Institut für Fluiddynamik und Schiffstheorie
Prof. Dr. Moustafa Abdel-Maksoud

Masterarbeit im Studiengang Schiffbau und Meerestechnik

Entwicklung von Methoden zum KI-gestützten Propellerentwurf im frühen Entwurfsstadium

Maike Strecker
21486180

10. August 2022

Erstprüfer: Prof. Dr.-Ing. Moustafa Abdel-Maksoud
Zweitprüfer: DSc. (Tech.) / PhD. Franz von Bock und Polach

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Hamburg, 10.08.2022 Maike Strecker

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Symbol	Einheit	Beschreibung
Lateinische Symbole		
A	m^2	Fläche
Ae/Ao	-	Flächenverhältnis
c_p	-	Druckbeiwert
c_{th}	-	Schubbelastungsgrad
D	m	Durchmesser
F_n	-	Froude-Zahl
J	-	Fortschrittsgrad
k_T	-	Momentenbeiwert
k_Q	-	Schubbeiwert
n	$\frac{1}{s}$	Drehzahl
p	Pa	Drücke
P	W	Leistung
P/D	-	Steigung der Propellerflügel
R	m	Radien
R_n	-	Reynolds-Zahl
R_T	kN	Widerstand
T	kN	Schub
t	-	Sogziffer
V, v	$\frac{m}{Ns}$	Geschwindigkeit
w	-	Nahcstromziffer
Z	-	Flügelzahl
Griechische Symbole		
η	-	Wirkungsgrad
μ	-	Kinematische Viskosität
ρ	$\frac{kg}{m^3}$	Dichte
σ_v	-	Kavitationszahl
τ_c	-	Koeffizient des mittleren Drucks

Abkürzungen

Adam	Adaptive moment estimation
ADDM	Advanced Data Driven Model
BA	Bland-Altman Analyse
BGD	Batch Gradient Descent
BEM	Boundary Element Method
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DDM	Data Driven Model
FFT	Fast Fourier Transformation
FWH	Ffowcs Williams - Hawkings Gleichung
HM	Hybrides Modell
KI	Künstliche Intelligenz
KNN	Nächste-Nachbarn-Klassifikation
LCB	Längs-Schwerpunktlage des Auftriebs
MBGD	Mini-Batch Gradient Descent
PM	Physikalisches Modell
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SWA	Stochastic Weight Averaging

Konstanten und Umrechnungen

g	9,81 $\frac{m}{s^2}$	Gravitationskonstante
ρ_{SW}	1,025 $\frac{t}{m^3}$	Dichte Seewasser
ρ	1,005 $\frac{t}{m^3}$	Dichte Süßwasser
kn	1 kn = 0,5144 $\frac{m}{s}$	

1 Einleitung

Bereits seit einigen Jahrzehnten nimmt die Leistungsstärke der Computer stetig weiter zu, sodass heute Anwendungen möglich sind, die früher nicht im Bereich der möglichen Rechenkapazitäten lagen. Früh haben sich mit der Entwicklung des Computers Überlegungen zur künstlichen Intelligenz ergeben, die aber nicht umgesetzt werden konnten, da die Rechenleistung fehlte. Seit einigen Jahren ist der Punkt erreicht, an dem ausreichend Rechenkapazität und auch genügend Speichermöglichkeit für Daten zur Verfügung steht. Damit sind die Voraussetzungen für künstliche Intelligenz geschaffen: viel Rechenleistung kombiniert mit ausreichend Daten.[20, S. 8 f.] Die Computer unterstützen immer mehr bei alltäglichen und speziellen Anwendungen, oftmals unbemerkt. Auch im technischen Bereich gibt es immer weitere marktreife Anwendungsmöglichkeiten für KI. So können inzwischen Fahrzeuge im Prinzip autonom fahren und es gibt Sprachassistenten in jedem Smartphone.[6, S. 18 ff.]

Entwurfsprozesse werden im technischen Bereich überwiegend von Menschen durchgeführt, da die Bewertungsgrundlagen eines Entwurfs für Maschinen nicht einfach definiert werden können. Menschen betrachten häufig viele Aspekte eines Entwurfs, benötigen aber auch dementsprechend viel Zeit dafür. Wenn eine künstliche Intelligenz über einen Menschen trainiert solche Prozesse unterstützen oder ganz übernehmen kann, können diese zeitlich deutlich optimiert werden.[2, S. 1 f.]

Diese Arbeit konzentriert sich auf den KI-gestützten Propellerentwurf. Für die Entwürfe kommen vielfach Propellerserien zum Einsatz, deren Daten in Freifahrtversuchen gewonnen worden sind und daher Phänomene wie Kavitation nicht betrachten. Sie sind jedoch schnell anwendbar, um eine erste Propellergeometrie zu erhalten. Für den KI-gestützten Entwurf soll anhand von mit CFD berechneten Freifahrtdiagrammen eine Geometrie und ein Betriebspunkt des Propellers gefunden werden. Dieser KI-gestützte Entwurf dient im ersten Schritt zur Grundlage für nachfolgende Optimierungen durch Menschen, zukünftig ist auch eine vollständige Optimierung über KI denkbar.

Zu Beginn der Arbeit soll ein Überblick über den aktuellen Stand der Forschung bezüglich des KI-gestützten Propellerentwurfes gegeben werden. Dann sollen für den Entwurf verschiedene Methoden der künstlichen Intelligenz verglichen und angewendet werden. Dabei soll insbesondere beachtet werden, ob die Kavitation auf den Propellerflügeln ausreichend abgeschätzt werden kann. Die optimalen Geometrien der KI sollen abschließend mit einer CFD-Rechnung an einem unbekannten Propeller verglichen werden.

2 Theoretischer Hintergrund

In dem folgenden Kapitel wird kurz die Geschichte der künstlichen Intelligenz erläutert und zusammengefasst. Außerdem wird auf die Grundlagen von maschinellem Lernen und neuronalen Netzen eingegangen. Der Entwurf und die Charakteristiken von Propellern werden ebenfalls kurz beschrieben.

2.1 Geschichte der Künstlichen Intelligenz

Bevor die Geschichte der künstlichen Intelligenz kurz zusammengefasst werden kann, stellt sich die Frage nach einer Definition von künstlicher Intelligenz. Der Begriff Intelligenz wird laut Duden definiert als "Fähigkeit [des Menschen], abstrakt und vernünftig zu denken und daraus zweckvolles Handeln abzuleiten"[10]. Damit wird schon angedeutet, dass Intelligenz nicht rein auf den Menschen bezogen ist. Laut Lämmel und Cleve[6] besteht Intelligenz aus mehreren Bestandteilen wie Lernen, Verstehen und Anpassung an veränderte Umgebungen. Künstliche Intelligenz wiederum lässt sich aus verschiedenen Blickwinkeln betrachten: als ein Teil der Informatikwissenschaft, als Teile von Lösungen im Hard- und Softwarebereich oder als gesamte Hard- und Softwaresysteme sowie als tatsächliche künstliche Wesen mit Intelligenz.[6, S. 9 ff.] In der vorliegenden Arbeit wird sich vor allem mit KI im Sinne von Softwarelösungen befasst, welche zur Unterstützung der Menschen genutzt werden.

Die Geschichte der künstlichen Intelligenz geht bis in die 1940er Jahre zurück. Dort begann die erste Welle der Kybernetik, welche unter anderem 1958 zum Perzeptron von *Rosenblatt* führte. So konnte bereits ein einzelnes Neuron trainiert werden. Da jedoch stark kritisiert wurde, dass diese ersten linearen Neuronen die XOR-Funktion nicht abbilden konnten, ebbte diese Welle ab und in den 1980ern begann die zweite Welle des Konnektionismus. Dort sind die Neuronen in mehreren Lagen kombiniert worden und konnten durch Rückwärtspropagation trainiert werden. Der heutige Trend Deep Learning hat in etwa 2006 begonnen und versucht unabhängig von der Inspiration des menschlichen Lernens Anwendungsmöglichkeiten und Verbesserungen im Bereich des maschinellen Lernens zu erzielen, sodass auch andere Lösungsmethoden möglich sind als die klassischen Neuronalen Netze, welche eng dem menschlichen Gehirn nachempfunden sind.[S. 12 ff.][8]

Die wichtigsten Meilensteine der künstlichen Intelligenz sind die Erfindung des Turing-Tests 1943[6, S. 8], bei dem eine Maschine als intelligent eingestuft wird, wenn sie zwischen zwei mit ihr schreibenden Personen den Menschen und die Maschine identifizieren kann, dann die Erfindung des Perzeptron 1958, welches ein einzelnes Neuron im Gehirn abbildete und trainierbar war, und 1996 als das Schachprogramm "Deep Blue" zum ersten Mal den Weltmeister Kasparow sowie 2016 "Alpha GO" in Go den Weltmeister besiegte.[6, S. 19]

Die Unterscheidung in Bereiche kann dabei anhand folgender Kriterien vorgenommen werden: Das Forschungsfeld im Ganzen heißt "Künstliche Intelligenz" und beinhaltet regelbasierte Systeme, wissensbasierte Systeme, maschinelles Lernen und Deep Learning. Die ersten intelligent anmutenden Systeme sind regelbasierte Systeme, wo der Computer anhand von vorher festgelegten Regeln Tätigkeiten ausführt. Ein bekanntes Beispiel dafür ist Schach, welches zwar ein komplexes Regelwerk enthält, jedoch einfach in Programmen abbildbar ist. Die nächste Art der Systeme ist wissensbasiert. Hierbei wird von Menschen ein gewisses Wissen vorgegeben und so verpackt, dass der Computer damit

arbeiten kann. Dieser erstellt dann beispielsweise anhand von aktuellen Verkehrsdaten optimierte Routen. Hierbei ist jedoch immer ein Engpass, dass die Daten von Menschen zur Verfügung gestellt werden müssen. Daraus entwickelt sich das Feld des maschinellen Lernens, bei dem der Computer anhand markierter Daten eigenständig die Zusammenhänge lernt. Dies bedeutet jedoch, dass Menschen immer noch die Daten vorbereiten und gewisse Merkmale auswählen müssen, anhand derer der Computer dann verlässliche Zusammenhänge lernen kann. So entwickelt sich das Feld des Deep Learnings, bei dem der Computer auch selbst nach relevanten Merkmalen in markierten Datensätzen sucht und sich dann die Zusammenhänge beibringt.[20, S. 2 ff.]

In den nächsten zwei Kapiteln soll nun im Detail das klassische maschinelle Lernen und neuronale Netze erklärt und die Unterschiede erläutert werden.

2.2 Maschinelles Lernen

Das Problem des maschinellen Lernens lässt sich wie folgt zusammenfassen: für bekannte Daten X und eine unbekannte Funktion f die den Zusammenhang der Daten darstellt, wird eine Funktion $\hat{f}$ gesucht, welche f abbildet. Dabei beschreibt X eine Menge von Merkmalen und $f(X)$ zum Beispiel Sätze unsortierter Daten oder kontinuierliche Funktionen. Je nach Art der Daten liegen zu X auch Ausgabemerkmale Y vor, also zu den X-Daten zugehörige Werte. Dies ist aber nicht immer der Fall.[15, S. 3 ff.] Grundsätzlich wird die Güte der Annäherung von $\hat{f}$ an f über eine Verlustfunktion $L(f(X), \hat{f}(X))$ bewertet, die je nach Art der vorhandenen Daten ebenfalls variiert. Meistens beschreibt sie aber die Abweichung der Vorhersagen $\hat{f}(X)$ zu den originalen Werten $f(X)$. Näheres zu Verlustfunktionen wird in Kapitel 2.2.2 beschrieben. Maschinelles Lernen lässt sich aufteilen in überwachtes Lernen, wo zu den Merkmalen X ein Wert Y gehört, sowie unüberwachtes Lernen, wo nur die Merkmale X vorliegen. Zudem lässt sich unterscheiden in finite oder kontinuierliche Datensätze.[15, S. 6] Diese vier Ausprägungen werden im Folgenden weiter erläutert.

Clustering

Beim Clustering handelt es sich um einen unüberwachten Lernalgorithmus mit finiten Datensätzen. Dabei sucht der Algorithmus selbstständig nach zusammengehörigen Klassen und klassifiziert die Datenpunkte anschließend nach diesen Mustern. Ein Beispiel für Clustering ist das Zuordnen von Fehlerursachen zu verschiedenen Festplatten.[17, S. 5]

Da beim Clustering keine Ausgabemerkmale Y für die Datenpunkte X vorliegen, kann auch keine Verlustfunktion berechnet werden. Die Bewertung erfolgt je nach verwendetem Algorithmus zum Beispiel anhand der Minimierung der Abstände von den einzelnen Punkten zu ihrem zugehörigen Cluster-Mittelpunkt.[15, S. 149 ff.]

Dichteschätzung

Die Dichteschätzung ist ein unüberwachter Lernalgorithmus mit kontinuierlichen Daten. Dabei wird eine Dichtefunktion über die vorhandenen Daten erstellt um die grundlegende Verteilung der Daten zu ermitteln.

Da auch bei der Dichteschätzung keine Ausgabemerkmale Y für die Datenpunkte X vorliegen, kann auch hier keine Verlustfunktion berechnet werden. Das Risiko der Falscheinordnung wird jedoch abgeschätzt, indem ein Vergleich zu einer zufälligen Zuordnung der Datenpunkte gemacht wird. [15, S. 35 ff.]

