausgangsgröße des Ausgangsgates  $o_c^{(t)}$ . In der Regel findet vor der Multiplikation eine Transformation des

---

Zustands  $s_c^{(t)}$  durch eine  $\tanh$ -Funktion statt, um den Wertebereich der Ausgangsgrößen festzulegen. [40]

Im Gegensatz zu anderen *rekurrenten Netzen* ist es bei *LSTM-Netzen* möglich, den *Backpropagation-Through-Time-Algorithmus* anzuwenden, ohne dass es zu einem *Explodieren* oder *Verschwinden der Gradienten* kommt. Dies liegt daran, dass durch die konstante Fehlerrückführung durch die *rekurrente Kante* der Fehler im internen Zustand der *LSTM-Zelle* gespeichert wird. Auf diese Weise lernen die Gates, wann sie den Fehler blockieren oder durchlassen. *LSTM-Netze* sind vor allem dafür geeignet, langfristige Abhängigkeiten abzubilden. [40]

---

#### 2.5.3.4 Klassifikation rekurrenter Netze

---

Abschließend sei darauf hingewiesen, dass die eben vorgestellten Netze nur einen Bruchteil aller *rekurrenten Netze* darstellen. Nach [60] lassen sich *rekurrente Netze* gemeinhin aufteilen in Netze mit interner und externer Dynamik. Zu den Netzen mit interner Dynamik gehören die *vollrekurrenten* und die *partiell rekurrenten* Netze, wozu die in Abschnitt 2.5.3.1 vorgestellten Jordan- und Elman-Netze gehören. Davon unterscheiden sich sogenannte Netze mit externer Dynamik, wozu das *Time-Delay-Neural Network (TDNN)* in den beiden Variationen *Nonlinear-Output-Error (NOE)* und *Nonlinear-Autoregressiv-with-exogenous-Inputs (NARX)* gehört. Auch die Ansätze basierend auf der Volterra-Funktionalpotentialreihe in der *Nonlinear-Orthogonal-Basis-Functions (NOBF)*- und in der *Nonlinear-Finite-Impulse-Response (NFIR)*-Form gehören zu den Netzen mit externer Dynamik. [60]

---

## 2.6 Regelungskonzepte basierend auf Methoden des maschinellen Lernens

---

In Abschnitt 2.2 ist bereits das allgemeine Regelungskonzept vorgestellt worden. Nachfolgend erfolgte eine Reihe von Erläuterungen über verschiedene Typen von neuronalen Netzen, welche in der Lage sind, dynamische Nichtlinearitäten eines Systems durch die Adaption der Gewichte zu approximieren bzw. zu lernen. Nun ist es denkbar, auf Basis neuronaler Netze ein Blackbox-Modell eines nichtlinearen dynamischen Systems zu entwickeln, welches entweder als Hilfsglied in einem Regelkreis oder direkt als Regler fungiert. Dafür sind verschiedene Regelungskonzepte in der Literatur gegenwärtig. Es erfolgt nun eine Übersicht der Regelungskonzepte auf Basis der Methoden des maschinellen Lernens in den Kategorien *überwachtes* und *bestärkendes Lernen*.

---

### 2.6.1 Überwachtes Lernen

---

Im Bereich des *überwachten Lernens* gibt es eine Reihe von verschiedenen Regelungskonzepten auf Basis neuronaler Netze, für die nun ein Überblick gegeben wird.

### 2.6.1.1 Direkte inverse Regelung

Das von [68] erstmals eingeführte Regelungskonzept der *direkten inversen Regelung* basiert auf der Annahme, dass für ein dynamisches System eine inverse Abbildung des Ausgangszustandes in den Eingangszustand existiert. Ist dies der Fall, kann ein neuronales Netz offline, also nicht während der Betriebseinsatzes, so trainiert werden, dass es die inverse Dynamik der Regelstrecke (in Abbildung 2.14 als Roboter bezeichnet) abbildet.

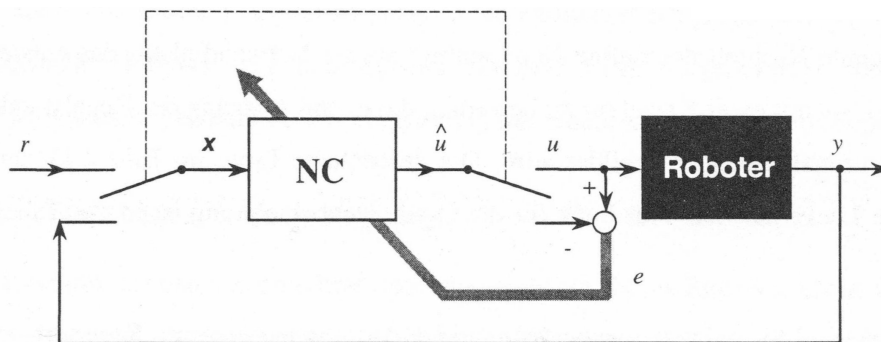


Abbildung 2.14: Direkte inverse Regelung mit offline-Netztraining [62]

Daraufhin ist eine direkte Schaltung der neuronalen Netzes in den Regelkreis möglich. In der Regel findet dann eine Übergabe der Sollgröße  $r$  an das invertierte neuronale Netz statt, siehe auch [3]. Die Ausgangsgröße des neuronalen Netzes entspricht dann den Stellgrößen der Regelstrecke. Entsprechen die Ausgangsgrößen der Regelstrecke  $y$  nicht im ausreichenden Maße den Sollgrößen  $r$ , ist es zweckmäßig, das Netz offline erneut zu trainieren. Jedoch findet während des Betriebes keine direkte Fehlerrückführung statt, sodass eine stabile Regelstrecke Voraussetzung ist. Offline trainierte Netze gehören zur Klasse der nicht adaptiven Regler. Beim Konzept der *direkten inversen Regelung* ist es auch möglich, die Adaption der Netzwerkgewichte während des Betriebes, also online, vorzunehmen, siehe Abbildung 2.15.

