HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Masterarbeit

Maschinenbau

Physics-guided Neural Networks zur Identifikation dynamischer Systeme

vorgelegt von

B.Sc. Oliver Schön
Matr.-Nr.: 7026058

Betreuer:

Prof. Dr.-Ing. habil. Ansgar Trächtler
Dr.-Ing. Julia Timmermann
M.Sc. Ricarda-Samantha Götte

Paderborn, 15.06.2021

Masterarbeit

Physics-guided Neural Networks
zur Identifikation dynamischer Systeme
am: 15.06.2021

Heinz Nixdorf Institut
Universität Paderborn
Regelungstechnik und Mechatronik
Prof. Dr.-Ing. habil. Ansgar Trächtler
Fürstenallee 11
33102 Paderborn

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Masterarbeit

Physics-guided Neural Networks zur Identifikation dynamischer Systeme

Motivation

Durch die zunehmende Bedeutung von Machine Learning auch in den Ingenieurwissenschaften werden immer mehr hybride Modellansätze bestehend aus physikalischen und datengetriebenen Teilmodellen erforscht. Diese Kombination bietet Synergien, denn durch das komplementäre Wissen beider Perspektiven ergeben sich neue Chancen dynamische Systeme vollumfänglich zu identifizieren. Dabei werden ganzheitliche Ansätze erforscht, welche einen gleichwertigen Zusammenschluss beider Modellierungsarten statt eines datengetriebenen Fehlermodells als Ergänzung zum physikalisch basierten Modell verfolgen. Neben der konkreten Wahl der Modellstruktur sind die Berücksichtigung unterschiedlicher Wissensarten sowie die Qualität eines hybriden Modells verglichen zu dem physikalisch und/oder datenbasierten Modell offene Forschungsfragen.

Aufgabenstellung

Zu Beginn dieser Arbeit ist eine Einarbeitung in die Thematik der hybriden Modellbildung durch „Physics-guided Neural Networks“ (PGNN) basierend auf den Arbeiten von Karpatne et al. und Muralidhar et al. vorgesehen. Hier werden neuronale Netze mit einem zusätzlichen Eingang aus einem physikalischen Modell konstruiert, wobei beim Training des NN auch physikalische Nebenbedingungen des Systems berücksichtigt werden können. Anschließend soll die Methodik für ein akademisches Anwendungsbeispiel getestet und hinsichtlich der physikalischen Nebenbedingungen für regelungstechnische Anwendungen analysiert werden. Die Arbeiten dazu erfolgen in Matlab/Simulink. Zum einen soll untersucht werden, inwieweit PGNN in der Lage sind Parameter aus dynamischen Systemen zu identifizieren. Zum anderen soll die Abbildung der Dynamik durch vorwärts gerichtete und rekurrente NN untersucht und Vor- bzw. Nachteile der jeweiligen Ansätze diskutiert werden. Außerdem sollen Vergleiche des hybriden Ansatzes zum physikalischen und zum rein datenbasierten Modell durchgeführt werden.

Je nach Fortschritt der Arbeit können weitere datenbasierte Verfahren (SINDY, Bayes‘sche Optimierung, Bayes‘sche NN) mit physikalischen Nebenbedingungen im hybriden Modellansatz untersucht werden.

Über Vorgehen und Ergebnisse ist eine schriftliche Ausarbeitung anzufertigen.

Arbeitsaufwand: 660 h

Bearbeitungszeit: 6 Monate

Bearbeiter: Oliver Schön

Betreuer: Ricarda Götte
Julia Timmermann

Beginn der Bearbeitung: 15.12.2020

Prof. Dr.-Ing. habil. Ansgar Trächtler

Erklärung:

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Paderborn, 15.06.2021

Zusammenfassung

Durch die Einführung von Physics-guided Neural Networks (PGNN), einer jüngst begründeten Klasse von Hybridmodellen, konnten bereits für verschiedene Anwendungsbereiche Erfolge im Bezug auf synergetische Effekte sowie physikalisch valide Modelle erzielt werden. Im Rahmen dieser Arbeit wird erstmals das Potential von PGNNs zur Identifizierung dynamischer Systeme aus regelungstechnischer Sicht untersucht. Es werden dabei diverse Ansätze identifiziert und erste Untersuchungen angestellt.

Das in diesem Zusammenhang entwickelte Physics-guided Recurrent Neural Network (PGRNN) leitet sich aus der Kombination eines Rekurrenten Neuronalen Netzes mit einem physikalischen Dynamikmodell ab. Dabei kann gezeigt werden, dass das PGRNN im Allgemeinen einem rein datengetriebenen Ansatz wesentlich überlegen ist. Weitere Untersuchungen machen deutlich, dass dabei die Güte des eingebrachten Dynamikmodells lediglich von untergeordneter Bedeutung für die resultierenden Leistungssteigerungen ist.

In konsekutiven Schritten wird das PGRNN um eine physikalische Nebenbedingung zur Induzierung energieerhaltender Lösungen sowie eine Funktionsbibliothek nichtlinearer Terme erweitert. Durch Letztere können Defizite infolge fehlerhafter dynamischer Modelle vollständig kompensiert werden.

Abstract

Since their introduction, Physics-guided Neural Networks (PGNN), a novel class of hybrid models, have already been successfully implemented in several domains of application. As a result, both synergetic effects as well as physically sound models were obtained. Within the context of this thesis, for the first time, the potential of PGNNs for the identification of dynamic systems is investigated from a control engineering point of view. Various approaches are identified and first investigations are performed.

By combining a recurrent neural network with a physical dynamics model, a Physics-guided Recurrent Neural Network (PGRNN) is constructed. It is demonstrated that the PGRNN generally outperforms a purely data-driven approach by a substantial margin. Further investigations indicate that the quality of the introduced dynamics model is merely of minor importance to the resulting performance benefits.

Consecutively, the PGRNN is augmented by a physics-based constraint, inciting energy conserving solutions as well as a function library of nonlinear terms. The latter resulted in the full compensation of prediction error deficiencies due to inaccurate dynamic models.

Physics-guided Neural Networks zur Identifikation dynamischer Systeme

Inhaltsverzeichnis

Anhang

Abkürzungsverzeichnis

BNN	Bayessches Neuronales Netz
BO	Bayessche Optimierung (⇒ siehe Kap. 2.1.6)
DNN	Deep Neural Network (dt. tiefes Neuronales Netz) (⇒ Glossar)
ELU	Exponential Linear Unit (⇒ siehe Kap. 2.1.5)
GAN	Generative Adversarial Network (⇒ Glossar)
GRU	Gated Recurrent Unit (⇒ Glossar)
GT	Ground Truth
HCI	Human-Machine-Interface
HNN	Hamiltonian Neural Network (⇒ siehe Kap. 3.1 sowie [GDY19])
HP	Hyperparameter (⇒ siehe Kap. 2.1.6)
idR.	in der Regel
iFv.	in Form von
KI	Künstliche Intelligenz (engl. artificial intelligence (AI))
LNN	Lagrangian Neural Network (⇒ siehe Kap. 3.1 sowie [CGH+20])
LogMAE	Logarithmisch skalierte MAE-Fehlerfunktion (⇒ siehe Kap. 2.1.3)
LogMSE	Logarithmisch skalierte MSE-Fehlerfunktion (⇒ siehe Kap. 2.1.3)
LReLU	Leaky ReLU (⇒ siehe Kap. 2.1.5)
LSTM	Long-Short Term Memory (⇒ siehe Kap. 2.1.4)
MAE	Mean Absolute Error (⇒ Glossar)
ML	Maschinelles Lernen (engl. Machine Learning)
MPC	Model Predictive Control
MSE	Mean Squared Error (⇒ Glossar)
NN	Neuronales Netz (⇒ Glossar)
ODE	Ordinary Differential Equation (dt. gewöhnliche Differentialgleichung)
PDE	Partial Differential Equation (dt. partielle Differentialgleichung)
PGNN	Physics-guided Neural Network (⇒ Glossar)
PGRNN	Physics-guided Recurrent Neural Network (⇒ siehe Kap. 3.6)
PGRNN+	PGRNN mit Funktionsbibliothek (⇒ siehe Kap. 3.8)
POMDP	Partially-Observable Markov Decision Process
ReLU	Rectified Linear Activation Unit (⇒ siehe Kap. 2.1.5)

RMSE	Root Mean Squared Error
RNN	Rekurrentes Neuronales Netz (⇒ Glossar)
SINDY	Sparse Identification of Nonlinear Dynamics (⇒ Glossar)
VFzg	Viertelfahrzeugmodell (⇒ siehe Kap. 3.3)

Glossar

D

Deep Neural Network (DNN): DNNs umfassen Neuronale Netze mit mehr als einer versteckten Schicht. Sie zeichnen sich durch ihre herausragenden Fähigkeiten in der Abbildung komplexer Zusammenhänge aus. ⇒ Kap. 2.1.1

dt. tiefes Neuronales Netz

G

Gated Recurrent Unit (GRU): Die zwei populärsten Grundbausteinvarianten zur Konstruktion Rekurrenter Neuronaler Netze sind GRU und LSTM. GRUs beschreiben einen Mechanismus, infolge dessen Informationen in diskreten Zeitschritten dem Hidden State hinzugefügt oder aus ihm entfernt werden. ⇒ Kap. 2.1.4

Generative Adversarial Network (GAN): Unter GANs wird eine besondere Klasse Neuronaler Netze zusammengefasst, bei der durch einen Netzteil (Generator) Kandidatenvektoren erzeugt und von einem nachgestellten Netzteil (Diskriminator) bewertet werden. Das Trainingsziel des Generators besteht darin, z.B. Bilder zu generieren, die nicht von den echten Bilddaten zu unterscheiden sind.

M

Mean Absolute Error (MAE): Fehlerfunktion mit linearem Eingang des Prädiktionsfehlers. ⇒ Kap. 2.1.3

Mean Squared Error (MSE): Fehlerfunktion mit quadratischem Eingang des Prädiktionsfehlers. ⇒ Kap. 2.1.3

N

Netzgüte: Es ist zwischen der Netzgüte im wörtlichen Sinne sowie der Verwendung als Gütemaß J_{sim} im Rahmen der Bayesschen Optimierung zu unterscheiden. Letztere umfasst den Simulationsfehler sowie die Trainingsdauer und ist zu minimieren. ⇒ Gl. 4-1

Neuronales Netz (NN): Beschreibt ein lineares Regressionsmodell, das als gerichteter Graph aus Neuronen und Gewichten einen algebraischen Zusammenhang zwischen Ein- und Ausgangsgrößen beschreibt. Siehe auch vorwärtsgerichtete Netze. ⇒ Kap. 2.1

engl. neural network (NN)

P

Physics-guided Neural Network (PGNN): PGNNs stehen im Fokus dieser Arbeit und umfassen eine neuartige Klasse von Hybridmodellen. Neben einer Vielzahl weiterer Bezeichnungen sind im Rahmen dieser Arbeit insbesondere die rekurrente Version (PGRNN) sowie die um eine Funktionsbibliothek erweiterte Variante (PGRNN+) von Bedeutung. ⇒ Kap. 3

R

Rekurrentes Neuronales Netz (RNN): Zyklische Neuronale Netze werden auch als RNNs bezeichnet. Aufgrund numerischer Limitationen werden diese heutzutage aus GRUs und LSTM-Zellen konstruiert. ⇒ Kap. 2.1.4

engl. recurrent neural network

S

Simulationsfehler: Der simulative Prädiktionsfehler, kurz Simulationsfehler E_{sim}, ist das im Rahmen dieser Arbeit genutzte Gütemaß zur Klassifizierung der Güte eines trainierten Netzes. ⇒ Kap. 3.4

Sparse Identification of Nonlinear Dynamics (SINDY): Das von Brunton et al. eingeführte Modellierungsverfahren zur Systemidentifikation dynamischer Systeme basiert auf einer Funktionsbibliothek, mithilfe derer die Dynamikgleichung eines Systems aus Messdaten identifiziert werden kann. ⇒ Kap. 2.2

V

Vorwärtsgerichtetes Neuronales Netz: Neuronale Netze mit azyklischer Anordnung werden auch als vorwärtsgerichtete Neuronale Netze bezeichnet. Oft ist im Kontext mit NN ein vorwärtsgerichtetes Neuronales Netz gemeint. ⇒ Kap. 2.1.1

engl. feed-forward neural network

1 Einleitung

Zunächst soll eine Motivation des angestrebten Forschungsobjekts Physics-guided Neural Networks zur Identifizierung dynamischer Systeme in der Regelungstechnik gegeben werden. Dabei wird das betrachtete Thema in den Gesamtkontext sowohl datengetriebener Methoden des Maschinellen Lernens sowie des Regelungsentwurfs eingeordnet. Im Anschluss folgen eine konkrete Zielsetzung sowie Anmerkungen zu Vorgehensweise und Aufbau der Arbeit.

1.1 Motivation

Maschinen, die sich in agiler Weise ihrem Umfeld anpassen, aus getroffenen Handlungen lernen und in Folge Entscheidungsprozesse optimieren; aktive Kooperation von Mensch und Maschine physisch und durch sogenannte Human-Machine-Interfaces (HCI) sowie sich selbst optimierende Prozessanlagen gehören längst nicht mehr der Zukunft an.

> „The future is now: AI's impact is everywhere“ [Tho19]

Auch wenn es sich bei der Definition Künstlicher Intelligenz[1] (KI) weiterhin um ein kryptisches und unscharfes Subjekt handelt, konnten durch den Einzug des Subbereichs *Machine Learning* (ML) in den ingenieurwissenschaftlichen Disziplinen bereits außerordentliche Erfolge erzielt werden.

Ob Robotersysteme zur Unterstützung chirurgischer Eingriffe, autonom gesteuerte Automobilflotten oder nach dem Start automatisch zur Erde zurückkehrende Raketenantriebssysteme, all diese und viele weitere Anwendungsbeispiele haben gemeinsam, dass im Vorhinein umfangreiche modellbasierte Simulationen diverser Betriebssituationen durchgeführt werden müssen, um Funktion und Sicherheit gewährleisten zu können.

Die Erstellung eines solchen Modells ist jedoch mitunter enorm zeit-, kosten- und arbeitsintensiv. Zu den größten Herausforderungen bei der Modellierung komplexer Systeme gehören umfangreiche Nichtlinearitäten, die Identifikation diverser Modellparameter sowie in manchen Fällen schier fehlendes Systemverständnis. Genau an diesen Problemstellungen greifen ML-basierte Ansätze in den Ingenieurwissenschaften, insbesondere in der Regelungstechnik, an.

Die Potentiale Maschinellen Lernens im Bereich der Modellbildung und Systemidentifikation bestehen in dem „Erlernen“ komplexer Systemdynamiken auf Basis von Messdaten. War dies für simple Systeme auch in der Vergangenheit mit z.B. Autoregressionsmodellen möglich, gehen die mächtigen Möglichkeiten von Deep Learning erheblich tiefer. So können nicht nur Parameter identifiziert und somit generalistische Modelle an spezifische Anwendungsfälle „kalibriert“ werden, sondern, sogar ohne jegliches Vorwissen, komplette Dynamikgleichungen aus Messdaten extrahiert werden.

Die Möglichkeit, umfassende Modelle ohne jegliches Systemwissen generieren zu können, ist sowohl Fluch und Segen zugleich. So sind übliche differentialgleichungsbasierte

[1] engl. artificial intelligence (AI)

Dynamikmodelle auf fundamentale physikalische Gesetzmäßigkeiten zurückführbar und verifizier- und interpretierbar. Für rein datengetriebene Modelle gilt dies aber idR. nicht. So kann im Fall auftretender Anomalien nicht mit Sicherheit gesagt werden, ob diese auf den modellierten Prozess oder das Modell an sich zurückzuführen sind.

Als *Hybridmodelle* werden eben solche Modelle bezeichnet, die datengetriebene Modellierungsverfahren mit herkömmlichem Domänenwissen vereinen. Als Objekt aktueller Forschungsbemühungen erhofft man sich im Rahmen der Modellierung dynamischer Systeme mit Hybridmodellen synergetische Effekte sowie die Produktion valider – insbesondere physikalische Gesetzmäßigkeiten respektierender – Modelle.

Bei *Physics-guided Neural Networks* (PGNN) handelt es sich um eine neuartige Klasse hybrider Modelle, die seit ihrer Einführung in 2017 durch Karpatne et al. unaufhaltsam Einzug in diverse Fachgebiete erfährt. Bisher wurden PGNNs allerdings, nach bestem Wissen und Gewissen des Autors, noch nicht auf ihre Eignung zur Systemidentifikation in der Regelungstechnik untersucht.

Dabei verspricht man sich durch den Einsatz von PGNNs die Berücksichtigung physikalischer Zusammenhänge, wie z.B. des Energieerhaltungssatzes, aber auch die Nutzung bestehender physikalischer Modelle. So ist in vielen Szenarien mit kleinem Aufwand ein physikalisches Dynamikmodell erstellbar, das die Dynamik des betrachteten Systems nur ansatzweise beschreibt. Inwiefern ein solches Modell im Rahmen der Modellierung mit PGNNs berücksichtigt werden kann und welche synergetischen Effekte sich zum rein datengetriebenen Ansatz ergeben, gilt es im Rahmen dieser Arbeit zu untersuchen.

1.2 Zielsetzung

Da es sich um die erste Arbeit handelt, die sich mit der Applizierung von PGRNNs zur Identifikation dynamischer Systeme im Bereich der Regelungstechnik beschäftigt, soll im Rahmen dieser Arbeit eine umfassende Untersuchung des Themengebiets sowie eine Implementierung in MATLAB erfolgen. Dazu sollen geeignete datenbasierte Modellierungsverfahren um Domänenwissen iFv. physikalischen Dynamikmodellen sowie physikalischen Nebenbedingungen erweitert und untersucht werden. Ziel ist die Identifikation erfolgversprechender Ansätze für weiterführende Arbeiten.

1.3 Vorgehensweise und Aufbau

Aufbau und Vorgehen dieser Arbeit gliedern sich in 3 Schritte. Zunächst wurden die Grundlagen Neuronaler Netze behandelt. Insbesondere erfolgt die Erweiterung zu Rekurrenten Neuronalen Netzen und der Untersuchung derer Potential zur Modellierung dynamischer Systeme. Erfahrene Leser können direkt zu Kapitel 3 springen und haben, je nach Bedarf, an geeigneter Stelle durch Verweise oder durch aktives Nachschlagen die Möglichkeit, zu den Grundlagen zurückzuspringen.

Die Behandlung von PGNNs umfasst eine ausgiebige Literaturrecherche, deren Ergebnisse Kapitel 3.1 und insbesondere Tabelle 3-1 zu entnehmen sind. Dem angeschlossen sind diverse Unterkapitel, in denen die in Kapitel 4 untersuchten Ansätze erläutert werden. Es

haben sich an diversen Stellen Analogien zur Regelungstechnik ergeben, wobei Kapitel 3.5 einen guten Überblick darüber gibt.

Kapitel 4 beschäftigt sich ausschließlich mit den Untersuchungen, die im Rahmen dieser Arbeit durchgeführt wurden. Diese erfolgten anhand dreier Beispielsysteme und beinhalten insbesondere die Untersuchung infolge der Inkorporation eines physikalischen Dynamikmodells, der Konditionierung der Trainingsdaten, der Erweiterung des Ansatzes um eine Funktionsbibliothek sowie einer physikalischen Nebenbedingung.

Sollten Begrifflichkeiten nicht bekannt sein oder ist eine kurze Definition bzw. ein Verweis auf das entsprechende Unterkapitel gewünscht, kann eine Reihe zentraler Begriffe im Glossar nachgeschlagen werden.

2 Datengetriebene Methoden der Systemidentifikation

Im Rahmen dieses Kapitels sollen die Grundlagen zweier datengetriebener Systemidentifikationsverfahren beschrieben werden. Dabei wird ein kurzer Überblick über vorwärtsgerichtete Neuronale Netze (Kap. 2.1.1) gegeben. Darauf aufbauend folgt eine detaillierte Beschreibung Rekurrenter Neuronaler Netze (Kap. 2.1.4) sowie diverser Komponenten und Faktoren, die es bei dem Training dieses Netztyps zu beachten gilt. Abschließend wird ein knapper Überblick über das Systemidentifikationsverfahren SINDY (Kap. 2.2) angeführt.

Da es sich bei diesem Kapitel lediglich um Grundlagen zum Verständnis der folgenden Inhalte handelt, wird dieses so umfangreich wie nötig, aber so kurz wie möglich gehalten. An geeigneten Stellen wird daher auf weiterführende Literatur verwiesen.

2.1 Neuronale Netze

Ein *(künstliches) Neuronales Netz*[2] (NN) ist ein Netzwerk aus *Knoten* und *Kanten*, auch als (gerichteter) Graph bezeichnet, dessen Knoten bzw. Neuronen reellwertige Zustände einnehmen und deren Kanten bzw. Verbindungen jeweils ein Gewicht zugeordnet bekommen. Die Neuronen eines Neuronalen Netzes können dabei grundsätzlich in Eingabeneuronen, Ausgabeneuronen und *versteckte Neuronen* unterschieden werden. [vgl. KBK$^+$12, S. 33 f.] Werden den Eingabeneuronen extrinsisch Zustandswerte zugeordnet, so ergeben sich durch die gewichteten Verbindungen zwischen den Neuronen entsprechende Zustandswerte für alle nachgestellten versteckten Neuronen, denen von außen kein initialer Wert zugeordnet wurde. Genauso ergeben sich für die Ausgabeneuronen eines Neuronalen Netzes entsprechende Zustandswerte, mit der Besonderheit, dass diese gleichzeitig als Ausgangsgrößen des Neuronalen Netzes fungieren. Es ergibt sich ein definiertes, deterministisches Ein-/Ausgangsverhalten, das durch die Gewichte des Neuronalen Netzes allein charakterisiert wird. Demnach kann das Übertragungsverhalten zwischen den Eingangsgrößen $\underline{x}$ und Ausgangsgrößen $\underline{y}$ eines Neuronalen Netzes mathematisch durch

$$\underline{y} = \underline{f}_{NN}(\underline{x}|\underline{w})$$

beschrieben werden, wobei die Gewichte $\underline{w}$ als variierbare Parameter fungieren.

Auch wenn der Begriff des (künstlichen) Neuronalen Netzwerkes ursprünglich den Versuchen der mathematischen Beschreibung der Informationsverarbeitung biologischer Systeme entstammt, haben Neuronale Netze in ihrer heutzutage prädominanten Form nur noch wenig mit ihrem biologischen Pendant zu tun [vgl. Bis09, S. 226]. Für die Zwecke dieser Ausarbeitung sind dabei lediglich die performantesten Derivate Neuronaler Netze zur Funktionsregression relevant.

Neuronale Netze gehören zur Klasse der linearen Regressionsmodelle – lineare Funktionen definierter Parameter – und können in mehrere aufeinanderfolgende Schichten von

[2] engl. (artificial) neural network

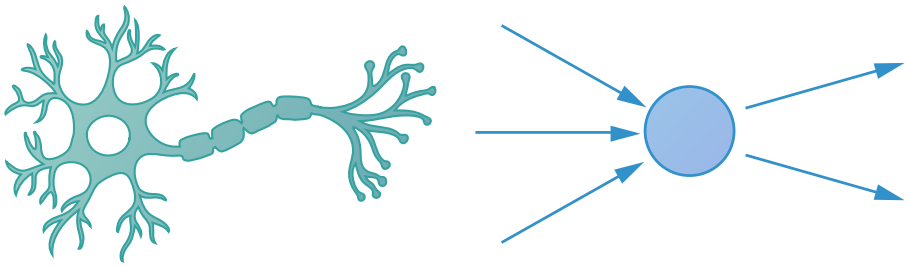

Bild 2-1: *Sowohl künstliche Neuronale Netze als auch das biologische Vorbild setzen sich aus einzelnen Logikeinheiten, den sogenannten Neuronen, zusammen. Die Verbindungen zwischen Neuronen, ähnlich den Synapsen zwischen Gehirnzellen, ermöglichen den Austausch von Signalen untereinander. Das Zusammenspiel des Netzwerks aus Neuronen und Verbindungen bewirkt in beiden Fällen die eigentliche Funktion der informationsverarbeitenden Systeme.*

Neuronen unterteilt werden. Zunächst soll eine solche Schicht eines Neuronalen Netzes betrachtet werden. Bei den Parametern dieser Schicht handelt es sich im einfachsten Fall um die Eingangsgrößen der Schicht und bei dem resultierenden linearen Regressionsmodell folglich um eine Linearkombination der Eingangsgrößen. Im mehrdimensionalen Fall entspricht dies der linksseitigen Multiplikation des Eingangsvektors $\underline{x}$ mit einer Gewichtsmatrix $\underline{W}^T$:

$$\underline{y} = \underline{W}^T \underline{x}$$

Deutlich nützlicher wird das resultierende Modell, wenn anstatt der Identität der Eingangsgrößen eine Linearkombination eines definierten Satzes nichtlinearer Funktionen $\underline{\phi}(\underline{x})$ des Eingangs, auch Basisfunktionen genannt, verwendet wird. Der Ausgangszustand $\underline{y}$ ergibt sich erneut durch linksseitige Multiplikation mit den Gewichten $\underline{W}^T$:

$$\underline{y} = \underline{W}^T \underline{\phi}(\underline{x})$$

Im Allgemeinen handelt es sich demnach bei einem aus aufeinanderfolgenden Schichten von Neuronen konstruierten Neuronalen Netz um eine Serie funktionaler Transformationen des Netzeingangs. Derartige Modelle sind aufgrund ihres linearen Aufbaus analytisch sehr einfach zu berechnen und bieten dennoch die Möglichkeit der Approximation von Nichtlinearitäten, wie z.B. nichtlinearer Dynamiken. [vgl. Bis09, S. 137 f.]

Die Wahl einer geeigneten Basisfunktion fällt idR. auf eine der populärsten Aktivierungsfunktionen, deren Charakteristika in Kapitel 2.1.5 gegenübergestellt werden. Zunächst soll anhand vorwärtsgerichteter NNe der typische Aufbau sowie die Funktionsweise eines grundständigen NNes untersucht werden. Auf deren Basis wird in Kapitel 2.1.4 eine weiterentwickelte Klasse NNe eingeführt, die die Berücksichtigung zeitlicher Zusammenhänge sequentieller Eingangsabfolgen ermöglichen.

2.1.1 Vorwärtsgerichtete Netze

Eine besonders verbreitete Unterklasse Neuronaler Netze belegen die sogenannten *vorwärtsgerichteten Neuronalen Netze*. Umfasst das Spektrum Neuronaler Netze allgemein alle erdenklichen Anordnungen von Knoten und Kanten – mit oder ohne Rekursionen –, beschränken sich vorwärtsgerichtete Neuronale Netze auf strikt unidirektionale, sogenannte azyklische Graphen [KBK+12, S. 33]. Auch wenn diese Einschränkung einer schwerwiegenden Eingrenzung der möglichen Architekturen[3] entspricht, sind die Derivate der vorwärtsgerichteten NNe von hoher Bedeutung und verkörpern das adaptivste und erfolgreichste Modellierungsverfahren für Pattern Recognition [vgl. Bis09, S. 225 f.]. Vorwärtsgerichtete NNe zeichnen sich neben ihrem guten Abbildungsvermögen besonders durch ihre kompakte Beschreibung, schnelle Auswertung und das damit einhergehende einfache Training aus [vgl. Bis09, S. 225 f.]. Aufgrund dieser Eigenschaften werden vorwärtsgerichtete Neuronale Netze häufig als Standardform der Neuronalen Netze behandelt und haben sich daher im Englischen die Bezeichnung *Vanilla Neural Network* verdient. Bis zur Einführung komplexerer Varianten Neuronaler Netze in Kap. 2.1.4 soll auch in dieser Arbeit zunächst standardmäßig die vorwärtsgerichtete Architektur betrachtet werden. Im Anschluss werden die Eigenheiten rekurrenter Netze ergänzt.

Aufbau

Strukturell können vorwärtsgerichtete NNe in mehrere Schichten untergliedert werden. Jedes NN verfügt dabei über mindestens eine Eingabe- und eine Ausgabeschicht. Die Anzahl der jeweiligen Ein- und Ausgabeneuronen ergibt sich durch die Dimension der Ein- und Ausgabegrößen. Das in Abbildung 2-2 skizzierte NN verfügt über eine Ein- und eine Ausgabeschicht, die jeweils aus drei und zwei Ein- bzw. Ausgabeneuronen bestehen. Die Ausgestaltung der logischen Verschaltung zwischen den beiden Schichten wird im Rahmen der Architekturwahl definiert. Das abgebildete Beispielnetzwerk verfügt über drei versteckte Schichten mit jeweils fünf versteckten Neuronen. Die Anzahl der in einem NN verwendeten versteckten Schichten sowie deren Besetzung mit versteckten Neuronen kann beliebig gewählt werden und ist primärer Bestandteil der Hyperparameteridentifikation (Kap. 2.1.6). Zunächst soll angenommen werden, dass die Anzahl der versteckten Neuronen und Schichten bekannt ist.

Das in Abbildung 2-2 dargestellte Netzwerk fällt überdies in eine weitere Unterklasse Neuronaler Netze. Alle NNe mit mehr als einer versteckten Schicht werden als *tiefe Neuronale Netze*[4] bezeichnet, was auf die hohe Abbildungsfähigkeit dieser Variante sowie deren Analogie zur Funktionsweise des menschlichen Gehirns zurückgeht. Der Einfachheit halber wird dabei idR. zunächst angenommen, dass zwischen allen Neuronen zweier aufeinanderfolgender Schichten Verbindungen bestehen, indem für jede Kante ein Gewicht gesetzt wird. Nullwertige Verbindungen werden im anschließenden Training neutralisiert.

[3] Im Allgemeinen wird bei dem Aufbau Neuronaler Netze und der Anordnung der beteiligten Neuronen von einer Architektur gesprochen.

[4] engl. deep neural networks (DNN)

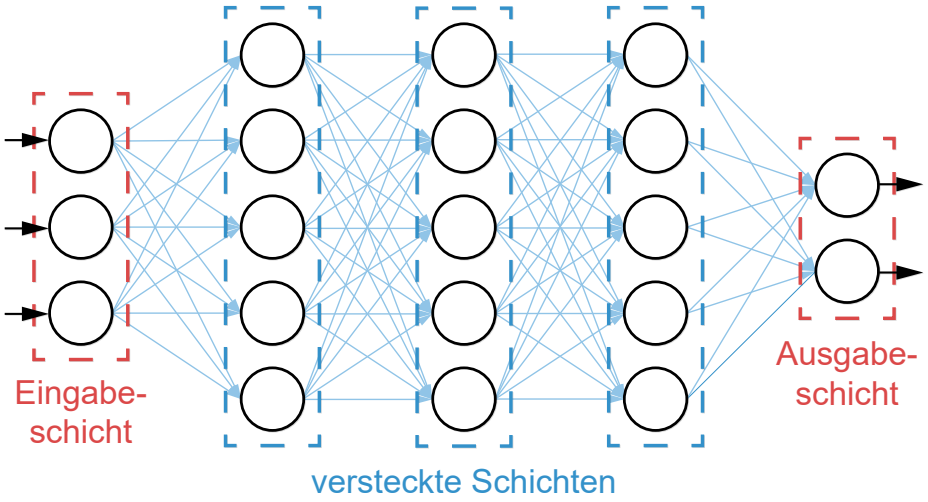

Bild 2-2: *Vorwärtsgerichtete Neuronale Netze können in Schichten untergliedert werden, die jeweils aus unterschiedlich vielen Neuronen bestehen. Netzeingaben werden dabei von der Eingabeschicht durch eine variable Anzahl versteckter Schichten zur Ausgabeschicht übermittelt. Die Verbindungen zwischen den Neuronen sind dabei durch Gewichte charakterisiert.*

Training

Zur Charakterisierung der Funktionsweise NNe soll zunächst eine simple Notation eingeführt werden. Jedem versteckten Knoten u des NN können vorausgehende $pred(u)$ und nachfolgende Knoten $succ(u)$ zugeordnet werden [vgl. KBK+12, S. 33–49]. Bei $out_j^{pred(u)}$ handelt es sich folglich um die Ausgabe des j-ten unmittelbaren Vorgängers von Knoten u und bei $in_j^{succ(u)}$ um die entsprechende Eingabe des j-ten unmittelbaren Nachfolgers.

Wie bereits in Kapitel 2.1.1 beschrieben, verkörpern NNe eine Serie funktioneller Transformationen der Netzeingabe, parametriert durch Gewichte. Die Bestimmung dieser Gewichte wird als *Training* bezeichnet und folgt dem Prinzip des Supervised Learnings. Das Training ist ein iterativer Prozess, bei dem durch progressive Anpassung der Gewichte die Netzausgabe, infolge einer definierten Eingabe, einer definierten Zielausgabe angenähert wird. Bei den definierten Eingaben und Zielausgaben spricht man auch von *Trainingsdaten*. Durch Auswertung eines Fehlermaßes zwischen Zielausgaben – auch *Target* oder *Ground Truth* genannt – und der tatsächlichen Netzausgabe, wird in jeder Iteration die Richtung bestimmt, in die die Gewichte bzw. Parameter des NNes angepasst werden müssen. Es handelt sich folglich um ein iteratives Optimierungsverfahren, bei dem die Gewichte die Optimierungsgröße und das auf dem Prädiktionsfehler basierende Fehlermaß das Gütemaß repräsentieren.

Das Training eines NNes in Form eines "Erlernens" eines definierten Ein-/Ausgangsverhaltens setzt sich grundsätzlich aus der wiederholten Abfolge dreier Schritte zusammen [vgl. KBK+12, S. 65]:

1. Prädiktion durch Vorwärtspropagation,
2. Bestimmung des Fehlerfaktors,
3. und der Gewichtsänderung durch Fehler-Rückpropagation.

Die Prädiktion der Netzausgabe erfordert die Initialisierung der Netzparameter bzw. Gewichte. Diese erfolgt idR. zufällig. Mithilfe der Parameter wird die Netzeingabe konsekutiv von Neuron zu Neuron propagiert und erfährt dabei, wie bereits zuvor erwähnt, eine Reihe funktioneller Transformationen. Die Auswahl geeigneter Basisfunktionen erfolgt im Vorhinein und beschränkt sich meist auf einen definierten Satz populärer Funktionen. Jedoch erlaubt eine parametrierte Ausgestaltung dieser sogenannten *Aktivierungsfunktionen* ein hohes Maß an Adaptivität [vgl. KBK+12, S. 225 f.]. Die Ausgabe out^u eines Neurons u ergibt sich aus der Ausgabe $out_i^{pred(u)}$ der Neuronen der vorausgehenden Schicht und den Gewichten w_i^u der Kanten zwischen diesen:

$$out^u = \sigma\left(\sum_{i=1}^{n} w_i^u out_i^{pred(u)} + b_i^u\right)$$

Dabei denotiert $\sigma(\cdot)$ die Aktivierungsfunktion von u sowie b_i^u eine weitere Form von Gewicht – den *Bias*. Die Biaswerte werden analog zu den Gewichten w_i^u behandelt. [KBK+12]

Angemerkt sei an dieser Stelle, dass nicht alle Aktivierungsfunktionen eines NNes zwangsweise dieselben sein müssen. Als Aktivierungsfunktion der Ausgabeneuronen wird zur Funktionsregression idR. die Identität verwendet. Je nach Wahl der Aktivierungsfunktion der versteckten Neuronen lassen sich mit einem NN verschiedene nichtlineare Funktionsverläufe besser oder schlechter approximieren. Die Verwendung von Aktivierungsfunktionen als einzige nichtlineare Komponente eines NNes sowie die Verwendung einer Vielzahl versteckter Neuronen und Schichten ermöglicht die Abbildung komplexen nichtlinearen Verhaltens, obgleich der andernfalls simplen mathematischen Konstruktionsweise. Ein ähnliches Konstrukt ist die Gauß-Prozess-Regression, bei der statt Aktivierungsfunktionen Gauß-Verteilungen in Form sogenannter Kernelfunktionen verwendet werden. Für Gauß-Prozesse sowie für NNe mit mehr als einer versteckten Schicht gilt jedoch, dass nur wenig über die konkrete Funktionsweise des eigentlichen Abbildungsvorgangs und somit auch über den Beweis abbildungstechnischer Fähigkeiten bekannt ist. [vgl. KBK+12, S. 55] Die Performanz eines Maschinellen Lernverfahrens wie dem NN wird daher erst anhand konkreter Anwendungen klar. Daher werden in der Literatur häufig bestimmte akademische Anwendungen (z.B. der Lorenz-Attraktor) als Benchmark gehalten.

Aus der Diskrepanz zwischen Netzausgabe und Trainings-Target ergibt sich mithilfe einer *Fehlerfunktion*[5] ein Fehlerfaktor als Maß für die Güte der Prädiktion. Die Prädiktionsgüte wiederum ist direktes Resultat der Gewichtswahl und lässt dementsprechend eine Aussage über die Güte der unterliegenden Gewichte im Sinne einer Optimierung zu.

Wurde der Fehler des NNes bestimmt, erfolgt die Gewichtsänderung durch *Fehler-Rückpropagation*[6], deren Funktionsweise an dieser Stelle nur kurz umrissen werden soll. Für eine ausführliche Beschreibung des Algorithmus wird auf [KBK+12, S. 58 ff.] verwiesen. Die Fehler-Rückpropagation, auch als *Backpropagation Algorithmus* bezeichnet, basiert auf dem *Gradientenabstiegsverfahren*, bei dem aus dem Gradienten der Fehlerfunktion Richtungen abgeleitet werden, in die die Gewichte und Biaswerte geändert werden müssen, um den Prädiktionsfehler zu verringern. Um den Gradienten der Fehlerfunktion bestimmen zu können, müssen durchweg differenzierbare Aktivierungsfunktionen

[5] engl. loss function
[6] engl. error backpropagation

verwendet werden, wodurch sich unmittelbar die Differenzierbarkeit der Fehlerfunktion ergibt.

Im Rahmen des Trainings wird nach Initialisierung der Gewichte w_i zunächst der Gradient der Fehlerfunktion bzgl. der Gewichte $\underline{\nabla}_{\underline{w}_u} e_u$ – hier für einen Knoten u und Fehlerfunktion half-MSE[7] (siehe MSE Kap. 2.1.3) – am aktuellen Punkt des durch die Gewichte aufgespannten Lösungsraums bestimmt:

$$\underline{\nabla}_{\underline{w}_u} e_u = -\sum_l \left(o_u^{(l)} - out_u^{(l)}\right) \frac{\partial out_u^{(l)}}{\partial net_u^{(l)}} \underline{in}_u^{(l)}, \tag{2-1}$$

Dabei wird über die Trainingsdaten $l \in \mathcal{L}$, bestehend aus Target $o_u^{(l)}$ und zugehöriger Netzeingabe $net_u^{(l)}$, summiert. Da das Training eines NNes im Grunde der Minimierung des Fehlermaßes entspricht, wird im darauffolgenden Schritt entgegen der bestimmten Gradientenrichtung gewandert (beachte negatives Vorzeichen in Gl. (2-2)). An dem sich daraus ergebenden neuen Punkt wird nun erneut der Gradient berechnet usw. bis das Minimum der Fehlerfunktion erreicht wurde. Bei diesem Minimum muss es sich jedoch nicht zwangsweise um das globale Minimum handeln, weshalb die Güte eines trainierten NNes idR. durch eine geeignete Initialisierung der Gewichte oder Wiederholung des Trainings verbessert werden kann.

Wie bereits beschrieben, wird nach Bestimmung des Gradienten am aktuellen Punkt entgegen des Gradienten $\underline{\nabla}_{\underline{w}_u} e_u$ gewandert. Dieses wiederkehrende Ereignis, bestehend aus der Bestimmung des Gradienten und Modifikation der Gewichte, wird auch als Trainingsepoche, kurz *Epoche*, bezeichnet und besteht jeweils aus mindestens einer Iteration. In jeder *Iteration* wird der Prädiktionsfehler für eine gewisse Teilmenge der Trainingsdaten, auch als *Batch* bezeichnet, ausgewertet und die Anpassung der Gewichte erfolgt erst nach Verarbeitung aller Trainingsdaten als Gradient des kumulierten Fehlers. Die Distanz, die je Epoche entgegen des Gradienten gewandert wird, bezeichnet man dabei als Lernrate η (mehr zur Wahl der Lernrate in Kapitel 2.1.6), mit der sich die allgemeine Gewichtsänderung eines Neurons u direkt aus dem Gradienten der Fehlerfunktion bzgl. der Gewichte (Gl. (2-1)) ergibt:

$$\Delta \underline{w}_u = -\eta \underline{\nabla}_{\underline{w}_u} e_u = \eta \sum_l \left(o_u^{(l)} - out_u^{(l)}\right) \frac{\partial out_u^{(l)}}{\partial net_u^{(l)}} \underline{in}_u^{(l)} \tag{2-2}$$

Analog erfolgt die Erweiterung der Fehler-Rückpropagation auf mehrschichtige NN, bei der der Fehler, ausgehend von der Ausgabeschicht des NNes, rückwärts durch die versteckten Schichten propagiert wird (siehe [KBK+12, S. 63 ff.]).

Moderne Trainingsalgorithmen nutzen in der Regel komplexere Varianten des Gradientenabstiegs. Ein weit verbreitetes Derivat ist der Optimierungsalgorithmus *Adam*, bei dem der Backpropagation Algorithmus um weitere adaptive Terme erweitert wurde. Dazu zählt etwa die Einführung eines Moment-Terms, der in einer Epoche jeweils anteilsmäßig den Gradienten aus den vorausgegangenen Epochen berücksichtigt und so zu einer schnelleren Konvergenz führen kann. Konträr kann durch die inhärente Adaptivität der zusätzlichen Terme auch eine gesteigerte Akkuranz verzeichnet werden. Aufgrund seiner durchweg überlegenen Performanz wird daher idR. Adam zur Anpassung der Gewichte verwendet. Weitergehende Ausführungen finden sich in [KBK+12, S. 69 ff.].

[7] Mean Squared Error Fehlerfunktion mal ein Halb

2.1.2 Modellkomplexität und Overfitting

Wie bereits in Kapitel 2.1.1 angesprochen, wird das Abbildungsvermögen eines NNes hauptsächlich durch die Anzahl an versteckten Neuronen und Schichten determiniert. Durch Wahl geeigneter Nichtlinearitäten iFv. Aktivierungsfunktionen und einer hohen Anzahl versteckter Neuronen können komplexe nichtlineare Dynamiken durch NNe approximiert werden. Allerdings wächst mit steigender Modellkomplexität auch der Bedarf an Trainingsdaten. Wird für eine bestimmte Menge von Trainingsdaten eine zu hohe Modellkomplexität angesetzt, tritt ein Effekt auf, der als *Overfitting*[8] bezeichnet wird. Hierbei erfolgt durch das NN zwar eine gute Anpassung an die verwendeten Trainingsdaten, das NN weist jedoch mangelhaftes Extrapolationsvermögen auf. Dieses klassische Problem

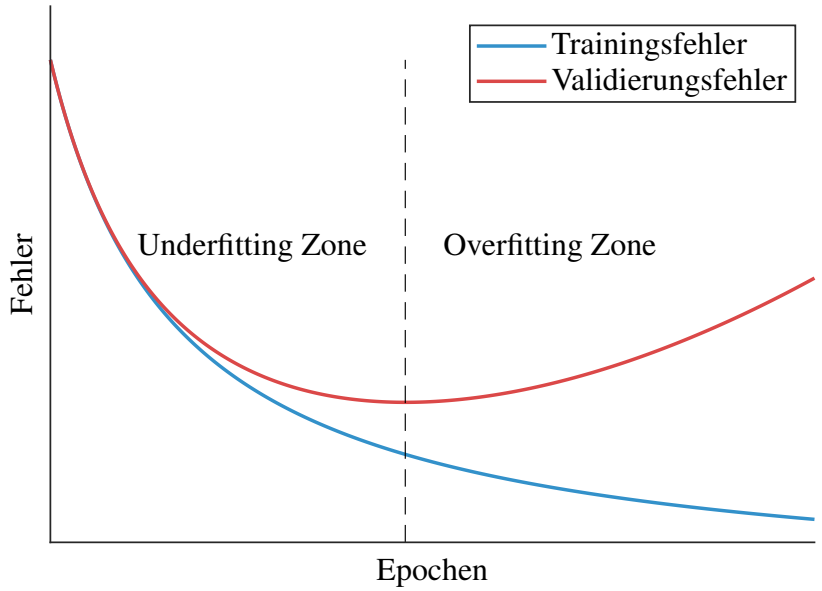

Bild 2-3: *Wird für eine bestimmte Menge an Trainingsdaten eine zu hohe Modellkomplexität gewählt, kann es zu Overfitting kommen, dabei steigt die Diskrepanz des Fehlers von Trainings- und Validierungsdaten über die Epochen. IdR. kommt es dabei an einem bestimmten Punkt zu einem Wiederanstieg des Validierungsfehlers aufgrund mangelhafter Extrapolationsfähigkeit des Neuronalen Netzes.*

NNe ist in Abbildung 2-3 veranschaulicht. Durch die hohe Anzahl an Neuronen und der daraus resultierenden hohen Dimensionalität des Gewichtsraums, können die Trainingsdaten beinah beliebig genau abgebildet werden. Dies spiegelt sich im kontinuierlich über die Epochen sinkenden Trainingsfehler wieder. Wird darüber hinaus jedoch auch der Fehler für einen von den Trainingsdaten verschiedenen *Validierungsdatensatz* bestimmt, so zeigt sich, dass der zugehörige *Validierungsfehler* zwar anfangs fällt, sich über die Trainingsepochen hinweg allerdings sukzessive eine Diskrepanz zwischen Trainings- und Validierungsfehler aufbaut, die in diesem Fall sogar in einem erneuten Aufstieg des Validierungsfehlers mündet. Den Effekt, dass das NN zwar die Trainingsdaten hinreichend genau abbilden kann, jedoch weitere, von den Trainingsdaten verschiedene Validierungsdaten

[8] dt. Überanpassung

nicht reproduzieren kann, bezeichnet man als mangelhaftes Extrapolationsvermögen. Da idR. ein allgemeingültiges Abbildungsvermögen, also ein gutes Extrapolationsvermögen gewünscht ist, wird im Rahmen des Trainings zusätzlich zu den Trainingsdaten oft ein Validierungsdatensatz betrachtet, der alleinig der Bestimmung des Scheitelpunktes entspricht, an dem idR. das Training abgebrochen und das Verhältnis von Trainingsdaten und Modellkomplexität abgeleitet wird. Der Prozess der Bestimmung einer geeigneten Modellkomplexität ist zentraler Bestandteil der Hyperparameteroptimierung (siehe Kap. 2.1.6).