Klassifikation
Bei der Klassifikation handelt es sich um einen überwachten Lernalgorithmus mit finiten Datensätzen.[15, S. 6] Zu den Problemen gehören beispielsweise die Vorhersage, ob ein Tumor bösartig ist (Binärwert 1) oder nicht (Binärwert 0). In diesem Fall handelt es sich um Zwei-Klassen-Klassifikation mit zwei Ausgabewerten, es gibt aber auch Mehrklassenklassifikation wo verschiedene Ausgabeklassen möglich sind. Dabei ist die Ausgabe aus dem Algorithmus jedoch immer ein diskreter Wert und die Wahrscheinlichkeiten werden zum Beispiel auf 0 oder 1 auf- bzw. abgerundet. Dadurch ist eine eindeutige Klassifizierung möglich.[17, S. 4]

Die Verlustfunktion $L(X)$ lässt sich für Klassifikationsprobleme über den Null-Verlust berechnen.[15, S. 6]

$$L(f(X), \hat{f}(X)) = 0 \text{ für } f(X) = \hat{f}(X) \tag{1}$$

$$L(f(X), \hat{f}(X)) = 1 \text{ für } f(X) \neq \hat{f}(X) \tag{2}$$

Regression
Die Regression ist ein überwachter Lernalgorithmus mit kontinuierlichen Datensätzen. [15, S. 6] Im Unterschied zur Klassifikation werden hier keine Klassen vorgegeben, sondern kontinuierliche Werte für $\hat{f}$ ausgegeben. Zu den möglichen Problemen zählt beispielsweise die Vorhersage von Verkaufszahlen im nächsten Quartal auf Basis der vorherigen Verkaufszahlen.[17, S. 4]

Die Verlustfunktion $L(X)$ lässt sich für Regressionsprobleme zum Beispiel über die quadratische Abweichung berechnen.[15, S. 6] Auf weitere Verlustfunktionen wird in Kapitel 2.2.2 eingegangen.

$$L(f(X), \hat{f}(X)) = (f(X) - \hat{f}(X))^2 \tag{3}$$

2.2.1 Klassifikation

Zwei weitverbreitete Klassifikationsarten sind Support Vector Machine (SVM) und die Nächste-Nachbarn-Klassifikation (k-nearest neighbor, KNN). Für SVM werden die Daten für beispielsweise zwei verschiedene bekannte Klassifikationskategorien mit einer Linie in diese zwei Bereiche aufgeteilt, wobei die Linie die Kategorien mit dem höchstmöglichen Abstand zu den Datenpunkten trennt. Dies kann auch im mehrdimensionalen Raum oder für mehrere Klassifikationskategorien geschehen, dann werden die Daten durch Hyperebenen oder entsprechend mehr Linien getrennt. [17, S. 177 ff.]

Bei KNN sind die Kategorien der zu klassifizierenden Daten ebenfalls bekannt, allerdings geschieht die Zuordnung zu einer Kategorie nicht anhand einer Trennlinie, sondern anhand der k nächsten Nachbarn. Das heißt wenn man einen neuen Datenpunkt zuordnen will, betrachtet man beispielsweise die drei Datenpunkte, die am dichtesten daran sind, und wählt die Kategorie für den unbekannten Punkt mit den meisten Übereinstimmungen. So können je nach Anzahl der k Nachbarn unterschiedliche Vorhersagen getroffen werden. Je höher k dabei ist, desto weniger anfällig ist das Modell für Rauschen in den Daten. Um das optimale k für KNN zu finden, kann es zum Beispiel mit k-facher Kreuzvalidierung und der maximalen Treffergenauigkeit gefunden werden. Dabei wird der Datensatz in k Teildatensätze aufgeteilt, und jeder dieser Datensätze wird einmal zur Fehlerberechnung genutzt, während mit allen anderen trainiert wird. So lässt sich eine gute Vorhersage für das optimale Modell für den gesamten Datensatz erstellen.[17, S. 205 ff.]

2.2.2 Regression

Da die Regression für die vorliegende Arbeit von größerer Bedeutung ist, wird diese Variante des maschinellen Lernens noch näher beschrieben. Es gibt verschiedene Arten der Regression. Random Forest Regression funktioniert vom Prinzip her ähnlich wie Random Forest Klassifikation, bei der über mehrere Entscheidungsbäume die passende Klasse vorhergesagt wird. Bei der Regression wird jedoch anhand der Entscheidungsbäume eine numerische Vorhersage aus den Eingangsdaten der Blattknoten erstellt. Die Ergebnisse aller Entscheidungsbäume werden dann gemittelt.[15, S. 32 ff.]

Weitere klassische Arten der Regression lassen sich aufteilen in lineare und polynomiale Regression. Zusätzlich kann man diese in Regressionen mit einer oder mehreren Eingangsvariablen unterscheiden. Bei der linearen Regression wird zwischen den Variablen ein linearer Zusammenhang angenommen. Dabei kann eine Ausgangsvariable von einer oder mehreren Eingangsvariablen linear abhängig sein. Bei mehreren Eingangsvariablen spricht man von linearer Mehrfachregression.[17, S. 120] Als Formel lässt sich die lineare Regression für M Eingangsvariablen wie folgt schreiben: [15, S. 16]

$$f(X_i) = \Theta_0 + \Theta_1 X_1 + ... + \Theta_M X_M = \Theta_0 + \sum_{m=1}^{M} \Theta_m X_m \tag{4}$$

Bei der polynomialen Regression wird zwischen Eingangs- und Ausgangsvariable ein polynomialer Zusammenhang beliebigen Grades angenommen. Auch hier kann in Regression von einer oder mehreren Eingangsvariablen unterschieden werden. Für polynomiale Mehrfachregression zweiten Grades und mit drei Eingangsvariablen verändert sich Gleichung 4 dann zu Gleichung 5 und es ergeben sich für diesen Fall zehn Regressionsparameter Θ.[17, S. 145]

$$f(X) = \Theta_0 + \sum_{m=1}^{3} (\Theta_{1m} X_m + \Theta_{2m} X_m^2) + \Theta_1 X_1 X_2 + \Theta_2 X_1 X_3 + \Theta_3 X_2 X_3 \tag{5}$$

Lineare und polynomiale Regression können in Python direkt mit der Bibliothek `scikit-learn`[9] berechnet werden. Für die polynomiale Regression wird die lineare Regression verwendet, aber mit dem Modul `PolynomialFeatures` zuvor zusätzliche polynomiale Merkmale erzeugt. Diese können dann wieder in dem Modul `LinearRegression` verwendet werden.[17, S. 144 f.] `scikit-learn` verwendet dabei zur Optimierung der Parameter Θ die mittlere quadratische Abweichung (siehe nächster Abschnitt), diese wird also iterativ minimiert während die Parameter verändert werden. Bei polynomialer Regression stellt sich die Frage, welchen Grad diese zur besten Annäherung an die Daten haben soll. Dazu wird in Abschnitt 2.4 auf die Verzerrung-Varianz-Problematik eingegangen. Diese Problematik ist für alle Problemstellungen des maschinellen Lernens von Bedeutung. Zuvor wird noch kurz auf verschiedene Verlustfunktionen und Datensätze eingegangen.

Verlustfunktionen für Regression

Eine häufig genutzte Verlustfunktion ist die quadratische Abweichung. Dafür werden von den echten Datenpunkten $f(X)$ die vorhergesagten Werte $\hat{f}(X)$ abgezogen, dieser Term dann quadriert und aufsummiert. Wird dann noch durch die Gesamtanzahl der Datenpunkte geteilt, erhält man die mittlere quadratische Abweichung (MSE, mean

squared error). Ebenfalls häufig verwendet wird der absolute Fehler. Hier wird der Betrag der Differenz aus den echten Datenpunkten $f(X)$ und den vorhergesagten Werten $\hat{f}(X)$ gebildet. Mittelt man diesen ebenfalls, erhält man die mittlere absolute Abweichung (MAE, mean absolute error). Die Formeln sind in Gleichung 6 für den MSE und in Gleichung 7 für den MAE gezeigt.[15, S. 17][20, S. 153 f.]

$$L(f(X), \hat{f}(X)) = \frac{1}{N} \sum_{n=1}^{N} (f(X_n) - \hat{f}(X_n))^2 \tag{6}$$

$$L(f(X), \hat{f}(X)) = \frac{1}{N} \sum_{n=1}^{N} |(f(X_n) - \hat{f}(X_n))| \tag{7}$$

Eine weitere ähnliche Möglichkeit den Verlust zu bewerten ist über das Bestimmtheitsmaß R^2. Dabei setzt sich der Wert aus der durch die Regression bestimmten Quadratsumme RSS und der totalen Quadratsumme TSS zusammen. Er gibt prozentual an, wie gut die Regression zu den Daten passt.[17, S. 105 f.].

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{n} (f(X_i) - \hat{f}(X_i))^2}{\sum_{i=1}^{n} (f(X_i) - \overline{f(X)})^2} \tag{8}$$

2.3 Datensätze

Für das Training von neuronalen Netzen und anderen Modellen, bei deren Datensätzen die Ergebnisse Y bekannt sind, ist eine vorherige Aufteilung der Daten wichtig, die sowohl das Training als auch die Auswertung ermöglicht. Dazu werden die zu verarbeitenden Daten in Trainingsdaten, Validierungsdaten und Testdaten aufgeteilt. Mit den Trainingsdaten werden die Modelle trainiert, dies sind also die Daten, welche direkt das Ergebnis des Modells beeinflussen. Die Validierungsdaten werden zur Anpassung der Hyperparameter wie der Lernrate verwendet und wählen aus den trainierten Modellen das optimale Modell aus. Mit den Testdaten wird zum Abschluss die Leistung des Modells bei der Vorhersage vollkommen unbekannter Daten bewertet. Dabei sollte insgesamt darauf geachtet werden, dass die Verteilung der Daten in allen drei Datensätzen vergleichbar ist.[20, S. 156 f.]

Sollten wenige Daten zur Verfügung stehen oder die Daten in den Datensätzen stark voneinander abweichen, ist k-fache Kreuzvalidierung eine gute Möglichkeit um trotzdem das optimale Modell auswählen zu können. Dabei wird der Datensatz in k Teile aufgeteilt, und jeder Teildatensatz wird als Validierungsdatensatz verwendet, während alle anderen Datensätze Trainingsdaten sind. Dies wird k-fach wiederholt und die Leistung dann über alle k Trainingsläufe gemittelt. Das Training erfolgt dann mit dem so ermittelten besten Modell und wird auf dem gesamten Datensatz durchgeführt. [17, S. 216]

2.4 Verzerrung und Varianz / Unter- und Überanpassung

Die Problematik von Verzerrung und Varianz tritt bei allen Annäherungsproblemen auf. Die grundlegende Problematik lässt sich mit Unter- und Überanpassung beschreiben: wenn das Modell die vorgegebenen Werte zu genau trifft, kann es schlecht zwischen den Werten interpolieren und bildet damit die Trainingsdaten zu genau ab. Wenn es die Werte gar nicht trifft, bildet es die Trainingsdaten kaum ab und die Vorhersagen sind

ebenfalls ungenau.[15, S. 97 f.] Gut vorstellen lässt sich die Problematik am Beispiel der Regression: ein in der Realität quadratischer Verlauf lässt sich kaum mit einer linearen Funktion sinnvoll annähern (Unteranpassung), aber bei einer Funktion mit 8er-Potenz trifft die Funktion die exakten Punkte und schwingt dazwischen durch (Überanpassung), vergleiche Abbildung 1. Dort sind neun Punkte mit zufälligen Fehlern aus einer quadratischen Funktion erzeugt worden, welche dann mit einer linearen, einer quadratischen und einer polynomialen Regression 8. Grades verbunden worden sind.

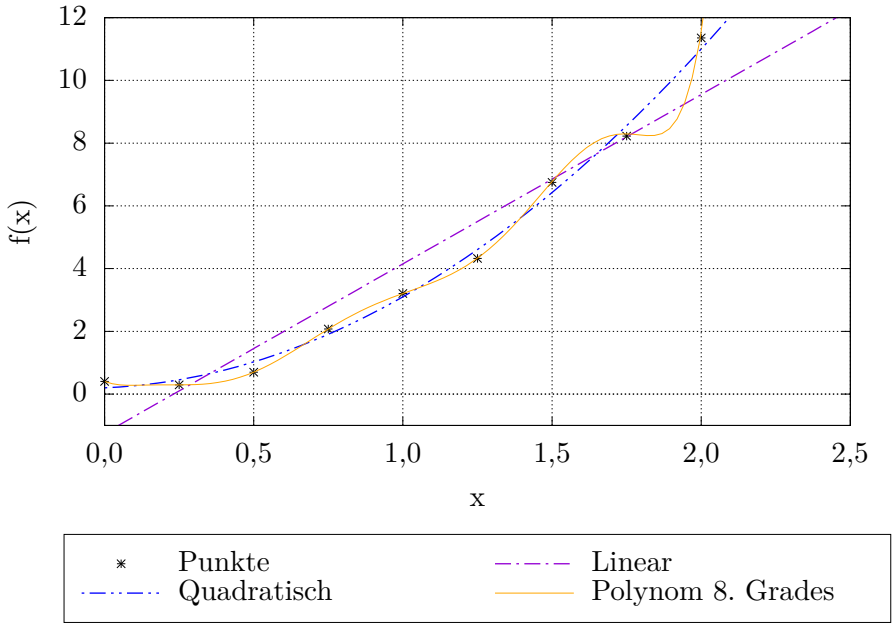

Abbildung 1: Über- und Unteranpassung am Beispiel von verschiedenen Regressionen zu neun Punkten

Mittlere quadratische Abweichung bei Regression

Der zuvor genannte Verlustterm der mittleren quadratischen Abweichung Gleichung 6 lässt sich nun für Regressionsprobleme noch weiter aufteilen, indem man mehrere Regressionen $\hat{f}^l(X)$ durchführt und dadurch zusätzlich zur gesamten Verlustfunktion noch die Verzerrung und die Varianz berechnet. Dabei ist in Gleichung 9 der erste Term der Summe der Verlust, welcher in den Daten selbst liegt, und somit durch die Regressionsmethode nicht beeinflusst werden kann. Der zweite Term wird als quadrierte Verzerrung bezeichnet und berechnet den Unterschied zwischen den realen Daten zu dem Mittel der mehrfach vorhergesagten Daten. Der letzte Term heißt Varianz und beschreibt den Unterschied zwischen den mehrfach vorhergesagten Daten zu dem Mittelwert dieser vorhergesagten Daten. [15, S. 98] [3, S. 149 ff.]

$$L(f(X), \hat{f}^l(X)) = \frac{1}{N}\sum_{n=1}^{N}\left((Y - f(X_n))^2 + (f(X_n) - \overline{\hat{f}^l(X_n)})^2 + (\hat{f}(X_n) - \overline{\hat{f}^l(X_n)})^2\right) \quad (9)$$

$$\text{mit } \overline{\hat{f}^l(X_n)} = \frac{1}{L}\sum_{l=1}^{L}\hat{f}^l(X_n) \quad (10)$$

Zusammenhang zu Über- und Unteranpassung

Der Zusammenhang des Fehlers zu Über- und Unterpassung lässt sich auch mit dem Beispiel in Abbildung 1 zeigen. In Tabelle 1 sind die berechneten Werte für den unveränderlichen Verlust, die Verzerrung und die Varianz gezeigt. Dabei ist zu erkennen, dass die Verzerrung mit steigendem Polynomgrad sinkt und die Varianz steigt. Dennoch gibt es einen minimalen Gesamtfehler bei einer quadratischen Annäherung, da im MSE die Summe aus allen Anteilen gebildet wird. So kann bei einem Trainingsprozess durch Minimieren des MSE das Optimum aus Über- und Unteranpassung gefunden werden.[3, S. 149 ff.] Ein höherer Polynomgrad bedeutet nicht unbedingt bessere Vorhersagen für noch unbekannte Daten.[20, S. 21 f.]