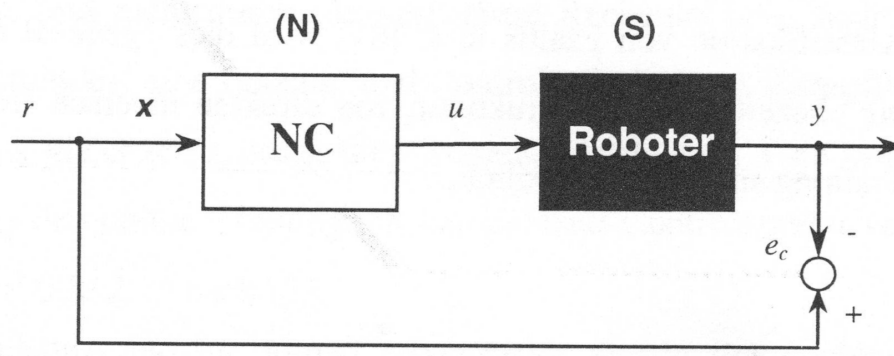


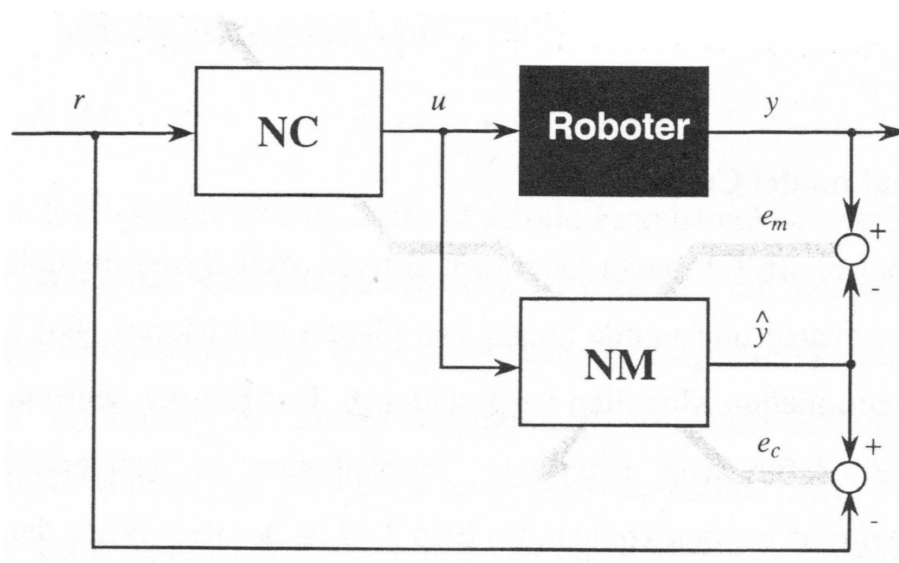
Abbildung 2.15: Direkte inverse Regelung mit online-Netztraining [62]

Diese Regler gehören zur Klasse der adaptiven Regler, da eine Anpassung der Regelparameter während des Betriebs stattfindet. Dabei soll das Netz möglichst den Fehler  $e_c$  minimieren, was jedoch nicht immer möglich ist, da die Jacobi-Matrix des nichtlinearen dynamischen Regelstrecke in der Regel nicht bekannt ist. Die Kenntnis der Jacobi-Matrix ist allerdings für Minimierung der Fehlerfunktion unabdingbar. Nach [62] ist es jedoch möglich, die Elemente der Jacobi-Matrix numerisch mit einer Differenzengleichung zu ermitteln. Ein weiterer Nachteil ergibt sich dadurch, dass die Regelstrecke anfangs mit einem untrainierten neuronalen Netz betrieben wird, was zu unkontrolliertem Systemverhalten führen kann.

Häufig kommen *inverse neuronale* Regler in Kombination mit anderen Reglern zum Einsatz. Beispielsweise nutzt [44] ein neuronales Netz in Kombination mit einem P-Regler zur Steuerung eines Industrieroboters. [48] und [34] kombinieren ein *inverses neuronales* Netz mit einem PD-Regler zur Regelung eines gekoppelten Pendelsystems und zur Regelung eines Industrieroboters. [6] dagegen nutzt einen PID-Regler in Kombination mit einem neuronalen Netz, um die optimalen Schweißparameter beim Lichtbogenschweißen zu ermitteln.

#### 2.6.1.2 Indirekte neuronale Regelung mit Referenzmodell

Wie in Abschnitt 2.6.1.1 erläutert, stellt sich bei der *direkten inversen* neuronalen Regelung das Problem, dass in vielen regelungstechnischen Anwendungen die Jacobi-Matrix der dynamischen nichtlinearen Regelstrecke nicht bekannt ist. Damit gestaltet sich die Gewichtsadaptierung des inversen neuronalen Netzes als schwierig, um die Differenz zwischen der Ausgangsgröße des dynamischen Systems  $y$  und der Sollgröße  $r$  zu minimieren. Dieses Problem umgeht die indirekte neuronale Regelung mit Referenzmodell, siehe Abbildung 2.16.