2.1.3 Fehlerfunktionen

Die Menge möglicher Fehlerfunktionen zur Bestimmung des Prädiktionsfehlers eines NNes ist schier endlos. Dennoch haben sich einige populäre Varianten über die Jahre herausgestellt, von denen in diesem Kapitel zwei der Bekanntesten vorgestellt werden sollen. Generell gilt jedoch, dass die Wahl der Fehlerfunkton individuell auf das vorliegende Optimierungsproblem abgestimmt werden muss und keine finale Wahl im Vorhinein getroffen werden kann. Die endgültige Selektion der für ein explizites Optimierungsproblem am besten geeigneten Fehlerfunktion erfolgt daher idR. im Rahmen der Hyperparameteridentifikation (siehe Kap. 2.1.6). Dennoch können für die einzelnen Fehlerfunktionen allgemein gültige Kriterien aufgeführt werden, die der Orientierung und Vorauswahl zweckdienlich sein können.

Wahl der Fehlerfunktion

Die Wahl der bestmöglichen Fehlerfunktion ist individuell auf das betrachtete Optimierungsproblem abzustimmen. Kriterien zur Vorauswahl geeigneter Kandidatenfunktionen sind den Unterkapiteln von 2.1.3 zu entnehmen.

Mean Squared Error

Eine der wohl populärsten Fehlerfunktionen wird im ML-Jargon vornehmlich als *Mean Squared Error*[9] (MSE) bezeichnet. Dabei leitet sich MSE durch Quadrieren direkt von *Root Mean Squared Error* (RMSE) ab. Analog zur euklidischen Norm folgt RMSE der geometrischen Interpretation und beschreibt die Distanz zweier Punkte. Während RMSE somit der gleichen Skalierung wie Target und Prädiktion unterliegt und seine Interpretation daher sehr intuitiv ausfällt, wird idR. zum Training MSE verwendet. Für einen Trainingsdatensatz $\underline{X} = [\underline{x}_1, \ldots, \underline{x}_n]$, $\underline{Y} = \left[\underline{y}_1, \ldots, \underline{y}_n\right]$ von n Trainingsdatenpunkten $\left(\underline{x}_i, \underline{y}_i\right)$, ergibt sich MSE als Mittelwert der quadrierten euklidischen Norm zwischen Prädiktionen $\underline{y}_i \equiv \underline{y}\left(\underline{x}_i\right)$ und Targets $\underline{\tau}_i \equiv \underline{\tau}\left(\underline{x}_i\right)$ mit $\underline{\boldsymbol{\tau}} = [\underline{\tau}_1, \ldots, \underline{\tau}_n]$:

$$\mathrm{MSE}\left(\underline{Y}, \underline{\boldsymbol{\tau}}\right) := \frac{1}{n} \sum_{i=1}^{n} \left\| \underline{y}_i - \underline{\tau}_i \right\|_2^2 \qquad \text{(MSE-Fehlerfunktion)}$$

NNe, die mit MSE trainiert wurden, weisen idR. ein für die meisten Trainingsdatenpunkte gleichmäßig präzises Abbildungsverhalten auf. Das liegt daran, dass hohe Abweichungen

[9] dt. etwa „mittlerer quadrierter Fehler“

zwischen Prädiktion und Target bereits für einzelne Datenpunkte – sogenannte Ausreißer – zu überproportionalen Fehlern führen und den kumulierten Fehler stark verzerren. Geringe Abweichungen ≤ 1 führen dagegen zu kaum nennenswerten Fehlerwerten, sodass das Trainingsziel durch Wahl von MSE eine hinreichend genaue Approximation aller Trainingsdatenpunkte, jedoch nicht deren exakte Abbildung ist. Liegt zwischen Signal- und Modellkomplexität eine gewisse Diskrepanz vor, kann die Wahl von MSE als Fehlerfunktion schnell zu einer Überanpassung führen. Infolge der Untersuchungen, die im Rahmen dieser Arbeit bzgl. der Modellierung von Zeitreihendaten angestellt wurden, hat sich herausgestellt, dass MSE besonders dann eine gute Wahl ist, wenn der Anspruch des Modells vornehmlich auf der Verfolgung der durch die Datensequenzen beschriebenen Trajektorien und weniger der exakten Abbildung der einzelnen Schwingungskomponenten liegt. Analog führt die Verwendung von MSE im Rahmen des Trainingsprozesses dazu, dass die Gewichtsmatrizen des NNes idR. dicht besetzt[10] sind bzw. keine Nullstellung von Gewichten angeregt wird.

Kriterien für die Wahl von MSE

Durch die Wahl von MSE als Fehlerfunktion im Rahmen der Zeitreihenprädiktion wird insbesondere eine approximative Abbildung der Zustandstrajektorien über den gesamten Zeithorizont, weniger jedoch die exakte Beschreibung der Schwingung angeregt. Dabei ist bei Verwendung von MSE insbesondere auf eine Entfernung möglicher Ausreißer zu achten.

Mean Absolute Error

Die Fehlerfunktion *Mean Absolute Error*[11] (MAE) zählt zu einer der am häufigsten verwendeten Fehlerfunktionen. Analog zur MSE-Fehlerfunktion wird bei MAE ebenfalls die Differenz zwischen Prädiktion $\underline{Y}$ und Target $\underline{\tau}$ für gegebene Trainingsdaten $\underline{x}_i, \underline{\tau}_i, i \in [1, n]$ als Maß für die Güte der Prädiktion angenommen, allerdings geht die Differenz bei MAE linear in das Fehlermaß ein:

$$\mathrm{MAE}\left(\underline{Y}, \underline{\tau}\right) := \frac{1}{n} \sum_{i=1}^{n} \left| \underline{y}_i - \underline{\tau}_i \right| \qquad \text{(MAE-Fehlerfunktion)}$$

Der lineare Eintrag des Fehlers zwischen Netzprädiktion und Target hat zur Folge, dass Ausreißer, d.h. Punkte des Datensatzes, die hohe Streuung aufweisen, keinen überproportionalen Einfluss auf die allgemeine Abbildung haben, wie es etwa bei MSE der Fall ist. Aufgrund der relativ niedrigen Anfälligkeit ggü. Messrauschen und Ausreißern kann das durch Wahl von MAE implizierte Modellierungsziel wie folgt formuliert werden: die Minimierung des Fehlers für den Großteil der Datenpunktemenge bei gleichzeitiger Akzeptanz außerordentlich hoher Fehler für einzelne Punkte. Symptome der mit MAE trainierten NNe treten bei der Prädiktion von Zeitreihen bspw. in Form unzureichender Abbildung einzelner Ausschwingungen auf. So liefern die resultierenden NNe zwar für

[10] engl. non-sparse
[11] dt. etwa „mittlerer absoluter Fehler“

die Hauptmenge des Zeitbereichs akkurate Ergebnisse, tun dies jedoch auf Kosten der Abbildung vereinzelter Subschwingungen.

Zudem sei an dieser Stelle angemerkt, dass aufgrund der linearen Eigenschaften MAE idR. zu dünnbesetzten[12] Gewichtsmatrizen führt. Diese Eigenschaft ist kritisch für das Systemidentifikationsverfahren SINDY (siehe Kap. 2.2), bei dem es besonders darauf ankommt, dass nur einzelne Kandidatenfunktionen durch Gewichte > 0 eingehen, die meisten jedoch zu Null gesetzt werden. Analog dazu ergibt sich ebenfalls die Eignung von MAE für bestimmte Netzarchitekturen, die in Kapitel 4.6 beschrieben werden.

Kriterien für die Wahl von MAE

MAE eignet sich besonders dann, wenn die abzubildenden Messdaten ein komplexes dynamisches Verhalten aufweisen, das infolge einer vergleichsweise niedrigen Modellkomplexität nicht vollständig abgebildet werden kann. In solchen Fällen würde bei Wahl von MSE ein durchschnittlicher Trajektorienverlauf modelliert. Durch Wahl von MAE kann das Signal zu weiten Teilen präzise modelliert werden, wobei auf die Modellierung einzelner kurzzeitiger, teils signifikanter Ausschwingungen verzichtet wird. I.A. gilt es jedoch, verschiedene Fehlerfunktionen zu testen.

Logarithmisch skalierte Fehlerfunktionen

Die Modellierung technischer Systeme bedingt die Anforderung der Einhaltung minimaler Toleranzen, da bereits leichte Abweichungen starke Divergenz des Systemzustands zur Folge haben können. Dementsprechend klein fallen auch die erforderlichen Prädiktionsfehler aus, die, wie im Fall von MSE und MAE, im Mittel sogar erheblich kleiner Null sein sollten. Um auch für derart kleine Fehlermaße numerische Instabilitäten während des Trainingsprozesses zu vermeiden, haben sich in dieser Arbeit vorzüglich logarithmisch skalierte Fehlerfunktionen als besonders stabil herausgestellt. Die entsprechenden logarithmisch skalierten Varianten von MSE und MAE ergeben sich zu

$$\mathrm{LogMSE}\left(\underline{Y}, \underline{\tau}\right) := \log\left(\frac{1}{n}\sum_{i=1}^{n}\left(\underline{y}_i - \underline{\tau}_i\right)\right) \text{ und} \qquad \text{(LogMSE-Fehlerfunktion)}$$

$$\mathrm{LogMAE}\left(\underline{Y}, \underline{\tau}\right) := \log\left(\frac{1}{n}\sum_{i=1}^{n}\left|\underline{y}_i - \underline{\tau}_i\right|\right). \qquad \text{(LogMAE-Fehlerfunktion)}$$

Kriterien für die Wahl log. skalierter Fehlerfunktionen

Sind die erforderlichen Fehlermaße im Mittel deutlich kleiner Null, kann die logarithmische Skalierung der verwendeten Fehlerfunktion numerische Vorteile bieten.

[12] engl. sparse

2.1.4 Rekurrente Netze

Mit diesem Kapitel soll die Betrachtung azyklischer NNe abgeschlossen werden. Stattdessen soll nun jene Klasse NNe betrachtet werden, die besonders durch ihr Potential in der Modellierung sequentieller, bzw. zeitlich zusammenhängender Daten weiter an Bekanntheit gewinnt. Ein solcher zeitlicher Zusammenhang besteht z.B. bei der Aufnahme des Ein-/Ausgangsverhaltens eines dynamischen Systems zwischen den einzelnen Messungen. Zur Modellierung dieses Zusammenhangs wird zunächst angenommen, dass ein derartiges dynamisches System zu jedem Zeitpunkt t_k durch einen *Hidden State* $\underline{h}(t_k)$ charakterisiert werden kann. Dieses Vorgehen erfolgt also völlig analog zu der regelungstechnischen Betrachtung eines Systemzustands. Die Berücksichtigung zeitlich aufeinanderfolgender Systemanregung erfolgt beim *Rekurrenten Neuronalen Netz* (RNN) mithilfe eines rekurrenten Gliedes, d.h. einer Rückführung des Hidden States. Somit ergibt sich der aktuelle Hidden State $\underline{h}(t_k)$ immer aus dem Hidden State des vorangegangenen Zeitpunkts $\underline{h}(t_{k-1})$ sowie der aktuellen Netzeingabe $\underline{x}(t_k)$. Da sich auch der vorangegangene Hidden State wiederum aus dem ihm vorangegangenen Hidden State $\underline{h}(t_{k-2})$ sowie der Netzeingabe $\underline{x}(t_{k-1})$ berechnen lässt, kann der Hidden State $\underline{h}_k \equiv \underline{h}(t_k)$ demnach zum einen als Funktion aller vorausgegangenen Netzeingaben $\underline{x}_k \equiv \underline{x}(t_k)$ sowie eines initialen Hidden States $\underline{h}_0$ beschrieben werden:

$$\underline{h}_k = \underline{f}\left(\underline{x}_k, \underline{x}_{k-1}, \underline{x}_{k-2}, \ldots, \underline{x}_0, \underline{h}_0\right), \tag{2-3}$$

aber auch als Funktion des vorausgegangenen Hidden States $\underline{h}_{k-1}$ und der aktuellen Netzeingabe $\underline{x}_k$ aufgefasst werden:

$$\underline{h}_k = \underline{g}\left(\underline{h}_{k-1}, \underline{x}_k\right) \tag{2-4}$$

Nimmt man $\underline{h}_0$ als gegeben an, kann man ein RNN anhand Gleichung (2-3) ebenfalls als ein unendlich tiefes NN verstehen, bei dem sich die Netzeingabe aus allen aufeinanderfolgenden Eingaben $\underline{x}_k$ zusammensetzt sowie die Gewichte aufgrund des rekurrenten Aufbau des RNNes gekoppelt sind [SMG14, vgl.].

Grundlegender Ansatz

Grundsätzlich kann ein RNN, wie bereits erwähnt, in ausgeklappter Form durch ein (bis zu) unendlich tiefes vorwärtsgerichtetes NN mit gekoppelten Gewichten beschrieben werden (siehe Abb. 2-4). Das Verhalten eines RNNes wird dabei hauptsächlich durch die rekurrenten Zellen charakterisiert, die, analog Gleichung (2-4), den vorangegangenen Hidden State $\underline{h}_{k-1}$ und aktuellen Eingang $\underline{x}_k$ auf den neuen Hidden State $\underline{h}_k$ abbilden. Die Kopplung der Netzgewichte drückt sich dadurch aus, dass die wiederkehrenden Zellen jeweils dieselben Gewichte haben – die Gewichte werden also zellübergreifend einheitlich und simultan gelernt.

Das Training vorwärtsgerichteter NNe ist i.A. wenig fehleranfällig. Dies trifft nicht auf RNNe zu. Aufgrund ihrer extremen Tiefe und der Kopplung der Netzgewichte dauert das Training RNNe idR. vergleichsweise deutlich länger und wirft schwerwiegende numerische Probleme auf. Das wohl bekannteste numerische Phänomen RNNe, und Grund dafür, dass RNNe in einer gesonderten Architektur aufgebaut werden, wird auch als *Problem der verschwindenden bzw. explodierenden Gradienten*[13] bezeichnet. Da für RNNe der wäh-

[13] engl. vanishing/exploding gradients problem

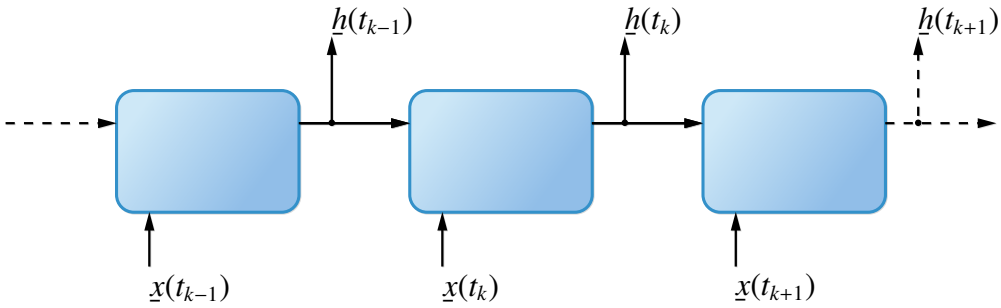

Bild 2-4: *Rekurrente Netze sind, im Vergleich zu vorwärtsgerichteten Netzen, aus Zellen aufgebaut. Dabei wird der interne Hidden State $\underline{h}_k$ zu einem bestimmten Zeitpunkt t_k aus dem jeweils vergangenen Hidden State $\underline{h}_{k-1}$ sowie der aktuellen Netzeingabe $\underline{x}_k$ berechnet. Das RNN ist hier in aufgefalteter Darstellung abgebildet.*

rend des Gradientenabstiegs (siehe Kap. 2.1.1) rückwärts propagierte Fehler „durch die Zeit“ exponentiell von den gekoppelten Gewichten abhängt, können während des Trainings Phänomene wie Oszillation der Gewichte oder extremes Ansteigen bzw. Abfallen des Fehlers auftreten [vgl. HS97]. Da die Änderung der Gewichte anhand des Gradienten der Fehlerfunktion bestimmt wird, kann diese Kopplung dazu führen, dass für die Gewichte keine nennenswerten Änderungen mehr bestimmt werden können – das Training wird extrem verzögert oder kommt gar zu einem kompletten Halt. So führen etwa Gewichtswerte < 4 in Kombination mit Sigmoid-Aktivierungsfunktionen (siehe Kap. 2.1.5) i.A. zu einem „Verschwinden“ des propagierten Fehlers. Die entsprechenden Neuronen des RNNes sind dann nicht weiter trainierbar. Dieser Effekt kann auch durch Anpassung anderer Faktoren wie z.B. der initialen Gewichtswerte sowie der Lernrate nicht umgangen werden. [HS97]

Die Literatur zum grundlegenden Ansatz RNNe zur Modellierung dynamischer Systeme geht teils weit zurück bis in die 90er Jahre. In chronologischer Reihenfolge: [FN93; KGW00; KGW00; SUZ06; SZ06; TD15]

Erweiterung des Ansatzes

Die genannten numerischen Probleme haben historisch zur Einführung von *Long-Short Term Memory* (LSTM), Memory Cells und sogenannten *Gates* geführt. Letztere, zu Deutsch „Gatter“, dienen der Lenkung des Informationsflusses durch das RNN. Durch sie können numerische Probleme wie verschwindende bzw. explodierende Gradienten weitestgehend umgangen werden. Das Prinzip dieses erweiterten Ansatzes stützt sich weiterhin auf das Konzept eines Hidden States $\underline{h}(t_k)$, der gemäß Gleichung (2-4) aus dem jeweils vorangegangenen Hidden State, sowie der Netzeingabe hervorgeht. Der Ein*fluss* dieser Größen wird nun allerdings mithilfe der Gates gesteuert. Mit diesem Ansatz werden Gewichtskonflikte umgangen, die aufgrund verschwindender oder explodierender Fehler auftreten. Durch die Einführung von LSTMs konnte somit zum ersten Mal ein vorwiegend stabi-

les Trainingsverhalten bewerkstelligt werden, mithilfe dessen es möglich war, RNNe zur Abbildung von Langzeiteffekten zu befähigen. Eine ausführliche Darlegung des Aufbaus und der Funktionsweise von LSTMs kann [HS97] entnommen werden.

Auch wenn die Einführung von LSTMs die Modellierung von Langzeiteffekten revolutioniert hat, ist das damit verbundene Training sowie die Prädiktion mitunter sehr rechenintensiv und dauert entsprechend lange. Die Motivation zur Echtzeitfähigkeit führte etwa zwei Dekaden später zur Einführung der *Gated Recurrent Unit* (GRU), die trotz ihrer Ähnlichkeit zum LSTM deutlich performanter ausfällt. Aus diesem Grund sind GRU-basierte RNNe auch im Rahmen dieser Arbeit die RNN-Variante der Wahl. Die folgenden Ausführungen sollen einen Überblick über die Funktionsweise von GRU geben, wobei für weitergehende Informationen an dieser Stelle auf [CvG+14] verwiesen werden soll.

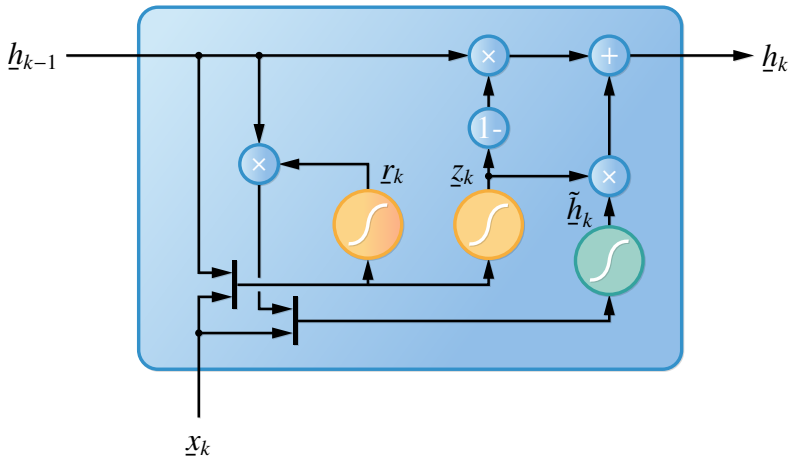

Bild 2-5: *Die Dynamik eines RNNs wird in diesem Fall über den Aufbau der GRU-Zelle definiert. Dabei wird durch das Update Gate $\underline{z}_k$ und das Reset Gate $\underline{r}_k$ der Fluss von Informationen zu und aus dem Hidden State $\underline{h}_k$ gesteuert.*

Im Vergleich zu LSTMs zeichnen sich GRUs durch einen simpleren Aufbau aus. Darüber hinaus ist auch die Funktionsweise einer GRU verhältnismäßig intuitiver (siehe Abb. 2-5). GRUs beschränken sich auf zwei Arten von Gates, *Update Gates* und *Reset Gates*. Dabei werden, analog zu ihrer Benamung, mithilfe der Update Gates $\underline{z}_k \equiv \underline{z}(t_k)$ Informationen zum Hidden State hinzugefügt und mithilfe der Reset Gates $\underline{r}_k \equiv \underline{r}(t_k)$ Informationen entfernt. Die Zustände der Gates zum Zeitpunkt t_k ergeben sich aus der Netzeingabe $\underline{x}_k$ und dem vorausgegangenen Hidden State $\underline{h}_{k-1}$:

$$\underline{z}_k = \underline{\sigma}_g \left(\underline{W}_z \underline{x}_k + \underline{b}_z + \underline{R}_z \underline{h}_{k-1} \right)$$
$$\underline{r}_k = \underline{\sigma}_g \left(\underline{W}_r \underline{x}_k + \underline{b}_r + \underline{R}_r \underline{h}_{k-1} \right)$$

Dabei wird als Gate-Aktivierungsfunktion $\underline{\sigma}_g$ idR. die Sigmoid-Funktion verwendet. Die Eingangsgewichte $\underline{W}_z$ bzw. $\underline{W}_r$ regeln den Einfluss der Netzeingaben $\underline{x}_k$ sowie die rekurrenten Gewichte $\underline{R}_z$ bzw. $\underline{R}_r$ den Einfluss des vorangegangenen Hidden States auf die jeweiligen Gates. [Mat] Mithilfe der Update und Reset Gates wird gesteuert, wie viel der vergangenen Zustandsinformationen für nachgeschaltete Zeitschritte relevant ist. Es kann bewiesen werden, dass durch die Einführung der Gates das Vanishing Gradient Problem verhindert werden kann [vgl. Kos17].

Die Auswertung der GRU erfolgt in drei konsekutiven Schritten:

1. Bestimmung der Reset und Update Gates
2. Berechnung des neuen Candidate States
3. Evaluation des neuen Hidden States

Nach Bestimmung von $\underline{z}_k$ und $\underline{r}_k$ wird im darauffolgenden Schritt ein *Candidate State* $\tilde{\underline{h}}_k$ berechnet,

$$\tilde{\underline{h}}_k = \underline{\sigma}_s \left(\underline{W}_{\tilde{h}} \underline{x}_k + \underline{b}_{\tilde{h}} + \underline{r}_k \odot \left(\underline{R}_{\tilde{h}} \underline{h}_{k-1} \right) \right)$$

aus dem sich im finalen Schritt der neue Hidden State ergibt:

$$\underline{h}_k = \left(1 - \underline{z}_k \right) \odot \underline{h}_{k-1} + \underline{z}_k \odot \tilde{\underline{h}}_k$$

Dabei denotiert $\underline{\sigma}_s$ die State-Aktivierungsfunktion, für die idR. der Tangens hyperbolicus (Tanh) (Kap. 2.1.5) verwendet wird. Der Einfluss des Reset Gates $\underline{r}_k$ auf die Weitergabe des vergangenen Hidden States $\underline{h}_{k-1}$ an den neuen Candidate State $\tilde{\underline{h}}_k$ sowie der Gewichtung des vergangenen Hidden States ggü. dem Candidate State durch das Update Gate $\underline{z}_k$ ist klar nachzuvollziehen. [Mat]

Trotz der numerischen Vorteile von LSTM und GRU gibt es weitere Komplikationen, die im Rahmen des Trainings auftreten können. Dazu zählen insbesondere die Stabilitätseigenschaften der verwendeten Aktivierungsfunktionen, die mehr oder weniger für die Tiefe der RNNe geeignet sind. Die Wahl geeigneter Aktivierungsfunktionen kann daher essentiell für die Gewährleistung eines stabilen Trainingsprozesses sein. Bislang wurde davon ausgegangen, dass entsprechende Aktivierungsfunktionen gewählt wurden. Das folgende Kapitel soll nun den unterschiedlichen Varianten von Aktivierungsfunktionen gewidmet werden.

2.1.5 Aktivierungsfunktionen

Eine Aktivierungsfunktion, oder auch Basisfunktion genannt, ist ein zentraler Bestandteil eines NNes, da sie die einzige Nichtlinearität in einem ansonsten strikt linearen Regressionsmodell darstellt. Analogien zu Kernelfunktionen von Gauß-Prozessen, Basisfunktionen bei Wavelet-Transformationen und Kandidatenfunktionen im Rahmen von SINDY (Kap. 2.2) sind hier zu vermerken. Bislang wurde in dieser Arbeit davon ausgegangen, dass jeweils geeignete Aktivierungsfunktionen gewählt wurden. Allerdings gibt es zahlreiche Faktoren, die bei der Wahl geeigneter Kandidatenfunktionen in Betracht gezogen werden müssen. In diesem Kapitel sollen ein paar der populärsten Aktivierungsfunktionen vorgestellt werden. Dabei erfolgt zudem eine Charakterisierung der Aktivierungsfunktionen für ihren Einsatz im Rahmen der Funktionsregression mit rekurrenten bzw. tiefen NNen.

Eine Aktivierungsfunktion beschreibt grundsätzlich eine Funktion, die einem skalaren Eingangssignal x ein entsprechendes Ausgangssignal $y = \mathcal{A}(x)$, oder auch *Aktivierung* genannt, zuordnet. Dabei werden im Rahmen von NNen bestimmte Aktivierungsfunktionen idR. aufgrund gewisser nichtlinearer Eigenschaften verwendet. Allerdings gibt es auch Einsatzszenarios, in denen als Aktivierungsfunktion die Identität eingesetzt wird.

So wählt man idR. die Identität als Aktivierungsfunktion für die Ausgabeneuronen von Regressionsnetzen. Darüber hinaus geht es bei der Auswahl allerdings hauptsächlich um die inhärenten Nichtlinearitäten sowie Sättigungseffekte. Auf diese soll nun weiter eingegangen werden.

Sigmoid

Die Sigmoid-Funktion, oder auch als logistische Funktion bezeichnet, ist eine der populärsten Aktivierungsfunktionen. Der Eingangswert der Sigmoid-Funktion wird auf einen Wertebereich zwischen 0 und 1 transformiert, wobei alle Eingangswerte deutlich größer 0 durch den inhärenten Sättigungseffekt der Sigmoid-Funktion auf 1 transformiert werden. Analog werden Eingangssignale deutlich kleiner 0 durchweg auf 0 transformiert.

$$\sigma(x) = (1 + \exp(-x))^{-1} \qquad \text{(Sigmoid-Aktivierungsfunktion)}$$

Dieser Sättigungseffekt ist Grund dafür, dass die Sigmoid-Funktion lediglich für einen Wertebereich um 0, jedoch nicht für Werte deutlich größer oder kleiner 0 geeignet ist. Wegen der Beschränkung des Ausgangs auf den Intervall $[0, 1]$ werden die Trainingsdaten daher idR. auf einen Wertebereich von 0 bis 1 normiert. Außerhalb dieses Sensitivitätsbereichs um den Mittelwert 0 werden entsprechende Signale nur noch mangelhaft prozessiert. Besonders im Rahmen der Fehler-Rückpropagation können unnormierte Trainingsdaten zu einem Verschwinden der Gradienten führen. Dies hat im schlechtesten Fall zur Folge, dass ein Training nicht möglich ist. [Bro19]

Limitationen von Sigmoid-Aktivierungsfunktionen

Aufgrund der mangelhaften Sensitivität außerhalb des Sättigungsbereichs um den Mittelwert 0 kann es im Rahmen des Trainings rekurrenter bzw. tiefer NNe zu numerischen Problemen bei der Bestimmung der Fehlergradienten kommen.

Tangens hyperbolicus

Eine weitere, populäre Aktivierungsfunktion ist der Tangens hyperbolicus (Tanh), dessen Eigenschaften sehr ähnlich zu denen der Sigmoid-Funktion sind. Historisch wird der Einsatz des Tanh über die Sigmoid-Funktion präferiert, da auf Tanh basierende Netze idR. eine bessere Trainierbarkeit sowie prädiktive Performanz aufweisen.

$$\sigma(x) = \tanh(x) \qquad \text{(Tanh-Aktivierungsfunktion)}$$

Analog zur Sigmoid-Aktivierungsfunktion ist der Tanh ebenfalls eine Sättigungsfunktion. Allerdings liegen die Ausgaben in diesem Fall zwischen -1 und +1, weshalb die Trainingsdaten idR. auf den Wertebereich $[-1, +1]$ normiert werden. Fallen Werte deutlich außerhalb dieses Bereichs, resultieren verschwindende Gradienten während des Trainings (siehe Vanishing Gradients Problem in Kap. 2.1.4). [Bro19]

Limitationen von Tanh-Aktivierungsfunktionen

Analog zur Sigmoid-Funktion treten bei Verwendung von Tanh-Aktivierungsfunktionen numerische Probleme im Zusammenhang mit tiefen NN auf. Dabei sind Tanh-Funktionen der Verwendung von Sigmoid.Aktivierungsfunktionen i.A. vorzuziehen.

ReLU

Durch die steigende Leistungsfähigkeit moderner Rechner wurde auch die Entwicklung immer tiefer werdender Netzwerkarchitekturen vorangetrieben. Aufgrund der schlechten Trainingscharakteristika der populären Aktivierungsfunktionen Sigmoid und Tanh für den Einsatz mit tiefen NNen stieg historisch bedingt die Popularität teilweise linearer Aktivierungsfunktionen wie der *Rectified Linear Activation Unit* (ReLU). Dabei handelt es sich um eine für positive Eingangswerte lineare Aktivierungsfunktion ohne Sättigung. Für Werte ≤ 0 ist die Funktion konstant gleich 0.

$$\sigma(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad \text{(ReLU-Aktivierungsfunktion)}$$

Die umschaltende Charakteristik von ReLU vereinigt die Vorteile linearer Aktivierungsfunktionen bzgl. ihrer guten Trainierbarkeit mit gradientenbasierten Optimierungsverfahren mit den Vorteilen nichtlinearer Aktivierungsfunktionen hinsichtlich ihres Abbildungsvermögens. Neben der daraus resultierenden Eignung für tiefe NNe existiert eine Reihe weiterer, entscheidender Vorteile. Aufgrund der simplen algebraischen Formulierung von ReLU ist ihre Berechnung weniger rechenintensiv ggü. Sigmoid und Tanh. Darüber hinaus ermöglicht das Plateau für Werte ≤ 0 ein numerisch robusteres Lernen von Gewichten exakt gleich Null, wohingegen Sigmoid und Tanh idR. nie das Trainieren exakt nullwertiger Gewichte zulassen. Damit ist eine dünne Besetzung[14] der Gewichtsmatrix möglich. [Bro19]

All diese Vorteile führen dazu, dass das Training tiefer NNe mithilfe von ReLU-Aktivierungsfunktionen vielfach schneller ist, als mit Tanh [KSH12]. Daher wird für moderne NNe der Einsatz von ReLU-Aktivierungsfunktionen empfohlen [GBC16]. Auch wenn ReLU-Funktionen gerade wegen der fehlenden Sättigung in der positiven Domäne beliebig große Werte ausgeben können, wurde zunächst vermutet, dass sie dadurch für RNNe ungeeignet seien. Es konnte allerdings gezeigt werden, dass bei entsprechend konservativer Initialisierung der Gewichte, und somit der Gewährleistung eines stabilen Netzzustandes zu Anfang des Trainings, ein entsprechendes „Explodieren“ der Netzausgaben vermieden werden kann [QNG15; Bro19].

Mögliche Limitationen von ReLU-Aktivierungsfunktionen werden in den Kapiteln zu L2-Regularisierung (2.1.7) sowie LReLU und ELU 2.1.5 adressiert.

[14] engl. sparsity

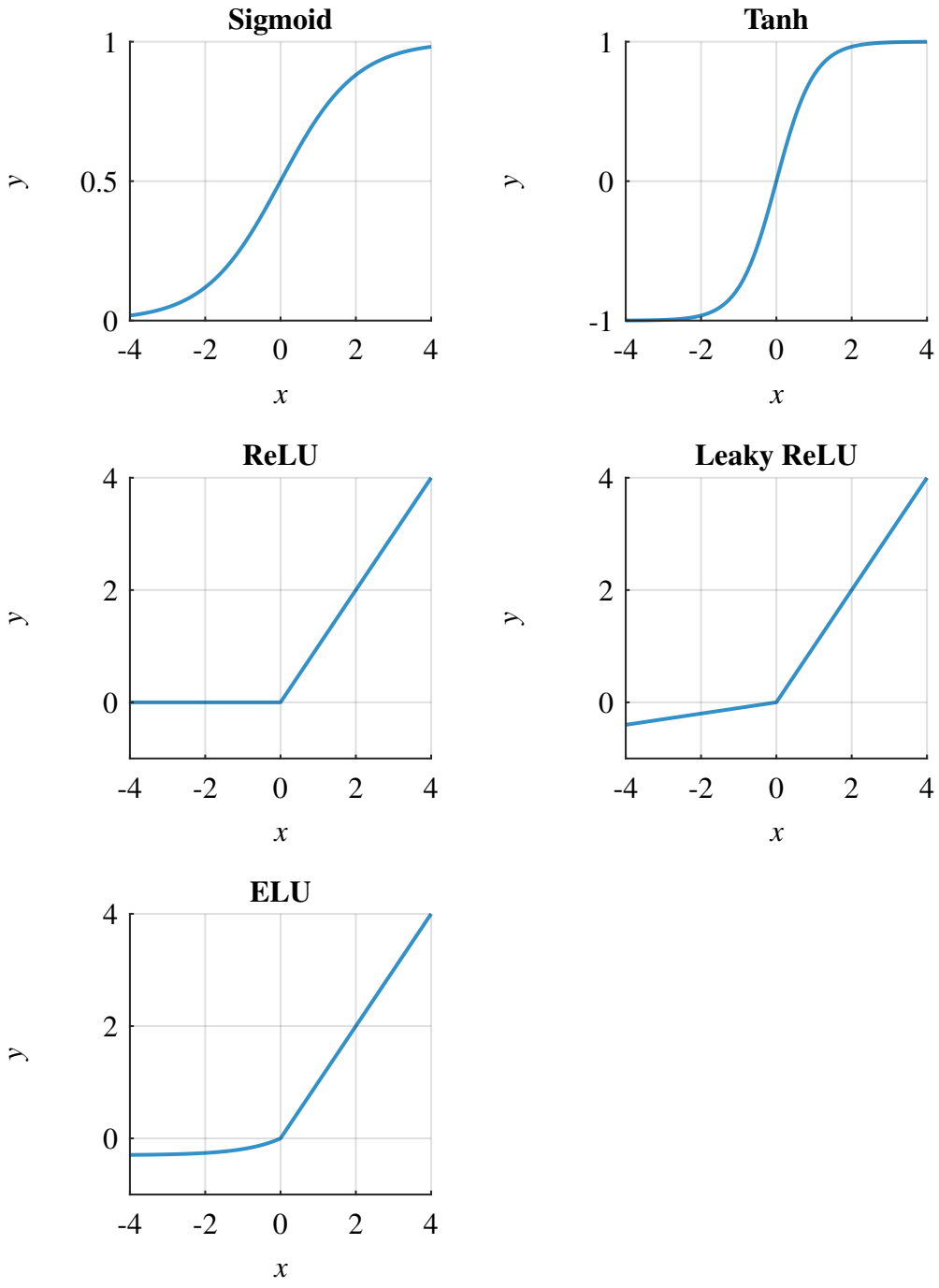

Bild 2-6: *Die Abbildung enthält eine Übersicht der im Rahmen dieser Arbeit beschriebenen Aktivierungsfunktionen. Dabei werden die jeweiligen nichtlinearen Eigenschaften deutlich. Für die Aktivierungsfunktionen Leaky ReLU und ELU wurde zur Veranschaulichung ein relativ hoher Glättungsfaktor α von 0.1 bzw. 0.3 gewählt.*

Einsatz von ReLU-Aktivierungsfunktionen

ReLU bietet ggü. Sigmoid und Tanh enorme Vorteile bzgl. des Trainingsverhaltens. Durch die Kombination linearer und nichtlinearer Charakteristika gewährleisten ReLU-Aktivierungsfunktionen neben einer guten Abbildung insbesondere Stabilität der gradientenbasierten Optimierungen, Sparsity und niedriger Berechnungsanforderungen. Sollen ReLU-Funktionen in Kombination mit RNNen verwendet werden, muss eine stabile Gewichtsinitialisierung gewährleistet werden. Limitationen können durch Ergänzung um L2-Regularisierung oder erweiterte Ansätze wie LReLU oder ELU adressiert werden.

Leaky ReLU und ELU

Trotz der überlegenen Eigenschaften von ReLU-Aktivierungsfunktionen birgt das inhärente Plateau für Werte ≤ 0 auch numerische Schwierigkeiten. Im Rahmen des Trainings können vereinzelt Neuronen dauerhaft mit negativen Eingaben beaufschlagt sein. Infolge des Plateaus für negative Eingaben sind dann alle Gradienten gleich Null und die Ausgabe verweilt für den Rest des Trainings bei einem konstanten Wert. Bei diesem Effekt spricht man von einem Absterben von Neuronen[15] bzw. *toten Neuronen*. Um dieses Absterben zu verhindern, wurden Modifikationen von ReLU eingeführt, die lediglich die negative Domäne der Funktion betreffen. Zu diesen zählen etwa Leaky ReLU (LReLU) und Exponential Linear Units (ELU). [Bro19]

$$\sigma(x) = \left\{ \begin{array}{ll} x, & x \geq 0 \\ \alpha x, & x < 0 \end{array} \right\} = \max(\alpha x, x) \qquad \text{(Leaky-ReLU-Aktivierungsfunktion)}$$

$$\sigma(x) = \left\{ \begin{array}{ll} x, & x \geq 0 \\ \alpha\left(e^x - 1\right), & x < 0 \end{array} \right\} = \max\left(\alpha\left(e^x - 1\right), x\right) \qquad \text{(ELU-Aktivierungsfunktion)}$$

Im Falle von LReLU wird durch das Hinzufügen einer leichten Steigung im negativen Funktionsbereich die Berechnung von Gradienten ungleich Null ermöglicht und somit ein Absterben der Neuronen verhindert [And13]. Analog wird die Berechnung eines entsprechenden Gradienten und die Erhaltung der Neuronen im Fall von ELU durch das Hinzufügen eines exponentiellen Verlaufs im negativen Wertebereich erreicht [CUH15]. Der Faktor α bestimmt dabei die Stärke der Ausprägung der beschriebenen Modifikationen (in Abb. 2-6 mit α gleich 0.1 für LReLU bzw. 0.3 für ELU). Ebenfalls hilfreich kann darüber hinaus die Regularisierung der Netzgewichte (Kap. 2.1.7) sein. [Bro19]

Einsatz von Leaky ReLU und ELU

Falls bei Einsatz von ReLU-Aktivierungsfunktionen sogenannte tote Neuronen auftreten, kann dem durch Verwendung von Leaky ReLU oder ELU entgegengewirkt werden.

[15] engl. dying ReLU

2.1.6 Hyperparameteridentifikation

Unter dem Begriff der *Hyperparameteridentifikation* oder *Hyperparameteroptimierung* versteht man die Bestimmung der Hyperparameter (HP) in Form eines der eigentlichen Optimierung übergeordneten Optimierungsprozesses. Zur Bestimmung der (näherungsweise) optimalen Hyperparametrierung werden iterationsweise verschiedene Hyperparametrierungen gewählt und der unterlagerte Optimierungsprozess, in diesem Fall das Training des NNes, durchgeführt. Durch Vergleich der mit unterschiedlichen Hyperparametrierungen erzielten Prädiktionsgüten der resultierenden Netze kann sich sukzessive der optimalen Parametrierung angenähert werden.

Die Bestimmung der Prädiktionsgüte als Maß für die Güte der gewählten HPer erfolgt anhand eines separaten *Test*datensatzes, der von den Trainings- und Validierungsdaten verschieden ist. Auf diese Weise wird sichergestellt, dass das resultierende NN ein gutes Extrapolationsverhalten aufweist. Dass die Verwendung eines unabhängigen Datensatzes von äußerster Wichtigkeit ist, wird klar, wenn die Schar möglicher Hyperparameter betrachtet wird. Eine der wohl am häufigsten im Rahmen von HP-Optimierungen betrachtete Optimierungsgröße ist die Anzahl der versteckten Neuronen des NNes. IdR. gilt, je höher die Anzahl gewählt wird, desto umfangreicher wird das Abbildungsvermögen des NNes. Gleichzeitig führt eine höhere Neuronenzahl allerdings auch zu einem verminderten Extrapolationsvermögen (siehe Overfitting in Kap. 2.1.2) sowie einer steigenden Berechnungsdauer – jedenfalls bei beschränkter Datenmenge. Es gilt demnach im HP-Optimierungsprozess ein Optimum zwischen den beiden gegenläufigen Faktoren zu finden. Als Gütemaß kann dafür die Prädiktionsgüte auf Basis eines unabhängigen Datensatzes zurate gezogen werden.

Neben der Anzahl der versteckten Neuronen bzw. der Modellkomplexität, können eine Vielzahl verschiedener Trainingsparameter optimiert werden. Grundsätzlich gilt allerdings, dass manche Parameter, wie etwa die Modellkomplexität, für jedes Optimierungsproblem und die Menge an Trainingsdaten individuell justiert werden muss, während andere Parameter kaum Anpassung benötigen. Um den übergeordneten Optimierungsprozess möglichst schlank zu halten, wird daher darauf verzichtet, eine Vielzahl unterschiedlicher Parameter zu optimieren und es wird sich auf die wichtigsten Parameter beschränkt. Im Rahmen dieser Arbeit wurde daher idR. lediglich die Anzahl der versteckten Neuronen optimiert. Da eine geeignete Hyperparametrierung zu finden jedoch i.A. kritisch für das Training eines NNes ist, sollte im Zweifel bzw. bei schlechten resultierenden Trainingsfehlern oder großen Diskrepanzen zwischen Trainings-, Validierungs- und/oder Testfehlern eine unpassende Hyperparametrierung in Betracht gezogen werden.

Wahl der Hyperparametrierung

Eine geeignete Hyperparametrierung ist essentiell für das Training eines NNes. So kann eine abweichende Hyperparametrierung zu völlig verschiedenen Ergebnissen führen. Für die meisten Parameter gibt es allerdings spezifische Wertebereiche und Toleranzen, in denen diese recht frei gewählt werden können.

Methoden

Zur Optimierung der Hyperparameter können verschiedene Methoden herangezogen werden, die jeweils unterschiedliche Charakteristika aufweisen. Die drei wohl gängigsten Methoden sind Random Search, Grid Search und Bayessche Optimierung.

Das wohl simpelste Vorgehen zur Optimierung der Hyperparameter besteht in dem schieren „Ausprobieren“ zufälliger Parametrierungen und der anschließenden Auswahl der performantesten Variante. Dieses Vorgehen wird als ***Random Search***[16] bezeichnet, ist sehr einfach zu implementieren und erfordert am wenigsten Rechenleistung zur Bestimmung einer geeigneten Hyperparametrierung.