Tabelle 1: Veränderung von Verzerrung, Varianz und des MSE

	Linear	Quadratisch	Polynom 8. Grades
Verzerrung	1.046	0.067	0.072
Varianz	0.003	0.009	0.046
Unveränderlicher Fehler	0.063	0.063	0.063
MSE	1.112	0.139	0.182

In Abbildung 2 sind die Varianz, die Verzerrung und der Verlust graphisch aufgetragen. Zusätzlich sind Trendlinien für die Varianz und Verzerrung eingefügt. Auch hier lässt sich erkennen, dass die Verzerrung sinkt, während die Varianz steigt. Das Optimum bei Grad zwei ist ebenfalls zu erkennen.

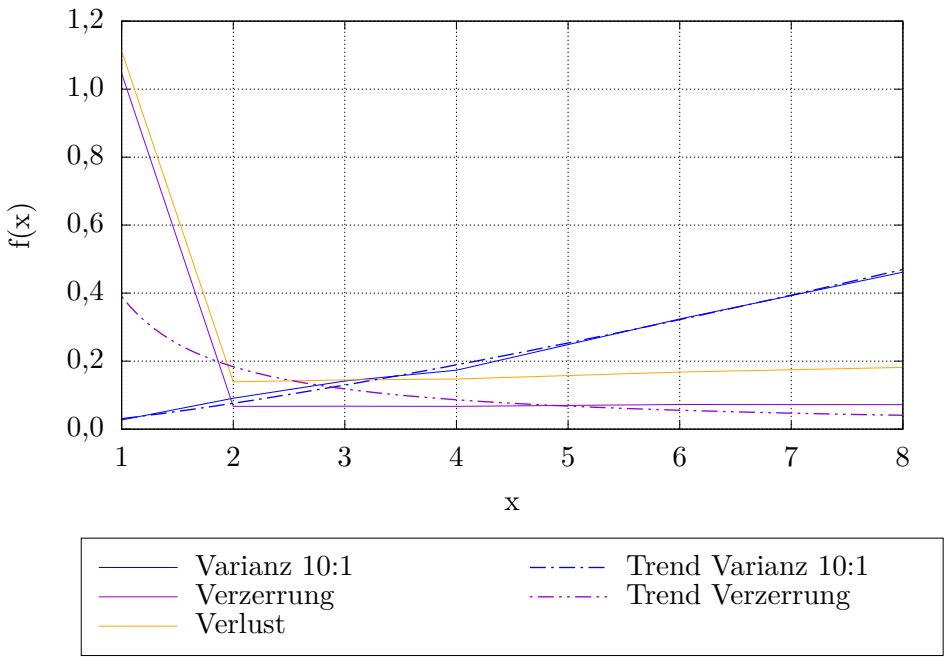

Abbildung 2: Verlauf von Verlust, Varianz und Verzerrung über polynomischen Grad

2.5 Neuronale Netze

Bei neuronalen Netzen handelt es sich wie eingangs beschrieben um grob den Neuronen im Gehirn nachempfundenen Strukturen. Ein Neuron im Gehirn verarbeitet Eingangsreize und leitet sie entweder weiter oder nicht, der Ausgang ist also binär. Damit das Neuron einen Reiz weiterleitet, muss eine gewisse Schwelle an Aktivierung überschritten werden. Dieser Prozess soll nun im Computer abgebildet werden. Dazu werden beispielhaft ein Neuron mit drei Eingängen e1, e2 und e3, einer sogenannten BIAS-Einheit und einem Ausgang a verwendet, siehe Abbildung 3. Zur Verbindung werden zwischen Eingang und Ausgang die Gewichte w1, w2 und w3 eingefügt, sowie Θ für die BIAS-Einheit. Mathematisch lässt sich der Ausgang nun wie in Gleichung 11 berechnen, wobei f eine sogenannte Aktivierungsfunktion darstellt. Diese verarbeitet die Eingangswerte zu dem Ausgangswert und die verschiedenen Möglichkeiten dieser Funktion werden im folgenden Unterkapitel weiter erläutert.[14]

$$\mathrm{a} = f(\mathrm{e1} \cdot \mathrm{w1} + \mathrm{e2} \cdot \mathrm{w2} + \mathrm{e3} \cdot \mathrm{w3} - \Theta) \tag{11}$$

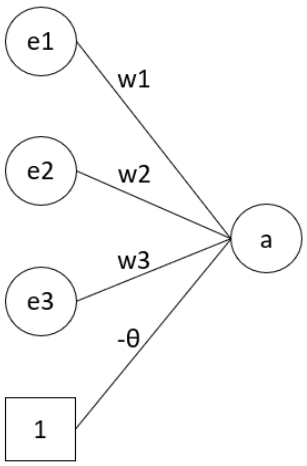

Abbildung 3: Beispielhaftes neuronales Netz mit drei Eingängen und einem Ausgang

Beispielhaft soll dieses Netz nun lernen, aus drei Eingangssignalen ein Ausgangssignal nach dem Muster in Tabelle 2 zu erstellen. Dabei hängt der Ausgabewert ausschließlich von dem ersten Eingabewert ab. Die Werte der Gewichte hängen von dem Wert Θ der BIAS-Einheit ab, dieser soll vereinfacht angenommen null sein. Wenn das Netz ideal trainiert ist, sollte somit das Gewicht w1 einen Wert nahe eins haben und w2 sowie w3 nahe null sein. Außerdem soll das Netz nach einer vereinfachten Aktivierungsfunktion die Ausgabe bewerten, bei der gelten soll:

$$f(x) = \begin{cases} 1, & x \geq 0,5 \\ 0, & x < 0,5 \end{cases} \tag{12}$$

Tabelle 2: Trainingsdaten für das beispielhafte neuronale Netz

Eingabe			Ausgabe
1	0	1	1
1	0	0	1
0	1	0	0
0	1	1	0

Die Gewichte werden zu Beginn alle mit $\frac{1}{3}$ initialisiert. Danach soll dann eine Gewichtsänderung wie folgt berechnet werden:

$$\triangle \mathrm{w}_{ij} = \sigma \cdot \mathrm{e}_i \cdot \triangle \mathrm{a}_j \tag{13}$$

σ ist dabei eine Konstante welche die Höhe der Veränderung der Gewichte bestimmt. Anhand der vier Trainingsdaten kann das Netz trainiert werden. Beispielhaft soll hier die Rechnung vorgestellt werden.

$$\mathrm{a}(1) = f(\frac{1}{3} \cdot 1 + 0 + \frac{1}{3} \cdot 1) = f(\frac{2}{3}) = 1 \text{ korrekt, keine Gewichtsänderung} \tag{14}$$

$$\mathrm{a}(2) = f(\frac{1}{3} \cdot 1 + 0 + 0) = f(\frac{1}{3}) = 0 \text{ Gewichtsänderung von w1:} \tag{15}$$

$$\mathrm{w1} = \frac{1}{3} + 0.2 * 1 * 1 = 0,53 \tag{16}$$

$$\mathrm{a}(3) = f(0 + \frac{1}{3} \cdot 1 + 0) = f(\frac{1}{3}) = 0 \text{ korrekt, keine Gewichtsänderung} \tag{17}$$

$$\mathrm{a}(4) = f(0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 1) = f(\frac{2}{3}) = 1 \text{ Gewichtsänderung von w2 und w3:} \tag{18}$$

$$\mathrm{w2} = \frac{1}{3} + 0.2 * 1 * -1 = 0,13 \tag{19}$$

$$\mathrm{w3} = \frac{1}{3} + 0.2 * 1 * -1 = 0,13 \tag{20}$$

Nun werden sich bei weiteren Trainingsdaten oder Beispielen die Gewichte nicht mehr anpassen und das Netz gibt die Ausgabewerte korrekt aus. Dies ist nur ein stark vereinfachtes Beispiel um die zugrundeliegenden Rechnungen zu erklären. In der Realität können die neuronalen Netze beliebig komplex und nicht mehr einfach nachvollziehbar sein. In Abbildung 4 ist schematisch ein Netz mit drei Eingangswerten, einer verborgenen Schicht mit zwei Knoten und drei Ausgabewerte gezeigt. Anhand von geschickter Wahl der Schichten- und Knotenanzahl sowie der Aktivierungsfunktion können so verschiedenste Funktionen erlernt werden. Dabei können nahezu beliebig viele Eingabe- und Ausgabewerte verwendet werden. [14, S. 42] Im Weiteren sollen die verschiedenen Aktivierungsfunktionen und Löser betrachtet werden.

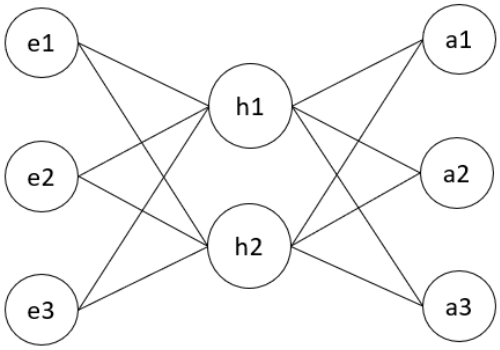

Abbildung 4: Neuronales Netz mit einer verborgenen Schicht

2.5.1 Aktivierungsfunktionen für neuronale Netze

Um aus den Eingangswerten die Ausgangswerte der einzelnen Knoten eines neuronalen Netzes zu bestimmen, gibt es wie zuvor erwähnt verschiedene Aktivierungsfunktionen. Ein neuronales Netz kann dann nichtlineares Verhalten lernen, wenn die Aktivierungsfunktion nicht linear ist.[20, S. 108] Alle Aktivierungsfunktionen sind in Abbildung 5 graphisch dargestellt.

Linear

Dies ist eine sehr simple Aktivierungsfunktion, bei der der Ausgabewert direkt dem Eingabewert entspricht. Sie entspricht einer linearen Aktivierung.[20, S. 109]

Sigmoid

Bei der Sigmoid-Aktivierung wird als Ausgabewert der Sigmoid des Eingabewertes berechnet. Diese Aktivierung wird hauptsächlich als Ausgabe von binären Klassifikationsproblemen verwendet.[20, S. 110]

$$f(x) = \frac{1}{1 + e^{-x}} \tag{21}$$

Softmax

Bei dieser Aktivierungsfunktion werden die einzelnen Ausgabewerte über dem Mittelwert gewichtet, es entsteht also eine Wahrscheinlichkeit von null bis eins für jeden Eingabewert. Diese Aktivierung wird häufig für mehrstufige Klassifikationen, also solche mit mehr als zwei Klassen, verwendet.[20, S.110]

$$f(x) = \frac{e^{x}}{\sum_{1}^{n} e^{x_n}} \tag{22}$$

Hyperbolischer Tangens

Der Ausgabewert ist der hyperbolische Tangens des Eingabewertes. Diese Aktivierungsfunktion wird auch häufig in verdeckten Schichten benutzt.[20, S.112]

$$f(x) = tanh(x) \tag{23}$$

Gleichgerichtete Lineareinheit (ReLU)

Die gleichgerichtete Lineareinheit ist für alle Werte kleiner oder gleich null null, und bildet im positiven Bereich einen linearen Verlauf ab. Es handelt sich dabei um eine nicht-lineare Aktivierungsfunktion mit Ableitung und wird häufig standardmäßig verwendet. Allerdings kann es bei Werten um null dazu kommen, dass die Ableitung immer kleiner wird und die Löser dadurch keine Lösung finden.[20, S.111]

$$f(x) = max(0, x) \tag{24}$$

Leaky ReLU

Leaky ReLU basiert auf der gleichgerichteten Lineareinheit, verändert allerdings den negativen Bereich, sodass eine minimale Steigung vorhanden ist und die Ausgabewerte dort nicht alle null sind. α ist dabei klein.[19, S. 45]

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases} \tag{25}$$

Softplus

Softplus ist ebenfalls eine Abwandlung der gleichgerichteten Lineareinheit, sie macht aus dem nicht stetigen Verlauf einen stetigen und ist immer größer null.[19, S. 47][27]

$$f(x) = \frac{1}{\beta} \star log(1 + e^{\beta x}) \tag{26}$$

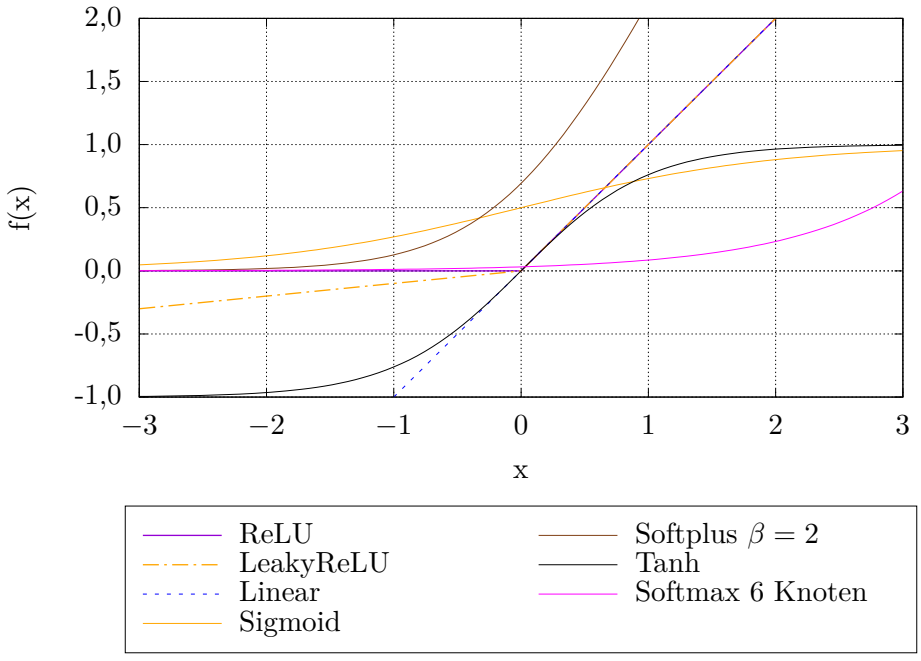

Abbildung 5: Übersicht über die Aktivierungsfunktionen für neuronale Netze

2.5.2 Löser für neuronale Netze

Um neuronale Netze zu trainieren wird Rückkopplung verwendet. Dabei wird die Ausgabe des Netzes verarbeitet und wieder zurückgegeben. So können die Gewichte angepasst werden, die Lernrate bestimmt dabei wie stark diese verändert werden.[14, S. 40, S. 50] Im Folgenden werden einige Verfahren dazu erläutert.