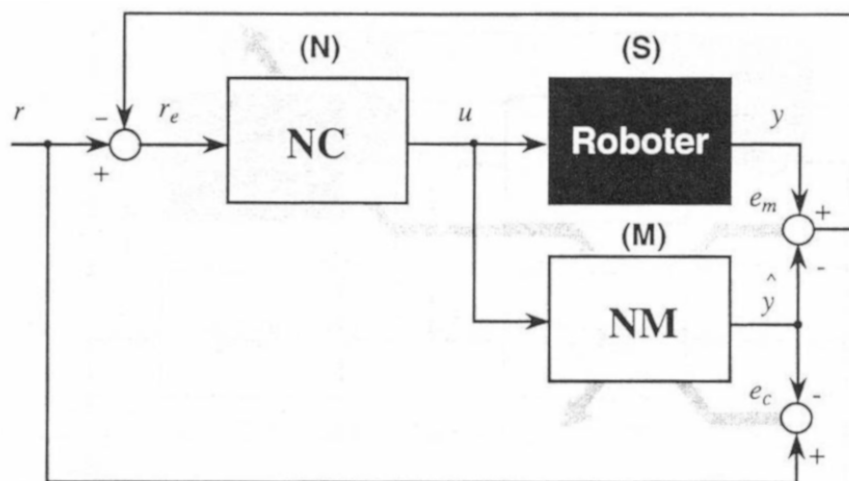


**Abbildung 2.16:** Indirekte neuronale Regelung mit Referenzmodell [62]

Im Unterschied zur *direkten inversen* neuronalen Regelung ist ebenfalls ein nicht invertiertes neuronales Netz Bestandteil der Regelstruktur (in Abbildung als NM bezeichnet), siehe [47], [8] und [29]. Dieses ist derartig offline trainiert, dass es die Eingangsgröße der Regelstrecke  $u$  in die Ausgangsgröße  $y$  der Regelstrecke transformiert. Während des Betriebs findet keine Gewichtsadaptierung des Netzes NM mehr statt, da dieses nun als Referenzmodell dient. Dagegen findet die Gewichtsadaptierung des Netzes NC während des Betriebes (online) derartig statt, dass die Differenz zwischen der Ausgangsgröße des Netzes NM  $\hat{y}$  und der Sollgröße  $r$  minimiert wird. Dies ist deshalb möglich, da der Fehler  $e_c$  durch das bekannte Netz NM hindurch propagiert (z.B. durch den Backpropagation-Algorithmus) werden kann. Die Regelgüte dieser Regelstruktur hängt sehr stark von der Approximationsfähigkeit des NM-Netzes in Bezug auf die aktuelle Regelstrecke ab. Kommt es im Laufe der Zeit zu Veränderungen in Bezug auf das dynamische Verhalten der Regelstrecke, muss das NM-Netz neu trainiert werden, um es an das aktuelle Verhalten der Regelstrecke anzupassen. Zudem gibt es keine Rückführung des Fehlers  $e_m$ , welcher die Differenz zwischen dem realen Systemausgang  $y$  und der Sollgröße  $r$  abbildet. Die Stabilität dieser Regelstruktur ist in [66] und [57] diskutiert.

### 2.6.1.3 Internal Model Control (IMC)

Wie im letzten Abschnitt erwähnt, findet bei der *indirekten neuronalen Regelung mit Referenzmodell* keine Rückführung des Fehlers  $e_m$  statt. Davon unterscheidet sich die IMC-Regelstruktur, da hier der Fehler  $e_m$  als zusätzliche Eingangsgröße für das Netz NC dient, siehe Abbildung 2.17.



**Abbildung 2.17:** Internal Model Control-Regelkreis [62]

Wie bei der *indirekten neuronalen Regelung mit Referenzmodell* kommt das invertierte Netz NC zum Einsatz, welches die Stellgrößen  $u$  für die Regelstrecke vorgibt. Das offline trainierte Netz NM dient wieder als Referenzmodell und generiert die Ausgangsgröße  $\hat{y}$ . Die Gewichtsadaptierung des Netzes NC findet nun derartig statt, dass der Fehler  $e_c$ , welcher die Differenz zwischen der Sollgröße  $r$  und der Ausgangsgröße  $\hat{y}$  abbildet, minimiert wird. Dies ist möglich, da der

Fehler durch die bekannte Netzstruktur NM propagiert werden kann. Nach [37] ist es üblich, bei IMC-Regelstrukturen Filter zur Erhöhung der Robustheit einzusetzen. Zudem diskutiert [37] den Stabilitätsnachweis und die Auswirkungen der Auswahl von Netzstrukturen (Feed-Forward vs. rekurrentes Netz) bei IMC-Regelkreisen.