Als direktes Pendant zu Random Search wird oft ***Grid Search***[17] angesetzt. Dabei wird der durch alle zu optimierenden HPer aufgespannte Optimierungsraum durch ein Raster (engl. grid) untergliedert und NNe für die Parametrierungen der Rasterpunkte ausgewertet. Entgegen der Annahme, dass dieses strategische Vorgehen Vorteile ggü. Random Search bieten könnte, liefert Random Search idR. deutlich bessere Ergebnisse als Grid Search, da der Wertebereich aller zu optimierender Parameter für mehr unterschiedliche Werte beobachtet wird [JY12]. Daher ist Random Search Grid Search idR. vorzuziehen.

Gilt es, die bestmögliche Hyperparametrierung mit möglichst wenigen Auswertungen bzw. Experimenten zu bestimmen, so ist weder Grid Search noch Random Search geeignet. In diesem Fall ist die Optimierung der Hyperparameter via ***Bayesscher Optimierung*** (BO) zu präferieren. Bei der BO werden parallel zur Auswertung von Hyperparametrierungen die dabei erzielten Netzgüten analysiert. Aus den erfassten Daten wird ein probabilistisches Modell erstellt, anhand dessen vor jeder Auswertung genau die Parametrierung bestimmt wird, für die eine höchstmögliche Güte am wahrscheinlichsten scheint. Mithilfe dieses modellbasierten Ansatzes können bereits nach wenigen Auswertungen nahezu optimale Hyperparametrierungen bestimmt werden. So ausgefeilt die BO zur Bestimmung der Hyperparameter sein mag, ist sie auch sehr rechenintensiv. Für Hyperparametrierungen mit mehr als 10 Dimensionen ist die traditionelle BO daher idR. nicht mehr geeignet. Ein möglicher Ansatz ist dann z.B. sparse BO. Für ausführliche Ausführungen soll an dieser Stelle auf [Bis09], [Sch19] und [Bij16] verwiesen werden.

Lernrate

Gleichwohl im Rahmen der HP-Optimierung diverse Parameter optimiert werden können, nimmt die Lernrate eine ganz besondere Rolle ein. Die bereits in Kapitel 2.1.1 beschriebene Lernrate[18] wird häufig nicht konstant sondern adaptiv gestaltet. Das bedeutet, dass sich der Wert der Lernrate über den Lauf der Epochen ändert. Das Regime bzw. die Strategie anhand derer die Lernrate angepasst wird, wird auch als *Learning Rate Schedule* bezeichnet. IdR. wird dafür eine initiale Lernrate definiert sowie ein *Learning Rate Decay Factor*, um den die Lernrate jede Epoche abgesenkt wird. Durch das Absenken der Lernrate soll ein Überschießen des zu erreichenden Optimierungsziels vermieden werden. Stellt man sich die Lernrate als ein Maß für die Distanz vor, die man im Parameter-

[16] dt. Zufallssuche
[17] dt. Rastersuche
[18] engl. learning rate

raum entlang des negativen Gradienten wandert, so nähert man sich iterativ, in progressiv kleiner werdenden Schritten, dem Optimierungsziel an. Dieses Verhalten ist für eine Konvergenz des Trainingsprozesses besonders erstrebenswert. Zur Kompensation der dadurch verursachten, sinkenden Konvergenzgeschwindigkeit, ist der vorwiegend genutzte Optimierungssolver Adam (siehe Kap. 2.1.1, letzter Absatz) mit einem epochenübergreifenden Momentterm ausgestattet.

Im Rahmen dieser Arbeit wurde ein adaptiver Learning Rate Schedule entwickelt und verwendet, bei dem abhängig vom Verlauf des Validierungsfehlers, Modifikationen der Lernrate angenommen werden. Im Fall eines „Überschießens“ des Minimums tritt ein Wiederanstieg des Validierungsfehlers auf, der mit einem Absenken der Lernrate bestraft wird. Darüber hinaus wird bei Stagnation des Validierungsfehlers ebenfalls die Lernrate abgesenkt, um die Auflösung der Optimierung zu erhöhen. Die Wahl der jeweiligen Faktoren wurde dabei so angenommen, dass ein zurückhaltender Ansatz resultierte. Die Rückhaltsamkeit drückt sich dadurch aus, dass die Konvergenz langsamer vorgenommen wird, als sie potentiell durchgeführt werden könnte, dadurch aber möglichst hohe Ergebnisgüten gewährleistet werden sollen. Durch einen vorschnellen Abstieg und frühen Abbruch des Optimierungsprozesses würde es möglicherweise nicht zu einer ausreichenden Konvergenz kommen. Auch wenn also durchaus schnellere Trainingsprozeduren möglich sind, liegt bei dem in dieser Arbeit verwendeten Trainingsansatz der Fokus auf dem Erreichen einer höchstmöglichen Güte.

2.1.7 Numerische Aspekte

Das Training RNNe stellt besonders im Hinblick auf die Numerik eine Herausforderung dar. Um ein stabiles Training sowie eine hinreichende Konvergenz zu gewährleisten, sind einige numerische Aspekte zu beachten. Werden diese Aspekte nicht berücksichtigt, kann es schnell zu schlechten Trainingsergebnissen sowie verschwindenden oder explodierenden Gradienten (Kap. 2.1.4) kommen.

Normierung

Wie bereits in Kapitel 2.1.5 beschrieben, werden die Trainingsdaten idR. normiert, um in den sensitiven Wertebereich der verwendeten Aktivierungsfunktionen zu fallen. Auch wenn in einem Szenario lediglich lineare Aktivierungsfunktionen verwendet werden, ist die Normierung der Trainingsdaten durchaus üblich. Die Normierung resultiert nämlich auch in der Beschränkung der resultierenden Fehlerwerte, sodass numerische Instabilitäten aufgrund sehr hoher oder extrem kleiner Werte vermieden werden. Generell gilt im Rahmen des Trainings NNe, dass der Wertebereich der Eingaben und Ausgaben sowie daraus resultierend, der Fehler, von hoher Bedeutung für die numerische Stabilität des Trainingsprozesses ist.

In dieser Arbeit werden RNNe zur Modellierung dynamischer Systeme verwendet. Die Zustandsvariablen eines Systems können dabei idR. grundsätzlich Werte zwischen $+\infty$ und $-\infty$ annehmen. Um dennoch eine Normierung gewährleisten zu können, ist entweder initial ein entsprechender Wertebereich zu bestimmen oder ein adaptiver Ansatz zu wählen, bei dem in der initialen Phase des Trainings die entsprechenden Extremwerte auf

Basis der Trainingsdaten identifiziert und als Orientierungswerte verwendet werden. Es resultiert idR. eine erhöhte Sensitivität und Akkuranz des NNes um den Mittelwert des Normierungsbereichs, sodass der Wertebereich nicht zu groß gewählt werden sollte.

Notwendigkeit der Normierung

Werden nichtlineare Aktivierungsfunktionen verwendet, so ist die Normierung der Trainingsdaten zu empfehlen, um die Daten auf den Sensitivitätsbereich der Aktivierungsfunktionen zu transformieren. Die Normierung ist besonders wichtig für ein numerisch stabiles Trainingsverhalten.

Toleranzen und Datengüte

Die in dieser Arbeit trainierten Netze werden zur Modellierung technischer Systeme und zur anschließenden Simulation verwendet. Daher häufen sich modellinhärente Fehler zeitlich an. Um trotz der Kummulation akkurate sowie stabile Simulationen durchführen zu können, müssen extrem kleine Prädiktionsabweichungen eingehalten werden. Um trotz der kleinen Fehlerwerte eine ausreichend numerisch stabile Bestimmung der Fehlergradienten zu gewährleisten, wurden in dieser Arbeit vornehmlich logarithmisch skalierte Fehlerfunktionen (Kap. 2.1.3) verwendet.

Reicht die Akkuranz der trainierten Netze nicht zur Prädiktion angemessen langer Zeitreihen aus, kann ein Multiscale Time-Stepper verwendet werden, bei dem mehrere Netzinstanzen parallel für unterschiedliche zeitdiskrete Schrittweiten trainiert werden. Durch hierarchische Sequenzierung der Netzvarianten kann damit das Langzeitverhalten eines Systems deutlich länger und präziser abgebildet werden. [LKB20]

Neben der Einhaltung gewisser Toleranzen ist auch eine bestimmte Güte der verwendeten Trainingsdaten notwendig, um entsprechend hochwertige Simulationsergebnisse zu generieren. Dabei bezieht sich die Datengüte in diesem Fall auf das inhärente Messrauschen. Allgemein ist daher eine gewisse Datenvorverarbeitung empfehlenswert und wird im Fortlauf vorausgesetzt. Es hat sich allerdings gezeigt, dass das Training sogar mit stark verrauschten Trainingsdaten hochwertige Ergebnisse erzielen kann, wenn ausreichend Trainingsdaten verfügbar sind. Dies lässt sich logisch insbesondere daraus ableiten, dass bei einer entsprechend kleinen Lernrate während des Trainingsprozesses in etwa der Mittelwert der durch die verrauschten Trainingsdaten beschriebenen Trajektorien und somit das zugrundeliegende Dynamikverhalten gelernt wird. Um im Fortlauf der Arbeit die Trainingsdaten möglichst gering halten zu können und da eine Datenvorverarbeitung mit oder ohne geeignetem Beobachter durchaus ein realistisches Szenario darstellt, wird in den folgenden Kapiteln von hochwertigen Trainingsdaten ausgegangen, die kein Rauschen beinhalten.

L2-Regularisierung und Dropout

Es gibt im Grunde zwei Ansätze, um Overfitting für eine gegebene Netzarchitektur zu minimieren:

1. Mehr Trainingsdaten

2. Regularisierung.

Da von einer relativen Armut an verfügbaren Daten ausgegangen wird, soll in diesem Kapitel kurz auf das Thema Regularisierung eingegangen werden. Für eine ausführliche Beschreibung des Themas sowie Herleitungen soll an dieser Stelle daher auf [Pei19] verwiesen werden.

Unter dem Begriff der *Regularisierung* versteht man grundsätzlich eine bestimmte Beeinflussung der Netzgewichte, die zur Regulierung der Modellkomplexität dient. Wie bereits in Kapitel 2.1.2 beschrieben, bestimmt die Wahl der Anzahl versteckter Neuronen und Schichten die Modellkomplexität eines NNes. Sie hat somit insbesondere einen großen Einfluss auf das Auftreten von Overfitting. Regularisierung nimmt nun auf eine andere Weise Einfluss auf die Modellkomplexität, um dem Overfitting entgegenzuwirken.

Die ***L2-Regularisierung***[19] dient zur Bestrafung großer Gewichtswerte. Dabei wird der Fehlerfunktion ein Regularisierungsterm hinzugefügt (siehe Gl. (2-5)). Durch die L2-Norm über alle Gewichte führen Gewichte deutlich größer 1 zu einem überproportionalen Anstieg des Trainingsfehlers – die Senkung der einzelnen Gewichte wird motiviert. Niedrige Gewichtswerte verringern den Einfluss der jeweiligen Aktivierungsfunktion und führen insgesamt zu einer Reduzierung der Modellkomplexität sowie Sparsity. Infolgedessen wird auch das Auftreten möglichen Overfittings reduziert. [Pei19] Werden ReLU-Aktivierungsfunktionen verwendet, kann der Einsatz einer L2-Regularisierung der Gewichte auf gleiche Weise dem unbeschränkten Anstieg der Neuronenausgaben in der positiven Domäne entgegenwirken [GBB11; Bro19].

$$J\left(\underline{Y}, \underline{\tau} | \underline{w}\right) = \text{Loss}\left(\underline{Y}, \underline{\tau}\right) + \alpha \left\| \underline{w} \right\|_2 \tag{2-5}$$

Eine weitere Form der Regularisierung wird auch als ***Dropout*** bezeichnet und basiert auf der Reduktion der Neuronenzahl. Dafür wird für alle Neuronen ein Threshold-Wert angenommen, der eine Wahrscheinlichkeit angibt, für die ein jedes Neuron entfernt werden soll oder nicht. Wird dieser Threshold bspw. zu 0.7 gesetzt, so gilt für jedes Neuron die 30%-ige Wahrscheinlichkeit, dass es während des Trainings entfernt wird. Diese Ausdünnung des NNes führt zu enormen Leistungssteigerungen, weil sich keine einzelnen Pfade hoher Gewichte herausbilden können, auf deren Existenz sich die Prädiktion stützt. Im Umkehrschluss induziert dies eine Verteilung der Informationsflüsse und eine Unterdrückung einzelner, großer Gewichtswerte. Analog zur L2-Regularisierung resultiert aus den beschriebenen Effekten die Reduktion der Modellkomplexität. [Pei19]

2.2 Sparse Identification of Nonlinear Dynamics (SINDY)

Mit *Sparse Identification of Nonlinear Dynamics*[20] (SINDY) [SJJ16] soll sich zuletzt einem jüngst (2016) eingeführten Modellierungsverfahren zur Systemidentifikation dynamischer Systeme angenommen werden. Im Rahmen von SINDY werden die Dynamikgleichungen eines Systems direkt gelernt. Dafür wird zunächst eine Bibliothek diverser

[19] engl. L2-regularization

[20] Genauer ist hier die erweiterte Variante Sparse Identification of Nonlinear Dynamics with Control, kurz SINDYc, gemeint.

Funktionen der Zustands- und Eingangsgrößen, $\underline{X}$ und $\underline{\Upsilon}$, in Form eines Spaltenvektors von Kandidatenfunktionen

$$\underline{\Theta}^T\left(\underline{X}, \underline{\Upsilon}\right) = \begin{bmatrix} \text{———} & \underline{1} & \text{———} \\ \text{———} & \underline{X} & \text{———} \\ \text{———} & \underline{\Upsilon} & \text{———} \\ \text{———} & \underline{X}^2 & \text{———} \\ \text{———} & \underline{\Upsilon}^2 & \text{———} \\ & \vdots & \\ \text{———} & \sin\left(\underline{X}\right) & \text{———} \\ \text{———} & \sin\left(\underline{\Upsilon}\right) & \text{———} \\ & \vdots & \end{bmatrix}$$

aufgestellt, wobei die zeitlichen Abfolgen von m Zustands- und Eingangsvektoren $\underline{x}_k$ bzw. $\underline{u}_k$ durch

$$\underline{X} = \begin{bmatrix} | & | & & | \\ \underline{x}_1 & \underline{x}_2 & \dots & \underline{x}_m \\ | & | & & | \end{bmatrix} \text{ und } \underline{\Upsilon} = \begin{bmatrix} | & | & & | \\ \underline{u}_1 & \underline{u}_2 & \dots & \underline{u}_m \\ | & | & & | \end{bmatrix}$$

denotiert werden. Wird nun in einem Supervised-Learning-Regime via *Sparse Regression*[21] eine Koeffizientenmatrix $\underline{\Xi}$ in Form eines Spaltenvektors von Koeffizienten

$$\underline{\xi}_k = \arg\min_{\underline{\xi}_k} \left\| \dot{\underline{X}}_k - \underline{\xi}_k \underline{\Theta}^T\left(\underline{X}, \underline{\Upsilon}\right) \right\|_2 + \alpha \left\| \underline{\xi}_k \right\|_1$$

bestimmt, so ergibt sich die Dynamikgleichung des Systems durch

$$\dot{\underline{X}} = \underline{\Xi}\underline{\Theta}^T\left(\underline{X}, \underline{\Upsilon}\right).$$

Der Parameter α dient dabei zur Einstellung des Verhältnisses von Modellkomplexität zu -akkuranz. Mithilfe von SINDY können die Dynamikgleichungen diverser komplexer technischer Systeme gelernt werden. Dabei ist jedoch auf eine ausreichende Datengüte und -verfügbarkeit sowie eine entsprechende Wahl der Kandidatenfunktionen zu achten.

Neben SINDY gibt es eine Reihe von Vertretern der Klasse der *Symbolischen Regression*, mit denen durch geschickte hierarchische Zerlegung der Dynamikgleichungen die direkte mathematische Bescheibung gelernt werden kann. Einer der fortschrittlichsten Algorithmen zur Symbolischen Regression ist *AI Feynman*, dessen Quellcode als Open Source zur Verfügung steht [UT20]. Im Rahmen dieser Arbeit wird das Konzept von SINDY in Kapitel 3.8 aufgegriffen und im Kontext von Physics-guided Neural Networks adaptiert.

[21] Regression mit L1-Regulierung bzw. LASSO-Regularisierung

3 Physics-guided Neural Networks

Seit ihrer Einführung in 2017 durch Karpatne et al. hat die Forschung rund um das Thema *Physics-guided Neural Networks* (PGNN) eine Vielzahl weiterer Entwicklungen und Veröffentlichungen mit sich gezogen. Dabei stehen diverse Termini wie *physics-guided*, *theory-guided*, *domain-adapted*, *physics-informed* oder *physics-based* alle für eine Klasse Neuronaler Netze, bei der zusätzliches, domänenspezifisches Wissen im Trainingsprozess eingebracht wird. In welcher Form dieses Domänenwissen eingebracht wird, variiert stark zwischen den verschiedenen Ansätzen. Ein Überblick über eine Reihe bekannter Ansätze wird im folgenden Kapitel 3.1 gegeben.

Bislang wurden PGNNs in diversen Fachgebieten zur Modellierung stationären Ein-/ Ausgangsverhaltens von Systemen sowie der Lösung von ODEs und PDEs eingesetzt. In dieser Arbeit soll nun untersucht werden, inwiefern sich PGNNs zur Identifikation der Dynamik technischer Systeme eignen und welches Potential sie im Bereich Regelungstechnik bieten.

3.1 Ansätze

Es gibt bereits eine Vielzahl unterschiedlicher Ansätze, domänenspezifisches Wissen in einen datengetriebenen Lernprozess einzubinden. Dabei wird jeweils auf mindestens einer traditionellen datengetriebenen Methode aufgebaut und diese um gewisse wissensbasierte Merkmale erweitert. Diese Erweiterungen können dabei diverse Formen annehmen. In diesem Kapitel soll zunächst ein Überblick über bereits untersuchte Kombinationen gegeben werden, wobei detaillierte Informationen Tabelle 3-1 entnommen werden können. Dabei wird mit dieser Auflistung kein Anspruch auf Vollständigkeit erhoben, sondern lediglich ein breiter Überblick angestrebt. Ebenfalls einen guten Überblick bieten [KAF+17] und [WJX+20].

Bislang wurden PGNNs hauptsächlich zur Modellierung der Dynamik autonomer Systeme ohne explizite Steuergröße verwendet. So untersuchten bspw. Karpatne et al. die zeitliche Entwicklung der Temperaturprofile in Seen infolge externer Störfaktoren wie z.B. Wettereinflüssen [KWRK17; JWK+18]. Viele Arbeiten beschäftigen sich zudem mit der Identifizierung und Lösung von PDEs bzw. ODEs aus Messdaten unter Berücksichtigung physikalischer Gesetzmäßigkeiten. Die Hinzunahme des Domänenwissens ermöglichte dabei u.a. eine Verkürzung der Trainingsdauer, die Identifikation und Lösung der Gleichungssysteme in Situationen sparsamer und/oder stark verrauschter Trainingsdaten [CZAB19], eine höhere Prädiktionsgüte [WWY20] sowie die Generierung physikalisch valider Modelle – in Anbetracht ausgewählter physikalischer Gesetzmäßigkeiten [LRP19]. In einer jüngst erschienenen Veröffentlichung von Antonelo et al. hat sich zum ersten Mal ein Team mit dem Einsatz von PGNNs in der Regelungstechnikdomäne, genauer zur Nutzung in Kombination mit MPC, beschäftigt [ACS+21]. Die in diesem Rahmen verwendeten PINCs ersetzen dabei einen numerischen Solver wie z.B. Runge-Kutta, basieren vollkommen auf der Annahme bekannter ODEs und sind zeitdiskret. Im Rahmen

dieser Arbeit wird dagegen angenommen, dass verfügbare ODEs alleine zu ungenau sind oder gar nicht vorliegen.

In den verschiedenen Veröffentlichungen kamen die unterschiedlichsten datengetriebenen Ansätze zum Einsatz. Dabei reichen die verwendeten Modelle von einfachen vorwärtsgerichteten NNen über DNNs, RNNen und GANs zu komplett maßgeschneiderten Architekturen wie LNNs und HNNs, welche ihrerseits wiederum aus den vorangegangenen Modellen zusammengesetzt sind. Generell gilt für PGNNs, liegt Vorwissen über Aufbau und Struktur des zu lösenden Problems bzw. seiner Lösung an sich vor, sollte dieses beim Architekturentwurf berücksichtigt werden. So kann beispielsweise ein komplexes Gesamtsystem auf Separabilität untersucht und in modulare Einzelsysteme[22] zerlegt werden, deren jeweilige Dynamik unabhängig voneinander identifiziert und gelöst werden kann. Das aus einzelnen Sub-PGNNs zusammengesetzte Gesamtmodell ließe sich dann in der Folge durch ein abschließendes Training mit geringer Lernrate finalisieren bzw. kalibrieren.

Ist darüber hinaus Wissen über bekannte Symmetrien oder Abhängigkeiten verfügbar, kann dies zur Modellordnungsreduktion eingebracht werden. Eine Klasse NNe mit erzwungener Symmetrie belegen Equivariant Neural Networks [WWY20]. Der Effekt durch Berücksichtigung einzelner Symmetrieeigenschaften drückt sich dort in einer erhöhten Prädiktionsakkuranz und Extrapolationsvermögen aus. Symmetriebeschränkungen werden hauptsächlich im Rahmen des Architekturentwurfs berücksichtigt.

Eine weitere Klasse domänenspezifischer Erweiterungen stellen physikalisch motivierte Nebenbedingungen dar. Physikalische Gesetzmäßigkeiten, Erfahrungswissen sowie -werte, Erhaltungssätze und Beschränkungen können als Nebenbedingungen beim Training berücksichtigt werden. Beschränkungen können dabei grob in Ungleichungsbeschränkungen – zur Berücksichtigung von z.B. Monotonie –, Gleichheits- bzw. approximativer Beschränkungen iFv. Toleranzbändern sowie nichtlineare Nebenbedingungen [DMC+20] untergliedert werden. Die Implementierung erfolgt idR. in Form sogenannter Strafterme in der Fehlerfunktion (siehe Gl. (3-1)). Wird eine physikalische Beschränkung verletzt, so führt dies zu einem Anstieg des Trainingsfehlers und einer entsprechenden Einflussnahme auf die Wahl der Netzgewichte. Grundsätzlich könnten derartige „problembezogene“ Nebenbedingungen durchaus auch als Nebenbedingungen im eigentlichen Optimierungssinne berücksichtigt werden. Diese „harten“ Beschränkungen führen allerdings zu unerwünschten Auswirkungen auf die Konditionierung des Optimierungsproblems und sind, wenn überhaupt, nur auf simple und effizient zu lösende Probleme anwendbar [vgl. DMC+20]. Daher werden Beschränkungen idR. als sogenannte „weiche“ Beschränkungen in Form zusätzlicher Strafterme implementiert. Die jeweilige Gewichtung der einzelnen Strafterme zum eigentlichen Gütemaß, dem Trainingsfehler, erfolgt mittels Skalarisierung als gewichtete Summe und erfordert i.A. exzessives Einstellen der Lagrange-Multiplikatoren λ_i. Vereinzelt werden für diese Einstellung auch automatisierte Ansätze auf Basis der Lagrange-Dualität verwendet [RZW20].

$$J\left(\underline{Y}, \underline{\tau}\right) = (1 - \lambda_{phy})\mathrm{Loss}_{data}\left(\underline{Y}, \underline{\tau}\right) + \underbrace{\lambda_{phy}\mathrm{Loss}_{phy}\left(\underline{Y}\right)}_{Strafterm} \tag{3-1}$$

[22] vgl. System-of-Systems (SOS)

Für die Implementierung von Erhaltungsgesetzen kann neben der Berücksichtigung iFv. Straftermen ebenso eine maßgeschneiderte Architektur verwendet werden. So wurden in diversen Untersuchungen Netze zur direkten Modellierung der Hamilton-Funktion oder Euler-Lagrange-Gleichung eingesetzt [GDY19; TRJ+19; LRP19; CGH+20; RZW20]. Im Fall der Euler-Lagrange-Gleichung wird durch die Vorgabe des Aufbaus die Gütefunktion S (Gl. 3.1) minimiert. Dies entspricht der Nullsetzung des zeitlichen Integrals der Differenz zwischen kinetischer T und potentieller Systemenergie V – kurz, es werden nur energieerhaltende Modelle gelernt:

$$\text{Euler-Lagrange-Gleichung: } 0 = \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i}$$

$$S = \int_{t_0}^{t_1} T\left(q_t, \dot{q}_t\right) - V\left(q_t, \dot{q}_t\right) \mathrm{d}t$$

Liegt bereits ein mehr oder weniger genaues Simulationsmodell des betrachteten Systems vor oder ist ein entsprechendes Modell mit relativ geringem Aufwand zu erstellen, so kann dies ebenfalls im PGNN-Kontext genutzt werden. Eine mögliche Implementierung besteht in der Erweiterung der PGNN-Eingabe. Die Ausgaben des Simulationsmodells werden dann z.B. wie in [KWRK17] parallel zu der eigentlichen Modelleingabe eingespeist (vgl. Abb. 3-1). Die Lernaufgabe des nachgeschalteten PGNNs besteht dann in der Modellierung des Residuums zwischen der Ausgabe des zusätzlichen physikalischen Modells und dem Trainingstarget. Diese Modifikation der zugrundeliegenden Optimierungsaufgabe kann zu signifikanten Vorteilen in Bezug auf die Konditionierung des Problems führen. Mögliche Vorteile bestehen in schnellerem und stabilerem Konvergenzverhalten, höherer Prädiktionsgüte sowie Interpretierbarkeit und gutem Extrapolationsvermögen. Ebenfalls

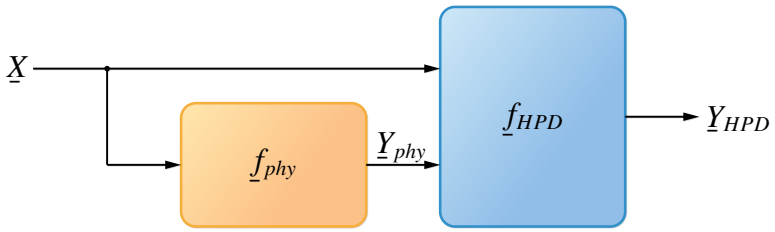

Bild 3-1: *Das Neuronale Netz $\underline{f}_{HPD}$ wird um eine weitere Eingabe erweitert. Die Auswertungen des physikalischen Dynamikmodells $\underline{f}_{phy}$ werden nun dem Neuronalen Netz als zusätzliche Eingaben zur Verfügung gestellt. Das sich ergebende Hybrid-Physics-Data (HPD) Modell bildet in Folge dessen die Netzeingaben $\underline{X}$ auf die Prädiktion $\underline{Y}_{HPD}$ ab, wobei $\underline{Y}_{phy}$ der intermediären Ausgabe des ODE-basierten physikalischen Modells entspricht.*

denkbar ist die Erzeugung synthetischer Trainingsdaten mithilfe des physikalischen Dynamikmodells. Dem Training auf Realdaten wird dabei ein Pre-Training mit synthetischen Daten vorgeschaltet. Diese Strategie birgt den Vorteil, dass mithilfe des Simulationsmodells unendlich viele zusätzliche Trainingsdaten erzeugt werden können. Ist dieses Simulationsmodell hinreichend genau, entspricht das nachgeschaltete Training auf Realdaten lediglich einer Kalibrierung, wie sie bspw. in [JWK+18] durchgeführt wird, um zuerst ein generalistisches Modell der Temperatur in einem See und in einem zweiten Schritt verschiedene, an die einzelnen geographischen Gegebenheiten angepassten Variationen zu lernen.

Tabelle 3-1: *Die folgende Liste enthält einen chronologischen Überblick vorausgegangener Arbeiten zum Thema PGNN.*

Bezeichnung	**Methode**	**Anwendung**
	2017	
Physics-guided Neural Network (PGNN) [KWRK17]	Erweiterung eines NNs um physikalisch motiviertes Simulationsmodell und Monotoniebeschränkungen in Fehlerfunktion	Modellierung von Seetemperaturprofilen
	2018	
Physics-guided Recurrent Neural Network (PGRNN) [JWK+18]	Erweiterung eines LSTMs um physikalisch motivierte Fehlerfunktion und Pre-Training auf Basis synthetischer Daten	Modellierung von Seetemperaturprofilen
Domain Adapted Neural Network (DANN) [MIM+18]	Erweiterung eines DNNs um Domänenwissen iFv. Monotonie- und Toleranzbandbeschränkungen	Akademisch: Bohachevsky-Funktion
HybridNet [LSM18]	Kombination eines LSTMs zur Prädiktion externer Perturbationen mit CeNN[23] zur Lösung und Identifikation von Koeffizienten gegebener PDEs	Prädiktion raumzeitlichen Verhaltens eines Wärme-Konvektions-Diffusions-Systems und eines Fluiddynamiksystems
Physics-Informed Generative Adversarial Network (PI-GAN) [YZK18]	Erweiterung von GANs um auf physikalischen Gesetzen basierende Gradienten-Terme	Approximation stochastischer Prozesse
Physics-Informed Deep Generative Model [YP18]	Erweiterung eines generativen Modells um auf physikalischen Gesetzen iFv. PDEs basierenden VI[24]-Beschränkungen zur Modellierung propagierender Unsicherheiten in komplexen physikalischen Systemen	Akademisch: Burgersgleichung
	2019	
Physics-informed Neural Network (PINN) [RPK19]	DNN mit Beschränkung des Lösungsraums durch physikalische Gesetze durch Erweiterung der Fehlerfunktion	Datengetriebene Identifikation und Lösung von PDEs für diverse Domänen

[23] Cellular Neural Network
[24] variational inference

Tabelle 3-1: *Fortsetzung der Liste vorausgegangener Arbeiten zum Thema PGNN.*

Bezeichnung	**Methode**	**Anwendung**
Deep Lagrangian Network (DeLaN) [LRP19]	DNN zur Parametrisierung der energieerhaltenden Euler-Lagrange-Gleichung; für Starrkörperdynamiken	Strukturdynamik-simulationen
Hamiltonian Neural Network (HNN) [GDY19]	NN zur Parametrisierung der energieerhaltenden Hamilton-Funktion	Ideales Feder-Masse-System; ideales Pendel; reales Pendel
Hamiltonian Generative Network (HGN) [TRJ+19]	GAN zur Modellierung der Hamilton-Funktion	Lernen der unterliegenden Systemdynamik von Bilddaten
Symplectic Recurrent Neural Network (SRNN) [CZAB19]	Modellierung die Hamilton-Funktion mithilfe eines RNN; nutzt als symplektischen Integrator die Leapfrog Methode	Akademisch: komplexes Feder-Masse-System
2020		
Bayesian Physics-Informed Neural Network (B-PINN) [YMK20]	Kombination aus BNN als probabilistisches Ersatzmodell und PINN zur Modellierung der Likelihood der Messungen	Lösung von PDEs auf Basis verrauschter Daten
Deep Residual Recurrent Neural Network (DR-RNN) [YYL20]	RNN modelliert Residuum zwischen physikalischem Simulationsmodell und Messdaten	Akademisch: verschiedene Mechanische Systeme
Lagrangian Neural Network (LNN) [CGH+20]	NN zur Parametrisierung der energieerhaltenden Euler-Lagrange-Gleichung; ähnlich DeLaN, aber generalistisch	Akademisch: u.a. Doppelpendel
Lagrangian Dual-based Theory-guided Deep Neural Network (TgNN-LD) [RZW20]	Multiple physikalisch motivierte Terme in Fehlerfunktion; automatisiert Bestimmung der zugehörigen Koeffizienten durch Lagrange-Dualität	Grundwasser-strömungsproblem
Encoder-Decoder Neural Network [DMC+20]	Nichtlineare physikalisch motivierte Terme in Fehlerfunktion; Zerlegung des Optimierungsproblems in unbeschränkte Teilprobleme und Lösung durch SGD[25]	Numerisches Kollisionsproblem

[25] Stochastic Gradient Descent

Tabelle 3-1: *Fortsetzung der Liste vorausgegangener Arbeiten zum Thema PGNN.*

Bezeichnung	**Methode**	**Anwendung**
Symmetry Constraints (dt. Symmetriebeschränkungen) [WWY20]	Modifikation der DNN-Architektur zur Einhaltung von Equivarianzen	Rayleigh-Bénard Konvektion; Meeresströmung
2021		
Physics-Informed Neural Nets-based Control (PINC) [ACS+21]	DNN als diskreter numerischer Solver gegebener ODEs als Strafterm	Regelungstechnik; MPC

3.2 Analogien zur Regelungstechnik

Trotz der großen Vielfalt der bereits untersuchten PGNN-Ansätze, eignen sich nur wenige davon für einen Einsatz zur Identifikation dynamischer Systeme im regelungstechnischen Kontext. Geeignete Ansätze sind hochperformant, besonders im Hinblick auf nichtautonome Systeme mit mehreren Stelleingängen, der Modellierung nicht-stationären Verhaltens sowie verfügen über hohes Extrapolationsvermögen bei Training mit geringen Datenmengen.

Einordnung in den Regelungsentwurfsprozess

Der Prozess zum Entwurf einer Regelung gliedert sich in aufeinanderfolgende Teilaufgaben. Je nach Quelle fallen die genaue Definition sowie die Anordnung leicht unterschiedlich aus. Das liegt daran, dass es sich beim Regelungsentwurf um einen adaptiven Prozess handelt, der der jeweiligen Problemstellung angepasst wird. Im Rahmen dieser Arbeit soll angenommen werden, dass zum Entwurf eines Reglers zunächst ein Modell der Strecke gebildet wird. Diese Modellbildung erfolgt entweder rein theoriegebunden oder datengetrieben bzw. experimentell. Im Anschluss an diesen Schritt, der auch als *Modellbildung und Systemidentifikation* bezeichnet wird, schließen sich eine Reihe weiterer Schritte an, die in der finalen Realisierung der Regelung münden:

1. Modellbildung und Systemidentifikation
2. Beobachterentwurf
3. Regelungsentwurf
4. Simulation des Gesamtsystems
5. Realisierung

Mit PGNNs wird in dieser Arbeit ein neuartiges datengetriebenes Verfahren zur Systemidentifikation verwendet, das die Berücksichtigung von Domänenwissen ermöglicht. Gebräuchliche Verfahren basieren üblicherweise auf der Modellierung des Frequenzgangs des betrachteten Systems. Allen Verfahren ist idR. eine entsprechende Datenvorverarbeitung vorgeschaltet, mit der eine ausreichende Datengüte und somit auch die Güte des

resultierenden Modells maßgeblich beeinflusst werden kann. Bemerkungen zur Datenvorverarbeitung im Rahmen von PGNNs wurden bereits in Kapitel 2.1.7 betrachtet.

Lineare Systemdynamiken

Einschichtige NNe mit linearen Aktivierungsfunktionen stellen wohl die simpelste Form NNe dar. Dennoch können die dabei trainierten Gewichte aus regelungstechnischer Sicht interpretiert werden. Werden als Netzeingaben der Zustand $\underline{x}$ sowie die Stellgröße $\underline{u}$ des zu modellierenden technischen Systems angenommen und als Ausgabe die zeitliche Ableitung des Zustandsvektors $\underline{\dot{x}}$, so entspricht die Netzarchitektur der Formulierung einer linearen Systemdynamik:

$$\begin{aligned} \text{Lineare Systemdynamik: } & \underline{\dot{x}}(t) = \underline{A}\,\underline{x}(t) + \underline{B}\,\underline{u}(t) \\ \text{Netzwerk: } & \underline{\dot{x}}_k = \underline{W}_{\underline{x}}^T\,\underline{x}_k + \underline{W}_{\underline{u}}^T\,\underline{u}_k + \underline{b} \end{aligned} \tag{3-2}$$

Dabei werden die Matrizen A und B durch die Netzgewichte $\underline{W}_{\underline{x}}^T$ und $\underline{W}_{\underline{u}}^T$ approximiert. Die dabei gelernten Matrizen können infolge hinsichtlich diverser regelungstechnischer Kriterien analysiert werden. So kann beispielsweise die Stabilität des Systems auf Basis des Realteils der Eigenwerte sowie die Beobachtbar- und Steuerbarkeit der Zustandsgrößen bestimmt werden.

Auch wenn im Falle einer derartig simplen Architektur auf eine Normierung der Trainingsdaten verzichtet werden könnte, wird idR. eine Normierung vorgenommen (siehe Kap 2.1.7). Die Normierung einer Größe $\underline{x}$ mit n Komponenten x_i mittels Mittelwert $\mu(x_i)$ und Standardabweichung $\sigma^2(x_i)$ ergibt

$$\underline{x}_{norm} := \begin{bmatrix} \sigma^2(x_1) & & \\ & \ddots & \\ & & \sigma^2(x_n) \end{bmatrix}^{-1} \left(\underline{x} - \underline{\mu}(\underline{x})\right) := \underline{S}_{\underline{x}}^{-1}\left(\underline{x} - \underline{\mu}_{\underline{x}}\right).$$

Durch Einsetzen der Normierung in Gleichung (3-2) und Auflösen nach den Netzgewichten und Biaswerten folgt:

$$\begin{aligned} & \underline{\dot{x}}_{norm} = \underline{W}_{\underline{x}}^T\,\underline{x}_{norm} + \underline{W}_{\underline{u}}^T\,\underline{u}_{norm} + \underline{b} \\ \Leftrightarrow & \underline{S}_{\underline{\dot{x}}}^{-1}\left(\underline{\dot{x}} - \underline{\mu}_{\underline{\dot{x}}}\right) = \underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1}\left(\underline{x} - \underline{\mu}_{\underline{x}}\right) + \underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}\left(\underline{u} - \underline{\mu}_{\underline{u}}\right) + \underline{b} \\ \Leftrightarrow & \underline{\dot{x}} = \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1}\left(\underline{x} - \underline{\mu}_{\underline{x}}\right) + \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}\left(\underline{u} - \underline{\mu}_{\underline{u}}\right) + \underline{\mu}_{\underline{\dot{x}}} + \underline{S}_{\underline{\dot{x}}}\underline{b} \\ \Leftrightarrow & \underline{\dot{x}} = \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1}\underline{x} + \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}\underline{u} + \underline{\mu}_{\underline{\dot{x}}} + \underline{S}_{\underline{\dot{x}}}\left(\underline{b} - \underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1}\underline{\mu}_{\underline{x}} - \underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}\underline{\mu}_{\underline{u}}\right) \\ \Rightarrow & \underline{b} \stackrel{!}{=} \underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1}\underline{\mu}_{\underline{x}} + \underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}\underline{\mu}_{\underline{u}} - \underline{S}_{\underline{\dot{x}}}^{-1}\underline{\mu}_{\underline{\dot{x}}} \end{aligned}$$

Dies zeigt, ist der Mittelwert von $\underline{\dot{x}}$, $\underline{x}$ und $\underline{u}$ nicht von Null verschieden, so fällt der Bias weg. Heben sich dagegen die Terme nicht auf, so ist ein Bias ungleich null notwendig. Die Dynamikmatrizen A und B können dann bestimmt werden zu

$$\begin{aligned} \underline{A} &= \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{x}}^T\underline{S}_{\underline{x}}^{-1} \text{ und} \\ \underline{B} &= \underline{S}_{\underline{\dot{x}}}\underline{W}_{\underline{u}}^T\underline{S}_{\underline{u}}^{-1}. \end{aligned} \tag{3-3}$$

Zeitkontinuität

Bisher wurde ein zeitkontinuierlicher Ansatz angenommen. Das resultierende NN kann demnach unter Zuhilfenahme eines numerischen Solvers wie bspw. dem klassischen Runge-Kutta-Verfahren zeitlich integriert werden. Dies birgt Vorteile bzgl. variabler Schrittweite und der daraus resultierenden Genauigkeit. Zudem kann auch in der Zeit rückwärts prädiziert werden. In der Literatur werden hingegen oft zeitdiskrete Ansätze mit fester Schrittweite verwendet. Für den Bereich der Regelungstechnik ist jedoch zunächst ein zeitkontinuierlicher Ansatz aufgrund der aufgeführten Vorteile hinsichtlich Interpretierbarkeit und Genauigkeit zu präferieren. Im Kontrast zur zeitdiskreten Variante müssen dafür jedoch zunächst entsprechende Messdaten erstellt werden, da die zeitliche Ableitung des Zustandsvektors als Target dient. Zur Generierung der Trainingsdaten aus Zeitreihendaten des Systemzustands kann die zugehörige zeitliche Ableitung bspw. durch Aufstellen des Differenzenquotienten gebildet werden. Dafür ist eine hinreichend hohe Abtastrate notwendig, sodass nicht grundsätzlich für alle verfügbaren Datenkonfigurationen ein zeitkontinuierlicher Ansatz infrage kommt.

Im Laufe dieser Arbeit werden u.a. rekurrente Ansätze betrachtet, die für die Berechnung der Prädiktion von $\dot{\underline{x}}_k$ zum Zeitpunkt t_k neben der aktuellen Netzeingabe zudem alle vorangegangenen Zustände und Netzeingaben erfordern (vgl. Kapitel 2.1.4). Um auch RNNs simulieren zu können, wurde im Rahmen dieser Arbeit ein auf dem klassischen Runge-Kutta-Verfahren basierender numerischer Solver mit variabler Schrittweite entwickelt (Anhang A4).

Nichtlineare Systemdynamiken

Nichtlineare Aktivierungsfunktionen und rekurrente Architekturen dienen zur Approximation nichtlinearer Dynamiken. Eine konkrete Nachvollziehbarkeit des Netzverhaltens sowie der trainierten Netzgewichte ist jedoch zunächst nicht gegeben. Vor allem die Analyse hinsichtlich regelungstechnischer Kriterien wie z.B. Stabilität, auf Basis der Netzgewichte, ist im Kontrast zum linearen Fall nicht unmittelbar möglich. Da in dieser Arbeit jedoch insbesondere nichtlineare Systeme modelliert und untersucht werden sollen, sind entsprechende Kompromisse hinsichtlich der Interpretierbarkeit des NN unumgänglich.

Soll die Dynamik eines technischen Systems identifiziert werden, das sich analog eines Blockschaltbilds in mehrere Module mit separierbarer Teildynamik zerlegen lässt, so ist dies hinsichtlich des Trainings und der anschließenden Interpretierbarkeit der Ergebnisse zu präferieren. Ob die Netzarchitektur entsprechend der einzelnen interpretierbaren Module ausgelegt wird, oder ob das direkte holistische Lernen der Systemdynamik angestrebt wird, hängt vom Einzelfall und den dabei verfügbaren Trainingsdaten ab und muss individuell geprüft werden.

3.3 Viertelfahrzeugmodell

Um die fortlaufenden Kapitel möglichst anschaulich gestalten zu können, soll zunächst ein beispielhaftes, repräsentatives technisches System angenommen werden. Das Viertel-

fahrzeugmodell erfüllt dazu alle Anforderungen hinsichtlich Relevanz und Steuerbarkeit. Darüber hinaus ist durch die Einführung eines nichtlinearen Dämpfers ein guter Kompromiss bzgl. Systemkomplexität und -verständnis gegeben.