Batch Gradientenabstieg - Batch Gradient Descent (BGD)

Hierbei wird über die Verlustfunktion die Abweichung der Ausgabe zum Sollwert ermittel. Dann wird in der Rückkopplung der Gradient der Verlustfunktion berechnet. Dieser zeigt an, in welche Richtung die Gewichte angepasst werden müssen, nämlich entgegengesetzt der Richtung des Gradienten. So wird iterativ die Verlustfunktion kleiner. Für die Berechnung wird der gesamte Datensatz verwendet, was sehr rechenintensiv ist. [20, S.114 f.]

Stochastischer Gradientenabstieg - Stochastic Gradient Descent (SGD)

Beim stochastischen Gradientenabstieg wird anstelle des gesamten Datensatzes eine zufällig gewählte Probe verwendet. Die Rückkopplung wird häufiger berechnet. Dadurch kann es zu Konvergenzproblemen kommen. Um das möglichst zu vermeiden wird die Lernrate angepasst.[20, S.115]

Mini-Batch Gradientenabstieg - Mini-Batch Gradient Descent (MBGD)

Der Mini-Batch Gradientenabstieg vereint nun die Nutzung einer zufällig gewählten größeren Stichprobe mit der Berechnungshäufigkeit von SGD.[20, S.115 f.]

Gradientenabstieg mit Momentum

Für den Gradientenabstieg mit Momentum wird anstelle des Gradienten der exponentiell gewichtete Durchschnitt dessen berechnet. Dieser ermöglicht ein exponentielles Abklingen des Einflusses der früheren Gewichte und die Gewichte zum Zeitpunkt der Rechnung haben den höchsten Einfluss. Damit sind die Schritte zur Änderung der Gewichte kleiner und so konvergiert die Rechnung schneller.[20, S.116 f.]

RMSprop

Für diesen Löser wird für jedes Gewicht der gleitende Durchschnitt der quadrierten Gradienten gebildet und die Gradienten dann durch die Wurzel aus dem Ergebnis geteilt. [20, S.117]

Adam

Adam (adaptive moment estimation) verbindet nun SGD mit RMSprop. Grundsätzlich wird SGD verwendet, aber mit dem gleitenden Durchschnitt der Gradienten. Zusätzlich werden die quadrierten Gradienten zur Veränderung der Lernrate verwendet.[20, S.118]

2.5.3 Regularisierung

Die in Abschnitt 2.4 bereits allgemein für alle Regressionsprobleme beschriebene Unter- und Überanpassung gilt ebenso für Regressionsprobleme, die mit neuronalen Netzen gelöst werden sollen. Hierbei entscheidet jedoch nicht die Höhe des Polynomgrades,

sondern die Anzahl der Knoten und Schichten, die Datenmenge sowie die Trainingsepochen über Über- und Unteranpassung.[20, S. 14 ff., S. 159] Mit dem Regularisieren von Modellen soll die Überanpassung verhindert werden. Um das zu erreichen, gibt es verschiedene häufig eingesetzte Methoden.[20, S. 163]

Frühzeitiges Stoppen

Beim frühzeitigen Stoppen wird die Verlustfunktion des Validierungsdatensatzes überwacht. Sobald diese über eine gewisse Anzahl an Trainingsepochen nicht gesunken ist, werden die Parameter des zuvor gespeicherten Netzes verwendet und das Training abgebrochen. Das verhindert, dass zwar die Verlustfunktion auf dem Trainingsdatensatz weiter sinkt, für den Validierungsdatensatz aber wieder ansteigt, was Überanpassung bedeuten würde.[8, S. 214 ff.]

Dropout

Wenn eine Dropout-Schicht in das neuronale Netz eingefügt wird, werden mit einer Dropout-Wahrscheinlichkeit zufällig in jedem Trainingsdurchlauf einige Neuronengewichte gelöscht, also auf null gesetzt und damit ihre Verbindung gekappt. So reduziert Dropout direkt die Komplexität der einzelnen Schichten und verändert das neuronale Netz laufend während des Trainingsprozesses.[20, S. 167 ff.]

Stochastic Weight Averaging

Bei der Nutzung von Stochastic Weight Averaging (SWA) werden während des Trainings gleitende Durchschnitte der Gewichte erzeugt. Diese werden am Ende der Optimierung als Gewichte für das Modell verwendet. Dadurch soll verhindert werden, dass der Löser ein Modell optimiert, welches schlecht generalisiert. Das bedeutet, der Lösungsraum ist am Ende mit dieser Methode breiter und der Testfehler sinkt im Vergleich zur herkömmlichen Optimierung, während der zugehörige Trainingsfehler höher ist.[29]

Normstrafen

Bei den Normstrafen wird zu dem Term der Verlustfunktion noch ein normierter Term addiert, sodass der Verlust höher wird. Dabei wird der Einfluss der Normstrafe durch den Regularisierungsparameter λ bestimmt. Zwei häufig verwendete Normstrafen sind L1- und L2-Regularisierung. Dabei werden der Verlustfunktion folgende Terme hinzuaddiert: [20, S. 165 ff.][8, S. 226 ff.]

$$\text{L1-Term} \quad : \quad \lambda \sum_i |w_i| \tag{27}$$

$$\text{L2-Term} \quad : \quad \frac{\lambda}{2} \sum_i |w_i|^2 \tag{28}$$

2.5.4 Weitere Typen neuronaler Netze

Zuvor sind die klassischen vorwärtsgerichteten neuronalen Netze beschrieben worden. Ein weiterer Typ dieser Netze sind beispielsweise die Convolutional Neural Networks (CNN), welche häufig zur Bildverarbeitung eingesetzt werden. Dabei werden mit einer Eingabe und einem Kernel eine Ausgabe erzeugt, und der Kernel verstärkt dabei die Bereiche der Eingangsdaten, zu denen er selbst am ähnlichsten ist. Die Convolutional Layer, also faltende Schichten, sind geringer vernetzt als die voll vernetzten verdeckten

Schichten, welche in den zuvor beschriebenen neuronalen Netzen verwendet worden sind.[20, S.198 ff.]

Noch unterschiedlicher sind die rückgekoppelten neuronalen Netze, welche keine reinen vorwärtsgerichteten Netze mehr sind, da die Ausgänge der Knoten auch wieder deren Eingänge, Eingänge anderer Knoten derselben Schicht oder Eingänge der Schichten vorher sein können. Damit ist das Verhalten dieser Netze insbesondere in zeitabhängigen Anwendungen wie der Spracherkennung im Vorteil zu den rein vorwärtsgerichteten Netzen.[20, S. 243]

2.6 Evolutionäre Algorithmen

Auch evolutionäre Algorithmen können dem maschinellen Lernen zugeordnet werden. Es gibt viele verschiedene Ausprägungen, grundsätzlich basiert das Prinzip jedoch auf der Nachbildung der Evolution. Es werden Merkmale definiert, die von einer Generation an die nächste weitergegeben werden. Dabei wird für die einzelnen Individuen in den verschiedenen Generationen jeweils ihre Fitness berechnet und nur die mit der höchsten Fitness werden für die nächste Generation wieder als Eltern eingesetzt. Zusätzlich zu den verschiedenen Rekombinationsmöglichkeiten der Merkmale der Eltern gibt es noch Mutationen, wo die Merkmale zufällig verändert werden. Über mehrere Generationen kann so ein Optimierungsproblem gelöst werden.[12, S. 3 ff.]

2.7 Propellerentwurf

In diesem Kapitel soll auf die Grundlagen der Bezeichnungen vom Schiffspropeller sowie dessen Entwurf eingegangen werden. Dazu sind in Abbildung 6 zuerst einige wichtige Bezeichnungen am Propeller dargestellt. Sehr wichtig für die Effektivität des Propellers ist sein Durchmesser D, dieser sollte je nach Gegebenheiten im Schiffsrumpf möglichst groß gewählt werden. Die Nabe hingegen sollte so klein wie aufgrund der Festigkeit möglich gestaltet werden, da sie keinen Schub generiert. Ein weiterer wichtiger Parameter ist die Flügelzahl Z. Diese sollte für die Wirkungsgrad-Optimierung gering gewählt werden, da die Flügel sich gegenseitig beeinflussen. Andererseits sollte sie aber hoch sein, um die Vibrationen durch eine hohe Belastung der einzelnen Flügel zu verringern. Häufig wird die Flügelzahl auch aufgrund der Taktzahl des Motors vorgegeben.[21, S. 4 ff.]

Bei dem Flächenverhältnis Ae/Ao handelt es sich um das Verhältnis aus der abgewickelten Fläche des Propellers und seiner gesamten Kreisfläche, die sich aus dem Durchmesser ergibt. Insbesondere um die Belastung auf einer größeren Fläche zu verteilen, ist ein hohes Flächenverhältnis anzustreben, denn in Bereichen von hoher Belastung neigt der Propeller eher zu Kavitation, was zu vermeiden ist. Ein weiterer sehr wichtiger Parameter im Hinblick auf Kavitation ist die Steigung P/D. Sie bezeichnet den vergleichbaren spiralförmigen Verlauf wie bei einem Schraubengewinde auf der zylindrischen Oberfläche. Eine hohe Steigung bedeutet bei gerader Anströmung einen tendenziell hohen Wirkungsgrad, jedoch auch erhöhte Kavitationsgefahr. Deshalb variiert die Steigung bei heutigen Entwürfen über den Propellerradius, häufig wird sie als Steigung bei 70% des Radius angegeben. Insbesondere an den Spitzen und im Wurzelbereich wird die Steigung oft reduziert, um Kavitation zu vermeiden. Heutzutage wird ebenfalls viel Skew oder auch Flügelrücklage verwendet, um die zeitliche Veränderung der Kavitation zu reduzieren. Um dabei Schub- und Momentenschwankungen zu verringern treten

die Radien der Propellerflügel versetzt in das Gebiet des langsamsten Nachstroms ein. [21, S. 7 ff.]

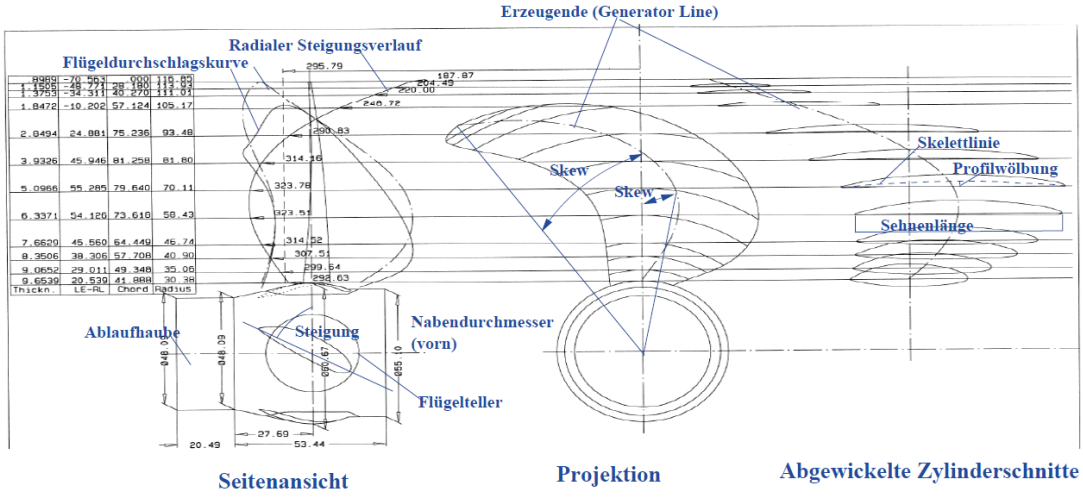

Abbildung 6: Bezeichnungen eines Propellerflügels aus *Krüger*, 2021 [21, S. 4]

In Abbildung 7 ist ein Schnitt eines einzelnen Profils gezeigt. Die Profildicke sollte aus hydrodynamischer Betrachtung möglichst gering sein, da dann der Widerstand kleiner ist. Jedoch muss auch die Festigkeit gewährleistet werden und dünne Profile neigen häufiger zu Kavitation im Nasenbereich. Bei Wölbung handelt es sich um eine Veränderung der Profilsehne, diese wird aufgewölbt. Dadurch wird der effektive Anstellwinkel des Profils beeinflusst, denn bei einem Profil ohne Wölbung liegt der stoßfreie Eintritt genau bei 0° auf der x-Achse. Dies ist bei dem ungewölbten Profil auch der Nullauftriebsanstellwinkel, bei dem es keinen Auftrieb generiert. Bei einem Profil ohne Wölbung entspricht die Profilsehne auch der Skelettlinie. Anders sind die Nullauftriebsanstellwinkel bei einem gewölbten Profil negativ, da die Skelettlinie nicht mehr exakt auf der Profilsehne liegt (vergleiche eingezeichnete Skelettlinie links in Abbildung 7). So lässt sich für einen bestimmten Betriebsbereich die Kavitation im Nasenbereich durch die verbesserte Strömungslenkung im Vergleich zum ungewölbten Profil bei demselben Anstellwinkel reduzieren, andererseits kann das Profil aber schlechter mit anderen Anstellwinkeln betrieben werden. [21, S. 10 ff.]