#### 2.6.1.4 Feedback Linearisierung mit neuronalen Netzen

Die Grundidee beim Verwenden linearisierter neuronaler Modelle besteht darin, nichtlineare Systeme für einen bestimmten Arbeitspunkt in lineare Modelle zu transformieren. Zunächst findet das Training eines neuronalen Netzes derartig statt, dass es die Dynamik der Regelstrecke abbildet. Daraufhin findet die Linearisierung des Modells bei einem bestimmten Arbeitspunkt statt, sodass das diskrete Modell durch

$$y(k+d) = f(\mathbf{y}, \mathbf{u}) + b(\mathbf{y}, \mathbf{u}) \cdot u(k+1) \quad (2.15)$$

mit  $\mathbf{y} = [y(k), y(k-1), \dots, y(k-n+1)]^T$  als diskreter Zustandsvektor,  $\mathbf{u} = [u(k), u(k-1), \dots, u(k-m+1)]^T$  als diskreter Stellgrößenvektor und  $b(\mathbf{y}, \mathbf{u})$  als weiterer diskreter Zustandsvektor beschrieben ist. Daraufhin kann die Stellgröße des nächsten Zeitschrittes  $u(k+1)$  nach [46] durch

$$u(k+1) = \frac{y_d(k+d) - f(\mathbf{y}, \mathbf{u})}{b(\mathbf{y}, \mathbf{u})} \quad (2.16)$$

ermittelt werden. Diese führt dazu, dass die Zustandsgröße  $y(k+d)$  gegen die Sollgröße  $y_d(k+d)$  konvergiert. Dabei approximiert das neuronale Netz die Funktionen  $f$  und  $b$ . Die Übergabe der Stellgröße  $u(k+1)$  kann dann durch einen vorher trainierten linearen Regler an die Regelstrecke stattfinden, siehe Abbildung 2.18.

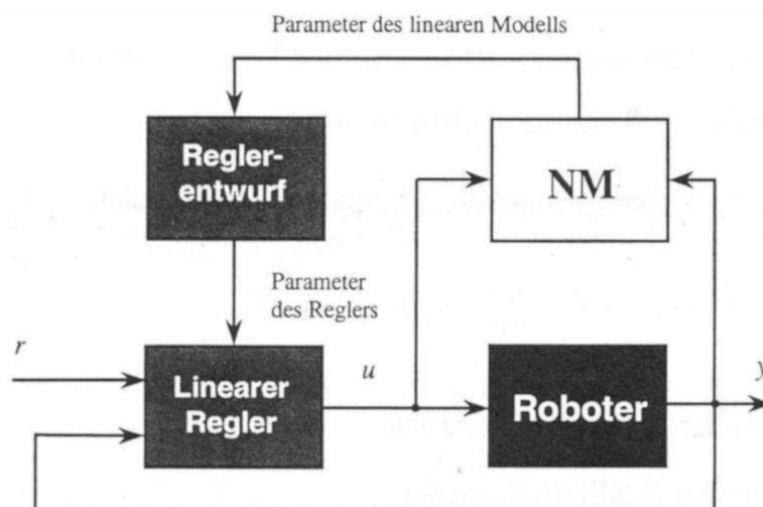
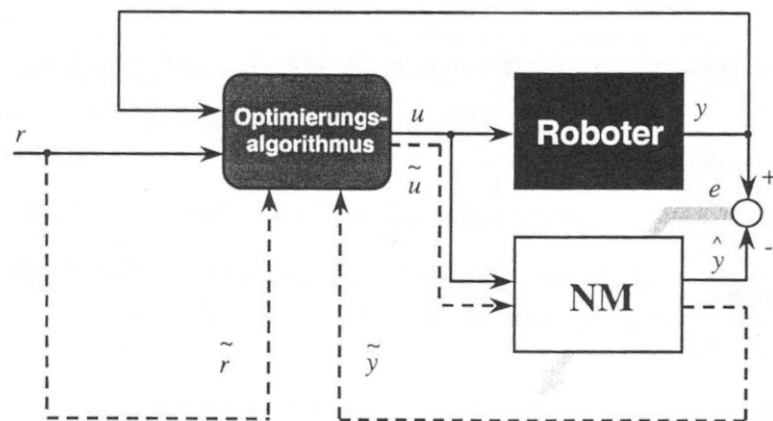


Abbildung 2.18: Feedback Linearisierung mit neuronalen Netzen [62]

Das neuronale Netz ist also nicht direkt als Regler, sondern als Hilfglied für den Entwurf bzw. Training des linearen Reglers verantwortlich. Die Bezeichnung der auf diesem Prinzip basierenden Regelsysteme lautet *NARMA-L2-Control*. Bei dieser Methode kommen sowohl MLP- [39] als auch RBF-Netze [12] zum Einsatz. Nach [39] kann sowohl eine Änderung des Regelgesetzes während des Betriebs (adaptive Regelung) als auch die Stabilität solcher Regelkonzepte nachgewiesen werden.

### 2.6.1.5 Neuronale prädiktive Regelung

Prädikative Regelungen verwenden neuronale Netze, um die Ausgangsgröße der Regelstrecke für einen zukünftigen Zeithorizont zu prädizieren, siehe Abbildung 2.19.



**Abbildung 2.19:** Neuronale prädiktive Regelung [62]

Dabei ist das neuronale Netz derartig trainiert, dass es die Eingangsgröße  $u(k)$  der Regelstrecke in die zukünftige Ausgangsgröße  $y(k+n)$  transformiert. Nach dem Training prädiziert das neuronale Netz die Ausgangsgröße  $\hat{y}(k+n)$  über  $n$  Zeitschritte hinweg. Nach der Prädiktion kommt eine Fehlerfunktion zum Einsatz, welche üblicherweise die Prädiktionsfehler und die Veränderung der Eingangsgrößen berücksichtigt. Abschließend minimiert ein Optimierungsalgorithmus die bekannte Fehlerfunktion.