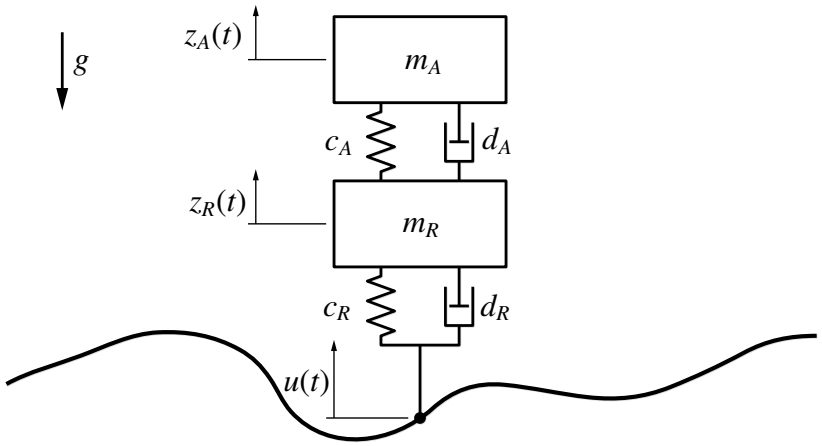

Bild 3-2: *Das Viertelfahrzeugmodell umfasst Rad- und Aufbaumassen m_R und m_A, die mittels eines Feder-Dämpfer-Gliedes miteinander verbunden sind. Es erfolgt eine Fußpunktanregung $u(t)$ infolge der Unebenheiten des Untergrunds.*

Das in Abbildung 3-2 dargestellte Viertelfahrzeugmodell (VFZG) besteht aus Rad und Aufbau, die mittels Federn und Dämpfern verbunden sind. Angeregt wird das System über eine Fußpunktanregung $u(t)$. Das lineare Viertelfahrzeugmodell kann dabei mithilfe von vier Zustandsgrößen $\underline{x}^T(t) = [\dot{z}_A(t), \dot{z}_R(t), z_A(t), z_R(t)]$ beschrieben werden:

$$\dot{\underline{x}}(t) = \underline{A}\underline{x}(t) + \underline{B}\underline{u}(t)$$

$$\Leftrightarrow \begin{bmatrix} \ddot{z}_A(t) \\ \ddot{z}_R(t) \\ \dot{z}_A(t) \\ \dot{z}_R(t) \end{bmatrix} = \begin{bmatrix} -\frac{d_A}{m_A} & \frac{d_A}{m_A} & -\frac{c_A}{m_A} & \frac{c_A}{m_A} \\ \frac{d_A}{m_R} & -\frac{d_A+d_R}{m_R} & \frac{c_A}{m_R} & -\frac{c_A+c_R}{m_R} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{z}_A(t) \\ \dot{z}_R(t) \\ z_A(t) \\ z_R(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{d_R}{m_R} & \frac{c_R}{m_R} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{u}(t) \\ u(t) \end{bmatrix} \qquad \text{(lin. VFzg)}$$

Durch Austausch des linearen Aufbaudämpfers d_A gegen einen nichtlinearen Dämpfer mit Dämpferkraft $F = d_A x^\alpha$ ergibt sich eine entsprechend nichtlineare Systemdynamik:

$$\dot{\underline{x}}(t) = \underline{f}\left(\underline{x}(t), \underline{u}(t)\right)$$

$$\Leftrightarrow \begin{bmatrix} \ddot{z}_A(t) \\ \ddot{z}_R(t) \\ \dot{z}_A(t) \\ \dot{z}_R(t) \end{bmatrix} = \begin{bmatrix} \frac{d_A \operatorname{sign}(\dot{z}_R(t)-\dot{z}_A(t))|\dot{z}_R(t)-\dot{z}_A(t)|^\alpha + c_A(z_R(t)-z_A(t))}{m_A} \\ \frac{d_A \operatorname{sign}(\dot{z}_A(t)-\dot{z}_R(t))|\dot{z}_A(t)-\dot{z}_R(t)|^\alpha + d_R(\dot{u}(t)-\dot{z}_R(t)) + c_A(z_A(t)-z_R(t)) + c_R(u(t)-z_R(t))}{m_R} \\ \dot{z}_A(t) \\ \dot{z}_R(t) \end{bmatrix}$$

(n-lin. VFzg)

Der Faktor $\alpha \in \mathbb{N}_+$ wird zunächst variabel angenommen, um in folgenden Kapiteln den Einfluss verschiedener Nichtlinearitäten untersuchen zu können. Darüber hinaus wird im Rahmen dieser Arbeit angenommen, dass als Stellgröße eine Sinusanregung

$$\underline{u}(t) = \begin{bmatrix} \dot{u}(t) \\ u(t) \end{bmatrix} = A \begin{bmatrix} f\cos(ft) \\ \sin(ft) \end{bmatrix}$$

mit der Amplitude $A = 0.4$ und der Frequenz $f = 20\text{Hz}$ aufgeschaltet wird. Die in dieser Arbeit verwendete „echte“ Parametrierung des Realsystems (Ground Truth) kann Tabelle A1-1 entnommen werden.

3.4 Einführung eines Gütemaßes

Die Güte der in dieser Arbeit trainierten PGNNs bestimmt sich hauptsächlich aus der Übereinstimmung der resultierenden Prädiktionen mit entsprechenden Referenztrajektorien. Um im weiteren Verlauf unterschiedliche Netze hinsichtlich ihrer Simulationsperformanz vergleichen zu können, soll zunächst ein entsprechendes Gütemaß eingeführt werden. Da es in dieser Arbeit vor allem um die Genauigkeit der Prädiktionen und weniger um z.B. die benötigte Trainingszeit geht, wird der simulative Prädiktionsfehler oder kurz *Simulationsfehler* E_{sim} als Maß für die Güte eines PGNNs herangezogen. Der Simulationsfehler für n_l Datensequenzen der Form

$$\underline{X}^{(l)} = \left[\underline{x}_0^{(l)}, \ldots, \underline{x}_n^{(l)}\right], \quad \underline{Y}^{(l)} = \left[\underline{y}_0^{(l)}, \ldots, \underline{y}_n^{(l)}\right],$$

wobei $t_0, \ldots, t_n$ gleichmäßig verteilt, $\underline{x}_k^{(l)} := \underline{x}(t_k)$, $\underline{y}_k^{(l)} := \underline{y}\left(\underline{x}_k^{(l)}\right)$ und $\underline{\tau}_k^{(l)} := \underline{\tau}\left(\underline{x}_k^{(l)}\right)$ sowie $d := \dim\left(\underline{y}_k^{(l)}\right)$, ergibt sich als normiertes kumuliertes Integral[26] der Abweichung zwischen Prädiktion $\underline{y}_k^{(l)}$ und Target $\underline{\tau}_k^{(l)}$ zu

$$\begin{aligned} E_{sim} &= \frac{1}{n_l}\frac{1}{d}\sum_l \int_{t_0}^{t_n} \frac{\left\|\underline{y}^{(l)}(t) - \underline{\tau}^{(l)}(t)\right\|_1}{1+\lambda t}\mathrm{d}t \\ &\approx \frac{1}{n_l}\frac{1}{d}\frac{t_n - t_0}{2n}\sum_l \sum_{k=0}^{n-1}\left(\frac{\left\|\underline{y}_k^{(l)} - \underline{\tau}_k^{(l)}\right\|_1}{1+\lambda t_k} + \frac{\left\|\underline{y}_{k+1}^{(l)} - \underline{\tau}_{k+1}^{(l)}\right\|_1}{1+\lambda t_{k+1}}\right). \end{aligned} \qquad \text{(Simulationsfehler)}$$

Dabei kann durch Wahl des *Discount Faktors* $\lambda \in \mathbb{R}_0^+$ die Gewichtung früher auftretender Abweichungen stärker gewichtet werden. Die Verwendung eines Discount Faktors ≥ 0 ist gerade im Kontext von MPC in Betracht zu ziehen, da der zeitliche Kontext der auftretenden Abweichungen von besonderer Relevanz ist und so spätere Abweichungen weniger ins Gewicht fallen. Für alle in dieser Arbeit angestellten Untersuchungen wurde dieser Faktor allerdings zu Null gesetzt.

Geometrisch entspricht der Simulationsfehler E_{sim} der (normierten) kumulierten Fläche zwischen den Prädiktions- und Targettrajektorien wie in Abbildung 3-3 dargestellt. Mithilfe dieses Gütemaßes ist nun ein Vergleich der Simulationsperformanz ungeachtet der dabei verwendeten Trainingsdatenmengen sowie Sequenzlänge und Dimension des betrachteten Systems möglich.

[26] Approximation mittels des impliziten Trapezverfahrens: $\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{2N}\sum_{n=1}^{N}\left(f(x_n) + f(x_{n+1})\right)$

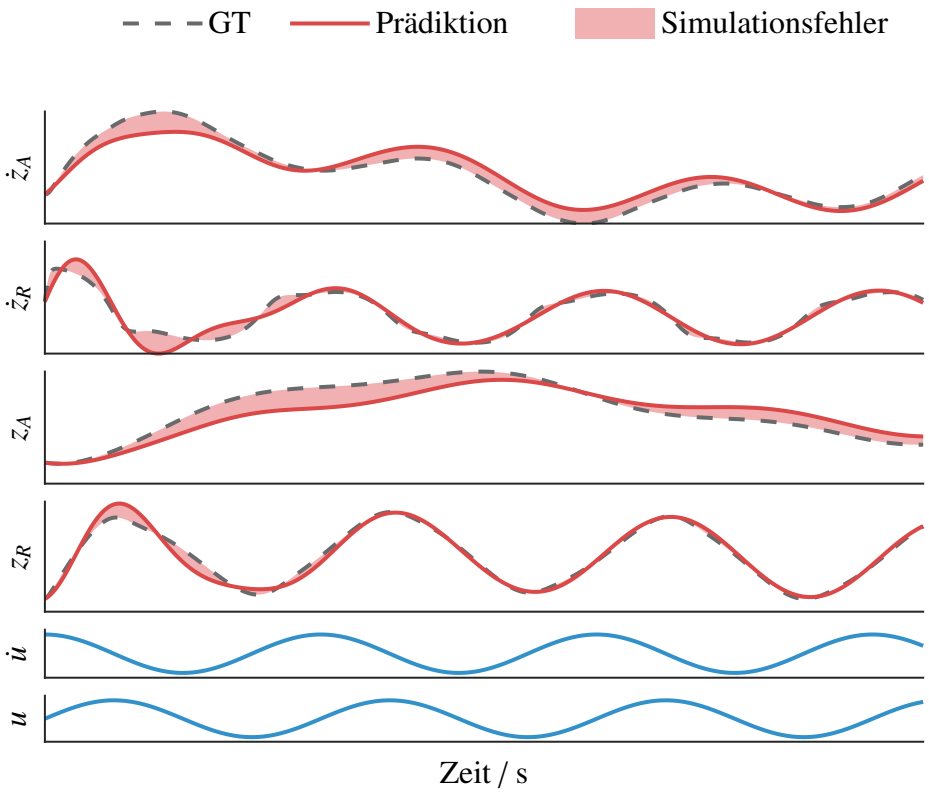

Bild 3-3: *Geometrisch interpretiert entspricht der simulative Prädiktionsfehler, kurz Simulationsfehler, der normierten Fläche zwischen Prädiktion und Ground Truth. Die Prädiktion der in dieser Abbildung dargestellten Trajektorien erfolgte durch ein lineares Neuronales Netz.*

3.5 Wahl der Architektur

Die Wahl einer geeigneten Netzarchitektur hat einen immensen Einfluss auf die resultierende Performanz in Training und Prädiktion. In diesem Kapitel soll anhand des Viertelfahrzeugmodells, stellvertretend für eine Vielzahl technischer Systeme, ein entsprechender Entwurfsprozess erläutert werden.

Beobachtbarkeit

Zunächst soll angenommen werden, dass eine vollständige Erfassung des Zustandsvektors möglich ist. Dies ist insbesondere dann zutreffend, wenn das zu modellierende System vollständig beobachtbar ist oder ein geeigneter Beobachter vorliegt. Ist dies nicht der Fall, können probabilistische Ansätze zur Prädiktion der benötigten Zustandsgrößen angestellt werden. Entsprechende Ansätze werden etwa im Bereich von Partially-Observable Markov Decision Processes (POMDP) sowie Variational Inference verwendet. Inwiefern ähnliche Ansätze zur Modellierung zeitkontinuierlicher Systemdynamiken verwendet werden können, ist jedoch fraglich. Ebenso ist fraglich, inwiefern ein PGNN simultan trainiert und als Beobachter verwendet werden kann. Dies entspräche in etwa ei-

nem Encoder-Decoder-Schema. Da der Fokus dieser Arbeit jedoch auf der Identifizierung der Systemdynamik liegt, soll fortan von einem vollständig beobachtbaren System ausgegangen werden. Es können daher nicht abgebildete, innere Dynamiken ausgeschlossen werden. Zudem wird, wie in Kapitel 2.1.7 postuliert, von einer ausreichend hohen Datengüte sowie geringem Messrauschen ausgegangen.

Zuzüglich zur Modellierung der Dynamik zwischen Ein-/Ausgangsgrößen und dem Systemzustand, kann auch der Einfluss von Störgrößen modelliert werden. Da diese allerdings, sofern diese von den Trainingsdaten erfasst werden, analog zu Eingangsgrößen behandelt werden, wurde im Rahmen dieser Arbeit auf die Betrachtung zusätzlicher Störeinflüsse verzichtet.

Architekturwahl

Komplexe technische Systeme können oft in eine Reihe weniger komplexe Subsysteme zerlegt werden. Wie bereits in Kapitel 3.1 und 3.2 erwähnt, können entsprechend umfangreiche Dynamiken hierarchisch untergliedert werden. Die sich dabei für die Untersysteme ergebenden Teildynamiken können robuster erlernt und im Nachhinein zu einem holistischen Modell zusammengefügt werden. Die im Rahmen dieser Arbeit betrachteten Systeme stellen jeweils entsprechend kompakte Modellierungsaufgaben dar und sind folglich holistisch zu lernen.

Dabei können jedoch unterschiedliche Netzwerkarchitekturen verwendet werden. Um die Performanz unterschiedlicher Architekturen am Beispiel des nichtlinearen Viertelfahrzeugmodells zu demonstrieren, sollen an dieser Stelle ein einfaches vorwärtsgerichetes NN mit linearen Aktivierungsfunktionen (lineares NN) und ein Rekurrentes Neuronales Netz (RNN) (Abb. 3-4) hinsichtlich ihrer Abbildungsfähigkeit untersucht werden. Die Eingabe der jeweiligen Netze setzt sich dabei aus dem Zustandsvektor $\underline{x}_k$ und den Stellgrößen $\underline{u}_k$ zusammen, wobei sich die Ausgabe durch Training der Gewichte $\underline{w}$ zu $\dot{\underline{x}}_k$ ergibt:

$$\begin{aligned} \dot{\underline{x}}_k &= \underline{f}_{NN}\left(\underline{x}_k, \underline{u}_k | \underline{w}\right) \\ \dot{\underline{x}}_k &= \underline{f}_{RNN}\left(\underline{x}_0, \underline{u}_0, \ldots, \underline{x}_k, \underline{u}_k | \underline{w}\right) \end{aligned} \tag{3-4}$$

Im Fall des RNNs ist dem GRU (Kap. 2.1.4) ein lineares einschichtiges NN nachgeschaltet. Das GRU gibt dabei für die Eingabesequenzen $\underline{x}_0, \ldots, \underline{x}_k$ und $\underline{u}_0, \ldots, \underline{u}_k$ den zu Zeitpunkt t_k gehörenden Hidden-State $\underline{h}_k$ zurück, welcher in der verwendeten Implementierung in MATLAB die gleiche Dimension wie $\left[\underline{x}_k, \underline{u}_k\right]^T$ hat. Die Netzausgabe $\dot{\underline{x}}_k$ wird durch die nachgeschaltete Schicht aus $\underline{h}_k$ bestimmt:

$$\begin{aligned} \underline{h}_k &= \underline{f}_{GRU}\left(\underline{x}_0, \underline{u}_0, \ldots, \underline{x}_k, \underline{u}_k | \underline{w}\right) \\ \dot{\underline{x}}_k &= \underline{g}\left(\underline{h}_k | \underline{w}\right) \end{aligned}$$

Lineares NN und RNN werden mit 10 Trainingsdatensequenzen der Form

$$\underline{X}_{train} = \left[\left[\underline{x}_1, \underline{u}_1\right]^T, \ldots, \left[\underline{x}_n, \underline{u}_n\right]^T\right], \quad \underline{\tau}_{train} = \left[\underline{\tau}_1, \ldots, \underline{\tau}_n\right]$$

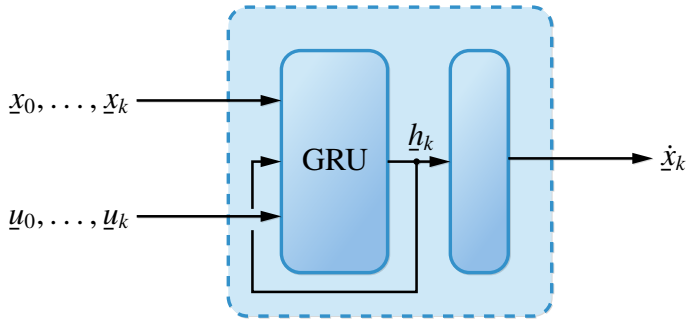

Bild 3-4: *Das RNN besteht aus einer GRU und einem einschichtigen linearen Neuronalen Netz. Da die Implementierung des GRU in MATLAB lediglich Hidden States mit Dimension des Eingangs zulässt, ist die nachfolgende lineare Schicht von Neuronen notwendig. Dabei gibt das RNN zum Zeitpunkt t_k auf Basis aller vergangener Netzeingaben von t_0 bis t_k $\dot{\underline{x}}_k$ als Lösung der Dynamikgleichung* (3-4) *zurück.*

trainiert. Jede Sequenz umfasst dabei einen Einschwingvorgang von einer Sekunde bei einer Abtastrate von 1000Hz. Das entsprechende Target $\underline{\tau}_k \equiv \dot{\underline{x}}_k$ wird, wie in Kapitel 3.2 beschrieben, mittels Differenzenquotienten gebildet. Beide Netze werden anschließend mittels dem klassischen bzw. erweiterten[27] Runge-Kutta-Ansatz für eine Sekunde simuliert. Die daraus resultierenden Ergebnisse können Tabelle 3-2 entnommen werden. Als Fehlerfunktion für das Viertelfahrzeugmodell hat sich LogMAE als am besten geeignet herausgestellt (siehe für Interpretationen Kap. 2.1.3).

Tabelle 3-2: *Das lineare Neuronale Netz verfügt über keine Aktivierungsfunktionen und kann infolgedessen keine nichtlinearen Dynamiken abbilden. Dies ist deutlich in der Diskrepanz des Simulationsfehlers zu erkennen. Dieser fällt im Fall des RNNs mehr als 2.5-mal niedriger aus.*

Architektur	**Lineares NN**	**RNN**
Anzahl Neuronen[28]	198	33
Trainingsdauer	2:54 min	4:48 min
Simulationsfehler	$5.6 \cdot 10^{-2}$	$2.2 \cdot 10^{-2}$

Es zeigt sich, dass der Simulationsfehler des RNN mehr als zweieinhalb mal niedriger ausfällt, als der des linearen NN. Damit ist deutlich zu erkennen, dass RNNs sich, aufgrund ihrer Fähigkeit zeitliche Zusammenhänge abzubilden, deutlich besser zur Modellierung der Dynamik technischer Systeme eignen. Anzumerken ist hierbei, dass das lineare NN trotz einer hohen Anzahl versteckter Neuronen keine nichtlinearen Dynamiken abbilden kann (siehe Abb. 3-3). Es ist deshalb, trotz seiner generell deutlich kürzeren Trainingsdauer nicht für die Identifikation nichtlinearer Systemdynamiken geeignet. Daher wird im Folgenden die lineare Architektur nicht weiter betrachtet und das RNN als Referenzarchitektur verwendet. Analog wird im Folgenden statt von Physics-guided Neural

[27] siehe Anhang A4
[28] Optimiert durch Bayessche Optimierung

Networks (PGNN) von Physics-guided Recurrent Neural Networks (PGRNN) gesprochen.

Trotz der überlegenen Genauigkeit des RNNs ist der damit verbundene Trainingsprozess sehr anspruchsvoll. Dabei ist besonders auf die numerische Konditionierung der Fehlerfunktion (Kapitel 2.1.3 und 2.1.7) zu achten. Insbesondere die Sensitivität bzgl. der gewählten Hyperparameter soll an dieser Stelle herausgestellt werden. Bereits kleine Abweichungen in gewissen Hyperparametern können zu ungenügenden Trainingsresultaten führen. Im Zweifelsfall sollte daher immer zuerst die numerische Konditionierung geprüft werden, bevor Rückschlüsse auf andere Faktoren erwogen werden.

Auch wenn für die in dieser Arbeit untersuchten technischen Systeme die rekurrente Architektur besonders vielversprechend ist, lässt dies keine generellen Rückschlüsse zu. Die Architekturwahl ist i.A. sehr vielfältig und basiert stark auf vorhandenem Vorwissen über die Struktur des zu modellierenden Systems. Allgemeine Aussagen können deshalb kaum getroffen werden. Auf Basis des Vorwissens können geeignete Kandidatenarchitekturen identifiziert werden.

3.6 Erweiterung um ein physikalisches Modell

Ein gängiges Szenario im Rahmen des Reglerentwurfs ist, dass ein mehr oder weniger detailliertes Systemmodell in Form eines Systems gewöhnlicher Differentialgleichungen[29] (ODE) vorliegt. In diesem Fall wäre der Anspruch bei Verwendung eines PGRNNs die Steigerung der Genauigkeit oder Robustheit dieses Modells. Im Rahmen des Trainings eines PGRNNs können entsprechende ODEs genutzt werden. Dazu sollen im Folgenden drei Ansätze beschrieben werden:

- Initialisierung der Gewichte
- Erzeugung synthetischer Daten
- Hybrider Ansatz

Wird ein lineares NN verwendet, kann Systemwissen, z.B. in Form der Dynamikmatrizen A und B zur Initialisierung der Netzgewichte, genutzt werden. Da es sich beim Training eines NNes i.A. um ein lokales Optimierungsverfahren handelt, ist ohne geeignete Initialisierung keine Konvergenz zum globalen Optimum zu gewährleisten. Für komplexe Systeme kann eine entsprechend hinreichende Initialisierung der Gewichte, auch *Initial Guess* oder *Educated Guess* genannt, essentiell sein. Verfügbares Vorwissen sollte demnach auf jeden Fall genutzt werden. Für komplexere Architekturen ist ein analoger Educated Guess aufgrund der fehlenden Interpretierbarkeit der Architektur nicht ohne Weiteres möglich.

Eine weitere Möglichkeit, Vorwissen iFv. ODEs für PGRNNs zu nutzen, besteht in der Erzeugung synthetischer Daten. Wie bereits in Kapitel 3.1 erwähnt, wurde bspw. in [JWK+18] ein vorhandenes Modell zur Erzeugung von Trainingsdaten verwendet und das PGRNN einem sogenannten Pre-Training unterzogen. Dem Training mit künstlichen Daten wurde das eigentliche Training mit Realdaten nachgeschaltet. Als Effekt konnte dabei eine Art

[29] engl. ordinary differential equations

Kalibrierung anhand der Realdaten festgehalten werden, wobei deutlich weniger Realdaten für das gesamte Training des PGRNNs nötig waren, um ein ausreichend niedriges RMSE[30]-Level zu erreichen. Für den regelungstechnischen Einsatz könnte dies bedeuten, dass zunächst ein PGRNN mit synthetischen Daten für den gesamten Zustandsraum trainiert wird und man sich im Hinblick auf die im anschließenden zweiten Training verwendeten Realdaten auf die Ausprägung der für die Regleraufgabe „wichtigen“ Teile des Zustandsraums beschränkt.

Der dritte und letzte Ansatz, der hier betrachtet werden soll, ist ein Hybridmodell. Dabei wird das vorhandene Modell iFv. ODEs, im Folgenden physikalisches Modell genannt, in das PGRNN eingebettet. Diese Modifikation erfolgt analog zum HPD-Modell aus [KWRK17]. Zunächst wird der Eingabevektor des RNNs um die abgeleiteten Zustandsgrößen $\dot{\tilde{\underline{x}}}_k$ des ODE-Systems $\underline{f}_{phy}$ ergänzt. Durch Erweiterung von Gleichung (3-4) ergibt sich:

$$\dot{\underline{x}}_k = \underline{f}_{PGRNN}\left(\underline{x}_0, \underline{u}_0, \tilde{\underline{x}}_0, \ldots, \underline{x}_k, \underline{u}_k, \tilde{\underline{x}}_k | \underline{w}\right) \tag{3-5}$$

$$\tilde{\underline{x}}_k = \underline{f}_{phy}\left(\underline{x}_k, \underline{u}_k\right) \tag{3-6}$$

Die Lernaufgabe des datengetriebenen RNN-Teils kann damit z.B. der Modellierung des Residuums zwischen $\dot{\underline{x}}_k$ und $\tilde{\underline{x}}_k$ entsprechen. Ist das physikalische Modell hinreichend genau, kann dies zu einer stark benefiziellen Konditionierung des Optimierungsproblems und somit zu einer deutlich schnelleren Konvergenz führen. Wie akkurat das physikalische Modell dafür sein muss und welches Potential das PGRNN ggü. dem eigenständigen RNNs birgt, wird in Kapitel 4.3 untersucht.

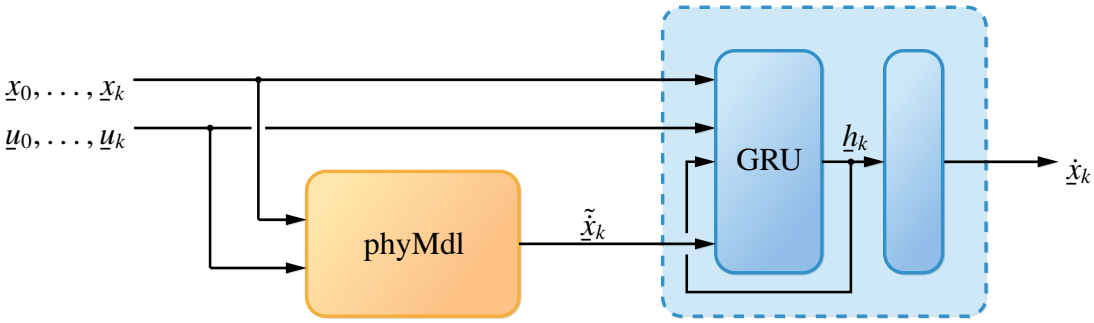

Bild 3-5: *Das RNN wird um einen weiteren Eingang erweitert, in dem zu jedem Zeitpunkt t_k der Vektor $\tilde{\underline{x}}_k$ als Lösung des physikalischen Modells* (3-6) *eingespeist wird.*

Neben der direkten Einbettung des physikalischen Modells in das PGRNN können die vorliegenden Trainingsdaten zudem zuerst zur Identifizierung der Parameter des physikalischen Modells genutzt werden. Dazu können neben gradientenlosen und -basierten Optimierungsverfahren auch die Bayessche Optimierung (Kap. 2.1.6) verwendet werden.

[30] engl. root mean squared error; wird häufig zum Vergleich der Genauigkeit von NNen verwendet

3.7 Erweiterung um eine physikalische Nebenbedingung

Wie bereits in Kapitel 3.1 beschrieben, können physikalische Nebenbedingungen als Strafterme in der Fehlerfunktion aufgenommen werden. Im Fall des Viertelfahrzeug kommt hier etwa die Energieerhaltung infrage. Ein entsprechender Term soll nun dazu dienen, solche Lösungen der Fehlerfunktion zu bevorzugen, die energieerhaltend sind. Mit anderen Worten wird die Wahl solcher Netzgewichtungen bevorzugt, deren Prädiktionsverhalten energieerhaltende Lösungen hervorbringt. Für lineare Architekturen wären ebenfalls physikalische Nebenbedingungen wie etwa die Negativität der Realteile der Dynamikeigenwerte in Betracht zu ziehen, die zu einer besseren Konditionierung des Optimierungsproblems beitragen könnten. Für komplexere Architekturen sind entsprechende Eigenwerte jedoch nicht ohne Weiteres identifizierbar. Auch die Beschränkung bestimmter Zustandsgrößen ist im Rahmen des Viertelfahrzeugs nicht zielführend.

Die Fehlerfunktion wird zunächst um den Strafterm für Energieerhaltung[31] erweitert:

$$J\left(\underline{X}, \underline{Y}, \underline{\tau}\right) = (1 - \lambda_{EC})\mathrm{LogMAE}_{data}\left(\underline{Y}, \underline{\tau}\right) + \underbrace{\lambda_{EC} \log\left(\mathrm{Loss}_{EC}\left(\underline{X}, \underline{Y}\right) + 1\right)}_{Strafterm}$$

Dieser ist nullwertig, wenn Energieerhaltung vorliegt und > 0 bei Anstieg oder Abfallen der Systemenergie über den simulierten Zeithorizont. Die logarithmische Skalierung erfolgt dabei analog zu LogMAE zur Verbesserung der numerischen Konditionierung. Für eine Trainingsdatensequenz l, bestehend aus Eingabesequenz $\underline{X}^{(l)}_{train}$ und Targetsequenz $\underline{\tau}^{(l)}_{train}$ ergibt sich mit den zugehörigen Prädiktionen $\underline{Y}^{(l)}_{train}$ der energieerhaltungsbezogene Fehler als Summe der Änderungen ΔE der Systemenergie:

$$\begin{aligned}
\mathrm{Loss}_{EC}\left(\underline{X}^{(l)}_{train}, \underline{Y}^{(l)}_{train}\right) &= \sum_{k=1}^{n} \left|\Delta E^{(l)}_{k}\right| \\
\underline{X}^{(l)}_{train} &= \left[\left[\underline{x}^{(l)}_{1}, \underline{u}^{(l)}_{1}\right]^{T}, \ldots, \left[\underline{x}^{(l)}_{n}, \underline{u}^{(l)}_{n}\right]^{T}\right] \\
\underline{\tau}^{(l)}_{train} &= \left[\underline{\dot{x}}^{(l)}_{1}, \ldots, \underline{\dot{x}}^{(l)}_{n}\right] \\
\underline{Y}^{(l)}_{train} &= \left[\underline{\hat{\dot{x}}}^{(l)}_{1}, \ldots, \underline{\hat{\dot{x}}}^{(l)}_{n}\right]
\end{aligned}$$

Zur Berechnung der Systemenergie soll zunächst das lineare Viertelfahrzeugmodell betrachtet werden. Die Dynamikgleichung des Viertelfahrzeugs kann umgeschrieben werden in Form der linearen zeitinvarianten Bewegungsgleichung der verallgemeinerten Koordinaten $\underline{q} = [z_A, z_R]^T$ zu

$$\begin{aligned}
&\underline{M}\underline{\ddot{q}}(t) + \left(\underline{D} + \underline{G}\right)\underline{\dot{q}}(t) + \left(\underline{K} + \underline{N}\right)\underline{q}(t) = \underline{f}(t). \\
&\underline{M} = \underline{M}^T > 0, \quad \underline{D} = \underline{D}^T, \quad \underline{G} = -\underline{G}^T, \quad \underline{K} = \underline{K}^T, \quad \underline{N} = -\underline{N}^T
\end{aligned}$$

[31] engl. energy conservation

$$\underline{M} = \begin{bmatrix} m_A & 0 \\ 0 & m_R \end{bmatrix}, \quad \underline{D} = \begin{bmatrix} d_A & -d_A \\ -d_A & d_A + d_R \end{bmatrix}, \quad \underline{G} = \underline{0}, \quad \underline{K} = \begin{bmatrix} c_A & -c_A \\ -c_A & c_A + c_R \end{bmatrix},$$
$$\underline{N} = \underline{0}, \quad \underline{f} = \begin{bmatrix} 0 \\ d_R\dot{u}(t) + c_R u(t) \end{bmatrix}$$

Durch linksseitige Multiplikation mit $\dot{\underline{q}}^T$ ergibt sich die Energiebilanz:

$$\underbrace{\dot{\underline{q}}^T \underline{M}\ddot{\underline{q}}}_{\frac{\mathrm{d}}{\mathrm{d}t}T} + \underbrace{\dot{\underline{q}}^T \underline{D}\dot{\underline{q}}}_{2R} + \underbrace{\dot{\underline{q}}^T \underline{G}\dot{\underline{q}}}_{0} + \underbrace{\dot{\underline{q}}^T \underline{K}\underline{q}}_{\frac{\mathrm{d}}{\mathrm{d}t}V} + \underbrace{\dot{\underline{q}}^T \underline{N}\underline{q}}_{2S} = \underbrace{\dot{\underline{q}}^T \underline{f}}_{P}$$

Dabei entspricht T der kinetischen und V der potentiellen Energie, R der Rayleighschen Dissipation sowie P der Leistung der zeitabhängigen Erregerkräfte $\underline{f}(t)$. [MPS08] Alle anderen Terme fallen im Falle des Viertelfahrzeugmodells weg. Die zeitliche Änderung der Systemenergie ergibt sich damit zu

$$\Delta E_k^{(l)} = \dot{\underline{q}}^T \underline{M}\ddot{\underline{q}} + \dot{\underline{q}}^T \underline{D}\dot{\underline{q}} + \dot{\underline{q}}^T \underline{K}\underline{q} - \dot{\underline{q}}^T \underline{f} \qquad \text{(Energiebilanz lin. VFzg)}$$
$$\dot{\underline{q}}^T \underline{M}\ddot{\underline{q}} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} m_A & 0 \\ 0 & m_R \end{bmatrix} \begin{bmatrix} \hat{\ddot{z}}_{A,k}^{(l)} \\ \hat{\ddot{z}}_{R,k}^{(l)} \end{bmatrix} = m_A \hat{\dot{z}}_{A,k}^{(l)} \hat{\ddot{z}}_{A,k}^{(l)} + m_R \hat{\dot{z}}_{R,k}^{(l)} \hat{\ddot{z}}_{R,k}^{(l)}$$
$$\dot{\underline{q}}^T \underline{D}\dot{\underline{q}} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} d_A & -d_A \\ -d_A & d_A + d_R \end{bmatrix} \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} \\ \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} = d_A \left(\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)} \right)^2 + d_R \left(\hat{\dot{z}}_{R,k}^{(l)} \right)^2$$
$$\dot{\underline{q}}^T \underline{K}\underline{q} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} c_A & -c_A \\ -c_A & c_A + c_R \end{bmatrix} \begin{bmatrix} z_{A,k}^{(l)} \\ z_{R,k}^{(l)} \end{bmatrix} = c_A \left(\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)} \right) \left(\hat{z}_{A,k}^{(l)} - \hat{z}_{R,k}^{(l)} \right) + c_R \hat{\dot{z}}_{R,k}^{(l)} z_{R,k}^{(l)}$$
$$\dot{\underline{q}}^T \underline{f} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} 0 \\ d_R \dot{u}_k^{(l)} + c_R u_k^{(l)} \end{bmatrix} = \hat{\dot{z}}_{R,k}^{(l)} \left(d_R \dot{u}_k^{(l)} + c_R u_k^{(l)} \right)$$

Es fällt auf, dass die einfachen und zweifachen zeitlichen Ableitungen der verallgemeinerten Koordinaten dabei der Prädiktion $\hat{\underline{x}}_k^{(l)}$ und die verallgemeinerten Koordinaten sowie die Erregerkräfte den Trainingseingaben $\left[\underline{x}_1^{(l)}, \underline{u}_1^{(l)}\right]^T$ entstammen. Damit kann für jedes Datenpaar aus Prädiktion und Eingabe zu Zeitpunkt t_k bestimmt werden, ob das System Energie konserviert, verliert oder gewinnt. Ist $\Delta E_k^{(l)}$ für mindestens ein Datenpaar > 0, so führt dies zu einem Anstieg der Fehlerfunktion und induziert eine Gewichtsänderung.

Analog zum linearen Viertelfahrzeugmodell kann die Bewegungsgleichung des nichtlinearen Viertelfahrzeugmodells aufgestellt werden:

$$\underbrace{\begin{bmatrix} m_A & 0 \\ 0 & m_R \end{bmatrix}}_{\underline{M}} \ddot{\underline{q}} + \underbrace{\begin{bmatrix} d_A \mathrm{sign}\left(\dot{z}_A - \dot{z}_R\right) |\dot{z}_A - \dot{z}_R|^\alpha \\ d_A \mathrm{sign}\left(\dot{z}_R - \dot{z}_A\right) |\dot{z}_R - \dot{z}_A|^\alpha + d_R \dot{z}_R \end{bmatrix}}_{\underline{D}\left(\dot{\underline{q}}\right)} + \underbrace{\begin{bmatrix} c_A & -c_A \\ -c_A & c_A + c_R \end{bmatrix}}_{\underline{K}} \underline{q} = \underbrace{\begin{bmatrix} 0 \\ d_R \dot{u} + c_R u \end{bmatrix}}_{\underline{f}}$$

Durch linksseitige Multiplikation mit $\dot{\underline{q}}^T$ ergibt sich wieder die Energiebilanz:

$$\Delta E_k^{(l)} = \dot{\underline{q}}^T \underline{M}\ddot{\underline{q}} + \dot{\underline{q}}^T \underline{D}\left(\dot{\underline{q}}\right) + \dot{\underline{q}}^T \underline{K}\underline{q} - \dot{\underline{q}}^T \underline{f} \qquad \text{(Energiebilanz n-lin. VFzg)}$$

$$\dot{\underline{q}}^T \underline{M}\ddot{\underline{q}} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} m_A & 0 \\ 0 & m_R \end{bmatrix} \begin{bmatrix} \hat{\ddot{z}}_{A,k}^{(l)} \\ \hat{\ddot{z}}_{R,k}^{(l)} \end{bmatrix} = m_A \hat{\dot{z}}_{A,k}^{(l)} \hat{\ddot{z}}_{A,k}^{(l)} + m_R \hat{\dot{z}}_{R,k}^{(l)} \hat{\ddot{z}}_{R,k}^{(l)}$$

$$\dot{\underline{q}}^T \underline{D}\left(\dot{\underline{q}}\right) = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} d_A \operatorname{sign}\left(\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)}\right) \left|\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)}\right|^{\alpha} \\ d_A \operatorname{sign}\left(\hat{\dot{z}}_{R,k}^{(l)} - \hat{\dot{z}}_{A,k}^{(l)}\right) \left|\hat{\dot{z}}_{R,k}^{(l)} - \hat{\dot{z}}_{A,k}^{(l)}\right|^{\alpha} + d_R \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix}$$

$$= d_A \left|\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)}\right|^{\alpha+1} + d_R \left(\hat{\dot{z}}_{R,k}^{(l)}\right)^2$$

$$\dot{\underline{q}}^T \underline{K}\underline{q} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} c_A & -c_A \\ -c_A & c_A + c_R \end{bmatrix} \begin{bmatrix} z_{A,k}^{(l)} \\ z_{R,k}^{(l)} \end{bmatrix}$$

$$= c_A \left(\hat{\dot{z}}_{A,k}^{(l)} - \hat{\dot{z}}_{R,k}^{(l)}\right)\left(\hat{z}_{A,k}^{(l)} - \hat{z}_{R,k}^{(l)}\right) + c_R \hat{\dot{z}}_{R,k}^{(l)} z_{R,k}^{(l)}$$

$$\dot{\underline{q}}^T \underline{f} = \begin{bmatrix} \hat{\dot{z}}_{A,k}^{(l)} & \hat{\dot{z}}_{R,k}^{(l)} \end{bmatrix} \begin{bmatrix} 0 \\ d_R \dot{u}_k^{(l)} + c_R u_k^{(l)} \end{bmatrix} = \hat{\dot{z}}_{R,k}^{(l)} \left(d_R \dot{u}_k^{(l)} + c_R u_k^{(l)}\right)$$

Wie bereits in Kapitel 3.1 angesprochen, erfordert die Gewichtung der Strafterme exzessives Einstellen der Lagrange-Multiplikatoren λ. Die Identifikation von λ_{EC} kann bspw. im Rahmen der HP-Optimierung durch Bayessche Optimierung erfolgen. Auch die Anwendbarkeit des hier vorgestellten, energieerhaltenden Strafterms beruht darauf, dass die zugrundeliegenden Netzeingabe- und Prädiktionswerte nur sehr geringes Rauschen beinhalten. Starke Schwankungen würden ebenfalls zu starken Abweichungen in dem resultierenden Strafmaß führen.

3.8 Erweiterung um eine Funktionsbibliothek

Im Rahmen der Modellierung nichtlinearer Systeme wird die nichtlineare Dynamik mittels RNNs lediglich approximiert. Eine exakte Abbildung der zugrundeliegenden Zusammenhänge ist idR. nicht möglich und fordert zudem eine hohe Modellkomplexität, die sich direkt in ein umfangreiches Training übersetzt. Ebenso steigen auch die Anforderungen an die Datengrundlage.

Das Modellierungsverfahren SINDY wurde bereits in Kapitel 2.2 vorgestellt und ermöglicht die Vorgabe einer Bibliothek verschiedener nichtlinearer Terme des Zustands sowie der Stellgrößen. Das Verfahren eignet sich daher besonders zur Modellierung nichtlinearer Systeme, weil lediglich skalare Faktoren von dem datenbasierten Modell gelernt werden müssen. Um auch im Fall von PGRNNs diesen Effekt zu nutzen, soll in diesem Kapitel das PGRNN um eine Funktionsbibliothek erweitert werden.

Analog zur Vorgehensweise in Kapitel 3.6 wird das PGRNN um einen weiteren Netzeingang erweitert (siehe Abb. 3-6). Wie beim Modellierungsverfahren SINDY wird eine Bibliothek nichtlinearer Kandidatenfunktionen gebildet (Gl. (3-7)). Die Terme sind dabei Funktionen der Netzeingaben $\underline{x}_k$ und $\underline{u}_k$. Zu jedem Zeitpunkt t_k wird dem RNN folglich ein

Vektor $\underline{\theta}_k$ als zusätzliche Eingabe zur Verfügung gestellt. Die um die Funktionsbibliothek erweiterte Netzvariante wird im Folgenden als PGRNN+ bezeichnet.

$$\underline{\theta}_k \equiv \underline{\theta}\left(\underline{x}_k, \underline{u}_k\right) = \begin{bmatrix} \underline{x}_k^2 \\ \underline{u}_k^2 \\ \vdots \\ \sin\left(\underline{x}_k\right) \\ \sin\left(\underline{u}_k\right) \\ \vdots \end{bmatrix} \tag{3-7}$$

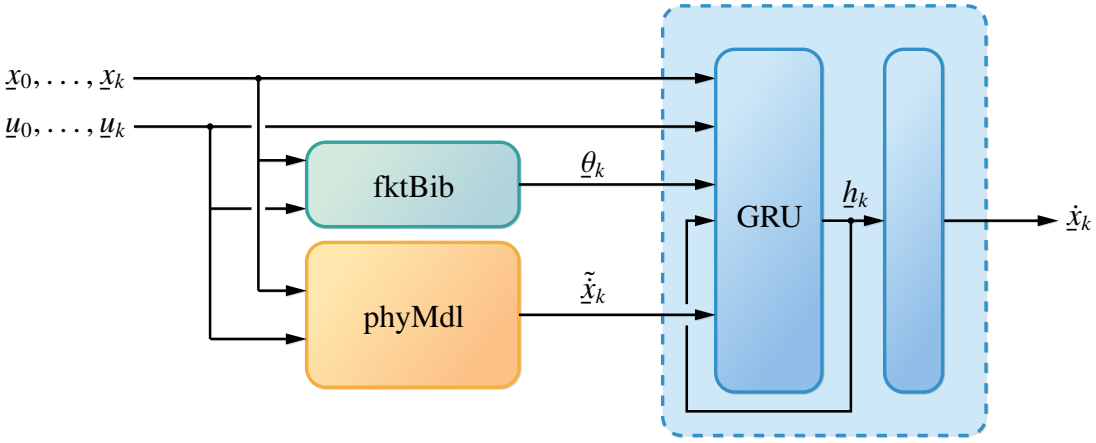

Bild 3-6: *Das Modell wird erneut um einen Eingang erweitert. Die Einspeisung der Auswertungen der Funktionsbibliothek $\underline{\theta}_k$ zum Zeitpunkt t_k erfolgt parallel zu den Ausgaben des physikalischen Modells $\tilde{\underline{x}}_k$. Die Dynamik des PGRNN+ $\dot{\underline{x}}_k$ zum Zeitpunkt t_k ergibt sich somit als Funktion aller vergangener und aktueller Zustandsvektoren und Stellgrößen $\underline{x}_i$ bzw. $\underline{u}_i$, der Funktionsauswertungen $\underline{\theta}_k$ sowie der physikalisch-approximativen Dynamik $\tilde{\underline{x}}_k$.*

Aufbauend auf Gleichung (3-5) ergibt sich damit die Dynamikgleichung des PGRNN+ durch

$$\begin{aligned} \dot{\underline{x}}_k &= \underline{f}_{PGRNN+}\left(\underline{x}_0, \underline{u}_0, \tilde{\underline{x}}_0, \underline{\theta}_0, \ldots, \underline{x}_k, \underline{u}_k, \tilde{\underline{x}}_k, \underline{\theta}_k | \underline{w}\right), \\ \tilde{\underline{x}}_k &= \underline{f}_{phy}\left(\underline{x}_k, \underline{u}_k\right). \end{aligned}$$

Die Funktionsbibliothek kann nun um eine Vielzahl verschiedener nichtlinearer Terme angereichert werden. Dadurch steigt allerdings auch die Anzahl an Neuronen. Daher sollte im besten Fall im Vorhinein eine Auswahl von Termen getroffen werden, deren Vorkommen in der zu modellierenden Systemdynamik erwartet wird.

Das Modellierungsverfahren SINDY basiert auf der LASSO-Regularisierung, die, wie die Fehlerfunktion MAE, auf der L1-Norm beruht und eine dünne Besetzung der Gewichtsmatrizen anregt. Es ist daher zu erwarten, dass zur Einbindung der Funktionsbibliothek im Rahmen des PGRNN+ eine MAE-Fehlerfunktion von Vorteil ist.

Die Untersuchungen zur Erweiterung des PGRNNs um eine Funktionsbibliothek sind Kapitel 4.6 zu entnehmen.

4 Untersuchungen: Physics-guided Recurrent Neural Networks

In diesem Kapitel sollen die zuvor beschriebenen Ansätze hinsichtlich ihrer Performanz untersucht werden. Dabei gilt als Gütemaß der Simulationsfehler über den Testdaten (eingeführt in Kap. 3.4). Es werden für diverse Untersuchungen drei Beispielsysteme betrachtet:

- das Viertelfahrzeugmodell,
- der Lorenz-Attraktor und
- der Cubli.

In einem ersten Schritt konnte in Kapitel 3.5 anhand des Viertelfahrzeugs bereits gezeigt werden, dass RNNs zur Modellierung nicht-autonomer nichtlinearer technischer Systeme geeignet sind. Im Folgenden soll nun gezeigt werden, dass PGRNNs einen Mehrwert ggü. RNNs liefern. Um den Untersuchungsfokus zu definieren, sollen zunächst anhand von vier Hypothesen gewisse Erwartungshaltungen festgehalten werden. Diese sollen in den fortlaufenden Kapiteln untersucht und verifiziert werden.