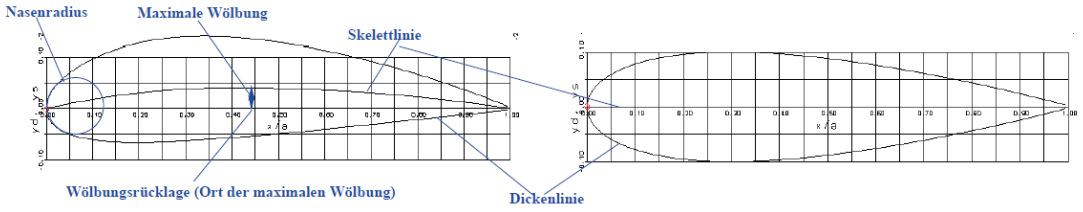

Abbildung 7: Bezeichnungen eines Profils aus *Krüger*, 2021 [21, S. 10]

Weitere wichtige Kenngrößen eines Propellers im Betrieb sind der Fortschrittsgrad J, der Freifahrtwirkungsgrad η_O und die Schub- und Momentenbeiwerte k_T und k_Q. Sie ergeben sich aus dem Freifahrtversuch, bei dem der Propeller ohne Nachstromeinfluss

bei konstanter Drehzahl n und Geschwindigkeit v_a durchs Wasser bewegt wird. Dabei werden Schub T und Drehmoment Q_o gemessen und die Kenngrößen können wie folgt berechnet werden:[21, S. 19 ff.]

$$J = \frac{v_a}{nD} \tag{29}$$

$$k_T = \frac{T}{\rho n^2 D^4} \tag{30}$$

$$k_Q = \frac{Q_o}{\rho n^2 D^5} \tag{31}$$

$$\eta_O = \frac{J}{2\pi}\frac{k_T}{k_Q} \tag{32}$$

Die Kennzahlen k_T, k_Q und η_O werden im Freifahrtdiagramm über J aufgetragen. Ein solches Freifahrtdiagramm ist beispielhaft in Abbildung 8 gezeigt.[21]

Um den Betriebspunkt des Schiffes zu beschreiben, werden weitere Größen benötigt. Zum einen die Drehleistung der Maschine, P_D. Diese setzt sich zusammen aus der Drehzahl und dem Arbeitsmoment Q des Propellers. Um k_Q zu berechnen wird das Freifahrtmoment Q_o des Propellers benötigt, es lässt sich mit dem Gütegrad der Anordnung η_R aus Q umrechnen. Der Widerstand R_T und der Schub T bedingen sich über die Sogziffer t gegenseitig. Mithilfe der Nachstromziffer w lässt sich die Schiffsgeschwindigkeit v_s in die Anströmungsgeschwindigkeit v_a am Propeller umrechnen.[22, S. 1 ff.]

$$P_D = 2\pi Q n \tag{33}$$

$$\eta_R = \frac{Q_o}{Q} \tag{34}$$

$$R_T = (1-t)T \tag{35}$$

$$(1-w) = \frac{v_a}{v_s} \tag{36}$$

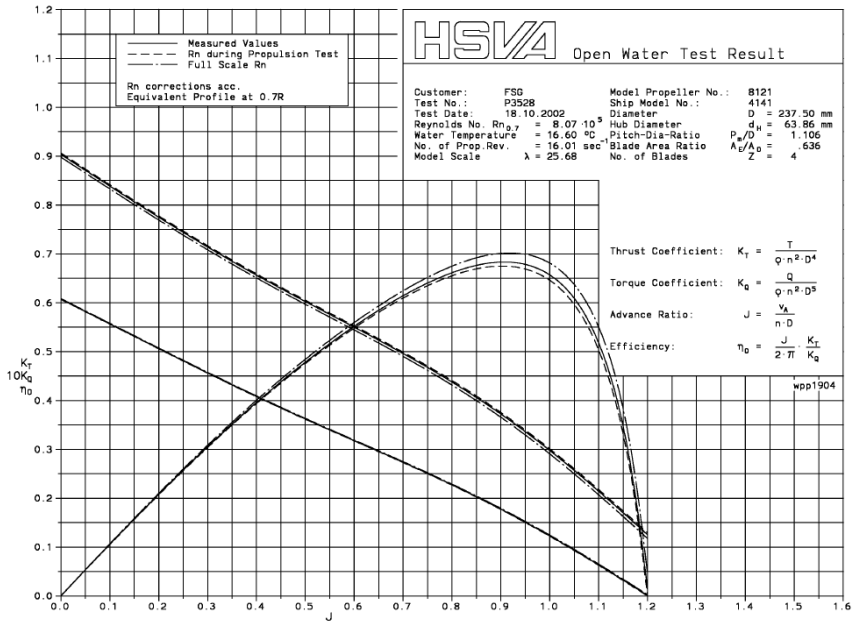

Abbildung 8: Beispielhaftes Freifahrtdiagramm aus *Krüger*, 2021 [21, S.21]

Um aus diesen Größen in einem vorhandenen Freifahrtdiagramm den korrekten Betriebspunkt zu finden, wird die sogenannte schiffsseitige Belastungskurve benötigt. Diese kann sowohl über den Schub- als auch den Momentenbeiwert berechnet werden. Geläufiger ist allerdings die Berechnung über den Schub. Dafür soll nun in der Gleichung vom Schubbeiwert die unbekannte Drehzahl herausgerechnet werden. Dazu kann der Beiwert durch den quadrierten Fortschrittsgrad geteilt werden, sodass sich Gleichung 38 ergibt. Es ergibt sich also eine von J über J^2 abhängige Kurve. Diese kann wieder in das zu dem Propeller gehörende Freifahrtdiagramm aus Abbildung 8 eingetragen werden. Der Schnittpunkt mit der Kurve des Schubbeiwertes ist nun der Betriebspunkt. Daraus kann der Fortschrittsgrad sowie der Momentenbeiwert abgelesen werden.[22, S. 12]

Dieselbe Rechnung ist auch mit dem Momentenbeiwert möglich, allerdings muss hier aufgrund der dritten Potenz durch J^3 geteilt werden. Dann ergibt sich Gleichung 40. Nun kann analog im Freifahrtdiagramm der Schnittpunkt aus der Kurve des Momentenbeiwertes und der schiffsseitigen Belastungskurve erfolgen und die anderen Größen abgelesen werden.

$$k_T = \frac{R_T}{\rho n^2 D^4 (1-t)} \tag{37}$$

$$\frac{k_T}{J^2} = \frac{R_T}{\rho D^2 (1-t) v_s^2 (1-w)^2} = k_T^* \tag{38}$$

$$k_Q = \frac{\eta_R \frac{P_D}{2\pi n}}{\rho n^2 D^5} = \frac{\eta_R P_D}{2\pi \rho n^3 D^5} \tag{39}$$

$$\frac{k_Q}{J^3} = \frac{\eta_R P_D}{2\pi \rho D^2 v_s^3 (1-w)^3} = k_Q^* \tag{40}$$

Mit dieser Abschätzung ist ein erster Entwurf eines Propellers möglich. Eine weitere wichtige Betrachtung ist die der Kavitation. Bei einem kavitierenden Propeller implodieren Gasblasen auf der Oberfläche des Propellers, da das Wasser wegen des geringen Druckes bereits bei Umgebungstemperatur anfängt zu verdampfen. Um Kavitation im Detail beurteilen zu können, sind Modellversuche oder intensive CFD-Rechnungen erforderlich. Allerdings sollte eine erste Abschätzung des Mindestflächenverhältnisses bereits im frühen Entwurfsstadium durchgeführt werden. Dies ist zum Beispiel anhand des Burill-Diagramms möglich. Dabei werden die Kavitationszahl σ_v und der mittlere Druck τ_c in ein Diagramm eingetragen, in dem eine definierte Grenzkurve eingetragen ist. Liegt der Wert des Propellers unterhalb der Grenzkurve, kavitiert die Strömung um den Propeller vermutlich nicht, liegt er darüber, kavitiert sie vermutlich. Die Grenzkurve ist in Abbildung 9 gezeigt. So lässt sich eine Abschätzung des mindestens nötigen Flächenverhältnisses treffen, welches zur Kavitationsvermeidung erforderlich ist.[26] Die Werte σ_v und τ_c lassen sich wie folgt berechnen:

$$\sigma_v = \frac{p_0 + \rho g h - p_v(T)}{\frac{\rho}{2} v_a^2 (1 + (\frac{0.7\pi}{J})^2)} \tag{41}$$

$$\tau_c = \frac{\frac{T}{A_p}}{\frac{\rho}{2}(1 + (\frac{0.7\pi}{J})^2)} \tag{42}$$

Dabei handelt es sich bei $p_v(T)$ um den Dampfdruck des Wassers zu dieser Temperatur und bei A_p um die projizierte Fläche des Propellers. Zusätzlich zu der Anströmungsge-

schwindigkeit vom Propeller wird noch die Umfangsgeschwindigkeit auf der Höhe von $0,7R$ berechnet, also dort wo typischerweise auch die Steigung angegeben wird.[26]

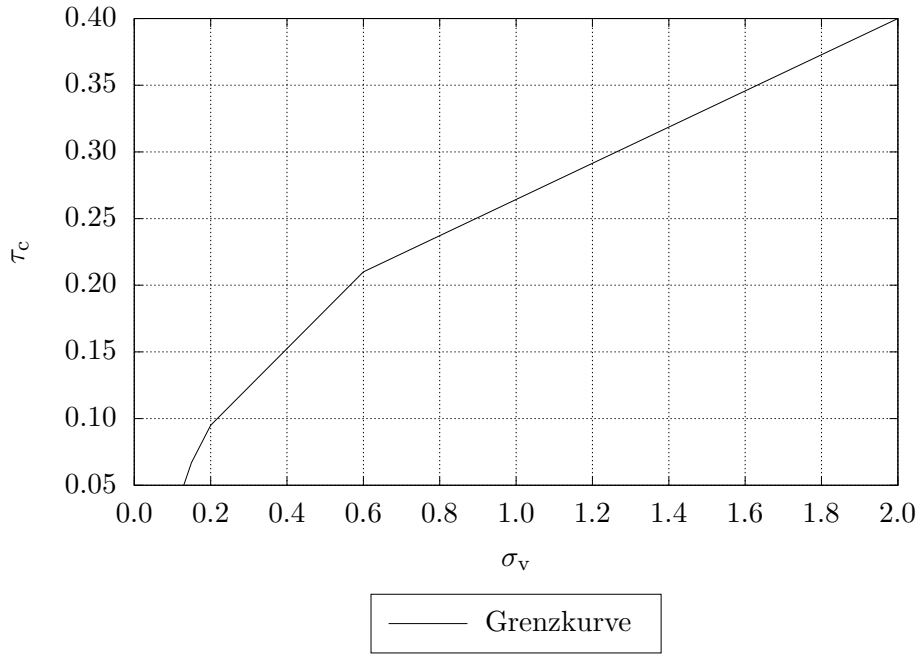

Abbildung 9: Grenzkurve des Burill-Diagramms für Kavitation

3 Literaturrecherche

Um erste Ansatzpunkte zur Lösung der Aufgabenstellung zu finden, werden in diesem Abschnitt verschiedene aktuelle Veröffentlichungen vorgestellt. Die Autoren haben bereits in Richtung des KI-gestützten Propellerentwurfs oder in vergleichbaren Feldern geforscht und die Methoden und Erkenntnisse können hilfreich bei der Lösung dieser Aufgabenstellung sein.

Machine Learning Assisted Propeller Design, Vanderbilt University

Es handelt sich bei dieser Veröffentlichung um ein kurzes Skript des Instituts für "Software Integrated Systems" der Vanderbilt Universität. Für diesen klassischen Propellerentwurf werden zuerst zwei verschiedene Sätze von geometrischen und physikalischen Parametern am Propeller definiert. In den geometrischen Parameterbereich fallen beispielsweise Steigung, Durchmesser, und Flügelzahl. Der sogenannte physikalische Parameterbereich beschreibt dann zum Beispiel den Schubbeiwert und Fortschrittsgrad. Für den Entwurf des Propellers wird eine Schiffsgeschwindigkeit sowie die Drehzahl vorgegeben. Dann soll der passende Entwurf mit der maximalen Effizienz gefunden werden.

Um die zuvor genannten Parametersätze nun mit maschinellem Lernen umsetzen zu können, werden aus den genannten Anforderungen zwei Merkmalsätze definiert: die Anforderungsmerkmale sowie die Geometriemerkmale. Als Anforderungsmerkmale werden so Schub, Geschwindigkeit und Drehzahl vorgegeben. Die Geometriemerkmale enthalten den Durchmesser und neun verschiedene Sehnenprofile. Als Ziel wird wie zuvor genannt die maximale Effizienz eines Propellers am Betriebspunkt angestrebt. Die untere Grenze wird dabei mit einer Effizienz von 50 % festgelegt. Die Größe des entstehenden gesamten Merkmalraumes ist 10^{38}.