Bei prädiktiven neuronalen Regelungen kommen in der Regel MLP- [45], RFB- [2] und Neuro-Fuzzy-Netzwerke [41] zum Einsatz. [50] und [2] trainierten die Netze online (während des Betriebsprozesses), sodass hier von adaptiven prädiktiven neuronalen Netzen gesprochen werden kann. Zudem passte [50] die Modellstruktur und das Regelgesetz so an, dass stabile Regelkreise entstehen. Als Optimierungsalgorithmen kommen unter anderem der Levenberg-Marquardt [45]- und der Fuzzy-Gradienten-Algorithmus zum Einsatz. Weitere Beispiele sind der lineare quadratische Gaußsche-Regler [35] und das sequentielle quadratische Programmieren [53] [50]. Auch neuronale Netze können als Optimierer fungieren [38].

---

#### 2.6.1.6 Regelungssysteme mit neuronalen Netzen als Kompensatoren

---

Häufig kommen neuronale Netze nicht als Regler, sondern als Hilfsglieder in Regelkreisen zum Einsatz, um Unsicherheiten dynamischer Systeme zu kompensieren. Dafür ist eine Zerlegung des Systems in der Zustandsraumdarstellung in bekannte und nicht bekannte Funktionen zweckmäßig. Die unbekannte Funktion ist von dem Zustandsvektor und dem Stellgrößenvektor abhängig und repräsentiert die Unsicherheiten des dynamischen Systems. Häufig kann dann ein Regelgesetz aufgestellt werden, in welchem die unbekannte Funktion durch ein neuronales Netz approximiert werden kann. Dadurch ergibt sich dann eine Stellgröße, welche die Unsicherheit des dynamischen Systems mit berücksichtigt bzw. kompensiert. In solchen hybriden Regelssystemen kommen neuronale Netze in Kombination mit P- [56], PID- [26], Sliding Mode- [22], Back-Stepping- [32], Feedback Linearisierungs- [69] und Modell-Referenz-Reglern [71] zum Einsatz. Dabei finden häufig MLP- [10], RFB- [69], Neuro-Fuzzy- [22], Sigma-Pi- [26] und Wavelet-basierte neuronale Netze [28] Anwendung.

---

#### 2.6.2 Bestärkendes Lernen

---

Die in Abschnitt 2.6.1 vorgestellten Regelungskonzepte basieren auf neuronalen Netzen, welche mit Hilfe von Trainingsdaten parametrisiert sind. Das heißt, dass die zu approximierende Größe (z.B. die Stößelhöhe für ein nicht invertiertes Netz oder die beiden Spindelvorschübe und der Exzenterwinkel für ein invertiertes Netz) in den Trainingsdaten vorgegeben sind. Allerdings ist es nicht für alle Anwendungen möglich, aus Experimenten die zu approximierende Größe zu ermitteln. Weiterhin stellt das experimentelle Ermitteln der Trainingsdaten einen Aufwand dar, der mit der Anzahl der Freiheitsgrade des Systems steigt. Aus diesen Gründen werden beim *bestärkenden Lernen* keine experimentell ermittelten Werte der zu approximierenden Größe vorgegeben. Beim *bestärkenden Lernen* befindet sich das System in einem sogenannte Markov Decision Process (MDP), bei dem das System versucht, eine größtmögliche Belohnung (*rewards*) zu erzielen. Dabei führt das System eine Menge von Aktionen (*actions*) aus, die dazu führen, dass sich das System durch verschiedene Zustände (*states*) bewegt. Welchen Zustand das System erreicht, kann durch eine Wahrscheinlichkeit, welche von den gewählten Aktionen und dem aktuellen Zustand abhängt, beschrieben werden. Diese Wahrscheinlichkeit ist durch eine Übergangsfunktion definiert. Dabei bekommt das System nach jedem Übergang in einen neuen Zustand auf Grundlage seines Ausgangszustands und seiner gewählten Aktion eine Belohnung. Dabei verwendet das System eine Strategie (*policy*), welche jedem Zustand eine Aktion zuordnet. Das Ziel besteht nun darin, eine Strategie zu finden, welche den höchsten Gesamtgewinn (die Summe aller erhaltenen Belohnungen) erreicht. In der Regel kommen dabei zwei Vorgehensweisen zum Einsatz, zum einen die *policy iteration (PI)* und die *value iteration (VI)*. Bei der *value iteration* findet im ersten Schritt die Ermittlung der optimalen Kostenfunktion statt. Im zweiten Schritte kann dann die optimale Strategie ermittelt werden (*policy extraction*). Bei der *policy iteration* findet im ersten Schritt die Strategiebewertung (*policy evaluation*) und im



---

zweiten Schritt die Verbesserung der Strategie (*policy improvement*) statt. Diese beiden Schritten werden so lange iterativ ausgeführt, bis die Verbesserung der Strategie konvergiert. Die Anwendung des *value iteration* in Kombination mit der *policy extraction* setzt die Kenntnis der Übergangsfunktionen und der Belohnungsfunktion voraus. Diese sind oftmals nicht bekannt. Im Bereich des *Reinforcement Learning* haben sich für diese Fälle eine Reihe von Algorithmen durchgesetzt, welche sich folgendermaßen klassifizieren lassen:

- Model Based Approaches
- Temporal Difference Learning
- Q-Values/Q-Learning
- Exploration
- State-Action-Reward-State-Action (SARSA)-Algorithmus
- Double Q-Learning
- Deep Q-Learning

[54] benutzte die Methode der *policy iteration*, um eine adaptive optimale Regelung für einen automatischen Spannungsregler zu entwickeln. Dabei verwendete [54] eine sogenannte *actor-critic*-Struktur. Der *actor* parametrisiert dabei die Regelungsstrategie und der *critic* approximiert die mit der Regelungsstrategie verbundenen Kosten, welche die Regelgüte beschreibt. Die Regelstrategie kann iterativ so lange verbessert werden, bis die Kosten konvergieren, also die optimale Regelstrategie gefunden wurde. Dabei ist nur eine teilweise Erkenntnis der internen Systemdynamik notwendig. Die Matrix  $A$  in der Zustandsraumdarstellung  $\dot{x}(t) = Ax(t) + Bu(t)$  wird dabei als nicht bekannt und die Matrix  $B$  als bekannt vorausgesetzt.