Hypothese 1. (1) *Durch Erweiterung um ein physikalisches Dynamikmodell ausreichender Güte kann der Simulationsfehler bei gleicher Datengrundlage reduziert werden. Die Güte des Dynamikmodells übersetzt sich dabei direkt in die Güte des resultierenden PGRNNs.*

(2) *Die Güte der Dynamikmodelle ist durch die inhärenten Modellparameter sowie die korrekten Terme bzw. Nichtlinearitäten charakterisiert. Je stärker Modellparameter und nichtlineare Terme vom GT abweichen, desto geringer ist der Mehrwert der Erweiterung um das Dynamikmodell.*

(3) *Chaotische Systeme wie etwa der Lorenz-Attraktor sind besonders sensitiv ggü. Parameterabweichungen. Für die Modellierung chaotischer Systeme durch PGRNNs ist eine ausreichend genaue Parametrierung des Dynamikmodells essentiell.*

Hypothese 2. *Trainingsdaten können von verschiedener Güte sein. Dabei zeichnet sich ein besonders geeigneter Datensatz durch eine hohe Varianz der durchlaufenen Zustände aus. Datensequenzen, die hauptsächlich periodische Verläufe im eingeschwungenen Zustand beinhalten, sind repetitiv und von geringem Mehrwert für das Training.*

Hypothese 3. *Die Beschränkung des Optimierungsraums auf energieerhaltende Lösungen wirkt sich positiv auf das Konvergenzverhalten aus. Die unter Einhaltung von Energieerhaltung trainierten Netze weisen infolgedessen niedrigere Simulationsfehler auf.*

Hypothese 4. *Auch geregelte Systeme können durch RNNs ausreichend gut abgebildet werden. Darüber hinaus kann durch Erweiterung um ein physikalisches Dynamikmodell ausreichender Güte eine Reduktion des Simulationsfehlers ggü. dem RNN verzeichnet werden.*

Im Folgenden werden zunächst die Beispielsysteme Lorenz-Attraktor und Cubli vorgestellt. Dem sind die Kapitel der Untersuchungen angestellt. Ziel der Untersuchungen ist die Verifizierung der aufgestellten Hypothesen.

4.1 Lorenz-Attraktor

Bei dem *Lorenz-System* oder auch *Lorenz-Attraktor* genannt, handelt es sich um ein sehr bekanntes Differentialgleichungssystem, das ursprünglich zur Beschreibung hydrodynamischer Systeme entwickelt wurde. Bei den Untersuchungen des Systemverhaltens fiel damals auf, dass die Lösung des DGL-Systems bereits für geringe Änderungen der Parametrierung a, b, c enorm divergiert. Für gewisse Parameterintervalle weist das System unregelmäßige, nicht-periodische Oszillationen auf, die sich auf einen begrenzten Teil des Phasenraums beschränken. Daher wird das System der Klasse der seltsamen Attraktoren zugeordnet. [vgl. JP03]

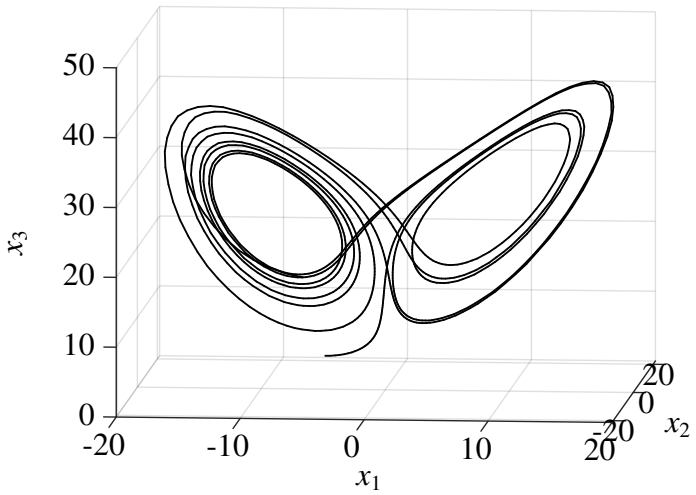

Bild 4-1: *Bei dem Lorenz-System handelt es sich um einen seltsamen Attraktor, dessen Trajektorien chaotisch in einem beschränkten Bereich des Zustandsraums oszillieren.*

$$\begin{aligned} \underline{\dot{x}}(t) &= \underline{f}\left(\underline{x}(t)\right) \\ \Leftrightarrow \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} &= \begin{bmatrix} a\left(x_2(t) - x_1(t)\right) \\ x_1(t)\left(b - x_3(t)\right) - x_2(t) \\ x_1(t) \cdot x_2(t) - c x_3(t) \end{bmatrix} \end{aligned} \qquad \text{(Lorenz-System)}$$

Für die im Rahmen dieser Arbeit durchgeführten Untersuchungen wurde die in Tabelle A1-2 gegebene Parametrierung als Ground Truth (GT) verwendet. Mit dieser Parametrierung stellt sich das für den Lorenz-Attraktor charakteristische Oszillationsverhalten um die zwei Ruhelagen ein (siehe Abb. 4-1). Neben der Sensitivität ggü. Parametervariationen handelt es sich bei dem Lorenz-System um ein chaotisches System. D.h., dass sich bereits für minimal verschiedene Anfangsbedingungen völlig verschiedene Trajektorien und somit auch komplett andere Endzustände einstellen. [vgl. JP03] Der Lorenz-Attraktor ist daher traditionell ein akademisches Benchmarking-System für Modellierungsmethoden zur

Untersuchung von Langzeitverhalten. In dieser Arbeit dient die Anschauung des Lorenz-Attraktors daher besonders dazu, bereits geringfügige Ungenauigkeiten der trainierten Netze und das daraus resultierende Langzeitprädiktionsvermögen zu untersuchen.

4.2 Cubli

Ein weiteres untersuchtes System ist der Cubli, der 2012 an der ETH Zürich entwickelt wurde [MMIR12]. Dabei handelt es sich um einen würfelförmigen Demonstrator (Abb. 4-2), der mittels Schwungrädern und einem geeigneten Regler auf seine Ecken und Kanten springen und balancieren kann. Im Rahmen des DART-Projekts des Heinz Nixdorf Instituts Paderborn soll ein ähnlicher Demonstrator entwickelt werden. Dazu wurden zunächst anhand des ursprünglichen, nichtlinearen Zustandsraummodells geeignete Parameter (siehe A1-3) bestimmt.

Bild 4-2: *Der Cubli verfügt über drei Schwungräder, mit denen er durch geschicktes Beschleunigen und Abbremsen aus der stabilen Ruhelage in einem ersten Schritt auf eine seiner Kanten springen und schlussendlich auf einer Ecke stabilisiert werden kann. [21]*

$$\dot{\underline{x}}(t) = \underline{f}\left(\underline{x}(t), u(t)\right)$$

$$\Leftrightarrow \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ \frac{-c_w x_4(t) - c_b x_2(t) - K_m u(t) + (l_b m_b + l m_w) g \sin(x_1(t))}{J_b + m_b l_b^2 + J_w + m_w l^2} \\ x_4(t) \\ \frac{(K_m u(t) - c_w x_4(t))\left(J_b + m_b l_b^2 + J_w + m_w l^2\right)}{J_w\left(J_b + m_b l_b^2 + m_w l^2\right)} - \frac{c_b x_2(t) + (l_b m_b + l m_w) g \sin(x_1(t))}{J_b + m_b l_b^2 + m_w l^2} \end{bmatrix} \quad \text{(n-lin. Cubli)}$$

Da für die Auslegung eines Reglers zur Einstellung der oberen Ruhelage ein lineares Dynamikmodell bestimmt werden muss, wurde das nichtlineare Modell außerdem um den Arbeitspunkt $\bar{\underline{x}} = \underline{0}$ linearisiert:

$$\dot{\underline{x}}(t) = \underline{A}\underline{x}(t) + \underline{b}u(t)$$

$$\Leftrightarrow \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{(l_b m_b + l m_w) g}{J_b + m_w l^2} & \frac{-c_b}{J_b + m_w l^2} & \frac{-c_w}{J_b + m_w l^2} \\ -\frac{(l_b m_b + l m_w) g}{J_b + m_w l^2} & \frac{c_b}{J_b + m_w l^2} & -\frac{c_w\left(J_b + J_w + m_w l^2\right)}{J_w\left(J_b + m_w l^2\right)} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 \\ -\frac{K_m}{J_b + m_w l^2} \\ \frac{K_m\left(J_b + J_w + m_w l^2\right)}{J_w\left(J_b + m_w l^2\right)} \end{bmatrix} u(t)$$

(lin. Cubli)

Da der Cubli im Rahmen dieser Ausarbeitung nur kurz betrachtet wird, soll an dieser Stelle für weitere Informationen auf [Reg21] verwiesen werden.

4.3 Untersuchung: Güte des physikalischen Modells

Das physikalische Dynamikmodell ist zentraler Bestandteil der behandelten PGRNN-Architektur. Dabei wird in der Realität die Parametrierung des physikalischen Modells von der Ground-Truth-Parametrierung abweichen, da einzelne Parameter entweder heuristisch bestimmt oder infolge einer Parameteridentifikation approximiert wurden. Es ist daher anzunehmen, dass die Parameter des physikalischen Modells prozentual von den GT-Parametern abweichen. Um zu untersuchen, welche Auswirkung unterschiedlich starke Abweichungen auf die Güte des resultierenden PGRNN haben, wird daher im Folgenden von einer weiteren Parameteridentifikation abgesehen.

Welchen Einfluss die Güte der Parametrierung des physikalischen Modells auf die Güte des resultierenden PGRNNs hat, soll anhand des Viertelfahrzeugs und des Lorenz-Attraktors untersucht werden.

4.3.1 Viertelfahrzeug

Im Bezug auf das Viertelfahrzeugmodell soll zunächst untersucht werden, welchen Einfluss die Güte des verwendeten physikalischen Modells, iFv. Differenzialgleichungen, auf die Güte des resultierenden PGRNNs hat. Die Güte des physikalischen Modells soll hier anhand zweier Faktoren untersucht werden:

1. abweichende Nichtlinearitäten und
2. abweichende Modellparameter.

Die Abweichungen sind dabei jeweils zwischen physikalischem Modell und GT-Modell zu verstehen. Auch wenn es sich bei Nichtlinearitäten wie einem Exponenten > 1 auch um einen Modellparameter handelt, sollen Parameter iFv. Faktoren grundsätzlich von Parametern iFv. Exponenten unterschieden werden. Diese Unterscheidung wird getroffen, da abweichende Exponenten einen stärkeren Einfluss auf die Systemdynamik haben als abweichende Faktoren. Im Folgenden wird daher von Nichtlinearitäten bzw. Exponenten und von (Modell-)Parametern gesprochen.

Untersuchungsvorgehen

Besonders interessant zu untersuchen ist, inwieweit PGRNNs zur Modellierung nichtlinearer dynamischer Systeme genutzt werden können. Um den Einfluss unterschiedlicher Güten des physikalischen Modells zu untersuchen, sollen die Parameter variiert und die Untersuchungsergebnisse gegenübergestellt werden. Für das in Kapitel 3.3 beschriebene nichtlineare Viertelfahrzeug werden dazu verschiedene Exponenten α zwischen 1 (Exp. 1) und 3 (Exp.3) betrachtet, während das als Referenz dienende GT-Modell den Exponenten $\alpha = 3$ zugrundelegt. Letzteres dient zur Generierung der Trainings-, Validierungs- und Testdaten.

Um den Einfluss der Variation des Exponenten α zu veranschaulichen, werden zunächst die physikalischen Dynamikmodelle mit Exponent 1 und 2 isoliert simuliert und mit den Trajektorien des GT-Modells (Exp. 3) verglichen. Die sich ergebenden Trajektorien (siehe Abb. 4-3) weisen hierbei jeweils nur für kurze Zeitbereiche nennenswerte Abweichungen voneinander auf, welche sich hauptsächlich auf den Einschwingvorgang beschränken.

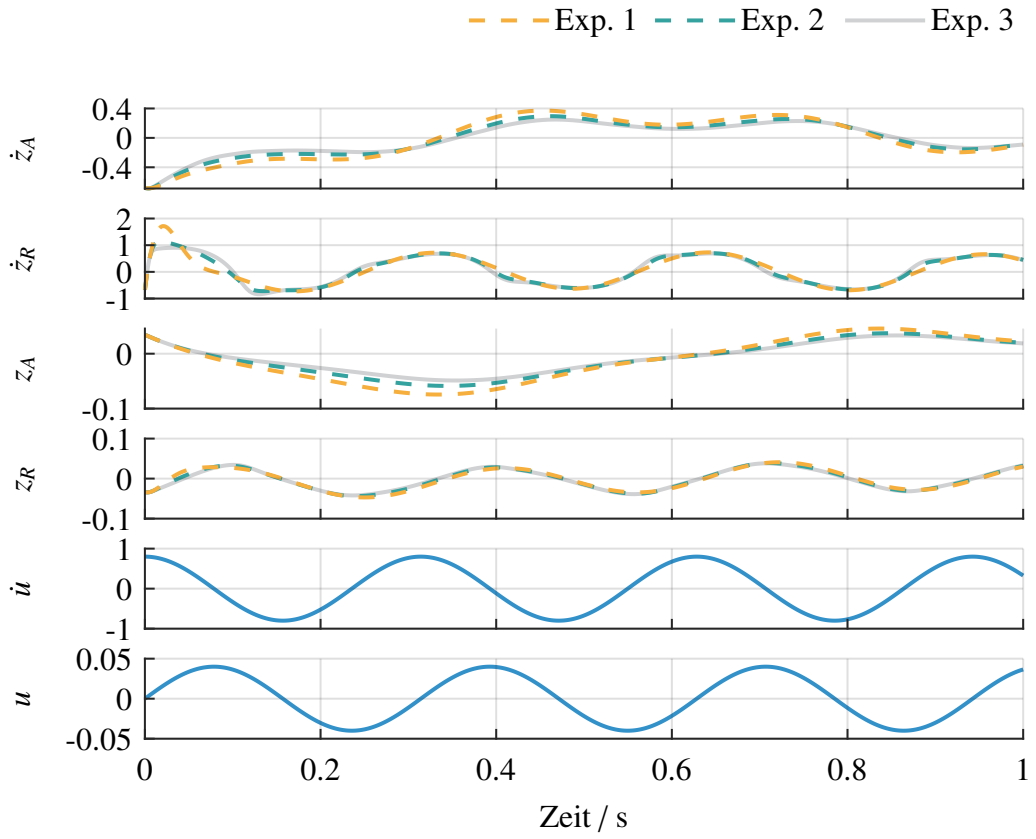

Bild 4-3: *Abgebildet sind die Trajektorien für die Modelle mit verschiedenen Exponenten* $\alpha \in \{1, 2, 3\}$. *Makroskopisch betrachtet, beschreiben alle drei Modelle in etwa die gleichen Verläufe, wobei auf mikroskopischer Ebene klare Diskrepanzen deutlich werden.*

Im Rahmen einer Systemidentifikation wird idR. ebenfalls eine Parameteridentifikation durchgeführt. Dazu wird auf Basis der Mess-/ bzw. Trainingsdaten die wahrscheinlichste Parametrierung des ODE-Systems bestimmt. Inwiefern die Parametrierung des physikalischen Modells Einfluss auf die Güte des resultierenden PGRNNs hat, soll anhand verschiedener, von der GT-Parametrierung abweichender Parametrierungen, untersucht werden. Dazu wird angenommen, dass die Masse des Fahzeugaufbaus m_A und des Rades m_R relativ präzise bestimmt werden können. Sie werden daher gleich den entsprechenden GT-Werten angenommen. Die Feder- sowie Dämpferkonstanten c_A, c_R bzw. d_A, d_R sollen dagegen als vom GT abweichend angenommen werden. Dazu werden die Werte prozentual von ihren entsprechenden GT-Werten variiert. Eine Abweichung von -10% bedeutet dabei etwa, dass c_A, c_R, d_A und d_R um 10% kleiner als ihre entsprechenden GT-Werte

ausfallen. Die GT-Parametrierung des Viertelfahrzeugs sowie alle anderen verwendeten Parametrierungen können Kap. A1 entnommen werden.

Um zu veranschaulichen, welche Auswirkungen die Parameterabweichungen auf das dynamische Verhalten des Gesamtsystems haben, sollen zunächst Dynamikmodelle mit einer Parameterabweichung von −10% und −75% isoliert simuliert werden. Es stellen sich verschiedene Trajektorien ein, die Abbildung 4-4 entnommen werden können. Aufgrund der relativ geringen Sensitivität des betrachteten Systems ggü. Parameteränderungen sind die Trajektorien mit GT und um −10% abweichender Parametrierung fast deckungsgleich. Bei einer Abweichung von −75% divergieren die Trajektorien dagegen recht stark.

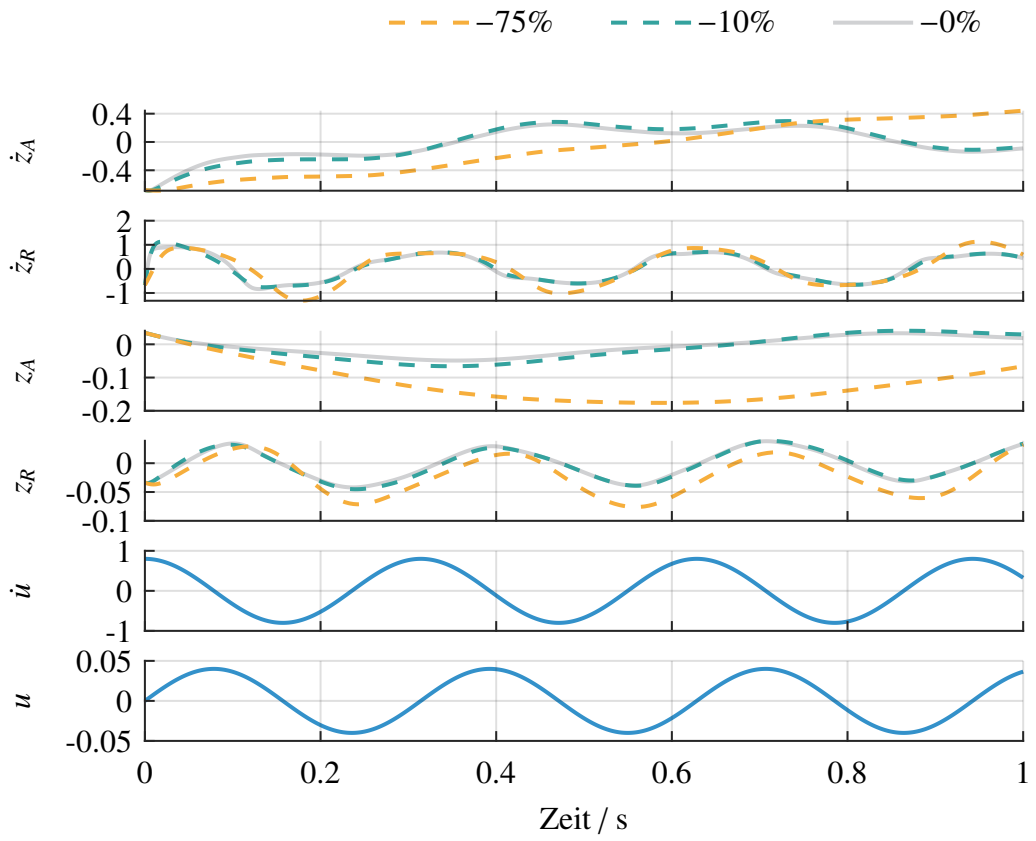

Bild 4-4: *Anders als für die verschiedenen Exponenten, wirkt sich die abweichende Parametrierung insbesondere auf makroskopischer Ebene aus. Dabei resultieren durch die gleichbleibenden Massenverhältnisse zeitlich gedehnte Trajektorien für steigende Parameterabweichungen.*

Für die verschieden parametrierten physikalischen Modelle wurden jeweils PGRNNs trainiert. Anschließend wurden die Netze für 1s simuliert und die resultierenden Simulationsfehler verglichen. Dazu wurde zunächst jeweils eine HP-Optimierung der versteckten Neuronen durchgeführt. Im Anschluss wurden dann 10 identische Netzinstanzen mit der besten beobachteten Hyperparametrierung – man spricht von *Best Observed* – trainiert und die beste Instanz sowie der zugehörige minimale Simulationsfehler identifiziert. Mit diesem Vorgehen wird auf die Erreichung eines möglichst geringen Simulationsfehlers und somit auf die höchstmögliche Netzgüte abgezielt. Auch wenn auf diese Weise

das Potential der betrachteten PGRNN-Konfigurationen am besten abgeschätzt werden kann, unterliegen die beobachteten Optima statistischer Streuung. Statistisch vergleichbarere Ergebnisse bietet dagegen der Mittelwert der 10 mit identischen HPs trainierten Netzinstanzen sowie der Standardabweichung. Da allerdings die dabei verwendeten HPs der besten beobachteten und nicht der besten geschätzten Parametrierung – man spricht von *Best Estimated* – entsprechen und diese nicht zwangsweise identisch sind, kann der Mittelwert teils deutlich vom beobachteten Optimum abweichen. Unter Berücksichtigung dieses Zusammenhangs sollen demnach primär die besten beobachteten Netzgüten verglichen werden, auch wenn diese einer gewissen Streuung unterliegen.

In vereinzelten Fällen treten Netzinstanzen auf, die als Ausreißer klassifiziert werden können und besonders schlechte oder besonders gute Netzgüten aufweisen. Da diese den Mittelwert und die Standardabweichung überproportional beeinflussen, wurden entsprechende Ausreißer identifiziert[32] und entfernt. Die entsprechenden Verteilungen sind dann rot gekennzeichnet.

Da neben dem Simulationsfehler auch die Trainingsdauer zu optimieren ist, wird eine Mehrzieloptimierung angestrebt. Zusätzlich zum Simulationsfehler auf den Testdaten wird daher eine hohe Trainingsdauer durch einen Strafterm in der Fehlerfunktion berücksichtigt. Da eine hohe Neuronenzahl idR. auch eine längere Trainingsdauer zufolge hat, wird so entsprechend ein Kompromiss zwischen Modellkomplexität und Trainingsdauer verfolgt. Die zu optimierende ***Netzgüte*** J_{sim} ergibt sich daher im Folgenden aus dem logarithmisch skalierten Simulationsfehler (Kap. 3.4) über den Testdaten, $\log\left(E_{sim}\left(\underline{Y}_{test}, \underline{\boldsymbol{\tau}}_{test}\right)\right)$ sowie der Trainingsdauer T_{train} in Sekunden:

$$J_{sim} = \log\left(E_{sim}\left(\underline{Y}_{test}, \underline{\boldsymbol{\tau}}_{test}\right)\right) + 10^{-4} \cdot \sqrt{T_{train}} \tag{4-1}$$

Die bei den Untersuchungen verwendeten Trainingseinstellungen können Tabelle A2-1 entnommen werden.

Vorgehen bei den Untersuchungen

- Bayessche Optimierung der Hyperparameter anhand Netzgüte
- Trainieren 10 identischer Netzinstanzen
- Bewertung der Netze in Bezug auf Simulationsfehler
- Entfernung von Ausreißern
- Identifikation bester Instanz, Mittelwert und Standardabweichung

Ergebnisse

Die Ergebnisse der Bayesschen Optimierung (Kap. A3) lassen einen deutlichen Unterschied zwischen dem Verhalten von RNN und PGRNNs erkennen. So ist für das RNN

[32] Eine Klassifikation als Ausreißer erfolgt, wenn der zugehörige Wert um mehr als drei skalierte mittlere absolute Abweichungen (MAD) vom Median abweicht.

im Gegensatz zu den PGRNNs ein markantes globales Minimum auszumachen, das ferner bei mehr als zweimal weniger Neuronen liegt, als die minimalste Neuronenzahl aller PGRNN-Konfigurationen. Im Gegensatz dazu ist für die PGRNNs für steigende Neuronenzahlen zunächst ein starker Abfall des Simulationsfehlers zu verzeichnen, der jedoch in einer Art Plateau mündet. Da für RNNs und PGRNNs mit steigender Anzahl versteckter Neuronen auch die Trainierbarkeit leidet, steigt der Simulationsfehler für sehr hohe Neuronenzahlen wieder oder entsprechende Auswertungen endeten mit einem Fehler. Zusätzlich wird eine hohe Trainingsdauer durch einen Strafterm berücksichtigt.

Das allgemeine Simulationsfehlerniveau variiert von Konfiguration zu Konfiguration, das Fehlerlevel des RNN ist jedoch i.A. deutlich höher als das der PGRNNs. Wie zuvor beschrieben, fallen das geschätzte Minimum und das beobachtete Minimum oft nicht zusammen. Aus genannten Gründen wurde idR. das beobachtete Minimum weiter betrachtet und zum Training der 10 identischen Netzinstanzen verwendet.

In Abbildung 4-5 sind die Ergebnisse der Versuchsreihe dargestellt. Dabei wurde zur Referenz ein standardmäßiges RNN ohne physikalisches Modell nach dem gleichen Vorgehen trainiert. Dabei fällt zunächst auf, dass die Hinzunahme eines physikalischen Modells in allen Fällen zu einer Verringerung des beobachteten Simulationsfehlers geführt hat. Diese Beobachtung stützt Hypothese 1 Abs. 1 S. 1.

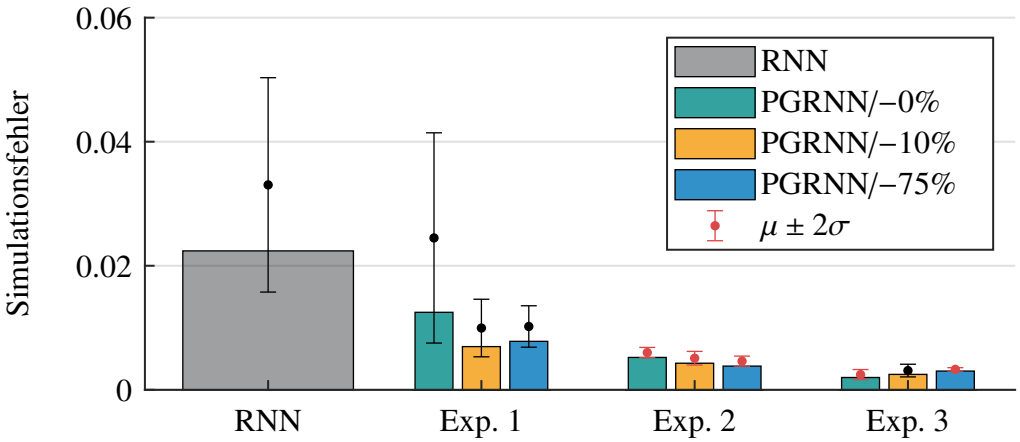

Bild 4-5: *Alle PGRNN-Varianten sind dem rein datengetriebenen RNN deutlich überlegen. Dabei hat die Parametrierung des physikalischen Modells keinen direkten Einfluss auf die resultierende Netzgüte. Für die Güte des Exponenten können dagegen vergleichsweise starke Korrelationen beobachtet werden.*

Ein Vergleich zweier beispielhafter Testtrajektorien von RNN und PGRNN/−10%/Exp.3 (Abb. 4-6) verdeutlicht die Unterschiede bei einem durchweg sehr niedrigen Simulationsfehlerlevel. Bereits das RNN ermöglicht eine sehr gute Abbildung der Systemdynamik des Viertelfahrzeugs, sodass die generierten Trajektorien nur vereinzelt sichtbar vom GT abweichen. Es kann ebenso festgehalten werden, dass der Exponent α des verwendeten physikalischen Modells einen erheblichen Einfluss auf die Güte des resultierenden Netzes hat. Dabei hatte bereits die Hinzunahme eines linearen Dynamikmodells (Exp. 1) einen

Abfall des Simulationsfehlers um bis zu $1.5 \cdot 10^{-2}$ zur Folge. Das entspricht einer Reduktion um fast 70%. Die Sublimierung des Exponenten zu $\alpha = 2$ (Exp. 2) und $\alpha = 3$ (Exp. 3) führte zu einer weiteren Verringerung um maximal $0.32 \cdot 10^{-2}$ bzw. $0.5 \cdot 10^{-2}$. Es gilt festzuhalten, dass bereits bei Verwendung eines linearen Modells (Exp. 1) 3/4 der Fehlerreduktion des Modells mit korrektem Exponenten (Exp. 3) erreicht werden konnte.

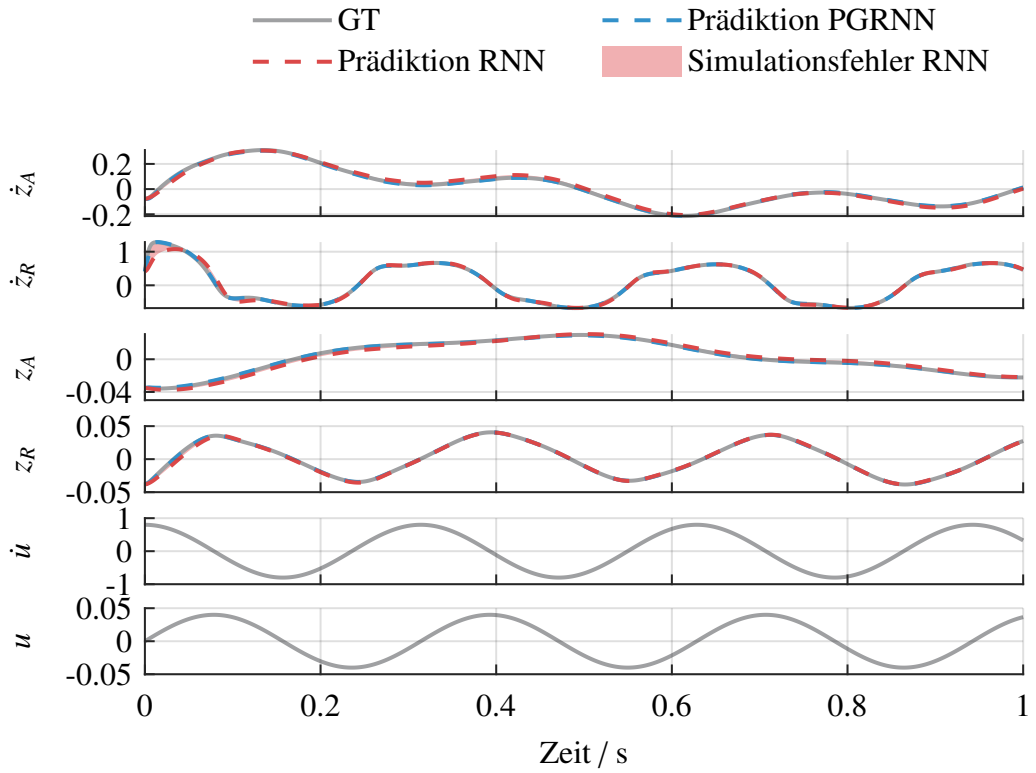

Bild 4-6: *Sowohl RNN als auch PGRNN lernen gute Modelle des nichtlinearen Viertelfahrzeugs. Die Trajektorien des RNN-Modells weichen dabei nur vereinzelt sichtbar von denen des PGRNNs und GT ab.*

Während die Ergebnisse für die verschiedenen Nichtlinearitäten konsistent ausfallen, ist dies für die unterschiedlichen Parametrierungen nicht der Fall. So kann nur für den Exponenten 3 – dem mit GT übereinstimmenden Exponenten – ein mit Hypothese 1 Abs. 2 konformes Resultat beobachtet werden. Für beide anderen Exponenten übersetzt sich die Güte der Modellparameter nicht unmittelbar in die Güte des resultierenden PGRNNs. Die Netze mit korrekter physikalischer Parametrierung (−0%) liefern nur marginal bessere Ergebnisse als die Modelle mit einer Abweichung von −75%, die, wie oben beschrieben, isoliert betrachtet zu relativ stark abweichenden Trajektorien führen. Die ähnlichen Mittelwerte deuten darauf hin, dass die Güte der Modellparametrierung in den betrachteten Fällen nur einen schwachen, wenn überhaupt existenten, Effekt auf die Güte des resultierenden Netzes hat. Die hohe Standardabweichung der Netzinstanzen PGRNN/−10%/Exp. 1 und PGRNN/−75%/Exp. 1 könnte ebenfalls auf ähnlichen, regelrecht willkürlichen Umständen basieren.

Zusammenfassend lässt sich sagen, dass mit allen betrachteten Netzvarianten – mit und ohne physikalischem Modell – sehr gute Modelle des Viertelfahrzeugs gelernt werden konnten. Die Netze mit physikalischem Modell sind dennoch deutlich überlegen, wobei

die Variierung der Parametrierung des physikalischen Modells in den Untersuchungen des Viertelfahrzeugmodells keinen nennenswerten Einfluss auf die resultierende Netzgüte hatte. Dagegen spielt die verwendete Nichtlinearität eine vergleichsweise große Rolle, wobei unter Verwendung eines linearen Modells bereits sehr gute Ergebnisse erzielt werden konnten und der Simulationsfehler infolge der Berücksichtigung „geeigneterer“ Exponenten nur geringfügig weiter verringert werden konnte.

Diese Erkenntnisse stützen durchweg Hypothese 1 Abs. 1 S. 1, dass die Erweiterung um ein Dynamikmodell zu einer Reduktion des Simulationsfehlers führt. Hypothese 1 Abs. 1 S. 2 sowie Hypothese 1 Abs. 2 sind dagegen differenziert zu betrachten. So konnte zwar ein Zusammenhang zwischen der Güte des Exponenten und der Güte der PGRNNs beobachtet werden, nicht aber zwischen der Güte der Modellparametrierung und der Güte der PGRNNs. Ob dies auf die relativ geringe Sensitivität des Viertelfahrzeugs ggü. Parameteränderungen zurückzuführen ist, soll im folgenden Kapitel anhand des Lorenz-Attraktors untersucht werden.

Zentrale Untersuchungsergebnisse

- Für die BO der Anzahl versteckter Neuronen liegt im Fall des RNNs ein lokales Minimum im niedrigen Neuronenbereich vor, wobei für die PGRNNs ein Plateau beobachtet werden kann, das erst für hohe Neuronenzahlen wieder ansteigt.
- Erweiterung um ein Dynamikmodell führte in allen Fällen zu einer deutlichen Reduktion des Simulationsfehlers (Hypothese 1 Abs. 1 S. 1)
- Bereits Verwendung eines linearen Dynamikmodells führte zu Reduktion um fast 70%
- Exponent des Dynamikmodells hat direkten Einfluss auf die Güte des resultierenden PGRNNs, während dies für die Modellparametrierung nicht beobachtet werden konnte (Hypothese 1 Abs. 1 S. 2, Abs. 2)

4.3.2 Lorenz-Attraktor

Bei dem Lorenz-System handelt es sich um ein chaotisches System. Dies drückt sich in einer starken Sensitivität bzgl. der Änderung der Modellparameter und Anfangsbedingungen aus. So führt die Änderung der Parametrierung des Lorenz-Systems um -10% bereits für eine Simulationsdauer von nur 1s zu stark divergierenden Trajektorien (siehe Abb. 4-7). Bei einer Änderung der Parameter um -25% stellen sich völlig verschiedene Trajektorien ein.

Das Vorgehen erfolgt analog zu dem der Untersuchung des Viertelfahrzeugs. Dabei werden für die betrachteten PGRNNs die Modellparameter a, b und c des physikalischen Modells variiert. Die Ergebnisse (Abb. 4-8) verdeutlichen erneut, dass die PGRNNs dem RNN deutlich überlegen sind. Dies ist analog zu den Ergebnissen am Viertelfahrzeug. Die geringste beobachtete Reduktion des Simulationsfehlers ggü. dem RNN beläuft sich dabei immer noch auf mehr als 75% (Hypothese 1 Abs. 1 S. 1).

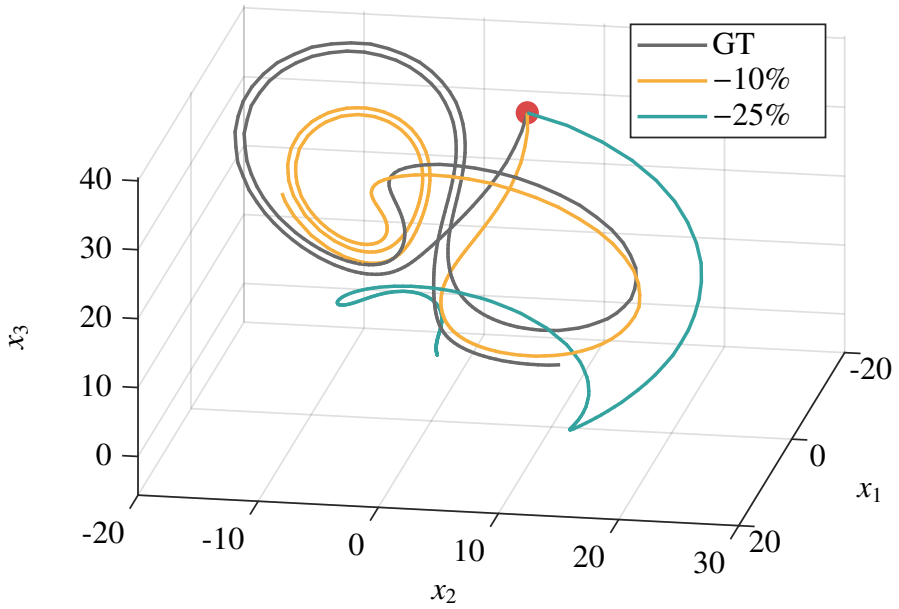

Bild 4-7: *Da es sich bei dem Lorenz-Attraktor um ein chaotisches System handelt, divergieren die Trajektorien für Variation der Modellparameter um −10% und −25% sehr stark vom GT. Die abgebildeten Trajektorien entsprechen einer Simulation des Systems von jeweils 1 Sekunde.*

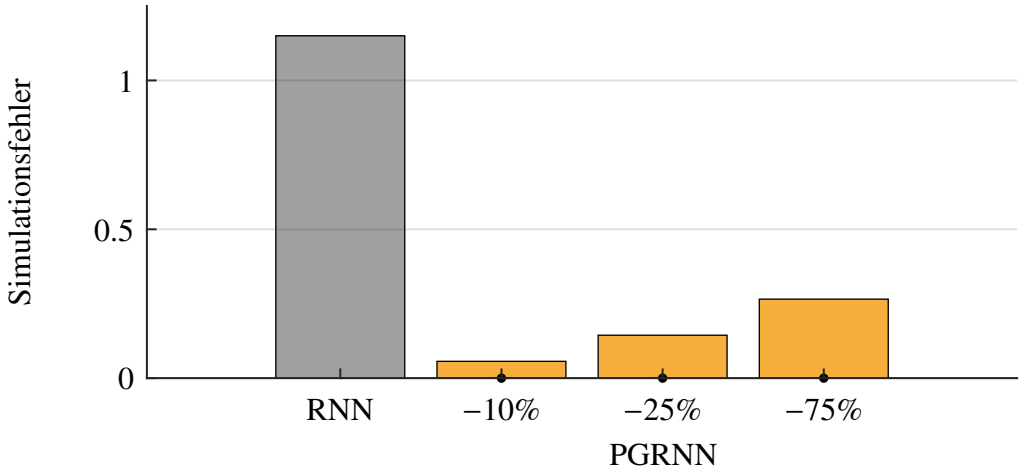

Bild 4-8: *Auch für das Lorenz-System konnte durch Erweiterung um ein physikalisches Dynamikmodell eine drastische Reduktion des Simulationsfehlers festgehalten werden. Im Gegensatz zum Viertelfahrzeug zeigt sich zudem eine Korrelation zwischen Parametergüte und Güte des resultierenden PGRNN.*

Darüber hinaus kann eine deutliche Korrelation zwischen steigender Parametergüte und Modellgüte beobachtet werden. Hatte die Parametrierung im Fall des Viertelfahrzeugs kaum einen Einfluss auf die Modellgüte, ist für das sensitive Lorenz-System ein Zusammenhang festzuhalten (Hypothese 1 Abs. 1 S. 2, Abs. 2). Dennoch kann nicht davon

gesprochen werden, dass eine ausreichend genaue Parametrierung des Dynamikmodells für eine Reduktion des Fehlers essentiell ist (Hypothese 1 Abs. 3).

Da das Fehlerlevel des RNNs vergleichsweise hoch ist – aufgrund der chaotischen Beschaffenheit des Lorenz-Systems –, können unter Hinzunahme eines physikalischen Modells (PGRNN) deutlich Erfolge bzgl. der Simulation erzielt werden (siehe Abb. 4-9). Dabei kann erst durch Einsatz eines PGRNN ein akzeptables Simulationsverhalten erreicht werden. Im Falle des Lorenz-Attraktors kann daher von einer deutlichen Überlegenheit des PGRNNs ggü. dem RNN gesprochen werden. Die Reduktion des RNN-Fehlers beträgt mit einer um -10% abweichenden Parametrierung 95%. Für eine Parameterabweichung von -25% sind es immerhin 87%.

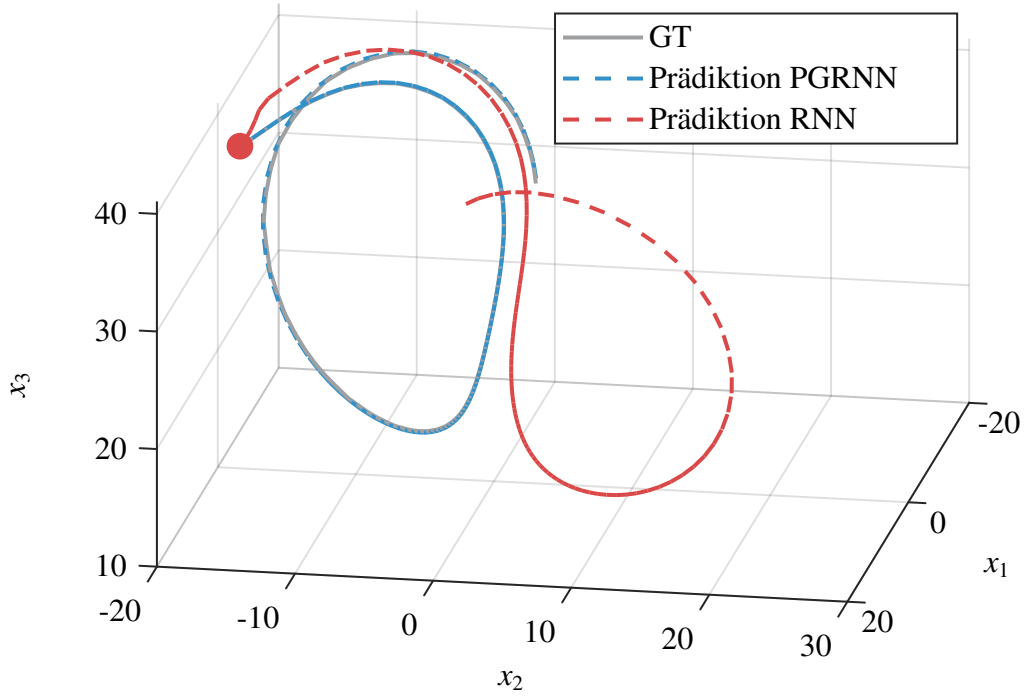

Bild 4-9: *Die PGRNN-Modelle weisen erheblich geringere Simulationsfehler als das als Referenz dienende RNN auf. Bei den hier abgebildeten Testtrajektorien handelt es sich um einen Extremfall, an dem deutlich wird, dass bereits infolge geringer initialer Abweichungen aufgrund der chaotischen Beschaffenheit des Systems völlig unterschiedliche Funktionsverläufe resultieren. Für das dargestellte Beispiel wurde Modellvariante PGRNN/−10% verwendet.*

Unter Berücksichtigung der Untersuchungsergebnisse am Viertelfahrzeug kann darauf geschlossen werden, dass bei zu modellierenden, parametersensitiven Systemen eine Korrelation zwischen Parametergüte und resultierender Netzgüte zu erwarten ist. Die Diskrepanz zwischen den einzelnen PGRNN-Varianten ist im Vergleich zur Reduktion des Fehlers ggü. des RNNs sehr gering. Da es sich bei dem Lorenz-Attraktor um ein chaotisches System handelt, ist zu erwarten, dass die Reduktion des Simulationsfehlers für längere Simulationsdauern stärker von der Güte des verwendeten physikalischen Modells abhängt. Entsprechende Untersuchungen werden in Kapitel 4.5 durchgeführt.

Zentrale Untersuchungsergebnisse

- Deutliche Überlegenheit des PGRNNs ggü. RNN (Hypothese 1 Abs. 1 S. 1)
- Korrelation zwischen Güte des verwendeten Dynamikmodells und der Güte des resultierenden PGRNNs (Hypothese 1 Abs. 1. S. 2, Abs. 2)
- Selbst für sehr hohe Parameterabweichungen starke Reduktion des Simulationsfehlers (Hypothese 1 Abs. 3)
- Einsatz eines Dynamikmodells mit Parameterabweichung von -10% führte zu Fehlerreduktion von 95% ggü. RNN

4.4 Untersuchung: Trainingsdaten

In diesem Kapitel soll untersucht werden, welchen Einfluss die Menge und Gestalt der Trainingsdaten auf die resultierenden Netze hat. Dabei definiert sich die Datengestalt durch die Länge der Datensequenzen sowie der Art von Informationen, die diese enthalten. So können Trainingsdaten vornehmlich dem Bereich des Einschwingvorgangs entnommen werden, aber auch zu weiten Teilen periodische Oszillationen im eingeschwungenen Zustand enthalten.