Nun soll als Vergleichsmethode eine Funktion erlernt werden, die aus den Anforderungsmerkmalen die Geometriemerkmale bei maximaler Effizienz erzeugt. Diese Funktion findet anhand der vorgegeben Anforderungen eine erste Geometrie und schätzt die zugehörige Effizienz. Danach wird die Geometrie mit OpenProp verbessert, indem durch evolutionäre Algorithmen in kleinem Rahmen lokale Optima gesucht werden. Die freie Software OpenProp berechnet dabei die Propellercharakteristiken basierend auf der Traglinientheorie. Diese Methode wird mit einem Random Forest Regression Modell verglichen, für welches 205.556 Trainingsdatenpunkte in dem zu beschreibenden Merkmalraum zufällig erzeugt und berechnet werden. Für dieses ML-Modell wird zehnfache Kreuzvalidierung und der mittlere quadratische Fehler zum Training benutzt. So sind unkonventionelle Designs mit einer Effizienz von 90 % gefunden worden. Bei 80 % Erfüllung der zuvor festgelegten Anforderungsmerkmale liegt der abweichende Fehler zum Design mit Openprop bei 7,5 %.

Diese Veröffentlichung enthält bereits einige interessante Aspekte zum klassischen Propellerentwurf mit maschinellem Lernen wie die Optimierung nach der maximalen Effizienz in einem vorgegebenen Geometrieraum, kann aber nur als erster Schritt gesehen werden. Zukünftig wollen die Autoren beispielsweise mit einem neuronalen Netz statt mit einer Random Forest Regression arbeiten. Weiterhin bleiben noch Fragen offen. Zum Beispiel ist nicht bekannt, ob die zufällig erzeugten Trainingsdaten auf ihre physikalische Eignung hin überprüft worden sind. Auch sind Phänomene wie Kavitation vollkommen außer Acht gelassen worden, stellen allerdings gerade in Bezug auf das Flächenverhältnis der Propeller eine wichtige Einschränkung dar. Der in der Veröffentli-

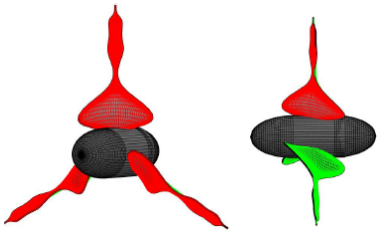

Abbildung 10: Mithilfe von KI erzeugte Propellergeometrie aus *Vardhan et al.*[24]

chung genannte optimale Beispielentwurf liegt bei einer Geschwindigkeit von 14 $\frac{\mathrm{m}}{\mathrm{s}}$, einer Drehzahl von 5,6 $\frac{1}{\mathrm{s}}$ und 30,6 kN Schub und soll einen Wirkungsgrad von 0,9 erzeugen. Das so erzeugte Profil ist in Abbildung 10 gezeigt. Der Verlauf der Steigung variiert stark, sodass vermutlich mit Kavitation zu rechnen ist. Dies deutet darauf hin, dass die Eingangsdaten für die neuronalen Netze nicht hinreichend geprüft worden sind.[24]

noiseNet: A neural network to predict marine propellers underwater radiated noise, Technische Universität Hamburg

In dieser Veröffentlichung des Instituts für Fluiddynamik und Schiffstheorie der technischen Universität Hamburg soll mithilfe von neuronalen Netzen die Prognose von Lärm und Vibration, welche vom Propeller ausgestrahlt werden, bereits für den frühen Entwurf verbessert werden. Dazu werden Datensätze mithilfe von panMARE und BEM-FWH (BEM = Boundary Element Method und FWH = Ffowcs Williams - Hawkings Gleichung für die akustischen Abschätzungen) erzeugt und damit ein neuronales Netz angelernt. Dabei wird Tonschall bis zur dritten Blattfrequenz betrachtet, da die BEM bis zu dieser Frequenz gut geeignet ist. Zudem werden die Daten in Schritten von 5° der Propellerumdrehung berechnet, sodass bei angemessener Rechenzeit genügend Daten erzeugt werden können. Insbesondere kavitierende Propeller sollen betrachtet werden, da hier der Lärm aufgrund des fluktuierenden Volumens der Kavitäten meist stärker ist.

Für den Datensatz werden Berechnungen durch die BEM-FWH Methode für 17 Propeller in 25 Nachstromfeldern durchgeführt, wobei acht Betriebspunkte verwendet werden. Das bedeutet einen Datensatz von 3400 Datenpunkten, wovon 302 aufgrund mangelnder Konvergenz in der Berechnung ausgeschlossen werden. Am Ende stehen für das Training 3098 Datenpunkte zur Verfügung. Für die Netzarchitektur werden vier verschiedene Blöcke verwendet, welche alle ein neuronales Netz mit verschiedenen Knoten und Zwischenschichten darstellen. Der Eingang wird aufgeteilt in einen skalaren Eingang und einen sequenziellen Eingang. Der sequenzielle Eingang enthält ein 3D-Array, indem über die Zeit und die betrachteten drei Radien die Eingangsvariablen des Flügels beschrieben werden. Diese bestehen aus dem Anstellungswinkel α, dem statischen Druck p_0, deren Zeitableitungen, dem Verhältnis von Wölbung sowie Dicke zu Sehnenlänge, der Rotationsgeschwindigkeit aus der Drehzahl sowie der Sehnenlänge. Für den skalaren Eingang werden die sechs zeitunabhängigen Variablen Durchmesser, Flügelzahl, Drehzahl und Abstand der Eintrittskante zur Mittellinie bei drei verschiedenen Radien festgehalten.

Die ersten beiden Blöcke berechnen aus sequenziellem und skalarem Eingang Aus-

gabewerte im Zeitbereich. Block 1 berechnet die hydrodynamischen Kräfte für jeden einzelnen Radius im Zeitbereich. Block 2 kombiniert die Einflüsse der Radien mithilfe der skalaren Eingangsdaten untereinander. Mit einem Convolutional Neural Network (CNN) wird das Volumen der Kavität in eine Änderung des Volumens übertragen. Über eine Fast Fourier Transformation (FFT) werden im dritten und vierten Block für die einzelnen Frequenzen ein Ausgabeblock mit Sound Pressure Level (SPL) erzeugt. Die FFT teilt die Daten dabei im Zeitbereich auf die einzelnen Blattfrequenzen im Frequenzbereich auf. So wird in Block 3 die Abhängigkeit der einzelnen Flügel in Bezug auf den Ton berechnet und in Block 4 noch ein möglicher Energietransfer von einer Frequenz zur anderen betrachtet. Insgesamt werden die Berechnungen für alle drei Blattfrequenzen durchgeführt. Die einzelnen Blöcke stellen jeweils neuronale Netze mit unterschiedlicher Anzahl an verdeckten Schichten und verschiedenen Aktivierungsfunktionen dar.

Die Bewertung der Ausgabe erfolgt über den mittleren quadratischen Fehler und den mittleren absoluten Fehler. Es stellt sich heraus, dass das Training der Parameter nur über eine vorgegebene Aufteilung der Daten geschehen kann. Wenn die Daten für die Training-Test-Aufteilung zufällig ausgewählt werden, kann es sein, dass in den Testdaten bereits bekannte Nachstromfelder und Propeller verwendet werden, sodass die Prognose für ungesehene Daten nicht verlässlich ist. Da Überanpassung eher auftritt, wenn die Propellergeometrien bereits bekannt sind, als wenn das Nachstromfeld bekannt ist, werden vier Propellergeometrien zufällig für die Tests verwendet und die anderen Propellergeometrien für das Training. Dadurch ist bei 70% aller Tests der absolute Fehler kleiner als 10 dB und die Standardabweichung liegt bei 0,41 dB. Damit ermöglicht dieses Netz bereits im frühen Entwurfsstadium verlässliche Abschätzungen zu Lärm und Vibration.

Für den Propellerentwurf mit maschinellem Lernen sind insbesondere die Erkenntnisse zur Überanpassung bei der Kenntnis von zu vielen Datenpunkten interessant. Da die Eingabeparameter für den Entwurf zum Teil ähnlich sind, kann ein solches Phänomen ebenfalls bei dem in dieser Arbeit zu entwickelnden Algorithmus auftreten und sollte betrachtet werden.[1]

Propeller Noise Detection with Deep Learning, Université Côte d'Azur

Unterwasser-Lärm in der Umgebung eines Schiffes soll anstatt von einem Menschen durch den Computer klassifiziert werden, es handelt sich um eine Veröffentlichung der Université Cote d'Azur. Dabei sollen nahe Umgebungsgeräusche von Schiffen aus denen von Tieren und weiter entferntem Verkehr herausgefiltert werden. Diese Veröffentlichung stellt den ersten Schritt dar, indem zwei Sekunden lange Schallsignale eines Schiffes mit einem vierflügeligen Propeller sowie verschiedene Signale von weißem Rauschen vorgegeben werden. Das weiße Rauschen stellt entferntere Umgebungsgeräusche dar. Die Klassifizierung dieser Daten in 0 - entfernte Geräusche und 1 - Schiff in der Nähe wird mit den Ergebnissen eines Neyman-Pearson-Tests verglichen.

Für die maschinelle Vorhersage werden die Daten über eine Deep Convolutional Neural Network (CNN) Architektur verarbeitet, welche L Feature Maps oder Ausgänge hat. L wird dabei variiert und über eine Rastersuche optimiert. Aktiviert wird im Netz über die Aktivierungsfunktiton ReLu. Im Vergleich mit dem Neyman-Pearson Test lieferte das beschriebene CNN noch etwas bessere Werte und stellt damit eine gute Entscheidungsbasis dar. In weitergehenden Untersuchungen sollen für die Eingangsdaten farbiges Rauschen genutzt werden, um das Modell realitätsnäher zu gestalten.

Insbesondere die in dieser Veröffentlichung für die Optimierung verwendete Rastersuche kann auch für den Propellerentwurf mit maschinellem Lernen nützlich sein.[16]

Deep Learning for Cavitating Marine Propeller Noise Prediction at Design Stage, University of Genoa / Strathclyde University

In dieser Veröffentlichung soll anhand einer Kombination aus physikalischen Modellen (PMs) und Data Driven Models (DDMs) ein hybrides Modell (HM) entwickelt werden, um den Lärm von kavitierenden Propellern bereits während des Entwurfs abschätzen zu können. Die drei Modellarten werden dabei hinsichtlich ihrer Effizienz verglichen. Zuerst werden fünf verschiedene Merkmalsätze erstellt, in denen Informationen zu Propellergeometrie und Nachstrom sowie Druckverteilung im Betrieb enthalten sind. Dabei werden zum einen Entwurfsparameter vorgegeben, wie zum Beispiel der Nachstrom, die grundlegende Propellergeometrie, Drehzahl sowie die Anströmwinkel der Profile in dem gegebenen Nachstrom. Anhand von BEM-Berechnungen werden außerdem die Druckverläufe auf dem Profil sowie die Geschwindigkeitsverläufe ermittelt. Als Ausgangsmerkmale werden zwei Spektren bestimmt, das erste enthält die Frequenz und den Spitzenpegel der Resonanz aus den Kavitationswirbeln, das zweite den gesamten abgestrahlten Geräuschpegel in Terzbanddarstellung. Um die Genauigkeit des Modells abzuschätzen werden der mittlere absolute Fehler sowie der mittlere prozentuale Fehler und der Produkt-Moment-Korrelationskoeffizient nach Pearson verwendet.

Für das physikalische Modell (PM) werden zwei Formeln betrachtet: die eine schätzt die Lärmabstrahlung von kavitierenden Wirbeln ab, die andere die Lärmabstrahlung von Schichtkavitation. Beide geben einen Dezibel-Wert aus. Bei der Netzwerkarchitektur wird als Data Driven Model (DDM) ein neuronales Netz verwendet, welches dem Typ Advanced DDM (ADDM) zuzuordnen ist. Dabei lernt das Netz selbst, welche Struktur es für die zu verarbeitenden Merkmale benötigt und bekommt nicht nur die relevanten Merkmale vorgegeben, sondern alle.

Die einzelnen Merkmale werden im ADDM unterschiedlichen Schichten zugeordnet. Für die Propellergeometrie wird ein einfaches neuronales Netz mit Random Layer verwendet, aktiviert wird mit dem hyperbolischen Tangens und reguliert mit L2-Regularisierung. Für die anderen 2D- und 3D-Merkmale werden mehrere Convolutional Layers (CNN) verwendet, welche parallel agieren. Diese münden in einen Concatenation Layer und werden ausgegeben. Die Gewichte des Modells werden vortrainiert, da ansonsten keine guten Ergebnisse zu erwarten sind. Test- und Training-Sets werden über die Bootstrap-Methode aus dem Gesamtdatensatz erstellt.

Für das hybride Modell (HM), also eine Kombination aus physikalischem Modell und DDM, werden die zuvor genannten Formeln des PMs hinterlegt und die berechneten Werte fließen ebenfalls mit in den Concatenation Layer ein. So gehen sie direkt mit in die Ausgangswerte ein und werden davor nicht weiter verarbeitet. Der Vergleich der Modelle zeigt, dass reine physikalische Modelle den Lärm eher ungenau abbilden und einige Trends nicht erfassen können (absoluter Fehler bei 5 dB). ADDMs sind deutlich besser (MAE 2 dB), können aber durch die Kombination mit den PMs noch weiter verbessert werden (MAE 1,5 dB). Die Werte sind beispielhaft für den Fehler des maximalen Pegel bei Resonanzfrequenz, sind aber für die anderen Messwerte vom Verhältnis zueinander in derselben Größenordnung. Allerdings gibt die Größe des Datensatzes noch nicht her, dass die Bewertung anhand vollständig unbekannter Daten durchgeführt werden kann. Dies stellt eine Verbesserung für die zukünftige Bewertung des Modells dar.

Die Kombination aus physikalischem Modell und neuronalem Netz ist auch für den Propellerentwurf möglicherweise vielversprechend, da es dort einige bekannte physikalische Formeln zur Berechnung gibt. Auch betont diese Veröffentlichung die Wichtigkeit eines großen und ausgewogenen Datensatzes.[7]

Machine Learning Based Classification of Ducted and Non-Ducted Propeller Type Quadcopter, DIAT

Mit dem in dieser Veröffentlichung des Defence Institute of Approved Technology in Indien beschriebenen Modell sollen Drohnen anhand ihres Radarquerschnittes in Drohnen mit und ohne Düsen eingeteilt werden. Dafür werden ein neuronales Netz, k-nearest Neighbor (k-NN) und Support Vector Machine (SVM) verglichen. In dem Datensatz sind die Radarquerschnitte in Dezibel und die Position über Azimut und Höhenwinkel für vier verschiedene Drohnen und mehrere Messpunkte gespeichert. Insgesamt sind so etwas mehr als 100.000 Datensätze erzeugt und in die Kategorien mit und ohne Düse eingeteilt worden.