[67] verwendet ebenfalls eine *actor-critic*-Struktur, um eine optimale Lastfrequenzsteuerung für ein Kraftwerk zu entwickeln. Auch hier ist die Erkenntnis internen Systemdynamik (repräsentiert durch die Matrix  $A$ ) nicht, aber dafür die Erkenntnis der Matrix  $B$  notwendig.



---

## 3 Ausblick



---

# Literaturverzeichnis

- [1] *Appendix G: Thirty Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation.* In: WIDROW, B. und E. WALACH (Hrsg.): *Adaptive inverse control*, IEEE Press Series on Power Engineering, S. 409–474. IEEE Press Wiley-Interscience and IEEE Xplore, Piscataway, New Jersey and Hoboken, NJ and Piscataway, New Jersey, 2008.
- [2] ALEXANDRIDIS, A. und H. SARIMVEIS: *Nonlinear adaptive model predictive control based on self-correcting neural network models.* *AIChE Journal*, 51(9):2495–2506, 2005.
- [3] ALMUSAWI, A. R. J., L. C. DÜLGER und S. KAPUCU: *A New Artificial Neural Network Approach in Solving Inverse Kinematics of Robotic Arm (Denso VP6242).* *Computational intelligence and neuroscience*, 2016:5720163, 2016.
- [4] ANASTASSIOU und KOLLIAS: *Adaptive training of multilayer neural networks using a least squares estimation technique.* In: *IEEE International Conference on Neural Networks 1993*, S. 383–390 vol.1, Piscataway, March 1993. IEEE.
- [5] ANDERL, R., A. PICARD, Y. WANG, S. DOSCH, B. KLEE und J. BAUER: *Leitfaden Industrie 4.0: Orientierungshilfe zur Einführung in den Mittelstand*, 2018.
- [6] ANDERSEN, K., G. E. COOK, G. KARSAI und K. RAMASWAMY: *Artificial neural networks applied to arc welding process modeling and control.* *IEEE Transactions on Industry Applications*, 26(5):824–830, 1990.
- [7] BAYER, J., D. WIERSTRA, J. TOGELIUS und J. SCHMIDHUBER: *Evolving Memory Cell Structures for Sequence Learning.* In: ALIPPI, C., M. POLYCARPOU, C. PANAYIOTOU und G. ELLINAS (Hrsg.): *Artificial neural networks - ICANN 2009*, Bd. 5769 d. Reihe *Lecture Notes in Computer Science*, S. 755–764. Springer, Berlin, 2009.
- [8] BEN NASR, M. und M. CHTOUROU: *Neural network control of nonlinear dynamic systems using hybrid algorithm.* *Applied Soft Computing*, 24:423–431, 2014.
- [9] BISHOP, C. M.: *Neural networks for pattern recognition.* Oxford Univ. Press, Oxford, Reprinted. Aufl., 2010.
- [10] CHENG, L., Z.-G. HOU und M. TAN: *Adaptive neural network tracking control for manipulators with uncertain kinematics, dynamics and actuator model.* *Automatica*, 45(10):2312–2318, 2009.
- [11] DAUPHIN, Y., R. PASCANU, C. GULCEHRE, K. CHO, S. GANGULI und Y. BENGIO: *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.*
- [12] DENG, H., H.-X. LI und Y.-H. WU: *Feedback-linearization-based neural adaptive control for unknown nonaffine nonlinear discrete-time systems.* *IEEE transactions on neural networks*,