4.4.1 Dateneffizienz

Im Rahmen aller anderen Untersuchungen wurde jeweils die Menge verwendeter Trainingsdaten konstant gehalten und somit als Einflussfaktor eliminiert. Im Rahmen dieses Kapitels soll untersucht werden, welchen Einfluss die Verminderung bzw. Steigerung der Trainingsdatenmenge auf die resultierende Netzgüte hat. Dazu wurden Netze mit 2, 5, 10 und 15 Datensequenzen mit einer Länge von jeweils 1s zur Modellierung des Viertelfahrzeugs trainiert. Zum Vergleich: In allen anderen Untersuchungen am Viertelfahrzeug wurde stets eine Datenmenge von 10 Datensequenzen à 1s verwendet. Eine äquivalente Datenmenge setzt sich aus N Samples zusammen und ergibt sich als Produkt der Sequenzanzahl N_{Seq}, -länge T und der verwendeten Zeitschrittweite ΔT:

$$N = N_{Seq} \cdot T \cdot \frac{1}{\Delta T} \tag{4-2}$$

Infolge der Erkenntnisse aus Kapitel 4.3.1 werden im Folgenden jeweils PGRNNs mit GT-Parametrierung sowie für alle drei Exponenten betrachtet.

In Abbildung 4-10 ist der Simulationsfehler für die verschiedenen Datenmengen aufgetragen. Es fällt auf, dass die Halbierung der Trainingsdatenmenge von 10 zu 5 Datensequenzen für keine der Netzvarianten zu einem nennenswerten Anstieg des Simulationsfehlers führt. Daher soll im Fortlauf angenommen werden, dass die verwendete Trainingsdatenmenge von $N = 10 \cdot 1\text{s} \cdot \frac{1}{10^{-3}\text{s}} = 10^4$ ausreichend ist.

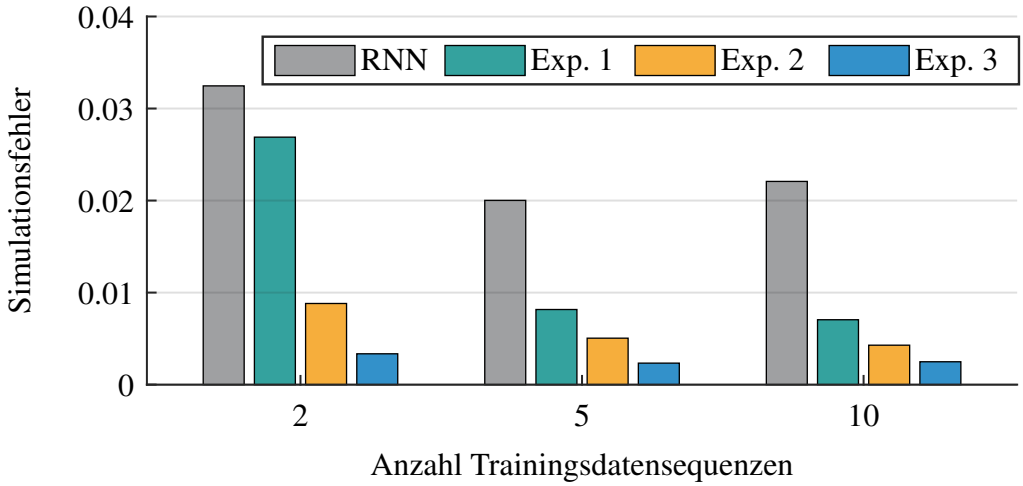

Bild 4-10: *Die Halbierung der Trainingsdatenmenge von 10 auf 5 Datensequenzen führt zu keinem Anstieg des Simulationsfehlers. Erst bei einer Absenkung der Trainingsdatenmenge auf 2 Sequenzen à* 1s *ist ein deutlicher Fehleranstieg zu verzeichnen.*

Für eine Verringerung der Trainingsdatenmenge zu lediglich 2 Sequenzen ist jedoch ein nennenswerter Anstieg des Simulationsfehlers zu verzeichnen. Fällt dieser für RNN, PGRNN/Exp.2 und PGRNN/Exp.3 mit 62%, 75% und 43% sogar noch vergleichsweise moderat aus, beträgt er für das PGRNN mit linearem physikalischen Modell fast 230%. Eine derart geringe Menge an Trainingsdaten sollte daher vermieden werden. Da eine größere Menge an Trainingsdaten zu einer längeren Trainingsdauer führt und demnach die Menge verwendeter Trainingsdaten einen Kompromiss zwischen Netzgüte und Trainingsdauer darstellt, soll im Folgenden an der verwendeten Menge an Trainingsdaten festgehalten werden. Ebenfalls kann aber auch abgeleitet werden, dass die Modelle bei der verwendeten Trainingsmenge nicht besonders sensibel ggü. Änderungen der Datenmenge reagieren.

Da es als allgemein realistisch angenommen werden kann, dass 5-10 Datensequenzen einer Länge von 1 Sekunde zum Training verfügbar sind, ist die Vorschaltung eines Pre-Trainings (siehe Kap. 3.6) mithilfe des verwendeten physikalischen Dynamikmodells an dieser Stelle nicht weiter relevant. Im Fall geringer Datenverfügbarkeit kann jedoch in Erwägung gezogen werden, ein entsprechendes Training mit synthetischen Daten dem Training mit Realdaten voranzustellen.

Zentrale Untersuchungsergebnisse

- Trainingsdatenmenge von 5 Sequenzen à 1s bei einer Zeitschrittweite von 10^{-3} ist bereits ausreichend
- Kein Anstieg des Simulationsfehlers infolge einer Halbierung der Datenmenge von 10 zu 5 Sequenzen beobachtbar

4.4.2 Datensequenzlänge

Neben der Anzahl an Trainingsdatensequenzen setzt sich die Trainingsdatenmenge nach Gleichung (4-2) aus der Länge der Sequenzen sowie der verwendeten Zeitschrittweite zusammen. Letztere ist abhängig von dem zu modellierenden System und somit festgesetzt. Welchen Einfluss die Sequenzlänge der Trainingsdaten auf die Güte der trainierten Netze hat, soll in diesem Kapitel untersucht werden. Dazu werden vier unterschiedliche Sequenzlängen zwischen 0.05 und 1 Sekunde unterschieden.

Der Einschwingvorgang beschränkt sich hauptsächlich auf die ersten 0.05 Sekunden (siehe Abb. 4-11), entsprechend kurze Sequenzen liefern kaum eine ausreichende Datenbasis über den durch die Netzeingaben aufgespannten Zustandsraum. Infolge der verwendeten sinusoidalen Anregung von 20Hz wird der volle Wertebereich der Stellgrößen in (2π) / (20Hz) $\approx$ 0.314s durchlaufen. Ebenso ergibt sich eine Periodizität der Zustandstrajektorien. Es gilt nun zu untersuchen, welche Sequenzlänge, bei gleichbleibender Trainingsdatenmenge (nach Gl. (4-2)) und Trainingseinstellungen, den Simulationsfehler minimiert.

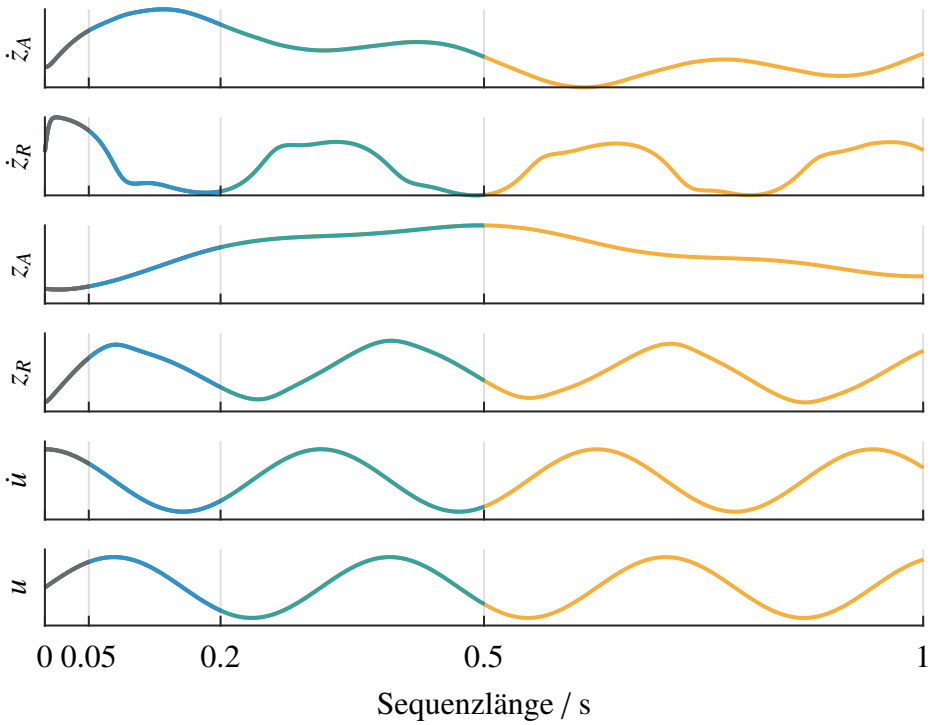

Bild 4-11: *Es werden vier Längen von Trainingsdatensequenzen unterschieden. Dabei beschränkt sich die kürzeste Variante auf den Übergangsbereich. Sequenzen mit einer Länge von mehr als ca.* 0.314s *enthalten repetitive Informationen.*

Vorab kann wiederum festgehalten werden, dass für alle Sequenzlängen die PGRNN-Varianten niedrigere Simulationsfehler als die Referenz-RNNs bieten (siehe Abb. 4-12). Die durchschnittliche Höhe des Simulationsfehlers sowie die Differenz zwischen den unterschiedlichen Netzkonfigurationen ist jedoch stark von der Sequenzlänge der jeweils verwendeten Trainingsdaten abhängig. So fällt der Simulationsfehler aller Netzvarianten

für Sequenzen der Länge 0.05s mit Abstand am schlechtesten aus. Der Referenzsimulationsfehler des RNNs konnte in diese Fall durch Hinzunahme eines physikalischen Modells um wenigstens 27% (Exp. 1) und bis zu 48% (Exp 3.) gesenkt werden. Allerdings sind alle Fehler unter Verwendung der Sequenzen mit Länge 0.05s vergleichsweise hoch. Dies deutet darauf hin, dass Trainingsdaten optimalerweise nicht nur dem Übergangsbereich entstammen sollten.

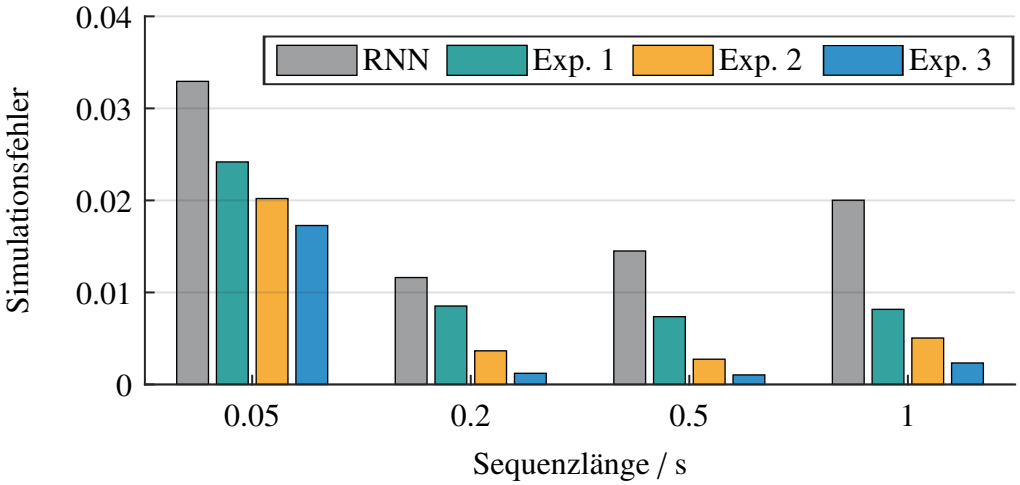

Bild 4-12: *Die für die Trainingsdaten verwendete Sequenzlänge hat direkte Auswirkungen auf die resultierenden Netze. Dabei führen zu kurze Sequenzen, die lediglich Informationen über das Übergangsverhalten enthalten, zu einem erheblich erhöhten Fehlerlevel, wohingegen eine Sequenzlänge von 0.2-0.5 Sekunden zu den geringsten Simulationsfehlern führte.*

Die in dieser Arbeit prädominant verwendete Sequenzlänge von 1s führt bereits zu deutlich besseren Ergebnissen. So konnte der Simulationsfehler des RNNs ggü. Sequenzlänge 0.05s um nahezu 40% gesenkt werden. Für die PGRNNs mit Exponent 1, 2 und 3 beträgt die Reduktion sogar bis zu 86% und jedenfalls 66%. Während der Fehler durch Verwendung der Sequenzlängen 0.5s und 0.2s in etwa genauso hoch ausfällt, kann der Fehler des RNNs weiter gesenkt werden. So kann eine Reduktion um etwa ein Viertel durch Verwendung halbsekündiger Sequenzen sowie eine Reduktion um fast die Hälfte durch Verwendung von Sequenzen der Länge 0.2s verzeichnet werden.

Es kann zusammenfassend gesagt werden, dass die Wahl einer geeigneten Sequenzlänge der Trainingsdaten einen großen Einfluss auf die Güte der resultierenden Netze haben kann. Dabei sollten die Trainingsdaten nicht nur Informationen über den Einschwingvorgang enthalten (Hypothese 2). Darüber hinaus konnte für das RNN festgestellt werden, dass ein repetitiver Verlauf der Trainingssequenzen optimalerweise vermieden werden sollte, da stattdessen andere Daten einen Mehrwert bieten würden. So werden im Fall einer periodischen Anregung optimalerweise Sequenzen mit ungefähr der Periodendauer verwendet. Inwiefern dies allerdings praxistauglich ist, bleibt offen. Um bei der Aufzeichnung von Messdaten den Informationsgehalt der resultierenden Trainingsdaten zu maximieren, könnte beispielsweise, wie es bei anderen Methoden zur Systemidentifikation der

Fall ist, eine Sweep-Anregung statt einer Anregung mit konstanter Frequenz verwendet werden.

Zentrale Untersuchungsergebnisse

- Trainingsdaten sollten möglichst von hoher Varianz sein. Repetitive Verläufe bieten keinen Mehrwert (Hypothese 2)
- Werden alle Trainingssequenzen dem Übergangsbereich entnommen, führt dies zu einem drastischen Anstieg des Fehlers

4.5 Untersuchung: Prädiktionshorizont

In Kapitel 4.3 wurde der Einfluss der Güte des physikalischen Dynamikmodells auf die Güte des resultierenden PGRNNs anhand des Viertelfahrzeugs und des Lorenz-Systems untersucht. Dabei wurden die Netze zur Bestimmung ihrer Güte jeweils für 1 Sekunde simuliert. Da es sich bei dem Lorenz-Attraktor um ein chaotisches System handelt, ist insbesondere interessant, wie sich der Simulationsfehler für unterschiedliche Prädiktionshorizonte verhält.

Zunächst werden dazu wieder wie in Kapitel 4.3.2 beschrieben verschiedene Netzkonfigurationen trainiert. Die dabei verwendete Sequenzlänge der Trainingsdaten beträgt 1s. Zusätzlich zur Reproduktion gleich langer Testsequenzen soll das Langzeitprädiktionsverhalten für 3, 5, 10 und 20 Sekunden untersucht werden (vgl. Abb. 4-13), auch wenn eine derart extensive Prädiktion in der Praxis idR. nicht appliziert wird, z.B. im Rahmen einer MPC [vgl. ACS+21].

Um eine Vergleichbarkeit des Simulationsfehlers für verschiedene Prädiktionshorizonte zu ermöglichen, wird der Simulationsfehler im Folgenden auf einen Prädiktionshorizont von 1s normiert. Dadurch kann für jede Versuchsreihe auf den durchschnittlichen Simulationsfehler und den Einfluss der Simulationsdauer geschlossen werden.

Die Ergebnisse (Abb. 4-14) für alle Modellkonfigurationen zeigen, dass der normierte Simulationsfehler für eine längere Prädiktionsdauer stark ansteigt. So ist der Anstieg für eine Prädiktion von bis zu 5s noch relativ moderat, die Trajektorien für Simulationen von 10 und 20 Sekunden weichen dagegen extrem vom GT ab. Dass der Fehleranstieg für eine Prädiktion von 10 Sekunden ggü. 1 Sekunde im Vergleich zu 20 zu 10 Sekunden erheblich signifikanter ausfällt, ist auf die Tatsache zurückzuführen, dass es sich bei dem Lorenz-System um einen Attraktor handelt und sich somit die Trajektorien in einem stark beschränkten Zustandsraum aufhalten. Das Fehlerniveau ist daher ebenso beschränkt.

Obwohl der Simulationsfehler aller 3 PGRNN-Varianten stets kleiner als der des Referenz-RNNs ausfällt, sinkt dieser Vorsprung mit steigender Prädiktionsdauer. Mit einer Parameterdeviation von -10% fiel der Fehler auf einen Prädiktionshorizont von 1 Sekunde noch mehr als 20-mal kleiner aus, als der des RNNs. Für eine Simulationsdauer von 5 Sekunden beläuft sich dieser Faktor nur noch auf 3.5. Es wird klar, dass selbst mit physikalischem Modell keine exakten Modelle des GT gelernt werden können und die entsprechenden Trajektorien für lange Simulationsdauern divergieren. Dass trotz der chaotischen Charakteristik des Lorenz-Systems selbst für Simulationslängen von 5 Sekunden noch moderate

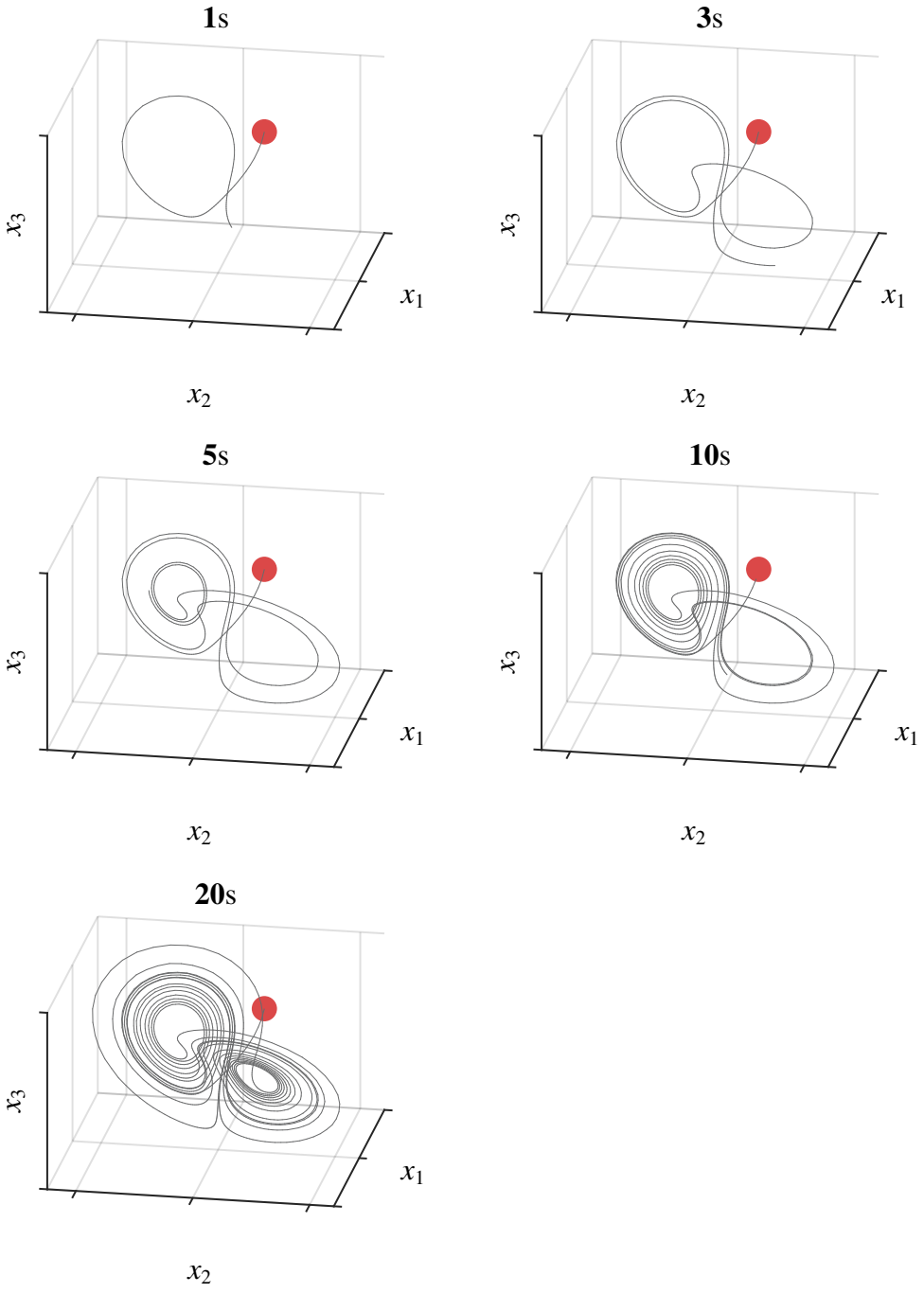

Bild 4-13: *Die Trajektorien des Lorenz-Attraktors oszillieren in chaotischer Weise in einem beschränkten Bereich des Zustandsraums. Dabei können für eine Prädiktionsdauer von 1, 3, 5, 10 und 20 Sekunden stets verschiedene Trajektorienverläufe beobachtet werden.*

Ergebnisse erzielt werden konnten, spricht in jedem Fall für die trainierten Netze. Für Langzeitprädiktionen chaotischer Systeme ist daher die Verwendung eines PGRNNs definitiv von Vorteil (Hypothese 1).

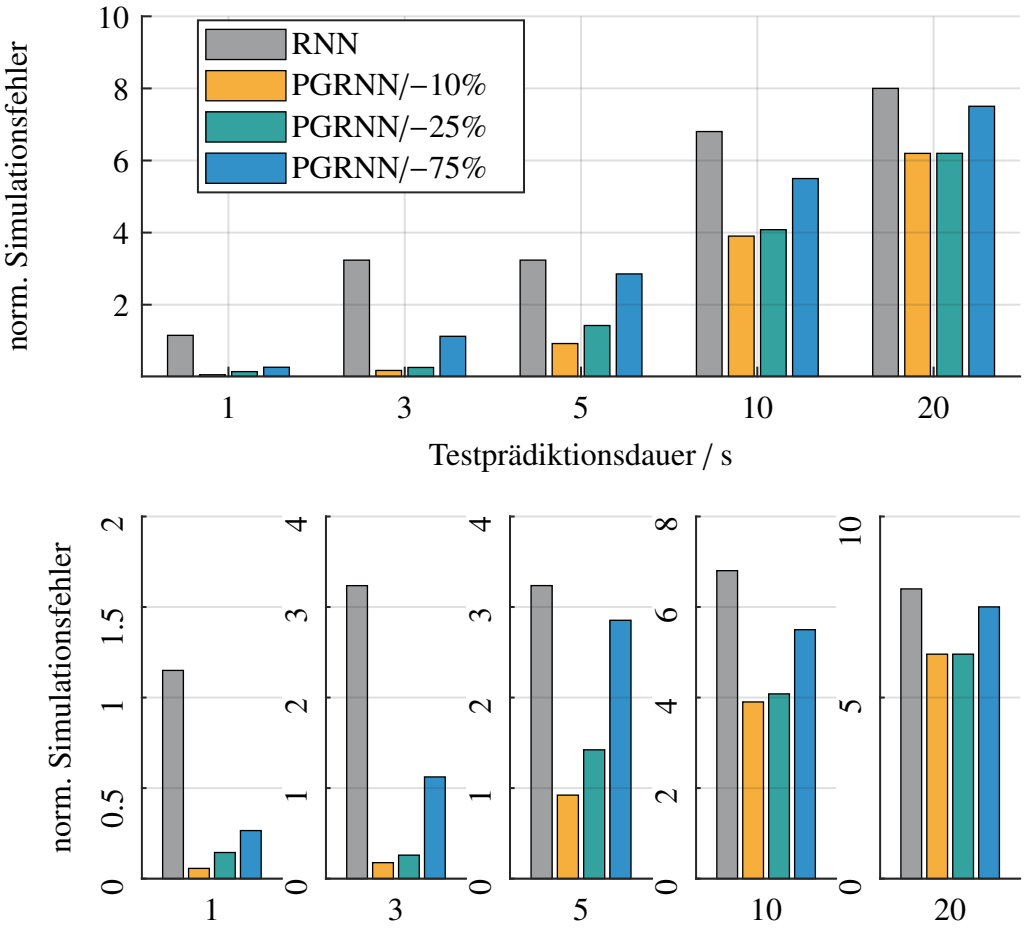

Bild 4-14: *Es ist der normierte Simulationsfehler der unterschiedlichen Netzvarianten für verschiedene Prädiktionsdauern aufgezeigt. Dabei enthalten das obere und die unteren Diagramme jeweils die gleichen Informationen und unterscheiden sich lediglich in der Skalierung der y-Achse. Es wird deutlich, dass der normierte Simulationsfehler für den Lorenz-Attraktor über einen höheren Prädiktionshorizont ansteigt, wobei auch der Vorsprung der PGRNN-Varianten mit steigender Prädiktionsdauer abnimmt.*

Wie bei den Untersuchungen für eine Simulationsdauer von 1s (Kap. 4.3.2) kann auch für längere Prädiktionshorizonte ein starker Zusammenhang zwischen Reduktion des Fehlers und Güte der Parametrierung des physikalischen Dynamikmodells observiert werden. Für alle Prädiktionshorizonte wächst mit der Güte der verwendeten Parametrierung auch die Güte des resultierenden Netzes. Dabei fällt auf, dass die Netze mit −10%-iger und −25%-iger Parametergüte gerade für längere Prädiktionen beinah gleich gute Ergebnisse bereiten. Der Simulationsfehler des PGRNN/−75% liegt hingegen ab einer Simulationsdauer

von 5 Sekunden in etwa auf einem Niveau mit dem RNN. Lediglich für kurze Dauern ist hier ein drastischer Kontrast festzustellen. Die Güte des verwendeten Dynamikmodells scheint daher für chaotische Systeme besonders für lange Prädiktionshorizonte hinreichend genau ($\approx 25\%$ oder genauer) sein zu müssen (Hypothese 1 Abs. 3).

Zusammenfassend lässt sich sagen, dass die Hinzunahme eines physikalischen Dynamikmodells auch für länge Prädiktionshorizonte zu einer Reduktion des Simulationsfehlers geführt hat. Dabei wird dieser Vorteil jedoch für längere Simulationen immer kleiner. Da in den meisten Kontexten, z.B. in Kombination mit einer MPC, entsprechende Modelle in einer geschlossenen Schleife eingesetzt werden werden und die Prädiktionen zu wiederkehrenden Zeitpunkten mit entsprechenden Messungen abgeglichen werden, sollte ein geeigneter Prädiktionshorizont $\leq$ 5s gewählt werden. Darüber hinaus hat sich herausgestellt, dass ein PGRNN mit einer Parametergüte von bereits -25% zu deutlich besseren Ergebnissen zum RNN geführt hat. Für 5 und 10 Sekunden Prädiktion beläuft sich die Reduktion dabei auf 2.27 und 1.7 ggü. dem RNN.

Zentrale Untersuchungsergebnisse

- Die Überlegenheit des PGRNNs zeigt sich auch für längere Prädiktionshorizonte, wobei der Vorsprung für steigende Simulationsdauern abnimmt.
- Mit einem PGRNN mit Parametergüte von -25% können über einen Prädiktionshorizont von 5 Sekunden gute Ergebnisse erzielt werden. Für Langzeitprädiktionen chaotischer Systeme scheint eine ausreichend genaue Parametrierung des Dynamikmodells wichtig zu sein (Hypothese 1 Abs. 3).

4.6 Untersuchung: Erweiterung um eine Funktionsbibliothek

Im Rahmen der Untersuchungen in Kapitel 4.3 und 4.5 wurde herausgestellt, dass die Güte des verwendeten Modells einen großen Einfluss auf die Güte des resultierenden PGRNNs haben kann. Insbesondere gilt dies für die verwendeten Nichtlinearitäten. Dies konnte am Beispiel des Viertelfahrzeugs gezeigt werden. Daher soll nun weitergehend untersucht werden, ob das schiere „Zur-Verfügung-Stellen“ der korrekten Nichtlinearitäten zu einer ähnlichen Reduktion des Simulationsfehlers führt.

4.6.1 Lorenz-Attraktor

Die nichtlinearen Terme werden, wie in Kapitel 3.8 beschrieben, in Form einer Funktionsbibliothek dem Netz durch einen zusätzlichen Netzeingang zur Verfügung gestellt. In der Realität würde diese Bibliothek aus diversen nichtlinearen Termen bestehen, da die korrekten Terme nicht bekannt sind und sonst auch vermutlich im Rahmen des physikalischen Modells verwendet würden. Gerade bei den Untersuchungen des Lorenz-Attraktors konnte ein starker Zusammenhang zwischen Modellgüte und resultierender Netzgüte beobachtet werden. Daher soll im Folgenden in Erweiterung von Hypothese 1 untersucht werden, ob das Hinzufügen einer Funktionsbibliothek der Lorenz-Nichtlinearitäten zu einer Reduktion des Simulationsfehlers führt.

Dazu wird zunächst der Netzeingang um einen dritten Eingang erweitert (vgl. Kap. 3.8). Die verwendete Funktionsbibliothek besteht aus den beiden nichtlinearen Termen des Lorenz-Systems:

$$\underline{\theta}\,(\underline{x}_k) := \begin{bmatrix} x_{k,1}\,x_{k,3} \\ x_{k,1}\,x_{k,2} \end{bmatrix}$$

Es werden erneut Netzkonfigurationen mit (PGRNN) und ohne (RNN) physikalischem Modell verglichen, wobei das „+“ hinter der Bezeichnung die Erweiterung um die beschriebene Funktionsbibliothek denotiert. Die Untersuchungsergebnisse sind in Abbildung 4-15 dargestellt.

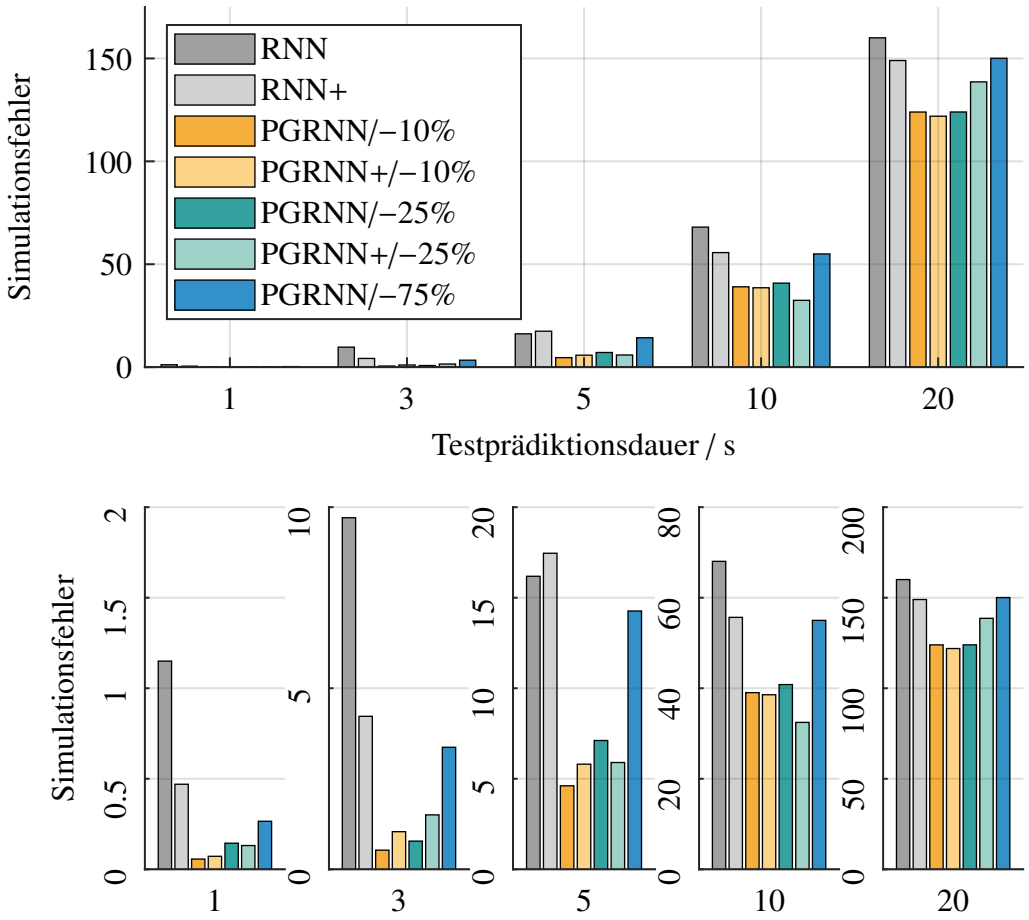

Bild 4-15: *Die Diagramme oben und unten enthalten erneut dieselben Informationen, wobei lediglich unterschiedlich skalierte y-Achsen vorliegen. Durch die Einführung der Funktionsbibliothek kann für den Lorenz-Attraktor keine weitere Reduktion des Simulationsfehlers verzeichnet werden.*

Es fällt zunächst auf, dass für die Netze mit Bibliothek der gleiche Zusammenhang zwischen Prädiktionsdauer und Simulationsfehler beobachtet werden kann, wie für die Netze

ohne Bibliothek. Zudem ist für die PGRNN+-Varianten keine weitere Fehlerreduktion festzustellen. Im Gegenteil führt die Erweiterung um die Funktionsbibliothek in vielen Fällen sogar zu einem geringfügigen Anstieg. Nur für die Netze mit −25% Abweichung kann für Simulationslängen von 5 und 10s eine messbare Fehlerreduktion verzeichnet werden. Aufgrund statistischer Schwankungen ist dies allerdings zu vernachlässigen.

Für die RNNe konnte beinah für alle Simulationslängen eine Reduktion des Fehlers infolge der zusätzlichen Bibliothek beobachtet werden, wobei diese für eine Dauer von 1 und 3 Sekunden maßgeblich ist und einer Reduktion von mehr als 100% entspricht. Dabei liegt das Fehlerniveau des RNN+ allerdings in allen Fällen weiterhin über dem der PGRNN-Varianten und liegt nur für sehr lange Simulationsdauern in etwa auf einem Niveau mit dem PGRNN mit der geringsten Parametergüte (PGRNN/−75%). Die Verwendung eines physikalischen Modells, selbst mit geringer Parametrierungsgüte, ist demnach der alleinigen Erweiterung um die korrekten isolierten nichtlinearen Terme in Form einer Funktionsbibliothek vorzuziehen.

Dass die Erweiterung der PGRNNs um die Funktionsbibliothek zu keiner weiteren Reduktion des Simulationsfehlers geführt hat, kann möglicherweise anhand zweier Ansätze begründet werden. Der erste und weniger ausschlaggebende Ansatz beruht auf der Wahl der Fehlerfunktion. Wie bereits in Kapitel 3.8 beschrieben, ist für das Erlernen dünnbesetzter Gewichtsmatrizen MAE MSE vorzuziehen. Die Dünnbesetztheit hat sich im Rahmen von SINDY – auf dem der Ansatz der verwendeten Funktionsbibliothek basiert – als äußerst wichtig herausgestellt, auch wenn dieser Faktor erst für höherbesetzte Funktionsbibliotheken an Relevanz gewinnt.

Der zweite und ausschlaggebende Faktor liegt in der Beschaffenheit des physikalischen Modells begründet. Dabei kann durch Wahl geeigneter Gewichte bereits aus dem physikalischen Modell auf die korrekten Nichtlinearitäten geschlossen werden. Entspricht $\underline{\tilde{x}}_k$ dem Ausgang des physikalischen Modells, so können mit den Gewichtsmatrizen $W_{\underline{x}}^T$ und $W_{\underline{\tilde{x}}}^T$ sowie dem Bias $\underline{b}$ die nichtlinearen Terme der Funktionsbibliothek isoliert werden:

$$
\begin{aligned}
W_{\underline{x}}^T \underline{x}_k + W_{\underline{\tilde{x}}}^T \underline{\tilde{x}}_k + \underline{b} &= \begin{bmatrix} b & -1 & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} x_{k,1} \\ x_{k,2} \\ x_{k,3} \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a\,(x_{k,2} - x_{k,1}) \\ x_{k,1}\,(b - x_{k,3}) - x_{k,2} \\ x_{k,1} \cdot x_{k,2} - c x_{k,3} \end{bmatrix} + \underline{0} \\
&= \begin{bmatrix} x_{k,1} x_{k,3} \\ x_{k,1} x_{k,2} \end{bmatrix}, \text{ qed.}
\end{aligned}
$$

Auch wenn die Terme in der Funktionsbibliothek in isolierter Form vorliegen und diese aus dem physikalischen Modell durch geeignete Anpassung der Gewichte erst extrahiert werden müssen, wurde durch Inkorporation der Bibliothek im Grunde kein zusätzlicher Informationsgehalt hinzugefügt.

Zentrale Untersuchungsergebnisse

- Durch Erweiterung des RNN um eine optimal besetzte Funktionsbibliothek konnte der Simulationsfehler auf einen Prädiktionshorizont von bis zu 3 Sekunden um mehr als 100% reduziert werden.
- Für PGRNNs: Können die korrekten nichtlinearen Terme durch geeignete Wahl der Netzgewichte aus dem physikalischen Modell isoliert werden, bietet eine Funktionsbibliothek der entsprechenden Terme keinen Mehrwert.

4.6.2 Viertelfahrzeug

Für das Viertelfahrzeug wurde ebenfalls ein starker Zusammenhang zwischen Modellgüte und resultierender Netzgüte beobachtet, insbesondere für variierende Exponenten α. Liegt der korrekte Exponent im physikalischen Modell vor (Exp. 3), so bietet die Bereitstellung einer Funktionsbibliothek hier erneut keinen Mehrwert. Für die PGRNNs deren Modelle vom GT abweichende Exponenten 1 und 2 beinhalten, kann infolge einer Erweiterung um eine Bibliothek zusätzlich der korrekte Exponent entnommen werden. Die Funktionsbibliothek besteht daher im Rahmen der nächsten Untersuchungen am Viertelfahrzeugs aus nur einem Term:

$$\underline{\theta}\left(\underline{x}_k\right) \equiv \underline{\theta}\left(\dot{z}_{A,k}, \dot{z}_{R,k}, z_{A,k}, z_{R,k}\right) := \left(\dot{z}_{A,k} - \dot{z}_{R,k}\right)^3$$

Dabei entspricht der Exponent 3 genau dem GT-Exponenten.

Im Folgenden wurden erneut Netze mit und ohne zusätzliche Funktionsbibliothek trainiert, wobei die PGRNNs jeweils über eine Parametrierungsgüte von -10% verfügen. Die Ergebnisse sind in Abbildung 4-16 dargestellt. Dabei fällt zunächst auf, dass die Verwendung einer Funktionsbibliothek i.A. zu einer Reduktion des Fehlers führt, wobei das Ausmaß der Fehlerreduktion mit steigender Modellgüte abnimmt. Im Fall des RNN konnte so eine Reduktion um ein Drittel beobachtet werden und auch für das PGRNN mit linearem physikalischen Modell konnte der Fehler mehr als halbiert werden. Für das PGRNN mit Exponent 2 beträgt diese Reduktion nur noch 27% und im Falle des PGRNN/Exp. 3 konnte keine weitere Verbesserung erreicht werden.

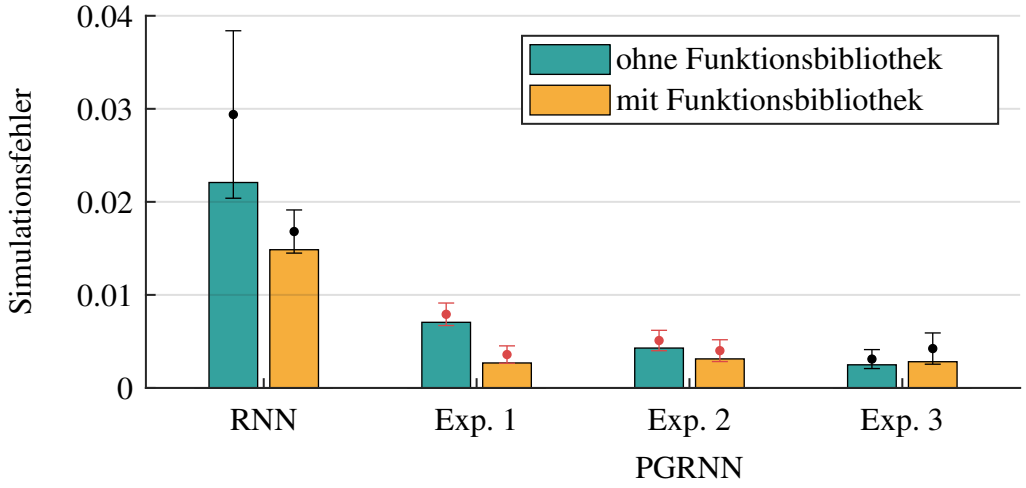

Bild 4-16: *Durch die Erweiterung um eine Funktionsbibliothek kann der Simulationsfehler für das Viertelfahrzeug weiter gesenkt werden. Damit liegen alle PGRNN-Varianten mit Funktionsbibliothek auf ungefähr einem Niveau. Die variierende Modellgüte infolge der verschiedenen Exponenten kann demnach durch die Verwendung einer Funktionsbibliothek kompensiert werden.*

Erneut können die Ergebnisse auf die gleichen beiden Faktoren wie im Fall des Lorenz-Attraktors zurückgeführt werden. Dabei hat sich allerdings im Rahmen des Viertelfahrzeugs MAE als am besten geeignete Fehlerfunktion herausgestellt. Bzgl. des Exponenten

ist deutlich zu beobachten, dass durch Erweiterung um eine adäquate Funktionsbibliothek die Diskrepanz bei Verwendung schlechterer physikalischer Modelle (z.B. eines linearen Modells) verkleinert oder gar vollständig kompensiert werden kann. So ist ein lineares physikalisches Modell mit Erweiterung um eine Funktionsbibliothek, die den korrekten nichtlinearen Term enthält, auf eine Ebene mit der Verwendung eines physikalischen Modells hoher Güte zu stellen.

Zentrale Untersuchungsergebnisse

- Wird ein PGRNN um eine Funktionsbibliothek erweitert, die den korrekten nichtlinearen Term enthält, wird dadurch die Diskrepanz eines Dynamikmodells mit falschem Exponenten vollständig kompensiert.
- Ein PGRNN mit linearem physikalischen Dynamikmodell und optimal besetzter Funktionsbibliothek ist mit einem PGRNN mit einem Dynamikmodell hoher Güte gleichzustellen (Hypothese 1 Abs. 1 S. 2, Abs. 2).

4.7 Untersuchung: Erweiterung um Energieerhaltung

Ein Ziel von PGNNs ist die datenbasierte Generierung von Modellen, die physikalisch interpretierbar sind und die Einhaltung dezidierter Gesetzmäßigkeiten gewährleisten. Dazu soll am Beispiel des Viertelfahrzeugs die Energieerhaltungsbedingung aus Kapitel 3.7 als Nebenbedingung eingebunden werden. Wie dort beschrieben, wird der Energieterm als zusätzlicher Strafterm zur Fehlerfunktion hinzugefügt. Die Bestimmung des zugehörigen Laplace-Multiplikators erfolgt durch BO. Es wird in diesem Fall sowohl die Anzahl versteckter Neuronen, als auch der Eintrag des energiebasierten Strafterms optimiert. Die resultierende, optimale Hyperparametrierung korrespondiert dabei mit eben der beobachteten Netzinstanz, die die höchste Netzgüte aufweist. Es wird also jeweils genau die Gewichtung des Strafterms gewählt, die zur Optimierung der Netzgüte führt. Ist der zusätzliche Strafterm destruktiv für das Erreichen eines niedrigen Simulationsfehlers infolge des Trainings, so wird ein entsprechender Laplace-Multiplikator ≈ 0 gewählt.

Wie zuvor werden PGRNNs für unterschiedliche Exponenten des physikalischen Modells trainiert. Die Abweichung der Parametrierung vom GT beträgt dabei in allen Fällen -10%. Um die Konditionierung des energieerhaltungsbasierten Strafterms zu verbessern, wurden die unstetigen Betragsfunktionen durch glatte Betragsfunktionen ersetzt:

$$\text{smoothabs}(x) := \sqrt{x^2 + \beta} \qquad \text{(glatte Betragsfunktion)}$$

Hierbei wurde im Rahmen der Untersuchungen am Viertelfahrzeug ein Glättungsfaktor β von 0.1 gewählt. Diese Modifikation hat keine direkten Auswirkungen auf Werte resultierender Netzgüten und dient lediglich der Konditionierung des Optimierungsproblems.