Die SVM und k-NN Strukturen werden als Standardtrainingssätze aus Matlab verwendet. Für das neuronale Netz wird ein vorwärts-gerichtetes Netz mit zwei verdeckten Schichten erstellt. Als Aktivierungsfunktionen werden Sigmoid und Softmax genutzt. Die vorhandenen Daten werden zu 70% für Training und zu je 15% für Test und Validierung verwendet. Am besten klassifiziert k-nearest neighbor mit Chebyshev für die Abstandsberechnung die Radarsequenzen mit einer Genauigkeit von 76,6%. Das neuronale Netz erreicht 75,5% und SVM 76,2%. Insgesamt sind die Ergebnisse den Autoren nach zufriedenstellend.

Diese Veröffentlichung zeigt, dass ein neuronales Netz nicht unbedingt die beste Lösung eines Problems darstellt. In diesem Fall sind die klassischen Klassifikationen etwas besser für die Aufgabe geeignet, es sollte daher auch in dieser Arbeit auf einen Vergleich aus neuronalen Netzen und klassischen Algorithmen geachtet werden.[11]

Use of Artificial Neural Network for Estimation of Propeller Torque Values in a CODLAG Propulsion System, University of Rijeka

Das Ziel dieser Veröffentlichung der University Rijeka ist die automatisierte Ermittlung des Schaft-Drehmomentes aus der Vorgabe von unter anderem Schiffgeschwindigkeit, Brennstoffverbrauch und Injektionsrate. Der Datensatz mit 11.000 Daten bestehend aus 14 verschiedenen Merkmalen wird zum Training eines einfachen vorwärts-gerichteten neuronalen Netzes genutzt. Dazu werden verschiedene Hyperparameter getestet und verglichen: die Anzahl der verdeckten Schichten und ihre Knotenzahlen, die Aktivierungsfunktionen, die Lernrate, die Regularisierungs-Parameter und der Löser. Gemessen wird die Genauigkeit des Modells anhand des mittleren absoluten Fehlers im Vergleich der vorhergesagten mit den gemessenen Werten und des Bestimmtheitsmaßes.

Der niedrigste mittlere Fehler liegt für die Steuerbord-Propeller und Backbord-Propeller bei 0,003 kNm, allerdings mit unterschiedlichen Modellen. Das heißt, die Hyperparameter-Konfiguration, die für Backbord genaue Ergebnisse liefert, kann für die Prognose auf Steuerbord nur mit einem höheren Fehler verwendet werden und umgekehrt. Die Fehler liegen für die andere Seite etwa um Faktor fünf bis 50 höher. Ein um einen geringen Fehler abweichendes Modell kann für die gegenüberliegende Seite auch akzeptiert werden, sodass der Rechenaufwand reduziert werden kann, weil nur ein Modell optimiert werden

muss. Der Fehler liegt mit den jeweils optimalen Modellen bei unter 0,0005% und das Modell weist damit für diesen Anwendungsfall grundsätzlich eine hohe Genauigkeit aus.

In dieser Veröffentlichung wird vor allem das Variieren und Optimieren der Parameter des neuronalen Netzes behandelt, da geringe Veränderungen der Eingangsdaten bereits andere optimale Parameter im Training erfordern. Anhand dieser Variationen kann auch ein Parameter-Raum für die Rechenmodelle in dieser Arbeit erstellt werden.[4]

Artificial neural network for predicting values of residuary resistance per unit weight of displacement, University of Rijeka / University of Prague / University of Ostrava

In dieser Veröffentlichung soll mithilfe eines neuronalen Netzes anhand von geometrischen Schiffsdaten die Veränderung des Gesamtwiderstandes bei Veränderung der Verdrängung vorhergesagt werden. Dazu werden als leicht ermittelbare Eingangsdaten die Längs-Schwerpunktlage des Auftriebs (LCB), prismatischer Koeffizient, $\frac{L}{\Delta}$, $\frac{B}{T}$, $\frac{L}{B}$ und Froudezahl genutzt. Das neuronale Netz ist ein einfaches vorwärts-gerichtetes Netz mit folgenden optimierbaren Hyperparametern: verdeckte Schichten und Knotenzahl, Aktivierungsfunktion, Lernratentyp und Lernrate sowie Regularisierungsparameter. 75 % der Daten werden zufällig für Training und 25 % für Validierung verwendet. Die Genauigkeit des Modells wird über das Bestimmtheitsmaß R^2 ermittelt. Zudem wird eine Bland-Altman (BA) Analyse für das beste Modell durchgeführt, bei der die tatsächlichen Werte mit den vorhergesagten Werten im Bereich der Standardabweichung verglichen werden.

Es zeigt sich, dass die besten Modelle alle eine Anfangslernrate von 0,01 haben und überwiegend ReLU als Aktivierungsfunktion nutzen. Außerdem liefern die komplexeren Netze mit Regularisierung tendenziell bessere Ergebnisse. Für das beste Modell wird ein R^2-Wert von 0,99 erreicht und auch nur drei von 77 Punkten liegen bei der BA-Analyse nicht im Bereich der Standardabweichung. Dies liegt daran, dass bei den Trainingsdaten dieser Bereich nicht ausreichend abgedeckt wird, ansonsten wäre die Abweichung der BA-Analyse noch geringer.

Um die komplexen neuronalen Netze in dieser Veröffentlichung optimal trainieren zu können, ist eine L2-Regularisierung nötig gewesen. Dies sollte bedacht werden, wenn komplexe Modelle trainiert werden müssen.[5]

Propeller optimization by interactive genetic algorithms and machine learning, Chalmers University of Technology

Diese Veröffentlichung der Chalmers University of Technology hat als Ziel anhand einer Kombination aus maschinellem Lernen und evolutionären Algorithmen den Entwurfsprozess vom Propeller zu beschleunigen. Vielfach ist in heutigen Entwürfen zu wenig Zeit für intensive Rechnungen des Propeller-Designs, sodass Experten anhand von Schätzungen und Erfahrung mithilfe überschlägiger Rechnungen den Entwurf finalisieren, wenn alle Parameter dafür feststehen. Dabei gibt es für automatisierte Entwurfsprozesse verschiedene Probleme: der Parameterraum ist für konvergierende Rechenmodelle schwierig zu definieren, es gibt oft nicht nur einen Betriebspunkt, welcher optimiert werden soll, sondern mehrere und die Realität kann durch halb-empirische Rechenmodelle oft nicht ausreichend abgebildet werden.

Der Vorschlag in dieser Veröffentlichung ist nun ein interaktiver evolutionärer Al-

gorithmus. Dabei werden ähnlich wie bei automatisierten evolutionären Algorithmen mehrere Generationen durch das Rechenmodell erzeugt und dann jeweils vor der nächsten Generation von menschlichen Experten in die Kategorien ”akzeptiert” und ”nicht akzeptiert” bewertet. Um zu verhindern, dass zu viele Generationen durch Experten bewertet werden müssen, soll anhand einiger bewerteter Entwürfe die Bewertung der anderen automatisch durchgeführt werden, dies soll über maschinelles Lernen mithilfe der Support Vector Machine (SVM) geschehen. Die SVM berechnet dabei anhand der bewerteten Entwürfe ihre Entscheidungsebene und bewertet anhand dessen die übrigen. Nach jeder Runde, in der die Experten die Entwürfe bewertet haben, legen sie die begrenzenden Parameter wie Anzahl der Generationen und Individuen für die nächste Runde fest. In dieser Veröffentlichung soll insbesondere auf Effizienz und Kavitationsverhalten der Entwürfe geachtet werden, wobei die menschlichen Bewertungen vor allem für die Kavitation nötig sind.

Für die abschließende Bewertung dieses Konzepts sind der interaktive Ansatz mit einem vollautomatisierten Ansatz und dem kombinierten Ansatz aus SVM und Interaktion in vier Generationen verglichen worden. Dabei sind im voll-automatisierten Ansatz 258 akzeptierte und 178 nicht akzeptierte Entwürfe entstanden. Für den interaktiven Ansatz sind insgesamt weniger Entwürfe erzeugt worden, dafür aber 189 davon akzeptiert und 49 nicht akzeptiert worden. Dadurch konnte die Strömungssimulation einiger Entwürfe eingespart werden. Für die Kombination aus interaktivem Ansatz und SVM bewertet der Experte in den ersten Runden die Entwürfe, die Bewertung der letzten Runde wird dann mithilfe der SVM durchgeführt. Dabei sind ähnliche Ergebnisse wie im rein interaktiven Ansatz durch eine Genauigkeit der Bewertung durch die SVM von 98,5 % möglich gewesen.

Insgesamt konnte mit jeder dieser Methoden die Effizienz und das Kavitationsverhalten der Entwürfe verbessert werden. Dabei hat der interaktive Ansatz im Vergleich zu dem voll-automatisierten bereits Entwürfe in einem engeren Parameterraum geschaffen, sodass die einzelnen Entwürfe bereits besser bezüglich der Effizienz und der Kavitationsprognose gewesen sind. Für den voll-automatisierten Algorithmus sind viele deutlich verschiedene Entwürfe erzeugt worden. Soll eine Bewertung der Entwürfe durch menschliche Experten stattfinden, ist der Einsatz von SVM sinnvoll, da dadurch Zeit bei der Bewertung eingespart werden kann. Diese Vereinfachung durch die SVM hat in diesem Beispiel keinen Einfluss auf das Ergebnis gehabt. Dennoch sollte die Methode in komplexeren Entwurfsprozessen eingesetzt werden, um eine deutlichere Verbesserung im Vergleich zum voll-automatisierten Ansatz zu finden. Außerdem könnten zukünftig bereits vorhandene Datenbanken von Propellern zum Training von SVMs vor dem Entwurfsprozess genutzt werden.

Diese Veröffentlichung zeigt einige Probleme im voll-automatisierten Propellerentwurf, welche ebenfalls für den KI-gestützten Propellerentwurf auftreten. So ist eine reine Bewertung anhand eines Parameters für die Komplexität des Problems zu gering, und insbesondere Kavitation stellt eine Herausforderung dar, die schwierig automatisiert zu beurteilen ist. Dort ist häufig die Erfahrung des Entwerfers nötig.[2]

Zusammenfassung der Literaturrecherche

Zusammenfassend scheinen insbesondere neuronale Netze gut geeignet zu sein, um den KI-gestützten Propellerentwurf zu ermöglichen. Sie sollten jedoch mit klassischen Algorithmen verglichen werden. Für das Training kann je nach Datensatz und Netzarchitektur die Validierung anhand vollkommen unbekannter Daten mit einem anderen Parameterraum nötig sein, um Überanpassung zu vermeiden. Auch Regularisierung ist bei komplexen Netzen mit kleineren Datensätzen eine sinnvolle Methode zur Vermeidung der Überanpassung. Dabei spielt die Größe und Qualität des Datensatzes ebenfalls eine wichtige Rolle. Kleine Änderungen des Datensatze können für optimale Ergebnisse eine erneute Parameter-Optimierung erfordern.

4 Beschreibung der Datensätze aus Propellerserien

Um die Modelle zu trainieren werden verschiedene Datensätze verwendet. Zum einem kommt die sogenannte Wageningen-B-Serie zum Einsatz, eine im Schiffbau weit verbreitete Propellerserie. Zum anderen wird die von der Firma ProMarin entwickelte Propellerserie F115 verwendet. Die beiden folgenden Kapitel behandeln die Art der vorhandenen Daten und ihre Herkunft. Dazu wird auch auf Korrelationen in den Daten eingegangen.

4.1 Wageningen B-Serie

Für die Wageningen B-Serie sind 1981 120 Modellpropeller in den Niederlanden in einem Freifahrtversuch getestet worden. Zu den Propellern sind Freifahrtdiagramme wie in Abschnitt 2.7 erläutert erstellt und daraus Polynome abgeleitet worden. Anhand dieser Polynome lassen sich nun die Momenten- und Schubbeiwerte in Abhängigkeit des Flächenverhältnisses, der Steigung und des Fortschrittsgrades vorhersagen.[13] Zur Berechnung der Datensätze ist ursprünglich ein Skript der TUHH verwendet worden. Dieses ist so abgewandelt worden, dass es Daten in einem bestimmten Bereich erzeugt und zusätzlich die beiden Beiwerte mit einem zufälligen Rauschen von 2 % bezogen auf den erzeugten Wert versieht. Das Rauschen dient zur Erzeugung einer gewissen Abweichung der Ergebnisse aus den Polynomen, sodass sie mit realen CFD-Rechnungen vergleichbar sind. In Tabelle 3 sind die Bereiche der erzeugten Daten aufgelistet. So können beliebig viele Trainingsdaten erzeugt werden. Damit im weiteren Verlauf der Arbeit der Entwurfsprozess wie in Abschnitt 2.7 beschrieben durchgeführt werden kann, müssen zusätzlich zu den erzeugten Daten k_Q^* und k_T^* berechnet werden.