- 
- 19(9):1615–1625, 2008.
- [13] DÖBEL, I., M. LEIS, M.-M. VOGELSANG, D. NEUSTROEV, H. PETZKA, S. RÜPING, A. VOSS, M. MENGELE und J. WELZ: *Maschinelles Lernen: Kompetenzen, Anwendungen und Forschungsbedarf*, 2018.
- [14] DUCHI, J., E. HAZAN und Y. SINGER: *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*, 2010.
- [15] DURIEZ, T., S. L. BRUNTON und B. R. NOACK: *Machine Learning Control - Taming Nonlinear Dynamics and Turbulence*, Bd. 116 d. Reihe *Fluid Mechanics and Its Applications*. Springer International Publishing, Cham and s.l., 2017.
- [16] ELMAN, J. L.: *Finding Structure in Time*. *Cognitive Science*, 14(2):179–211, 1990.
- [17] FUMUMOTO, Y., S. OWAKI und M. NAKAMURA: *Effect of number of neurons of a neural-network on compensation performance of SPM non-linear waveform distortion*. In: *2017 Opto-Electronics and Communications Conference (OECC) and Photonics Global Conference (PGC)*, S. 1–2, Piscataway, NJ, 2017. IEEE.
- [18] GERWIN, D.: *Manufacturing Flexibility: A Strategic Perspective*. *Management Science*, 39(4):395–410, 1993.
- [19] GROCHE, P., M. SCHEITZA, M. KRAFT und S. SCHMITT: *Increased total flexibility by 3D Servo Presses*. *CIRP Annals*, 59(1):267–270, 2010.
- [20] GROCHE, P. und R. SCHNEIDER: *Method for the Optimization of Forming Presses for the Manufacturing of Micro Parts*. *CIRP Annals*, 53(1):281–284, 2004.
- [21] HAN, S., J. POOL, J. TRAN und W. J. DALLY: *Learning both Weights and Connections for Efficient Neural Networks*.
- [22] HAN, S. I. und K. S. LEE: *Sliding mode-based friction control with adaptive dual friction observer and intelligent uncertainty compensator*. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 223(8):1129–1147, 2009.
- [23] HARRIS, S. A. und T. SAMAD: *Optimizing Neural Networks with Genetic Algorithms*. *Proceedings of the American Power Conference*, 54 pt 2:1138–1143, 1992.
- [24] HASSOUN, M. H.: *Fundamentals of Artificial Neural Networks*. *Proceedings of the IEEE*, 84(6):906, 1996.
- [25] HOCHREITER, S., Y. BENGIO und P. FRASCONI: *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*. In: KOLEN, J. und S. KREMER (Hrsg.): *Field Guide to Dynamical Recurrent Networks*. IEEE Press, 2001.
- [26] HONG, C.-H., K.-C. CHOI und B.-S. KIM: *Applications of adaptive neural network control to an unmanned airship*. *International Journal of Control, Automation and Systems*, 7(6):911–917, 2009.
- [27] HORNIK, K., M. STINCHCOMBE und H. WHITE: *Multilayer feedforward networks are universal*

---

approximators. *Neural Networks*, 2(5):359–366, 1989.

- [28] HSU, C.-F., K.-H. CHENG und T.-T. LEE: *Robust wavelet-based adaptive neural controller design with a fuzzy compensator*. *Neurocomputing*, 73(1-3):423–431, 2009.
- [29] HUSSAIN, M. A. und L. S. KERSHENBAUM: *SIMULATION AND EXPERIMENTAL IMPLEMENTATION OF A NEURAL-NETWORK-BASED INTERNAL-MODEL CONTROL STRATEGY ON A REACTOR SYSTEM*. *Chemical Engineering Communications*, 172(1):151–169, 1999.
- [30] ISERMANN, R.: *Modellbasierte Überwachung und Fehlerdiagnose von kontinuierlichen technischen Prozessen*. *at - Automatisierungstechnik*, 58(6), 2010.
- [31] ISERMANN, R. und M. MÜNCHHOF: *Identification of Dynamic Systems: An Introduction with Applications*. *Advanced Textbooks in Control and Signal Processing*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [32] JOLLY, K. G., R. SREERAMA KUMAR und R. VIJAYAKUMAR: *An artificial neural network based dynamic controller for a robot in a multi-agent system*. *Neurocomputing*, 73(1-3):283–294, 2009.
- [33] JORDAN, M. I.: *Serial Order: A Parallel Distributed Processing Approach*. In: DONAHOE, J. W. und V. P. DORSEL (Hrsg.): *Neural-network models of cognition*, Bd. 121 d. Reihe *Advances in Psychology*, S. 471–495. Elsevier, Amsterdam and New York, 1997.
- [34] JUNG, S. und T. C. HSIA: *A new neural network control technique for robot manipulators*. In: *Proceedings of the 1995 American Control Conference*, S. 878–882, Piscataway, N.J., 1995. Distributed through the IEEE Service Center.
- [35] K-KARAMODIN, A. und H. H-KAZEMI: *Semi-active control of structures using neuro-predictive algorithm for MR dampers*. *Structural Control and Health Monitoring*, 5(4):n/a–n/a, 2009.
- [36] KALMAN, B. L. und S. C. KWASNY: *Why tanh: choosing a sigmoidal function*. In: *IEEE-INNS International Joint Conference on Neural Networks, Baltimore, 1992*, S. 578–581, Piscataway, June 1992. IEEE.
- [37] KAMBHAMPATI, C., R. J. CRADDOCK, M. THAM und K. WARWICK: *Inverse model control using recurrent networks*. *Mathematics and Computers in Simulation*, 51(3-4):181–199, 2000.
- [38] KOKER, R.: *Design and performance of an intelligent predictive controller for a six-degree-of-freedom robot using the Elman network*. *Information Sciences*, 176(12):1781–1799, 2006.
- [39] LEEGHIM, H., I.-H. SEO und H. BANG: *Adaptive nonlinear control using input normalized neural networks*. *Journal of Mechanical Science and Technology*, 22(6):1073–1083, 2008.
- [40] LIPTON, Z. C., J. BERKOWITZ und C. ELKAN: *A Critical Review of Recurrent Neural Networks for Sequence Learning*.
- [41] LIU, X.-J. und J.-Z. LIU: *Neuro-Fuzzy Generalized Predictive Control of Boiler Steam Temperature*. In: CHAI, T. (Hrsg.): *The Sixth World Congress on Intelligent Control and Automation*,