Bzgl. der BO des Laplace-Multiplikators λ_{EC} kann eine Art umschaltendes Verhalten beobachtet werden (siehe Abb. 4-17). Allerdings findet dieses „Umschalten“ für die betrachtete Nebenbedingung bereits für sehr geringe Werte von λ_{EC} statt. Dies deutet darauf hin, dass die Einbringung des zusätzlichen Fehlerterms vornehmlich negativ zur Konditionierung des Trainings beiträgt und nicht, wie vermutet, zu einer schnelleren Konvergenz

führt. Dennoch wurde infolge der BO in vielen Fällen ein relativ niedriges $\lambda_{EC} > 0$ bestimmt (siehe Abb. 4-18). Für Modellvariante PGRNN/−10%/Exp.2 gilt sogar $\lambda_{EC} \approx 0$.

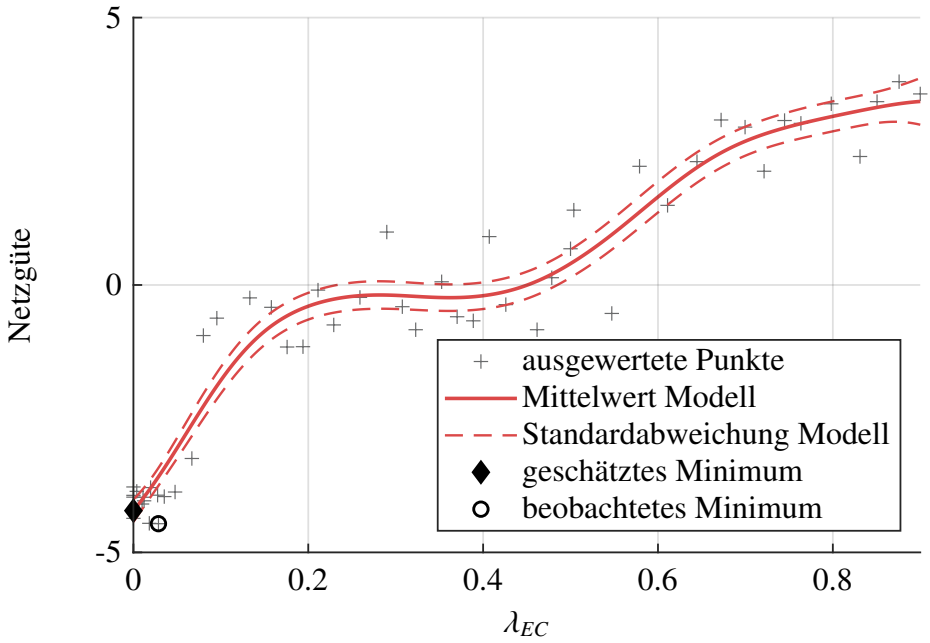

Bild 4-17: *Abgebildet ist die BO des Laplace-Multiplikators des energiebasierten Strafterms für die Modellvariante PGRNN/−10%/Exp.1. Es ist ein „Umschalten“ zu erkennen. Bereits bei geringem Eintrag der energieerhaltenden Bedingung kommt es zu einer starken Verschlechterung der Netzgüte* (4-1).

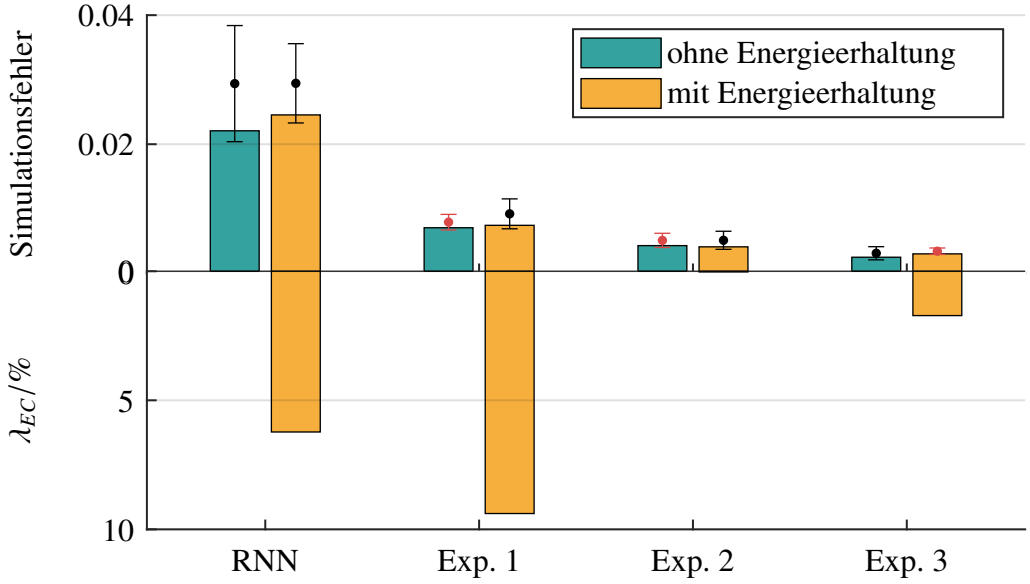

Bild 4-18: *Bei Hinzunahme des energieerhaltenden Strafterms wird der Laplace-Multiplikator infolge der BO durchweg sehr niedrig gewählt. Es konnte dabei keine Reduktion des Simulationsfehlers beobachtet werden.*

Für keines der trainierten PGRNNs mit Energieerhaltung konnte eine Reduktion des Simulationsfehlers verzeichnet werden (siehe Abb. 4-18) (Hypothese 3). Gleiches gilt für die respektiven Trainingsdauern. Allerdings führte der zusätzliche Strafterm für die bestimmten Laplace-Multiplikatoren ebenfalls zu keiner signifikanten Erhöhung des Simulationsfehlers. Stattdessen fällt auf, dass für das RNN bei einem relativ hohen Wert von λ_{EC} eine Senkung der Standardabweichung des Fehlers auftritt. Da bei dieser Netzvariante noch kein physikalisches Modell vorliegt, scheint die Einführung einer physikalischen Nebenbedingung zu einer marginalen Beschränkung des Suchraums zu führen.

Da das theoretische Optimum des Optimierungsproblems (möglicherweise liegen sogar mehrere Optima vor) der Modellierung des GT-Systems entspricht und dieses die Energiebedingung erfüllt, weist die Degradation des Konvergenzverhaltens infolge der Einführung des zusätzlichen Strafterms auf eine schlechtere Konditionierung des allgemeinen Optimierungsproblems hin. Dieser Umstand könnte darauf zurückzuführen sein, dass die untersuchte Nebenbedingung in dieser Form nicht geeignet ist bzw. keinen Mehrwert bietet. Für die Modellierung komplexerer Systeme könnte eine geeignete physikalische Nebenbedingung allerdings erheblich zu einer Beschränkung des Optimierungsraums beitragen.

Zentrale Untersuchungsergebnisse

- Infolge der Einführung des energiebasierten Strafterms konnte keine Reduktion des Simulationsfehlers sowie i.A. keine positive Beeinflussung beobachtet werden (Hypothese 3).
- Zur Modellierung komplexerer Systeme könnten ähnliche Nebenbedingungen dennoch zu einer Beschränkung des Suchraums beitragen.

4.8 Untersuchung: Identifikation eines geregelten Systems

Bisher wurden lediglich autonome sowie gesteuerte Systeme modelliert. Am Beispiel des Cubli soll nun ein geregeltes System identifiziert werden. Dazu wurde für den Cubli eine Zustandsrückführung mittels Riccati-Regler (LQR) entworfen. Aus Gründen der Simplifizierung wurde zum Entwurf der Reglermatrix das GT-System zugrunde gelegt.

Die Reglermatrix $\underline{K}$, sodass $u(t) = -\underline{K}\,\underline{x}(t)$, minimiert im Rahmen des LQRs die quadratische Gütefunktion

$$J(\underline{u}) = \int_0^\infty \left(\underline{x}^T \underline{Q}\underline{x} + uRu + \underline{x}^T \underline{N}u\right) \mathrm{d}t.$$

Es wurden dabei folgende Gewichtungsmatrizen verwendet:

$$\underline{Q} = \mathrm{diag}\,(100, 10, 1, 1)$$
$$R = 1000$$
$$\underline{N} = \underline{0}$$

Die Reglermatrix lässt sich damit in geschlossener Form berechnen, wobei sich $\underline{S}$ durch Lösung von Gleichung (4-3) ergibt:

$$\underline{K} = R^{-1}\left(\underline{B}^T\underline{S} + \underline{N}^T\right)$$
$$\underline{A}^T\underline{S} + \underline{S}\,\underline{A} - (\underline{S}\,\underline{B} + \underline{N})\,R^{-1}\left(\underline{B}^T\underline{S} + \underline{N}^T\right) + \underline{Q} = 0 \tag{4-3}$$

Für die PGRNNs wurden zwei verschiedene, vom GT abweichende Parametrierungen für die zugehörigen physikalischen Modelle verwendet. Eine der beiden Parametrierungen weicht nur leicht von der eigentlichen GT-Parametrierung ab, wobei die Zweite einer groben Schätzung des GTs entspricht. Insgesamt ist es realistisch anzunehmen, dass eine der groben zweiten Parametrierung ähnliche Parametrierung ohne weitere Probleme bestimmbar ist.

Zudem wurde als physikalisches Modell jeweils das nichtlineare und das um die instabile Ruhelage linearisierte Modell des Cublis verwendet. Hiermit soll untersucht werden, ob, wie bereits in vorausgegangenen Kapiteln beobachtet, schon bei Verwendung eines linearen Dynamikmodells eine signifikante Reduktion des Simulationsfehlers zu beobachten ist.

Die Ergebnisse sind in Abbildung 4-19 dargestellt. Es fällt zunächst auf, dass keines der PGRNNs einen kleineren Simulationsfehler ggü. dem Referenz-RNN aufweist. Im Gegenteil führt die Berücksichtigung eines nichtlinearen Dynamikmodells für beide Parametrierungen zu einem Anstieg des Fehlers ggü. dem RNN. Lediglich mithilfe des linearen physikalischen Modells konnten in etwa gleich gute Ergebnisse erzielt werden. Erwartungsgemäß hätte hier das nichtlineare Modell im Vorteil gewesen sein müssen. Dies widerspricht den Ergebnissen am Viertelfahrzeug. Generell kann aber auch anhand der Trajektorien der Modellvarianten RNN und PGRNN/NL/leicht (Abb. 4-20) beobachtet werden, dass beide Netzvarianten eine gute Abbildung des geregelten Cublis ermöglichen. Bei der abgebildeten Testsequenz handelt es sich zudem um ein Extrembeispiel. Für die meisten Testsequenzen ist der Simulationsfehler marginal.

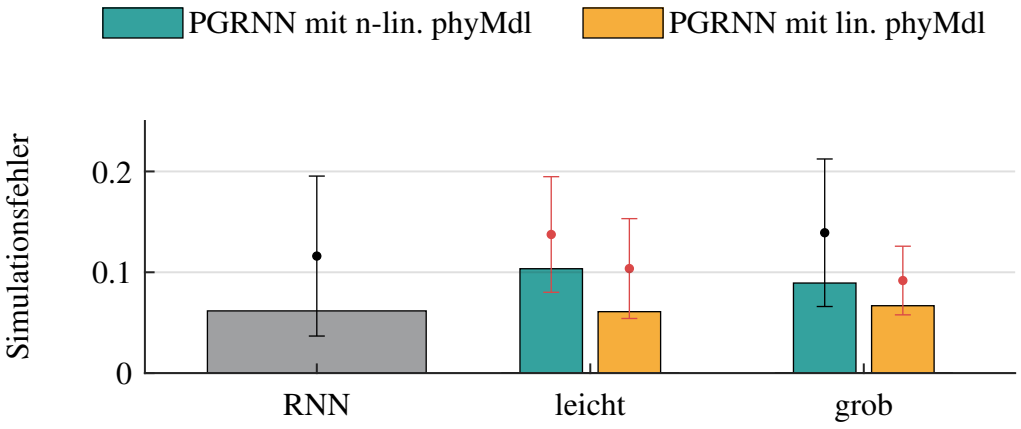

Bild 4-19: *Zur Modellierung des geregelten Cubli wurden RNN und PGRNNs mit linearem bzw. nichtlinearem physikalischen Modell herangezogen, wobei zwischen einer nur leicht vom GT abweichenden und einer groben Parametrierung unterschieden wird. Dabei konnte kein Vorteil der PGRNNs ggü. dem RNN verzeichnet werden. Im Falle der nichtlinearen Dynamikmodelle kam es sogar zu einem Anstieg des Simulationsfehlers.*

Zudem sind die Ergebnisse für die leicht und grob abweichende Modellparametrierung ungefähr auf einem Fehlerniveau. Hier wäre ein Vorsprung des Modells mit höherer Parametergüte erwartet worden. Ähnlich zum Viertelfahrzeug kann dies allerdings nicht beobachtet werden.

Der leicht niedrigere Mittelwert der Fehler für die PGRNNs mit linearem physikalischem Modell ggü. dem RNN deutet auf eine minimal benefizielle Auswirkung des zusätzlichen Dynamikmodells hin, dies ist jedoch angesichts statistischer Ungenauigkeiten zu vernachlässigen. Generell kann also gesagt werden, dass für den Cubli mit Zustandsrückführung keine Fehlerreduktion durch Einbringung eines physikalischen Dynamikmodells beobachtet werden konnte (Hypothese 4).

Zentrale Untersuchungsergebnisse

- Sowohl PGRNN als auch RNN erreichen eine gute Abbildung des Cubli mit Zustandsrückführung (Hypothese 4 S. 1).
- Für die PGRNNs konnte kein Vorteil ggü. dem RNN beobachtet werden (Hypothese 4 S. 2).
- Konträr zu den Ergebnissen am Viertelfahrzeug konnte mithilfe eines um die obere Ruhelage linearisierten Modells ein niedrigeres Fehlerniveau als mit dem nichtlinearen Cubli-Modell erzielt werden (Hypothese 1 Abs. 1 S. 2, Abs. 2).
- Die Güte der Modellparameter hatte wie beim Viertelfahrzeug keinen Einfluss auf die resultierende Netzgüte (Hypothese 1 Abs. 1 S. 2, Abs. 2).

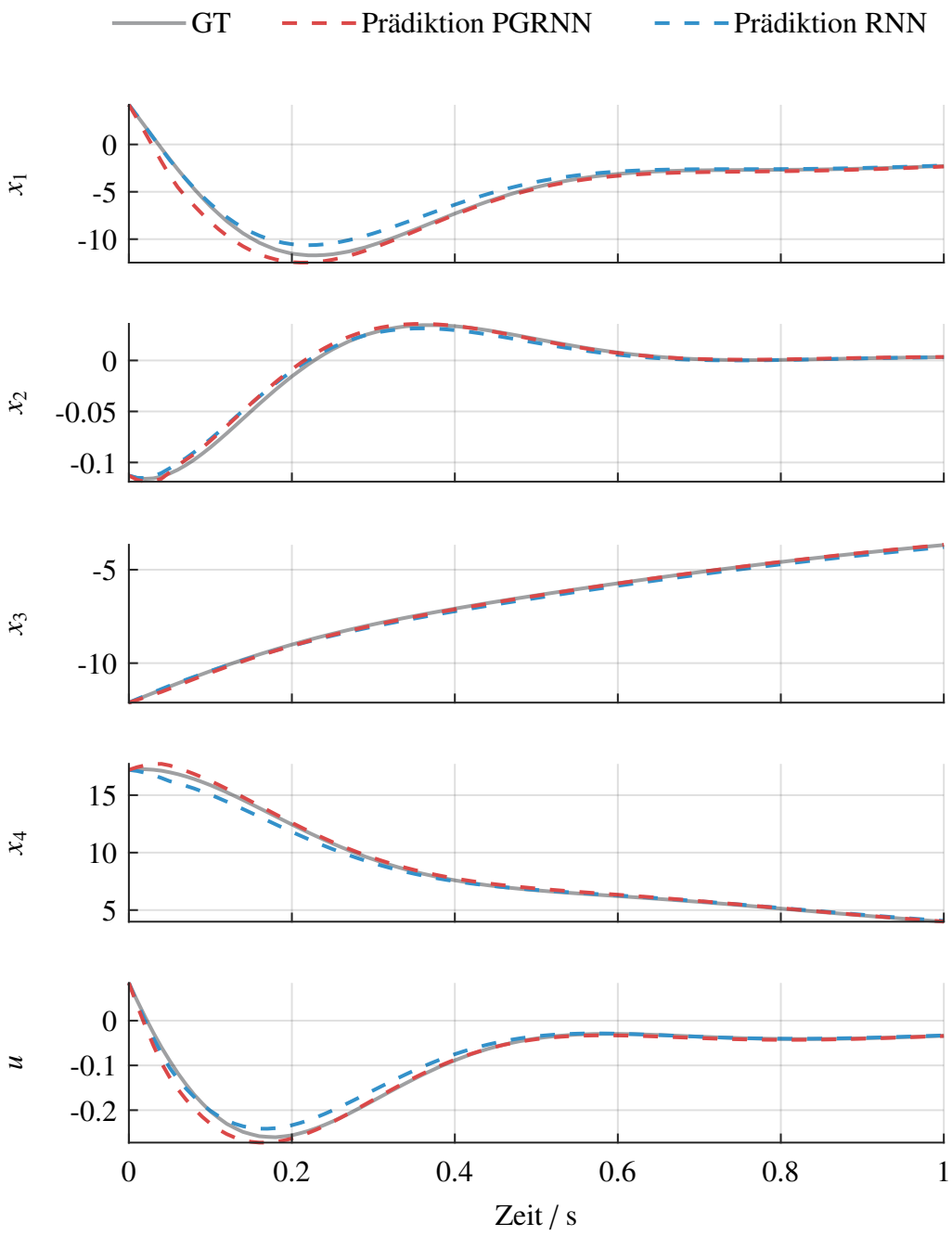

Bild 4-20: *Zur Veranschaulichung handelt es sich hierbei um ein Extrembeispiel bei Gegenüberstellung des RNN und des PGRNN/NL/leicht. Generell bieten sowohl RNN als auch PGRNN eine gute Abbildung des Systems. Das RNN ist in diesem Fall dem PGRNN knapp überlegen.*

5 Zusammenfassung und Ausblick

In diesem Kapitel wird zunächst eine Zusammenfassung der zentralen Ergebnisse der angestellten Untersuchungen gegeben. Diese umfasst einen Überblick über die dabei gefällten Entscheidungen und den sich daraus ergebenden Untersuchungsverlauf, aber auch Anmerkungen zu aufgekommenen Herausforderungen sowie Erfolgen.

Schlussendlich soll eine Interpretation der Untersuchungsergebnisse zur Identifikation erfolgversprechender Ansatzpunkte dienen. Diese Auswahl steht vollkommen vor dem Hintergrund des Einsatzes in der regelungstechnischen Domäne. Dieser Ausblick enthält zudem weitere Ideen und Suggestionen, die in konsekutiven Arbeiten zu untersuchen sind.

5.1 Zusammenfassung

Die Zusammenfassung ist in Sektionen untergliedert, die jeweils einem dezidierten Thema gewidmet sind. Sie umfasst einen holistischen Überblick der zentralen im Rahmen dieser Arbeit gewonnenen Erkenntnisse.

Motivation

Die Erweiterung üblicher datenbasierter Modellierungsverfahren um domänenspezifisches Wissen in Form von PGNNs hat bereits in einer Reihe von verschiedenen Fachbereichen zu bemerkenswerten Ergebnissen geführt. Besonders interessant für ingenieurwissenschaftliche Anwendungen ist dabei die Erweiterung bestehender Dynamikmodelle um datengetriebene Ansätze, die die Einhaltung dedizierter physikalischer Gesetzmäßigkeiten gewährleisten können. Die auf diese Weise modellierten Systeme können im Resultat nicht nur physikalisch interpretiert werden, sondern offenbaren eine neue Möglichkeit der Wissensgenerierung.

In dieser Arbeit sollen die Anwendbarkeit entsprechender PGNN-Ansätze auf regelungstechnische Anwendungen, insbesondere die Identifikation dynamischer Systeme geprüft werden. Dazu wurde zunächst die rein datengetriebene Modellierung mittels RNNs angestrebt. Ggü. vorwärtsgerichteten Netzen bieten diese aufgrund ihrer rekurrenten Struktur die Möglichkeit, zeitinvariante Dynamiken aus Datensequenzen zu identifizieren. Infolge eines zeitkontinuierlichen Modellierungsansatzes konnten entsprechend gute Modelle am Beispiel des Viertelfahrzeugs gelernt werden (Kap. 3.5).

Implementierung

Zur Erzielung dieser und aller folgenden Ergebnisse wurde in Mathworks MATLAB ein erweiterbares Framework zum Training diverser Netzarchitekturen entwickelt und implementiert. Die teils substantiellen Neuimplementierungen umfassen Training und Simula-

tion rekurrenter Netze mit und ohne zusätzlichem Dynamikmodell, komplexe Learning-Rate-Schedules, die Überwachung des Trainingsfortschritts anhand des Simulationsfehlers über den Validierungsdaten sowie die direkte Einbindung eines BO-basierten HP-Optimierungsalgorithmus. Neben dem Training der Netzwerke wurde für rekurrente Architekturen ein erweiterter ODE-Solver (Kap. A4) entwickelt und implementiert.

Zentrale Untersuchungsergebnisse für PGRNNs

Konnten mit rein datengetriebenen RNNs in vielen Fällen gute Ergebnisse erzielt werden, ist die Überlegenheit der PGRNNs eklatant.

Das PGRNN definiert sich im Rahmen dieser Arbeit über die Erweiterung des RNNs um ein physikalisches Dynamikmodell. Bei den Untersuchungen anhand der drei gewählten Beispielsysteme konnte gezeigt werden, dass selbst mit Dynamikmodellen geringer Güte deutliche Reduktionen des Simulationsfehlers beobachtet werden können. Die einzige Ausnahme betrifft dabei den geregelten Cubli, bei dem durch Hinzunahme des physikalischen Modells keine Fehlerreduktion festgestellt werden konnte. Hypothese 1 Abs. 1 Satz 1 gilt damit als verifiziert.

Hypothese 1 Abs. 1 Satz 2 sowie Abs. 2 suggeriert zudem, dass sich eine höhere Modellgüte direkt in eine höhere Güte des resultierenden PGRNNs übersetzt, wobei die Modellgüte durch die Güte der Modellparameter und der Nichtlinearitäten gekennzeichnet ist. I.A. können diese Hypothesen ebenfalls bejaht werden. Dabei konnte für das Viertelfahrzeug sowie den Cubli kein direkter Zusammenhang zwischen Parametergüte und resultierender Netzgüte verzeichnet werden. Lediglich die Wahl verschiedener Nichtlinearitäten, im Fall des Viertelfahrzeugs des Exponenten α, hatte einen deutlichen Einfluss auf die resultierende Netzgüte. So konnte ggü. dem RNN bereits bei Verwendung eines linearen Dynamikmodells eine Reduktion des Simulationsfehlers um fast 70% am Beispiel des Viertelfahrzeugs verzeichnet werden (Kap. 4.3.1). Am Beispiel des Lorenz-Attraktors belief sich die Reduktion durch Hinzunahme eines Modells sogar auf maximal 95%, mindestens immerhin 75% (Kap. 4.3.2).

Beim Lorenz-Attraktor handelt es sich darüber hinaus um ein chaotisches, ggü. Parameterdeviationen sensitives System. Hier konnte ein deutlicher Zusammenhang zwischen Netzgüte und Parametergüte beobachtet werden. Die Importanz einer ausreichend genauen Parametrierung konnte für Langzeitprädiktionen herausgestellt werden. Mit einer Parameterabweichung von -25% konnten auf einen Prädiktionshorizont von 5 Sekunden sehr gute Ergebnisse erzielt werden. Das RNN lieferte dagegen bei der Modellierung des Lorenz-Attraktors vergleichsweise schlechte Ergebnisse – der Fehler war hier etwa 3.5-mal höher. Für derart komplexe Systeme scheinen demnach PGRNN einen deutlich höheren Mehrwert zu bieten, womit Hypothese 1 Abs. 3 bestätigt werden kann.

Hypothese 2 konnte ebenfalls verifiziert werden. So konnte bei den Untersuchungen in Kapitel 4.4.2 am Viertelfahrzeug demonstriert werden, dass für eine periodische Anregung eine optimale Länge der Trainingssequenzen existiert. Datensequenzen, die vornehmlich repetitive Schwingungstrajektorien enthalten, sind von geringem Wert. Ein besonders drastischer Anstieg des Fehlers lag jedoch genau dann vor, wenn Trainingssequenzen ausschließlich dem Übergangsbereich entnommen werden und keine Informationen des eingeschwungenen Zustands enthalten. Genau dieser Fall bietet zudem bzgl. der

Stellgrößen u und $\dot{u}$ keine umfassende Datenbasis. Durch geeignete Wahl der Sequenzlänge konnte eine Senkung des Simulationsfehlers um bis zu 93% beobachtet werden.

Eine weitere Form des Domänenwissens, die im Rahmen von PGNNs eingebracht werden kann, sind physikalische Nebenbedingungen. Im Rahmen dieser Arbeit wurde dazu die Energiebilanz am Beispiel des nichtlinearen Viertelfahrzeugs betrachtet. Infolge der Einführung des energiebasierten Strafterms konnte jedoch keine Reduktion des Simulationsfehlers beobachtet werden. Dies könnte auf eine negative Beeinflussung der allgemeinen Konditionierung des Trainings zurückzuführen sein und ist system- sowie nebenbedingungsspezifisch. Aus dieser Beobachtung sollte im Umkehrschluss also nicht auf eine allgemein negative Auswirkung physikalisch-motivierter Nebenbedingung geschlossen werden. Hypothese 3 kann demnach weder bestätigt noch negiert werden.

Hypothese 4 postuliert, dass geregelte Systeme auf gleiche Weise wie gesteuerte und autonome Systeme durch RNNs und PGRNNs abgebildet werden können sowie eine Überlegenheit der PGRNNs. Ersteres konnte im Rahmen der Untersuchungen in Kapitel 4.8 bestätigt werden. Sowohl mit RNN als auch mit dem PGRNN konnten gute Modelle des Cubli gelernt werden. Jedoch führte eine Hinzunahme eines physikalischen Dynamikmodells iFv. ODEs in keinem Fall zu einer Reduktion des Simulationsfehlers. Ebenso konnte bzgl. unterschiedlicher Modellgüten kein konsistentes Ergebniss beobachtet werden. Die Hintergründe sind in folgenden Arbeiten zu analysieren.

Schlussendlich konnte durch Erweiterung des PGRNNs um eine Funktionsbibliothek ein weiterer Fortschritt erzielt werden. Am Beispiel des Viertelfahrzeugs wurde gezeigt, dass die Hinzunahme einer Funktionsbibliothek, die die korrekten nichtlinearen Terme des GT-Modells enthält, ein minderwertiges Dynamikmodell kompensiert werden kann. So konnten die PGRNN+-Varianten für alle Dynamikmodelle gleich niedrige Simulationsfehlerniveaus aufweisen. Dies bedeutet, dass im untersuchten Fall die Berücksichtigung eines lediglich linearen Dynamikmodells unter Hinzunahme einer Funktionsbibliothek mit einem sehr genauen Dynamikmodell gleichzusetzen ist. Ggü. dem rein datengetriebenen RNN konnte so eine Fehlerreduktion um 88% festgestellt werden.

5.2 Ausblick

Die im Rahmen dieser Arbeit gefundenen Untersuchungserkenntnisse werfen eine Vielzahl weiterer Ansatzpunkte auf. Dabei gilt es, beobachtete Effekte anhand weiterer Systeme zu verifizieren, aufgeworfene Fragestellungen und Diskrepanzen näher zu untersuchen sowie identifizierte erfolgversprechende Ansätze weiter zu verfolgen.

Folgende Ansätze werden im Folgenden näher beschrieben:

- Identifikation geregelter Systeme
- Leitfaden zur Generierung optimaler Trainingsdaten
- Physikalische Nebenbedingungen
- Neuimplementierung GRU
- Vorgabe der Lösungsstruktur
- Benchmarking

- Kontextualisierung bzw. Einbeziehung in den Reglerentwurfsprozess
- Substitution der HP-Akquisitionsfunktion
- Probabilistischer Ansatz
- Modellierung zeitvarianter Systemdynamiken

Identifikation geregelter Systeme

Die Untersuchungsergebnisse bzgl. des geregelten Cubli-Systems sind nicht konsistent. Zum einen konnte durch Hinzunahme eines physikalischen Dynamikmodells nur in diesem Fall kein Vorteil ggü. dem rein datengetriebenden RNN beobachtet werden. Zum anderen fällt der beobachtete Zusammenhang zwischen Parametergüte und resultierender Netzgüte konträr zu den Erkenntnissen am Viertelfahrzeug sowie des Lorenz-Attraktors aus.

Inwiefern dieser Umstand auf das betrachtete System, die Regelung an sich oder numerische Aspekte zurückzuführen ist, kann an dieser Stelle nicht beantwortet werden und sollte im Rahmen weiterer Arbeiten untersucht werden.

Leitfaden zur Generierung optimaler Trainingsdaten

Infolge der Untersuchungen in Kapitel 4.4.2 konnte verifiziert werden, dass Trainingsdaten möglichst variantenreiche Trajektorien umfassen sollten. Für eine periodische Anregung konnte dabei eine optimale Sequenzlänge bestimmt werden. Inwiefern dies auf andere Steuerungsszenarien übertragbar ist, bleibt fraglich.

Ebenso bleibt fraglich, auf welche Weise zuverlässig möglichst hochwertige Trainingsdaten generiert werden können. Dazu könnten etwa die Modellierung mit Sprungantworten oder Sweep-Anregungen untersucht werden.

Darüber hinaus wurde in Kapitel 4.4.1 die Datenmenge als Funktion der Sequenzzahl, Sequenzlänge und Zeitschrittweite definiert. Im Rahmen dieser Arbeit wurde stets eine Datenmenge zwischen $5 \cdot 10^3$ und $1 \cdot 10^4$ verwendet (siehe Kap. A2), wobei in Kapitel 4.4.1 eine Degradation der Modellgüte für deutlich niedrigere Datenmengen beobachtet wurde. In weiteren Arbeiten gilt es zu untersuchen, ob eine Faustformel für eine minimal notwendige Datenmenge aufgestellt werden kann, bei der noch eine ausreichende Modellgüte gewährleistet werden kann. Desweiteren gilt es zu untersuchen, ob die Hinzunahme eines physikalischen Dynamikmodells, mit oder ohne Pre-Training, zu einer Reduktion dieser Anforderung führt.

Physikalische Nebenbedingungen

Gerade bei der Modellierung komplexerer technischer Systeme kann von einer Beschränkung des Optimierungsraums profitiert werden, da dieser exponentiell mit der Anzahl Gewichte ansteigt. Daher gilt es im Rahmen weiterer Arbeiten für entsprechende Systeme

geeignete Nebenbedingungen zu identifizieren. Dabei können verschiedenste Bedingungen betrachtet werden. Die Implementierung erfolgt, wie in Kapitel 3.7 beschrieben, iFv. Straftermen.

Eine Nebenbedingung dient zweier Zwecke. Zum einen beschränkt die zusätzliche Beschränkung den Suchraum und sollte im besten Fall zu einer schnelleren Konvergenz sowie einem besseren Endergebnis führen. Der zweite im Rahmen von PGNNs verfolgte Zweck ist die Generierung physikalisch valider Modelle. Dazu werden entsprechende Gesetzmäßigkeiten ausgewählt und als Strafterme implementiert. Je nach Anwendungsfall ergeben sich unterschiedliche Möglichkeiten, wobei im Folgenden ein beispielhafter, erfolgversprechender und für den regelungstechnischen Bereich relevanter Ansatz gegeben werden soll.

Im Rahmen regelungstechnischer Anwendungen werden idR. die Eigenwerte der Dynamikmatrizen untersucht. So könnte beispielsweise durch Bestrafung positiver Realteile der Eigenwerte die Stabilität des gelernten Modells gewährleistet werden. Entsprechende Ansätze gelten jedoch idR. lediglich für lineare Modelle. Wie in diesen Fällen die Dynamikmatrizen zurückgewonnen werden können, wurde in Kapitel 3.2 anhand Gleichung (3-3) beschrieben. Inwiefern ähnliche Ansätze auf rekurrente Netze überführbar sind, bleibt fraglich. Möglicherweise können durch geeignete Modifikation des RNNs entsprechende Ansätze verwendet werden. Ebenso denkbar ist die Modellierung des Koopman-Operators mittles Neuronaler Netze, für den im gleichen Schritt entsprechende, auf lineare Systeme anwendbare, Ansätze implementiert werden könnten.

Neuimplementierung GRU

Die in dieser Arbeit verwendete GRU-Implementierung entspricht dem GRU-Layer in MATLAB. Der Nutzer ist dabei auf die State-Aktivierungsfunktion Tanh und die Sigmoid-Funktion als Gate-Aktivierungsfunktion beschränkt. Aus denen in Kapitel 2.1.5 aufgeführen Gründen entsprechen diese nicht mehr dem aktuellen Standard und führen zu numerischen Problemen. Dies könnte durch die Verwendung von ReLU- bzw. LReLU oder ELU-Aktivierungsfunktionen anstelle von Tanh und Sigmoid gemindert werden.

Durch Beseitigung der numerischen Limitationen ist möglicherweise ein stabileres Training rekurrenter Neuronaler Netze mit höherer Neuronenzahl möglich. Die höhere Modellkomplexität würde wiederum zu einer gesteigerten Performanz der entsprechenden Netze führen.

Vorgabe der Lösungsstruktur

Viele der in Kapitel 3.1 vorgestellten Ansätze basieren auf der Vorgabe einer speziellen Lösungsstruktur – sie sind *domain-adapted* oder *physics-based*. Zwei der Ansätze zur Modellierung technische Systeme sind LNNs und HNNs [CGH+20; GDY19]. Im Rahmen dieser speziellen Modellvarianten wird ein Modell in Form der Euler-Lagrange- bzw. der Hamilton-Funktion gelernt. Dabei dient das jeweilige Netz zum „Erlernen“ der Parameter der jeweiligen Gleichungsform.

Ebenso ist auch in der Regelungstechnikdomäne die Vorgabe einer konkreten Lösungsstruktur möglich. In Kapitel 3.5 wurde bspw. explizit ein lineares NN gewählt. Es könnte in weiteren Arbeiten untersucht werden, inwiefern LNNs, RNNs oder artverwandte PGNN-Varianten für nicht-autonome, regelungstechnische Systeme geeignet sind.

Benchmarking

Im Rahmen dieser Arbeit konnte gezeigt werden, dass PGRNNs zur Modellierung der Systemdynamik dem rein datengetriebenen Ansatz mittels RNNs deutlich überlegen sind. Darüber hinaus bieten PGRNNs die Möglichkeit, die Einhaltung physikalischer Gesetze festzustellen sowie synergetische Effekte.

Neben der Modellierung mittels RNNs bestehen weitere rein datengetriebene Modellierungsverfahren, deren Performanz mit der der PGRNNs verglichen werden könnte. So könnte bspw. untersucht werden, inwiefern PGRNNs Systemidentifikationsverfahren auf Basis von Autoregressionsmodellen überlegen sind. Dazu sollte ein Benchmark geeigneter Identifikationsverfahren sowie Beispielsysteme angestellt werden.

Kontextualisierung bzw. Einbeziehung in den Reglerentwurfsprozess

Bis jetzt wurden PGRNNs lediglich isoliert zur Modellbildung und Systemidentifikation untersucht. In konsekutiven Schritten wird idR. zusätzlich ein Beobachter sowie schlussendlich der Regler ausgelegt. In all diesen Schritten ist das PGRNN-Modell involviert. Entsprechend sollte geprüft werden, welche Anforderungen bzgl. aufbauender Schritte vorliegen und inwiefern diese bspw. in Form weiterer physikalischer Nebenbedingungen ausgedrückt werden können.

Es ist z.B. noch offen, ob ein PGRNN-Modell während des Trainings ebenfalls für einen Beobachter genutzt werden kann. Hier liegen speziell Anforderungen bzgl. der Beobachtbarkeit, Steuerbarkeit und Robustheit des Modells vor, deren Einbindung iFv. Nebenbedingungen getestet werden könnte. Ebenso ist in diesem Zusammenhang zu klären, ob ein probabilistischer Ansatz notwendig ist.

Soll das Modell darüber hinaus in Kombination mit einer MPC verwendet werden, so kann untersucht werden, ob das Modell während des Betriebs durch die erfassten Messdaten angepasst werden könnte. Dies entspricht einem Online-Training. Auf diese Weise könnte sich das Modell adaptiv an zeitliche Veränderungen der Systemdynamik anpassen.

Substitution der HP-Akquisitionsfunktion

Die Wahl geeigneter HPs ist von großer Bedeutung für den Trainingserfolg (siehe dazu auch Kap. 2.1.6). Entsprechend wichtig ist die Gewährleistung einer ausgiebigen aber auch effizienten HP-Optimierung. Im Rahmen der HP-Optimierung durch BO ist ein wichtiger Parameter die gewählte Akquisitionsfunktion. Für den Umfang dieser Arbeit wurde dazu Probability of Improvement Plus (PI+) verwendet.

Wird die zu optimierende Netzgüte auf den Simulationsfehler reduziert, ist der Wert des Optimums bekannt, da der minimal zu erreichende Simulationsfehler 0 bzw. im logarithmischen Fall $-\infty$ entspricht. Die Akquisitionsfunktion Max-Value Entropy Search (MES) gilt als state-of-the-art. Diese kann vereinfacht werden für den Fall, dass der Wert des gesuchten Minimums bekannt ist. Die reduzierte Akquisitionsfunktion für einen minimalen Wert von 0 ergibt sich damit zu MESmin0, wie in [Sch19] beschrieben. In weiterführenden Untersuchungen könnte erforscht werden, ob durch Substitution von PI+ durch MESmin0 eine schnellere Konvergenz des HP-Optimierungsprozesses zum Optimum erzielt werden kann.

Probabilistischer Ansatz

Die im Rahmen dieser Arbeit verwendeten Neuronalen Netze sind deterministischer Natur. Es besteht darüber hinaus die Möglichkeit, probabilistische Modelle auf Basis Bayesscher Neuronaler Netze (BNN) zu erstellen. So konnten bspw. im Fall von [YMK20] entsprechend hochwertige Modelle aus stark verrauschten Messdaten extrahiert werden.

Besonders interessant sind probabilistische Ansätze dann, wenn Bereiche des Zustandsraums oder einzelne Zustandsvariablen nicht beobachtet werden können. Inwiefern BNNs im Rahmen von PGNNs zur Modellierung in der Regelungstechnik eingesetzt werden können, gilt es zu untersuchen. Dabei stehen den möglichen Performanzgewinnen eine gesteigerte Modellkomplexität sowie höhere Anforderungen an den Trainingsablauf entgegen.

Modellierung zeitvarianter Systemdynamiken

Sowohl Viertelfahrzeug, Lorenz-Attraktor als auch der Cubli sind zeitinvariante Systeme. Sollen PGRNNs zur Modellierung zeitlich veränderlicher Systeme, bspw. degradierender Fertigungsanlagen verwendet werden, ist eben dieser Degradationsprozess ebenfalls zu modellieren.

Im Rahmen von RNNs wird die zeitliche Unabhängigkeit durch die Kopplung der Gewichte erzielt. Um nun eine Modellierung eines zeitlich veränderlichen Systemverhaltens zu ermöglichen, können entweder DNNs verwendet werden oder es wird eine Modifikation des RNNs durchgeführt. Ein entsprechender Ansatz wäre, in einem dem Systemzustand beschreibenden GRU übergeordneten Modellierungsprozess eine Modellierung der dynamischen Änderung des Systemverhaltens abzubilden. In diesem übergeordneten Modellierungsprozess könnte dazu ebenfalls ein GRU genutzt werden, dessen Hidden State den Gewichtsmatrizen der untergeordneten Modellierungsebene entspricht.

Literaturverzeichnis

[21] *Cubli*. 2021. https://idsc.ethz.ch/research-dandrea/research-projects/archive/cubli.html (besucht am 11. 06. 2021)

[ACS+21] Antonelo, E. A.; Camponogara, E.; Seman, L. O.; Souza, E. R. de; Jordanou, J. P.; Hubner, J. F.: *Physics-Informed Neural Nets-based Control*. 2021

[And13] Andrew L. Maas: Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2013

[Bij16] Bijl, H.: *Gaussian Process Regression Techniques: With Applications to Wind Turbines: student version*. 2016.

[Bis09] Bishop, C. M.: *Pattern recognition and machine learning*. Corrected at 8th printing 2009. Information science and statistics. New York, NY: Springer, 2009

[Bro19] Brownlee, J.: A Gentle Introduction to the Rectified Linear Unit (ReLU). *Machine Learning Mastery* (2019).

[CGH+20] Cranmer, M.; Greydanus, S.; Hoyer, S.; Battaglia, P.; Spergel, D.; Ho, S.: *Lagrangian Neural Networks*. 2020

[CUH15] Clevert, D.-A.; Unterthiner, T.; Hochreiter, S.: *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015

[CvG+14] Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y.: *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014

[CZAB19] Chen, Z.; Zhang, J.; Arjovsky, M.; Bottou, L.: Symplectic Recurrent Neural Networks. *CoRR* abs/1909.13334 (2019)

[DMC+20] Dener, A.; Miller, M. A.; Churchill, R. M.; Munson, T.; Chang, C.-S.: *Training neural networks under physical constraints using a stochastic augmented Lagrangian approach*. 2020

[FN93] Funahashi, K.-i.; Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks* 6 (1993), Nr. 6, S. 801–806

[GBB11] Glorot, X.; Bordes, A.; Bengio, Y.: Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Hrsg. von Gordon, G.; Dunson, D.; Dudík, M. Bd. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA, 2011, S. 315–323

[GBC16] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep learning*. Cambridge, Massachusetts und London, England: MIT Press, 2016

[GDY19] Greydanus, S.; Dzamba, M.; Yosinski, J.: *Hamiltonian Neural Networks*. 2019

[HS97] Hochreiter, S.; Schmidhuber, J.: Long Short-term Memory. *Neural Computation* 9 (1997), S. 1735–1780

[JP03] Jönck, U.; Prill, F.: *Das Lorenz-System: Seminar über gewöhnliche Differentialgleichungen*. 2003. www . math . uni - hamburg . de / home / lauterbach / scripts / seminar03 / prill . pdf (besucht am 12. 05. 2021)

[JWK+18] Jia, X.; Willard, J.; Karpatne, A.; Read, J.; Zwart, J.; Steinbach, M.; Kumar, V.: *Physics Guided RNNs for Modeling Dynamical Systems: A Case Study in Simulating Lake Temperature Profiles*. 2018

[JY12] James Bergstra; Yoshua Bengio: Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (2012), Nr. 10, S. 281–305

[KAF+17] Karpatne, A.; Atluri, G.; Faghmous, J.; Steinbach, M.; Banerjee, A.; Ganguly, A.; Shekhar, S.; Samatova, N.; Kumar, V.: Theory-guided Data Science: A New Paradigm for Scientific Discovery from Data. *IEEE Transactions on Knowledge and Data Engineering, 29(10), pp* (2017)

[KBK+12] Kruse, R. J.; Borgelt, C.; Klawonn, F.; Moewes, C.; Russ, G.; Steinbrecher, M.: *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Studium. Wiesbaden: Vieweg + Teubner, 2012

[KGW00] Kambhampati, C.; Garces, F.; Warwick, K.: Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks. *IJCNN 2000*. Hrsg. von Amari, S. New York, 2000, 64–69 vol.1

[Kos17] Kostadinov, S.: Understanding GRU Networks - Towards Data Science. *Towards Data Science* (2017).

[KSH12] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*. Hrsg. von F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger. Bd. 25. 2012

[KWRK17] Karpatne, A.; Watkins, W.; Read, J.; Kumar, V.: *Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling*. 2017

[LKB20] Liu, Y.; Kutz, J.; Brunton, S.: *Hierarchical Deep Learning of Multiscale Differential Equation Time-Steppers*. 2020

[LRP19] Lutter, M.; Ritter, C.; Peters, J.: *Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning*. 2019

[LSM18] Long, Y.; She, X.; Mukhopadhyay, S.: *HybridNet: Integrating Model-based and Data-driven Learning to Predict Evolution of Dynamical Systems*. 2018

[Mat] MathWorks: *Gated recurrent unit (GRU) layer: More About Gated Recurrent Unit Layer*. https://de.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.grulayer.html (besucht am 03. 05. 2021)

[MIM+18] Muralidhar, N.; Islam, M. R.; Marwah, M.; Karpatne, A.; Ramakrishnan, N.: Incorporating Prior Domain Knowledge into Deep Neural Networks. *2018 IEEE International Conference on Big Data (Big Data)*. 2018, S. 36–45

[MMIR12] M. Gajamohan; M. Merz; I. Thommen; R. D'Andrea: The Cubli: A Cube that can Jump Up and Balance. *International Conference on Intelligent Robots and Systems*. 2012, S. 3722–3727

[MPS08] Koppelschwingungen. *Schwingungen*. Hrsg. von Magnus, K.; Popp, K.; Sextro, W. Studium. Wiesbaden: Vieweg + Teubner, 2008, S. 178–220

[Pei19] Peixeiro, M.: How to Improve a Neural Network With Regularization. *Towards Data Science* (2019).

[QNG15] Quoc V. Le; Navdeep Jaitly; Geoffrey E. Hinton: A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *CoRR* abs/1504.00941 (2015)

[Reg21] Regel, F. M.; Trächtler, A. (BetreuerIn); Timmermann, J. (BetreuerIn); Junker, A. (BetreuerIn): *Entwicklung eines Demonstrators für hybride Methoden in der Regelungstechnik*. Masterarbeit. Paderborn: Universität Paderborn, 2021

[RPK19] Raissi, M.; Perdikaris, P.; Karniadakis, G. E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 378 (2019), S. 686–707

[RZW20] Rong, M.; Zhang, D.; Wang, N.: *A Lagrangian Dual-based Theory-guided Deep Neural Network*. 2020

[Sch19] Schön, O.; Trächtler, A. (BetreuerIn); Timmermann, J. (BetreuerIn); Hesse, M. (BetreuerIn): *Effiziente Bayessche Optimierung durch Berücksichtigung A-priori-Wissens in Form eines physikalischen Dynamikmodells*. Studienarbeit. Paderborn: Universität Paderborn, 2019

[SJJ16] Steven L. Brunton; Joshua L. Proctor; J. Nathan Kutz: Sparse Identification of Nonlinear Dynamics with Control (SINDYc). *IFAC-PapersOnLine* 49 (2016), Nr. 18, S. 710–715

[SMG14] Saxe, A. M.; McClelland, J. L.; Ganguli, S.: *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*. 2014

[SUZ06] Schäfer, A. M.; Udluft, S.; Zimmermann, H. G.: Learning Long Term Dependencies with Recurrent Neural Networks. *Artificial neural networks - ICANN 2006*. Hrsg. von Kollias, S.; Stafylopatis, A.; Duch, W.; Oja, E. Bd. 4131. Lecture Notes in Computer Science. Berlin: Springer, 2006, S. 71–80

[SZ06] Schäfer, A. M.; Zimmermann, H. G.: Recurrent Neural Networks Are Universal Approximators. *Artificial neural networks - ICANN 2006*. Hrsg. von Kollias, S.; Stafylopatis, A.; Duch, W.; Oja, E. Bd. 4131. Lecture Notes in Computer Science. Berlin: Springer, 2006, S. 632–640

[TD15] Trischler, A.; D'Eleuterio, G. M. T.: *Synthesis of recurrent neural networks for dynamical system simulation*. 2015

[Tho19] Thomas, M.: The Future of AI: How Artificial Intelligence Will Change the World. *Built In* (2019).

[TRJ+19] Toth, P.; Rezende, D. J.; Jaegle, A.; Racanière, S.; Botev, A.; Higgins, I.: *Hamiltonian Generative Networks*. 2019

[UT20] Udrescu, S.-M.; Tegmark, M.: AI Feynman: A physics-inspired method for symbolic regression. *Science advances* 6 (2020), Nr. 16, eaay2631

[WJX+20] Willard, J.; Jia, X.; Xu, S.; Steinbach, M.; Kumar, V.: *Integrating Physics-Based Modeling with Machine Learning: A Survey*. 2020

[WWY20] Wang, R.; Walters, R.; Yu, R.: *Incorporating Symmetry into Deep Dynamics Models for Improved Generalization*. 2020

[YMK20] Yang, L.; Meng, X.; Karniadakis, G. E.: *B-PINNs: Bayesian Physics-Informed Neural Networks for Forward and Inverse PDE Problems with Noisy Data*. 2020

[YP18] Yang, Y.; Perdikaris, P.: *Physics-informed deep generative models*. 2018

[YYL20] Yu, Y.; Yao, H.; Liu, Y.: Structural dynamics simulation using a novel physics-guided machine learning method. *Engineering Applications of Artificial Intelligence* 96 (2020), S. 103947

[YZK18] Yang, L.; Zhang, D.; Karniadakis, G. E.: *Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations*. 2018

Anhang

Inhaltsverzeichnis

A1 Modellparametrierungen

In diesem Kapitel sind die Parametrierungen der einzelnen Modelle aufgelistet, die in den Untersuchungen verwendet wurden. Dabei entspricht GT jeweils der Ground Truth Parametrierung sowie phyMdl den Parametrierungen des physikalischen Dynamikmodells, die im Falle des PGRNNs Anwendung finden.

A1.1 Viertelfahrzeugmodell

Tabelle A1-1: *Ground Truth und phyMdl Parametrierungen Viertelfahrzeugmodell*

Parameter	**Wert**			**Einheit**
	GT	**−10%**	**−75%**	
m_A	500	500	500	kg
m_R	20	20	20	kg
c_A	20000	18000	5000	$\mathrm{N\,m^{-1}}$
c_R	100000	90000	25000	$\mathrm{N\,m^{-1}}$
d_A	1400	1260	350	$\mathrm{N\,s\,m^{-1}}$
d_R	300	270	75	$\mathrm{N\,s\,m^{-1}}$

A1.2 Lorenz-Attraktor

Tabelle A1-2: *Ground Truth und phyMdl Parametrierungen Lorenz-Attraktor*

Parameter	**Wert**			
	GT	**−10%**	**−25%**	**−75%**
a	10.0	9.0	7.5	2.5
b	28.0	25.2	21.0	7.0
c	8/3	2.4	2.0	2/3

A1.3 Cubli

Tabelle A1-3: *Ground Truth und phyMdl Parametrierungen Cubli*

Parameter	**Wert**			**Einheit**
	GT	**leicht**	**grob**	
l_b	0.075	0.080	0.080	m
l_w	0.085	0.080	0.080	m
m_b	0.461	0.465	0.450	kg
m_w	0.187	0.189	0.200	kg
J_b	$2.23 \cdot 10^{-3}$	$2.20 \cdot 10^{-3}$	$3.00 \cdot 10^{-3}$	$\mathrm{kg\,m^2}$
J_w	$0.32 \cdot 10^{-3}$	$0.30 \cdot 10^{-3}$	$0.10 \cdot 10^{-3}$	$\mathrm{kg\,m^2}$
c_b	$1.02 \cdot 10^{-3}$	$1.00 \cdot 10^{-3}$	$1.50 \cdot 10^{-3}$	$\mathrm{kg\,m^2\,s^{-1}}$
c_w	$0.05 \cdot 10^{-3}$	$0.06 \cdot 10^{-3}$	$0.10 \cdot 10^{-3}$	$\mathrm{kg\,m^2\,s^{-1}}$
K_m	$40.4 \cdot 10^{-3}$	$40.0 \cdot 10^{-3}$	$35.0 \cdot 10^{-3}$	$\mathrm{N\,m\,A^{-1}}$

A2 Trainingseinstellungen

Die folgende Tabelle enthält die Trainingseinstellungen, die sich im Rahmen dieser Arbeit als passen herausgestellt haben und folglich bei den Untersuchungen verwendet wurden.

Tabelle A2-1: *Trainingseinstellungen, die im Rahmen der Untersuchungen verwendet wurden*

Parameter	**Viertelfahrzeug**	**Lorenz-Attraktor**	**Cubli**
Lernrate	0.0136	0.0136	0.0136
Fehlerfunktion	LogMAE	LogMSE	LogMAE
Anzahl Trainingssequenzen[33]	10	24	50
Zeitschrittweite	10^{-3}s	10^{-2}s	10^{-2}s
Länge der Trainingssequenzen[34]	1s	3s	1s
Anzahl Validierungssequenzen	5	6	4
Anzahl Testsequenzen	10	10	10
Länge der Testsequenzen[35]	1s	3s	1s
Anzahl Auswertungen BO	50	50	50
Min. Anzahl verst. Neuronen	5	5	4
Max. Anzahl verst. Neuronen	350	1000	500
Min. λ_{EC}[36]	0	–	–
Max. λ_{EC}[37]	0.9	–	–
Anzahl identischer Netzinstanzen	10	10	10
Learning Rate Drop Factor	0	0	0
Initial Learning Rate	0.6	0.6	0.6
Early Stopping Horizon	40	40	40
Maximale Epochenzahl	5000	5000	5000
Min. Learning Rate	10^{-7}	10^{-7}	10^{-7}
Min. Training Progress	10^{-6}	10^{-6}	10^{-6}
Min. Validation Progress	10^{-2}	10^{-2}	10^{-2}
Validation Frequency	50	20	20

A3 Ergebnisse der Bayesschen Optimierung

Zur Bestimmung geeigneter HPs wird dem Training eines NNes idR. eine Hyperparameteroptimierung vorgeschaltet. Diese erfolgt z.B. durch BO. Im Rahmen der in dieser Arbeit angestellten Untersuchungen wurde vorwiegend über die Anzahl versteckter Neuronen sowie vereinzelt über den Laplace-Multiplikator des energiebasierten Strafterms optimiert. Zur Veranschaulichung der intermediären Ergebnisse sowie der Diskrepanz zwischen geschätztem Minimum und beobachtetem Minimum sind im Folgenden einige Grafiken der BO angefügt.

A3.1 Viertelfahrzeugmodell

Die in diesem Kapitel angeführten Abbildungen bilden die BO im Fall des Viertelfahrzeugmodells ab. Dabei wurden in allen abgebildeten Fällen 10 Trainingssequenzen à 1s verwendet.

Zur Erinnerung: Die Netzgüte (4-1) **ist zu minimieren!**

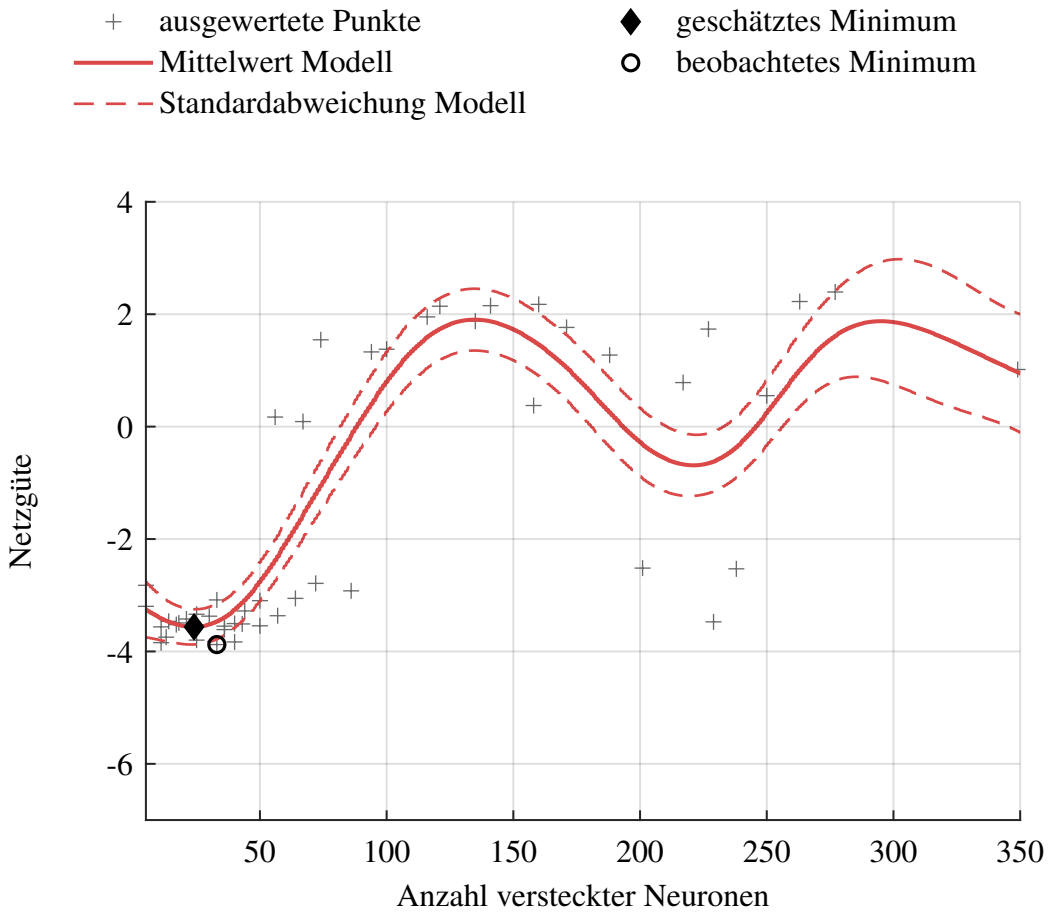

Bild A3-1: *Ergebnisse der BO für das RNN zur Modellierung des Viertelfahrzeugs.*

A3.1.1 PGRNN/-0%

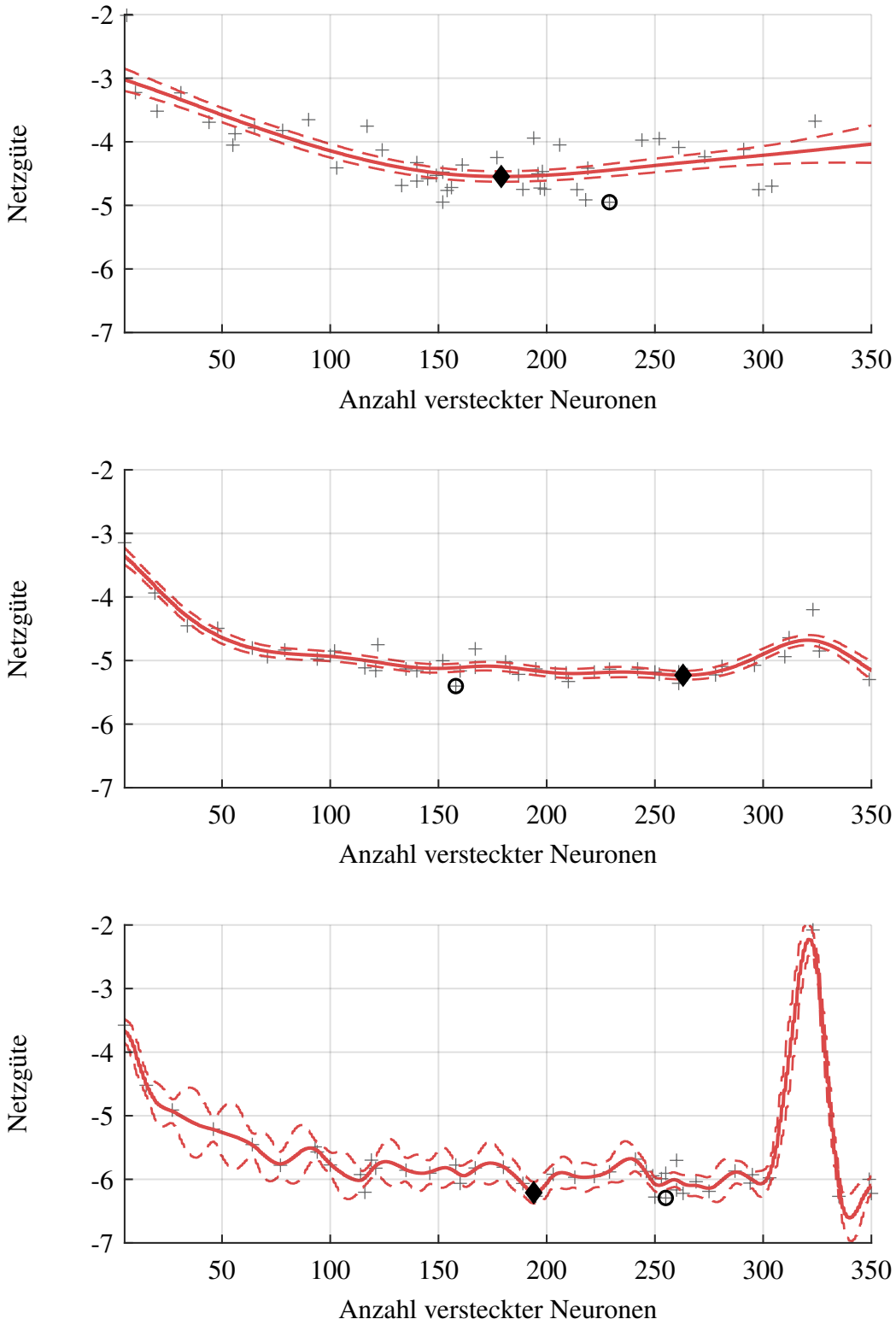

Bild A3-2: Ergebnisse der BO für PGRNN/−0%/Exp.1, PGRNN/−0%/Exp.2 und PGRNN/−0%/Exp.3 (von oben) zur Modellierung des Viertelfahrzeugs.

A3.1.2 PGRNN/-10%

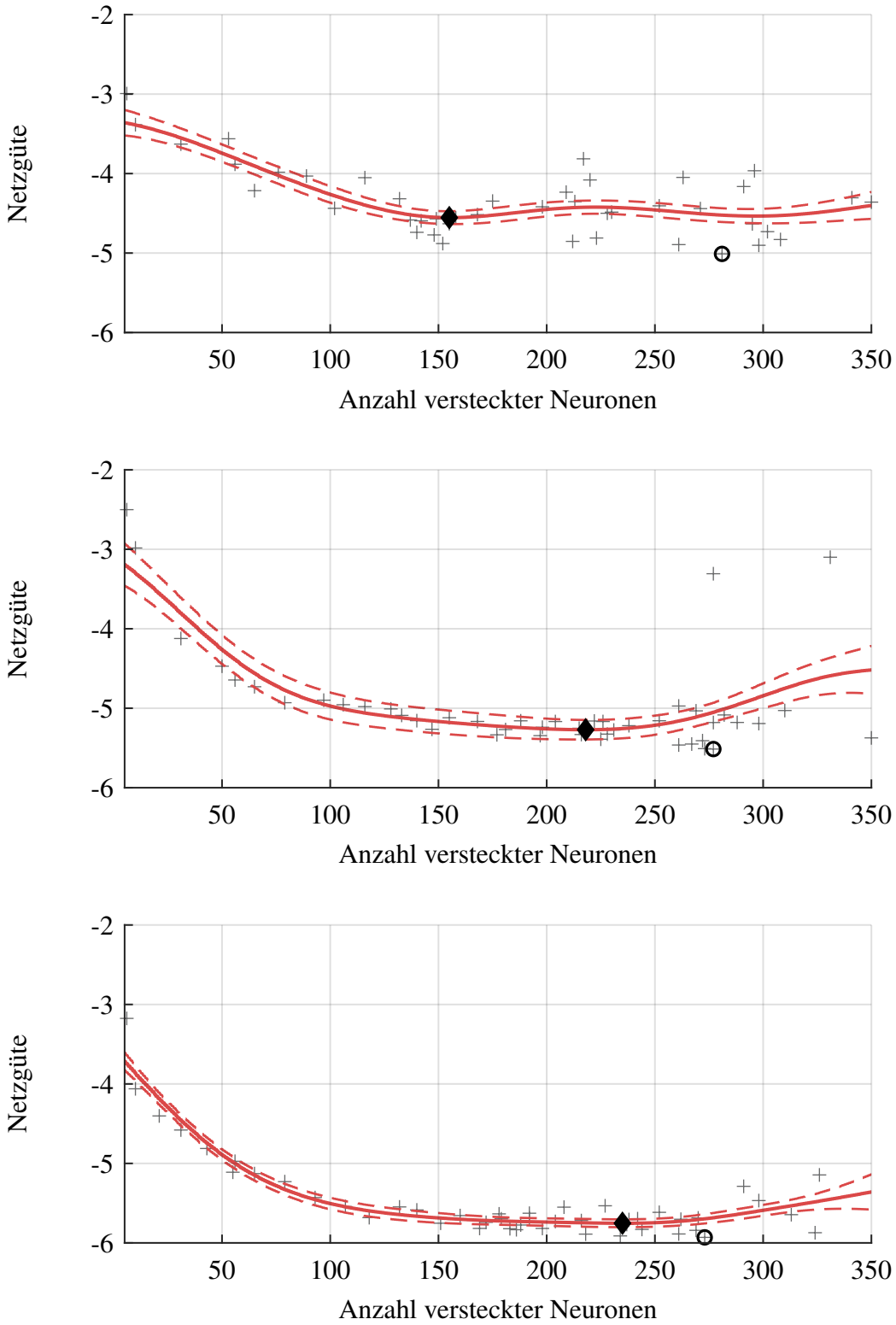

Bild A3-3: *Ergebnisse der BO für PGRNN/−10%/Exp.1, PGRNN/−10%/Exp.2 und PGRNN/−10%/Exp.3 (von oben) zur Modellierung des Viertelfahrzeugs.*

A3.1.3 PGRNN/-75%

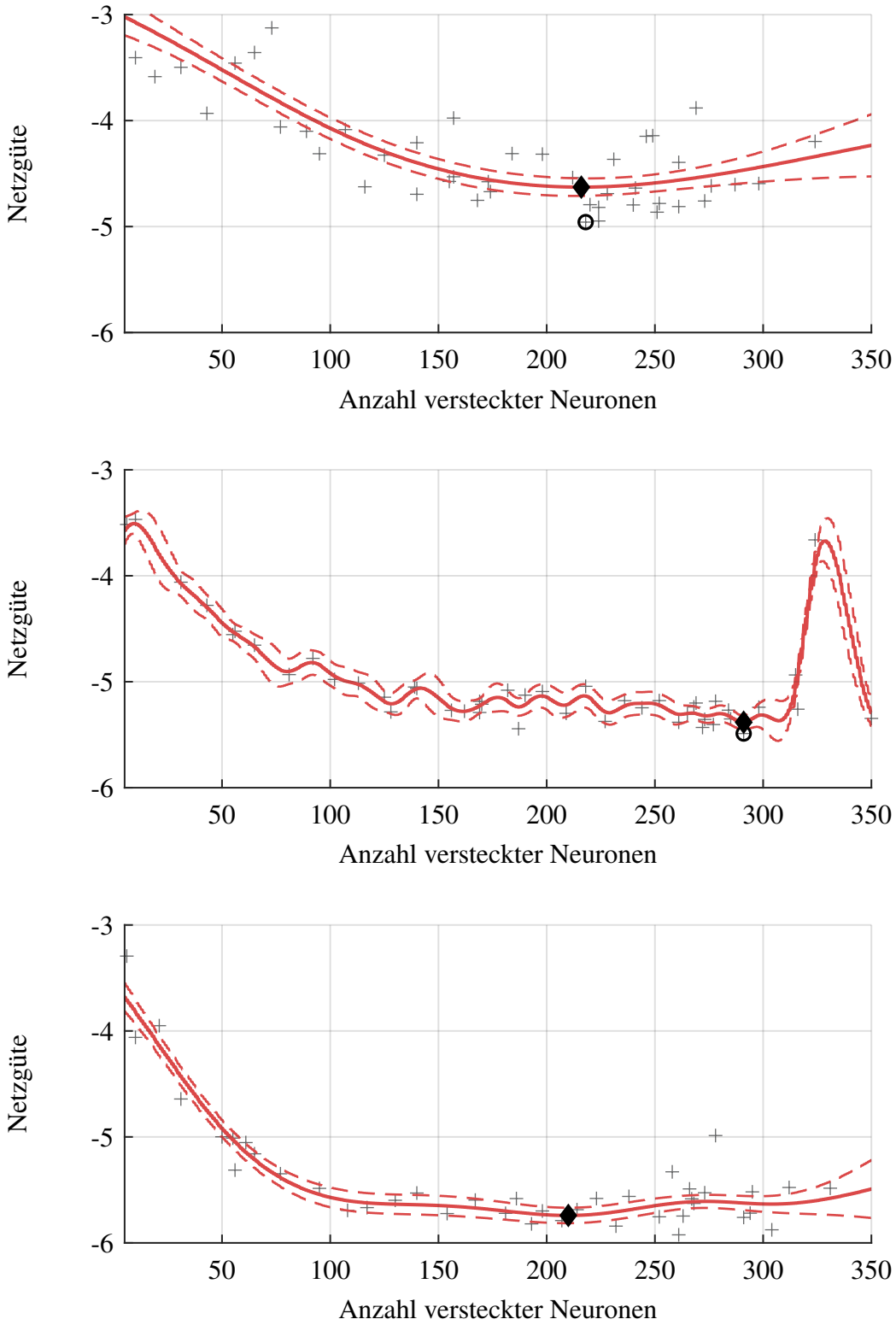

Bild A3-4: Ergebnisse der BO für PGRNN/−75%/Exp.1, PGRNN/−75%/Exp.2 und PGRNN/−75%/Exp.3 (von oben) zur Modellierung des Viertelfahrzeugs.

A3.1.4 PGRNN/-10% mit Energieerhaltung

Die in diesem Kapitel angeführten Abbildungen bilden die BO im Fall des Viertelfahrzeugmodells mit der Nebenbedingung Energieerhaltung ab. Dabei wurden in allen abgebildeten Fällen 10 Trainingssequenzen à 1s verwendet.

Zur Erinnerung: Die Netzgüte (4-1) **ist zu minimieren!**

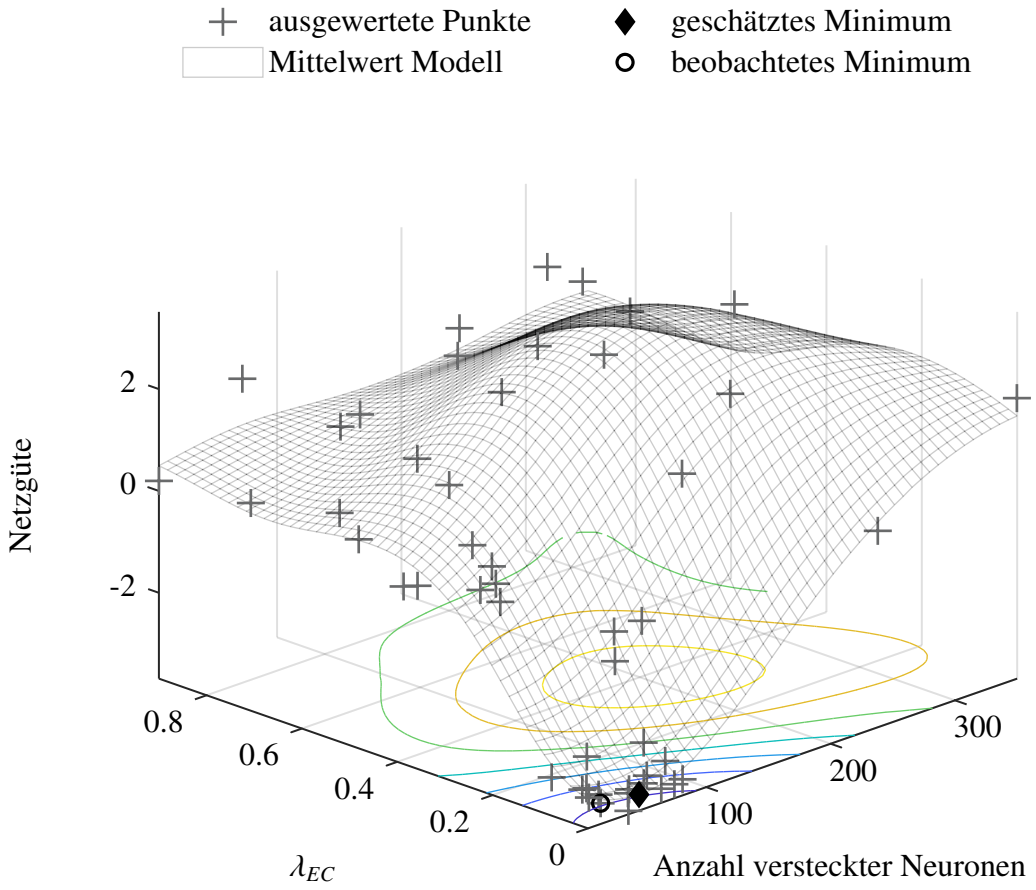

Bild A3-5: *Ergebnisse der BO für das RNN mit Energieerhaltung zur Modellierung des Viertelfahrzeugs.*

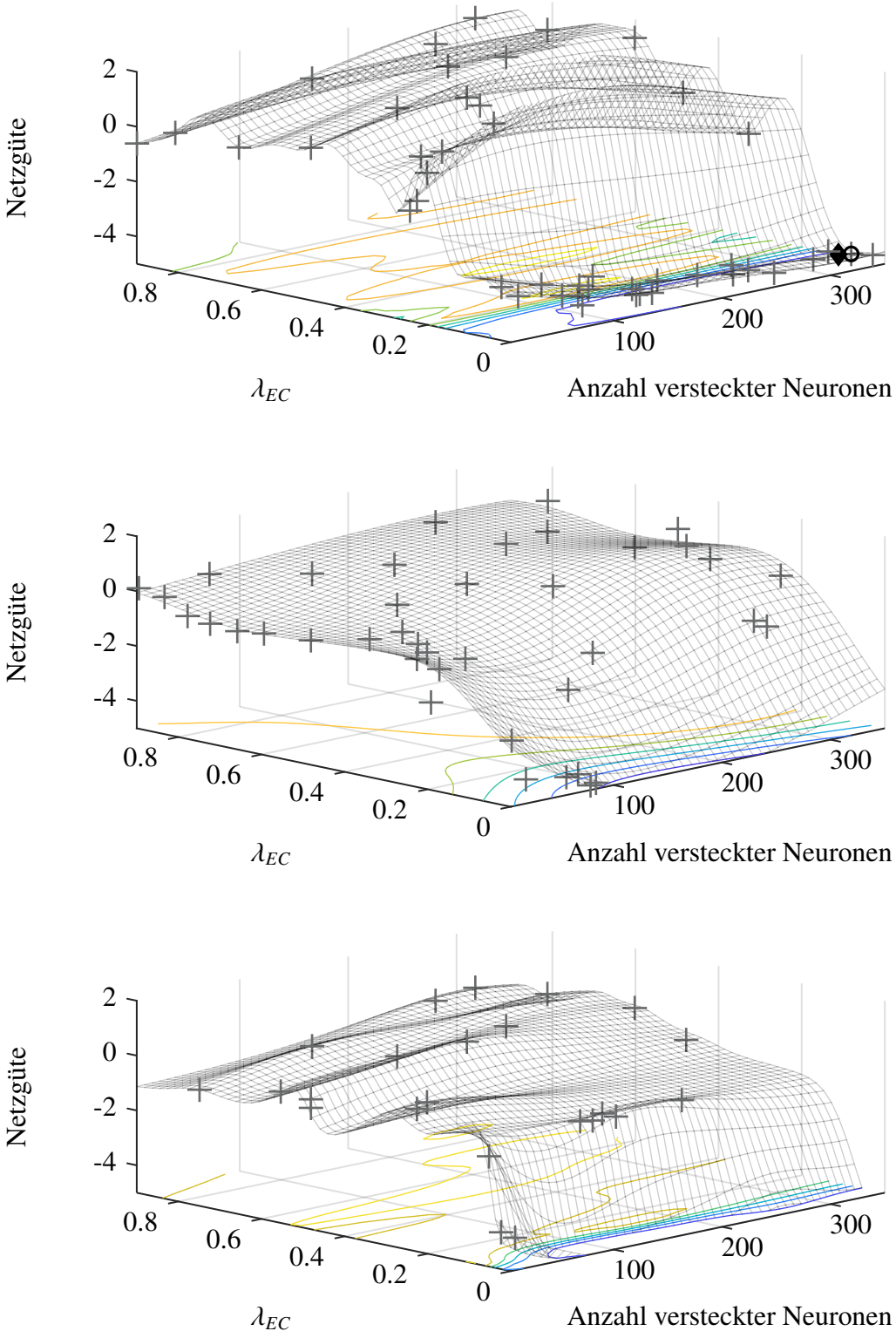

Bild A3-6: Ergebnisse der BO für PGRNN/−10%/Exp.1, PGRNN/−10%/Exp.2 und PGRNN/−10%/Exp.3 (von oben) jeweils mit Energieerhaltung zur Modellierung des Viertelfahrzeugs.

A3.2 Lorenz-Attraktor

Die in diesem Kapitel angeführten Abbildungen bilden die BO im Fall des Lorenz-Attraktors ab. Dabei wurden in allen abgebildeten Fällen 24 Trainingssequenzen à 3s verwendet.

Zur Erinnerung: Die Netzgüte (4-1) **ist zu minimieren!**

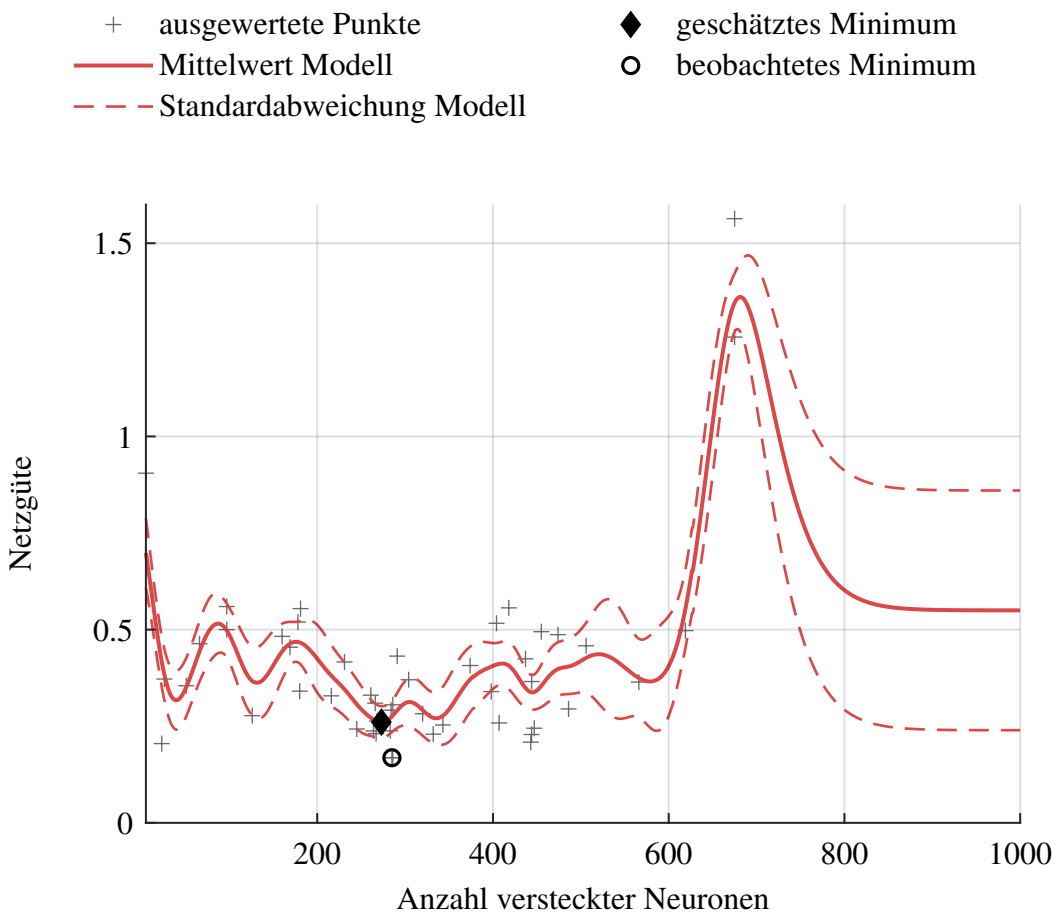

Bild A3-7: Ergebnisse der BO für das RNN zur Modellierung des Lorenz-Attraktors.

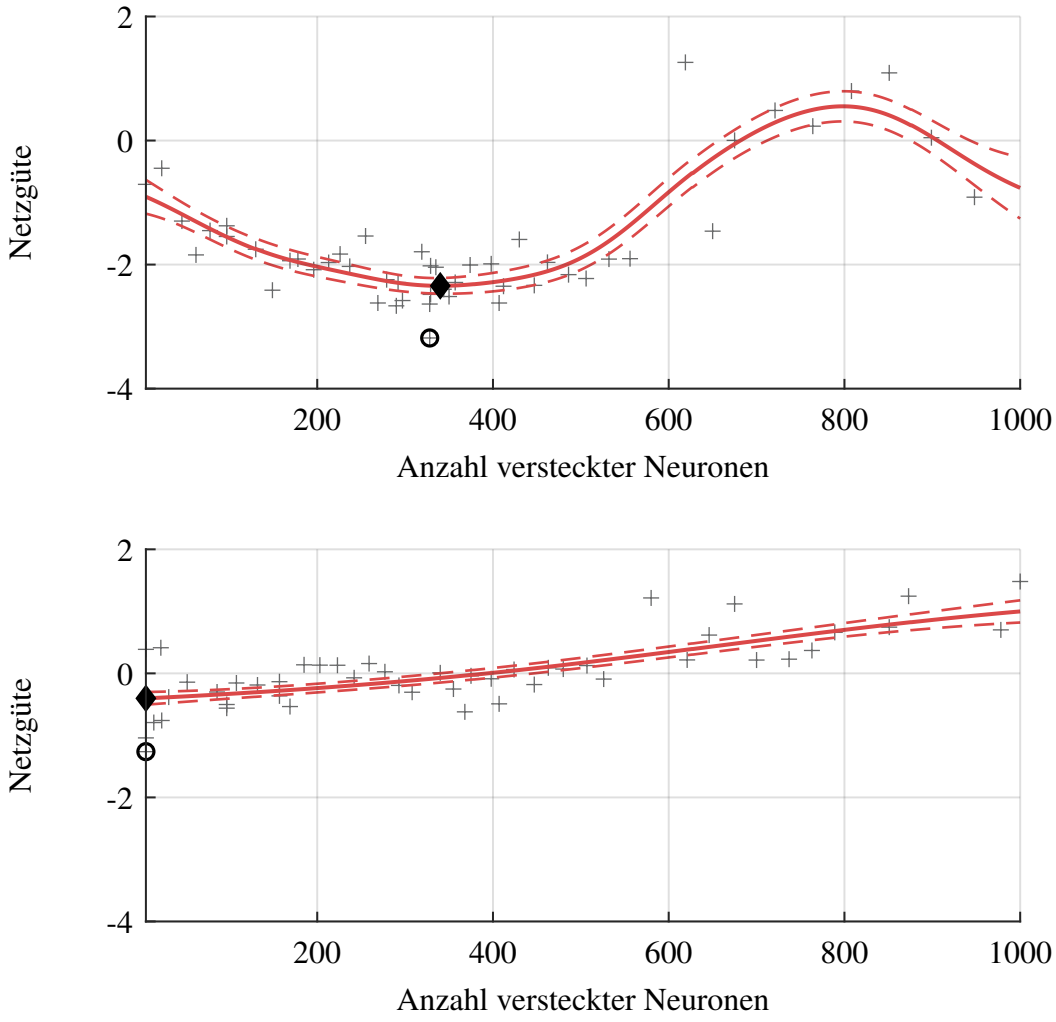

Bild A3-8: Ergebnisse der BO für PGRNN/−10% und PGRNN/−75% (von oben) zur Modellierung des Lorenz-Attraktors.

A4 Programmcode: ODE-Solver

Für alle im Rahmen dieser Arbeit untersuchten rekurrenten Netzarchitekturen wurde ein erweiterter ODE-Solver in MATLAB entwickelt und implementiert. Dieser basiert im Kern auf dem expliziten Runge-Kutta-Verfahren, das zur zeitlichen Integration der Dynamikgleichung dient. Die Dynamikgleichung wird dabei durch das respektive RNN bzw. PGRNN modelliert. Da die Auswertung der rekurrenten Modelle nicht für jeden Zeitschritt unabhängig voneinander erfolgt, sondern stets die vorausgegangene Trajektorie zu berücksichtigen ist, wird im Rahmen des entwickelten Solvers `ode4rnn` durch Interpolation jeweils die entsprechende Datengrundlage generiert und dem Algorithmus zugespeist.

```
% Copyright 2021 Oliver Schoen
function [T, D] = ode4rnn(predictFcn, controlFcn, tSpan,
   stepSize4Rnn, x_0, maxSequenceLength, UsePlotting)
if nargin < 7
    UsePlotting = false;
end

% Time steps for the RNN
T4ODE = tSpan(1):stepSize4Rnn:tSpan(2);

% Init closure object
[odeFcn, outputFcn] = closure(predictFcn, controlFcn,
   maxSequenceLength, tSpan, stepSize4Rnn, UsePlotting);

% Pass custom output function through odeset
opts = odeset('OutputFcn', outputFcn);

% Solve ODE
[T, X] = ode45(odeFcn, T4ODE, x_0, opts);

% Return results
U = controlFcn(T');
D = [X'; U];
end

% Function closure for broadcasting driver and time
   histories to several
% functions in use
function [odeFcn, outputFcn] = closure(predictFcn,
   controlFcn, ...
    maxSequenceLength, tSpan, stepSize4Rnn, UsePlotting)
% Init driver and time history
driverHist = dlarray([], 'CT');
```

```
30 timeHist = [];
31
32 % Return function handles
33 odeFcn = @odeFcn_;
34 outputFcn = @outputFcn_;
35
36 %% Functions
37     % Custom output function for fetching accepted
       intermediary data from
38     % the ode45 solver
39     function status = outputFcn_(T, X, flag)
40         status = [];
41
42         if isempty(flag) % Valid data incoming
43             % Calculate control for new time steps
44             U = controlFcn(T);
45
46             % Add new data to history
47             driverHist = [driverHist, [X; U]];
48             timeHist = [timeHist, T];
49
50             % Check length of history
51             [driverHist, timeHist] = historyLengthChecker(
                   driverHist, ...
52                  timeHist);
53         end
54     end
55
56     % Custom ode function calculating the response of the
       RNN
57     function dxdt_k = odeFcn_(t_k, x_k)
58         % Calculate network response (predict method
              normalizes input data)
59         % But first check if driver history contains any
              data
60         if isempty(driverHist)
61             % History is empty -> generate auxiliary data
62             u_k = controlFcn(t_k);
63             d_k = dlarray([x_k; u_k], 'CT');
64
65             % Calculate response
66             dxdt_k = predictNetwork(d_k);
67         else
68             % History contains data -> interpolate history
69             interpDriverHist = interpolateHistory(t_k, x_k)
                   ;
70
```

```
71              dxdt_k = predictNetwork(interpDriverHist);
72          end
73      end
74
75      % Function takes sequence of network drivers (dlarray)
           and calculates
76      % network response while giving back the response for
           the last time
77      % step (double)
78      function y_k = predictNetwork(X)
79          Y = double(gather(extractdata(predictFcn(X))));
80          y_k = Y(:, end);
81      end
82
83      % Due to ode45 not adhering to the stepSize4Rnn time
           grid, in order of
84      % supplying network with correctly spaced history,
           history has to be
85      % anchored by newest time stamp and interpolated to
           meet the grid
86      function interpHistory = interpolateHistory(t_k, x_k)
87          % Check if interpolation is necessary
88          if t_k == timeHist(end) + stepSize4Rnn
89              % No interpolation necessary because matches
                   stepSize4Rnn
90              interpHistory = driverHist;
91          else
92              % Interpolation necessary because doesn't match
                    stepSize4Rnn
93              % Create time grid to meet stepSize4Rnn
94              timeGrid = flip(t_k:-stepSize4Rnn:tSpan(1));
95
96              % Check length of history
97              timeGrid = timeHistoryLengthChecker(timeGrid);
98
99              % Build current driver
100             u_k = controlFcn(t_k);
101             d_k = [x_k; u_k];
102
103             % Interpolate
104             T_temp = [timeHist, t_k];
105             D_temp = [double(gather(extractdata(driverHist)
                   )), d_k];
106             temp = interp1(T_temp', D_temp', timeGrid', '
                   pchip', ...
107                 'extrap')';
108
```

```
109             % Plot data
110             if UsePlotting
111                 plot(T_temp', D_temp', 'k-o', 'MarkerSize',
                        5, ...
112                     'MarkerFaceColor', 'k', 'LineWidth',
                            1.5)
113                 hold on
114                 xlim(tSpan)
115                 ylim([-5, 8])
116                 grid on
117                 plot(timeGrid', temp', 'c--o', 'MarkerSize'
                        , 10, ...
118                     'LineWidth', 1.5)
119                 hold off
120                 drawnow limitrate
121             end
122
123             interpHistory = dlarray(temp, 'CT');
124         end
125     end
126
127     % Function checks time history to stay below the
           maximum sequence
128     % length allowed
129     function historyT = timeHistoryLengthChecker(historyT)
130         if length(historyT) > maxSequenceLength
131             % Pop oldest entries
132             historyT = historyT(end-maxSequenceLength+1:end
                    );
133         end
134     end
135
136     % Function checks time and driver histories to stay
           below the maximum
137     % sequence length allowed
138     function [historyD, historyT] = historyLengthChecker(
           historyD, ...
139             historyT)
140         if length(historyT) > maxSequenceLength
141             % Pop oldest entries
142             historyD = historyD(:, end-maxSequenceLength+1:
                    end);
143             historyT = historyT(end-maxSequenceLength+1:end
                    );
144         end
145     end
146 end
```