- 2006, WCICA 2006, S. 6531–6535, Piscataway, NJ, 2006. IEEE Operations Center.
- [42] LUNZE, J.: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Springer-Lehrbuch. Springer, Berlin, 9., überarb. Aufl. Aufl., 2013.
- [43] MASTERS, T.: *Advanced algorithms for neural networks A C++ sourcebook*. Wiley, New York, 1. Dr Aufl., 1995.
- [44] MILLER, W. T.: *Real-time application of neural networks for sensor-based control of robots with vision*. IEEE Transactions on Systems, Man, and Cybernetics, 19(4):825–831, 1989.
- [45] MOHAMMADZAHEDI, M. und L. CHEN: *Intelligent predictive control of a model helicopter's yaw angle*. Asian Journal of Control, 12(6):667–679, 2010.
- [46] MOHAMMADZAHEDI, M., L. CHEN und S. GRAINGER: *A critical review of the most popular types of neuro control*. Asian Journal of Control, 14(1):1–11, 2012.
- [47] NGUYEN, D. H. und B. WIDROW: *Neural networks for self-learning control systems*. IEEE Control Systems Magazine, 10(3):18–23, 1990.
- [48] NORDGREN, R. E. und P. H. MECKL: *An analytical comparison of a neural network and a model-based adaptive controller*. IEEE transactions on neural networks, 4(4):685–694, 1993.
- [49] PAREKH, R., J. YANG und V. HONAVAR: *Constructive neural-network learning algorithms for pattern classification*. IEEE transactions on neural networks, 11(2):436–451, 2000.
- [50] PARLOS, A. G., S. PARTHASARATHY und A. F. ATIYA: *Neuro-predictive process control using online controller adaptation*. IEEE Transactions on Control Systems Technology, 9(5):741–755, 2001.
- [51] PASCANU, R., T. MIKOLOV und Y. BENGIO: *On the difficulty of training Recurrent Neural Networks*.
- [52] PIOVOSO, M. J. und A. J. OWENS: *Sensor Data Analysis Using Neural Networks*. In: ARKUN, Y. und W. H. RAY (Hrsg.): *Chemical process control-CPCIV*, AIChE publication. CACHE, Austin Tex., 1991.
- [53] PRASAD, G., E. SWIDENBANK und B. W. HOGG: *A neural net model-based multivariable long-range predictive control strategy applied in thermal power plant control*. IEEE Transactions on Energy Conversion, 13(2):176–182, 1998.
- [54] PRASAD, L. B., H. O. GUPTA und B. TYAGI: *Application of policy iteration technique based adaptive optimal control design for automatic voltage regulator of power system*. International Journal of Electrical Power & Energy Systems, 63:940–949, 2014.
- [55] RAKOWITSCH, M.: *Modellierung der Dynamik eines Koppelgetriebes am Beispiel der 3D-Servo-Pressen: Studienarbeit*. Darmstadt, 2018.
- [56] REN, X. M. und A. B. RAD: *Adaptive non-linear compensation control based on neural networks for non-linear systems with time delay*. International Journal of Systems Science,



---

40(12):1283–1292, 2009.

- [57] RUAN, X., M. DING, D. GONG und J. QIAO: *On-line adaptive control for inverted pendulum balancing based on feedback-error-learning*. Neurocomputing, 70(4-6):770–776, 2007.
- [58] SARLE, W. S.: *Neural Network Implementation in SAS® Software: Proceedings of the Nineteenth Annual SAS Users Group International Conference*, 1994.
- [59] SCHMOECKEL, D.: *Developments in Automation, Flexibilization and Control of Forming Machinery*. CIRP Annals, 40(2):615–622, 1991.
- [60] SCHRÖDER, D.: *Intelligente Verfahren: Identifikation und Regelung nichtlinearer Systeme*. Springer-Verlag Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [61] SINZ, J.: *Die 3D-Servo-Pressen - von der Forschungsversion zur industriellen Standardmaschine*, 2018.
- [62] SKLYARENKO, Y.: *Anwendung neuronaler Netze zur Regelung von nichtlinearen Roboterantrieben: Dissertation*. 2002.
- [63] SON, Y. K. und C. S. PARK: *Economic measure of productivity, quality and flexibility in advanced manufacturing systems*. Journal of Manufacturing Systems, 6(3):193–207, 1987.
- [64] STURM, M.: *Neuronale Netze zur Modellbildung in der Regelungstechnik: Dissertation*. 2000.
- [65] VELIČKOVIĆ, P.: *Multilayer perceptron (MLP): A diagram representing an in-depth view at a single perceptron, along with its position within a larger-scale multilayer perceptron (~unrestricted feedforward neural network)*. 2018.
- [66] VIJAYA KUMAR, M., S. SURESH, S. N. OMKAR, R. GANGULI und P. SAMPATH: *A direct adaptive neural command controller design for an unstable helicopter*. Engineering Applications of Artificial Intelligence, 22(2):181–191, 2009.
- [67] VRABIE, D., O. PASTRAVANU, M. ABU-KHALAF und F. L. LEWIS: *Adaptive optimal control for continuous-time linear systems based on policy iteration*. Automatica, 45(2):477–484, 2009.
- [68] WERBOS, P. J.: *NEUROCONTROL AND RELATED TECHNIQUES*. In: MAREN, A. J., C. T. HARTON und R. M. PAP (Hrsg.): *Handbook of Neural Computing Applications*, S. 345–380. Elsevier Science, Burlington, 2014.
- [69] YANG, Y.-S. und X.-F. WANG: *Adaptive  $H_\infty$  tracking control for a class of uncertain nonlinear systems using radial-basis-function neural networks*. Neurocomputing, 70(4-6):932–941, 2007.
- [70] ZEILER, M. D.: *ADADELTA: An Adaptive Learning Rate Method*.
- [71] ZHAO, B. und H. HU: *A new inverse controller for servo-system based on neural network model reference adaptive control*. COMPEL - The international journal for computation and mathematics in electrical and electronic engineering, 28(6):1503–1515, 2009.