Tabelle 3: Verwendete Parameter und ihre Bereiche von den Wageningen-Daten

Abkürzung	Bedeutung	Variationsraum
Z	Flügelzahl	4 bis 6
Ae/Ao	Flächenverhältnis	0,7 bis 1,1
P/D	Gesamtsteigung der Flügel	0,4 bis 1,4
J	Fortschrittsgrad	0,1 bis 1,5
k_Q	Momentenbeiwert	0,0027 bis 0,1435
k_T	Schubbeiwert	0,0001 bis 0,7150
η	Wirkungsgrad	0,0042 bis 0,8738
k_Q^*	Momentenbeiwert / J^3	0,0001 bis 14,3521
k_T^*	Schubbeiwert / J^2	0,0001 bis 71,5030

Die erzeugten Datensätze liegen als `Pandas DataFrame`[25] vor. In Tabelle 4 ist ein solcher Datensatz von 4496 Datenpunkten auf Korrelationen hin untersucht worden. Dazu ist die Funktion `dataframe.corr` verwendet worden, die zugehörige Formel ist in Gleichung 43 gezeigt. Dabei stellen x_i und x_j jeweils die Datensätze dar, zwischen denen die Korrelation berechnet werden soll, mit $\bar{x}$ wird der Mittelwert der jeweiligen Daten bezeichnet und σ ist ihre Varianz. Bei den Ergebnissen bedeuten Werte von 1, dass die Daten korrelieren, und Werte von 0, dass keine Korrelation zwischen den Daten vorliegt. [3, S. 567] Werte im Betrag kleiner als 0,3 sind in dieser Tabelle vereinfachend als

null gekennzeichnet, da dann kaum eine Korrelation vorliegt. Daran lässt sich erkennen, dass wie zu erwarten eine beinahe perfekte Korrelation zwischen Momenten- und Schubbeiwerten, als auch Korrelationen zwischen Steigung, Fortschrittsgrad und den beiden Beiwerten vorliegt. Der Wirkungsgrad korreliert ebenfalls mit dem Fortschrittsgrad, da der Betriebspunkt einen hohen Einfluss auf ihn hat. k_Q^* und k_T^* korrelieren ebenfalls untereinander stark wie die beiden Beiwerte, sowie k_T^* leicht mit dem Fortschrittsgrad und dem Wirkungsgrad. Obwohl k_Q^* ebenfalls mit dem Fortschrittsgrad berechnet wird, korrelieren diese wenig. Die weiteren Werte Flügelzahl und Flächenverhältnis korrelieren nicht mit anderen Werten.

$$\rho_{ij} = \frac{1}{N} \sum_{n=1}^{N} \frac{(x_{ni} - \bar{x}_i) \cdot (x_{nj} - \bar{x}_j)}{\sigma_i \sigma_j} \tag{43}$$

Tabelle 4: Korrelationstabelle für die Wageningen-Daten

	Z	Ae/Ao	P/D	J	k_Q	k_T	η	k_Q^*	k_T^*
Z	1	0	0	0	0	0	0	0	0
Ae/Ao	0	1	0	0	0	0	0	0	0
P/D	0	0	1	0,4471	0,6507	0,4884	0	0	0
J	0	0	0,4471	1	-0,3641	-0,5571	0,7603	-0,4001	-0,5261
kQ	0	0	0,6507	-0,3641	1	0,9622	-0,4239	0,3985	0,4383
kT	0	0	0,4884	-0,5571	0,9622	1	-0,5452	0,4464	0,5250
η	0	0	0	0,7603	-0,4239	-0,5452	1	-0,4965	-0,6204
k_Q^*	0	0	0	-0,4001	0,3985	0,4464	-0,4965	1	0,9673
k_T^*	0	0	0	-0,6261	0,4383	0,5250	-0,6204	0,9673	1

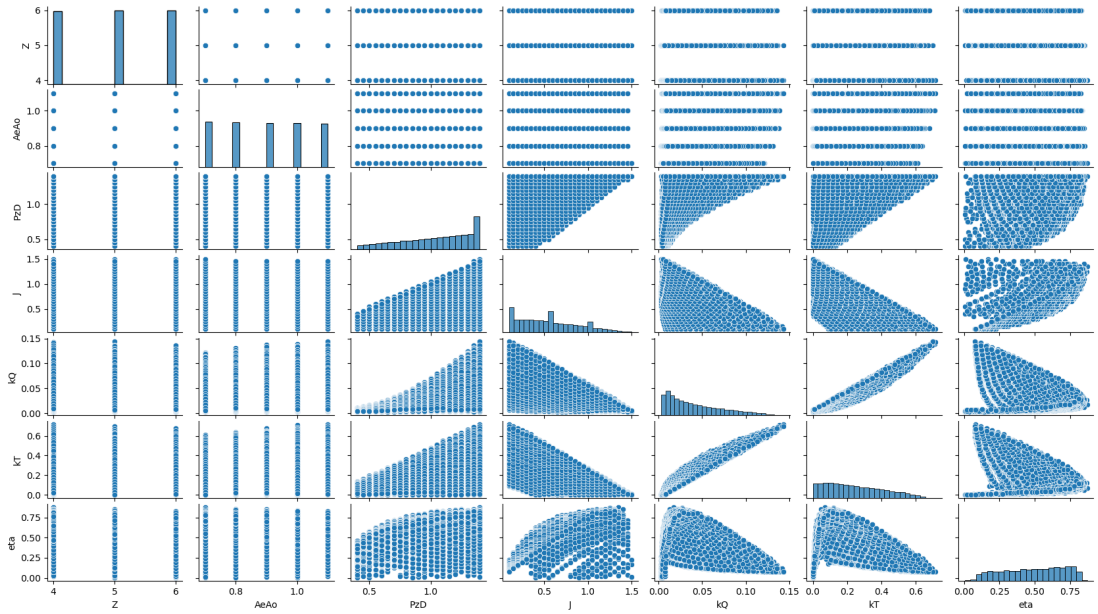

Abbildung 11: Grafische Darstellung der Datenverteilung und der Korrelationen der Wageningen-Daten

In Abbildung 11 sind die Ergebnisse noch einmal für die Grunddaten grafisch zusammengefasst. In dieser Darstellung sind auf der Diagonalen die Verteilungen der jeweiligen Größen zu erkennen. In den anderen Grafiken sind die beiden Größen jeweils zueinander aufgetragen. Je mehr diese Darstellung dann einer Geraden gleicht, desto höher ist die Korrelation. In dieser Darstellung lässt sich gut erkennen, dass k_Q und k_T stark korrelieren. Außerdem sind alle Parameter außer der Flügelzahl und dem Flächenverhältnis recht gleichmäßig über ihren Variationsraum verteilt. Bei der Flügelzahl und dem Flächenverhältnis sind gängige Abstufungen gewählt worden.

4.2 Propellerserie F115 von ProMarin

Dies ist wie eingangs erwähnt eine eigene Serie der Firma ProMarin, welche sowohl mit Modellversuchen als auch CFD-Rechnungen erstellt worden ist. Anhand der in der Serie festgelegten Geometrieparameter kann eine vollständige Propellergeometrie erzeugt werden. Die in dieser Arbeit verwendeten Daten stammen aus CFD-Rechnungen, um vorerst kein weiteres Rauschen aus beispielsweise Versuchsdaten herausfiltern zu müssen.

Tabelle 5: Verwendete Parameter und ihre Bereiche von den ProMarin-Daten

Abkürzung	Bedeutung	Wertebereich (744)	bereinigter Wertebereich (461)
Z	Flügelzahl	4 bis 6	4 bis 6
Ae/Ao	Flächenverhältnis	0,7; 0,9; 1,1	0,7; 0,9; 1,1
AreaTab	Serienbezeichnung	F115	F115
P/D	Gesamtsteigung der Flügel	0,4 bis 1,4	0,4 bis 1,4
cps	Steigung an den Flügelspitzen bezogen auf P/D	0,9; 0,95; 1,0	0,9; 0,95; 1,0
F02	Wölbung des Profils bei $r/R = 0{,}2$	0,025; 0,05; 0,075	0,025; 0,05; 0,075
F06	Wölbung des Profils bei $r/R = 0{,}6$	0,015; 0,03; 0,045	0,015; 0,03; 0,045
F10	Wölbung des Profils bei $r/R = 1{,}0$	0,005; 0,01; 0,015	0,005; 0,01; 0,015
J	Fortschrittsgrad	0,1 bis 1,5	0,1 bis 1,5
$10k_Q$	Momentenbeiwert	-0,6896 bis 1,5841	0,0708 bis 1,6106
k_T	Schubbeiwert	-0,7069 bis 0,7815	0,0003 bis 0,7815
η	Wirkungsgrad	-1138,5444 bis 82,0822	0,0047 bis 0,7467
$10k_Q^*$	Momentenbeiwert/J^3	-0,0204 bis 161,0554	0,0051 bis 161,554
k_T^*	Schubbeiwert/J^2	-0,3671 bis 78,1544	0,0007 bis 78,1544

In Tabelle 5 werden einmal die Abkürzungen zu den einzelnen Parametern sowie ihr Wertebereich erläutert. Die 744 Datenpunkte beschreiben wie die Wageningen-Daten jeweils Freifahrtdiagramme. Zu erkennen ist anhand der Tabellendaten, dass eine Vorauswahl der Daten basierend auf einem positiven Schub- und Drehmomentenbereich nötig ist. Wie bei den Wageningen-Daten sind k_Q^* und k_T^* berechnet worden. Zudem

ist auch der Wirkungsgrad berechnet worden, dieser enthält ebenfalls nicht realistische Werte. Deshalb ist in Tabelle 5 auch noch der bereinigte Wertebereich aufgetragen, der Datensatz besteht dann noch aus 461 Datenpunkten.

Von diesen 461 Datenpunkten entfallen noch 242 auf eine Variation von Betriebspunkt, Steigung und Flächenverhältnis ohne Veränderung der Spitzensteigung cps und der Profilwölbung, also vergleichbar mit den Daten aus der Wageningen-B-Serie. Zusätzlich zu den Ergebnissen von Betriebspunkt und Geometrie werden noch Druckdaten auf der Propelleroberfläche ausgewertet. Dabei liegen die Koordinaten der Propelleroberfläche und der zugehörige Druck in Tabellenform vor. Daraus wird der prozentuale Anteil der Koordinaten, bei denen der Dampfdruck unterschritten wird, berechnet und als kavitierende Fläche des Propellers verwendet. Es wird angenommen, dass die Netze auf der Propelleroberfläche ungefähr für alle Rechnungen gleich verteilt sind. Dadurch ist eine Vergleichbarkeit der Werte untereinander möglich, auch wenn die tatsächliche Kavitationsfläche abweicht. Eine solche Druckverteilung auf dem Flügel ist in Abbildung 12 gezeigt. Dabei liegen bei der Skala Werte unterhalb von -98 unter dem Dampfdruck. Die prozentualen Werte der kavitierenden Fläche variieren in den Daten zwischen 0,07 und 0,51.

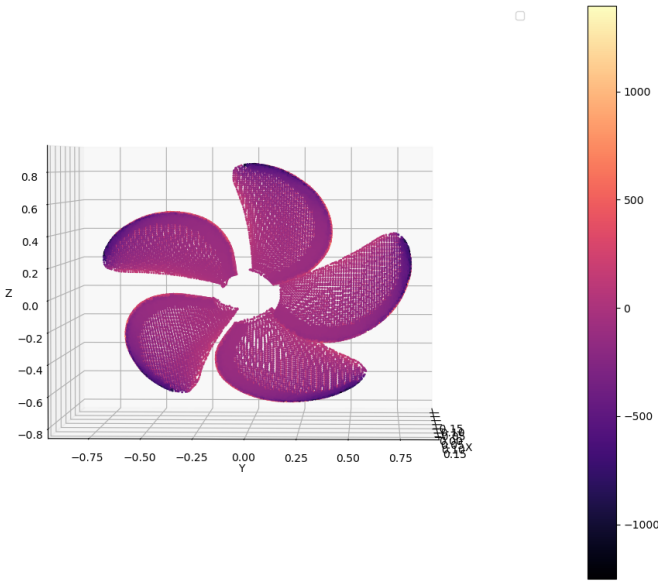

Abbildung 12: Beispielhafte Druckverteilung auf der Propelleroberfläche eines Propellers von ProMarin

Zuerst werden die Abhängigkeiten der 242 Datenpunkte für den Betriebspunkt bewertet, um einen Vergleich mit den Wageningen-Daten zu ermöglichen. Es ergibt sich die Korrelationsmatrix in Tabelle 6. Werte unter 0,3 sind dabei wieder vereinfacht worden zu null. Die größten Korrelationen sind wie zuvor und zu erwarten zwischen den Momenten- und Schubbeiwerten und $10k_Q^*$ und k_T^*, sowie von dem Momentenbeiwert zu der Steigung und von dem Schubbeiwert zum Fortschrittsgrad zu finden. Der Wirkungsgrad und der Fortschrittsgrad korrelieren ebenfalls wie bei den Wageningen-Daten. Sowohl $10k_Q^*$ als auch k_T^* korrelieren bei diesen Daten jedoch leicht mit Wirkungsgrad, Fortschrittsgrad und k_T. Fortschrittsgrad und Steigung korrelieren untereinander kaum,

auch die Flügelzahl und das Flächenverhältnis haben wieder keine Korrelation mit anderen Werten. Die Daten der Korrelationstabellen sind für die Wageningen-B-Serie und die Serie F115 insgesamt vergleichbar.

Tabelle 6: Korrelationstabelle für die Daten zur Betriebspunktbestimmung der ProMarin-Daten

	Z	Ae/Ao	P/D	J	$10k_Q$	k_T	η	$10k_Q^*$	k_T^*
Z	1	0	0	0	0	0	0	0	0
Ae/Ao	0	1	0	0	0	0	0	0	0
P/D	0	0	1	0,4459	0,6819	0,4756	0,3114	0	0
J	0	0	0,4459	1	-0,3227	-0,5717	0,6936	-0,5216	-0,6041
$10k_Q$	0	0	0,6819	-0.3227	1	0,9450	0	0,5116	0,4752
k_T	0	0	0,4756	-0,5717	0,9450	1	-0,3675	0,5888	0,5997
η	0	0	0,3114	0,6936	0	-0,3675	1	-0,6848	-0,6142
$10k_Q^*$	0	0	0	-0,5216	0,5116	0,5888	-0,6142	1	0,9791
k_T^*	0	0	0	-0,6041	0,4752	0,5997	-0,6848	0,9791	1

Diese Daten sind in Abbildung 13 ebenfalls grafisch dargestellt. Zu erkennen sind hier ähnliche Verläufe wie in Abbildung 11, jedoch sind weitaus weniger Daten vorhanden. Dadurch sind die Verläufe und Verteilungen nicht so stark gestreut.

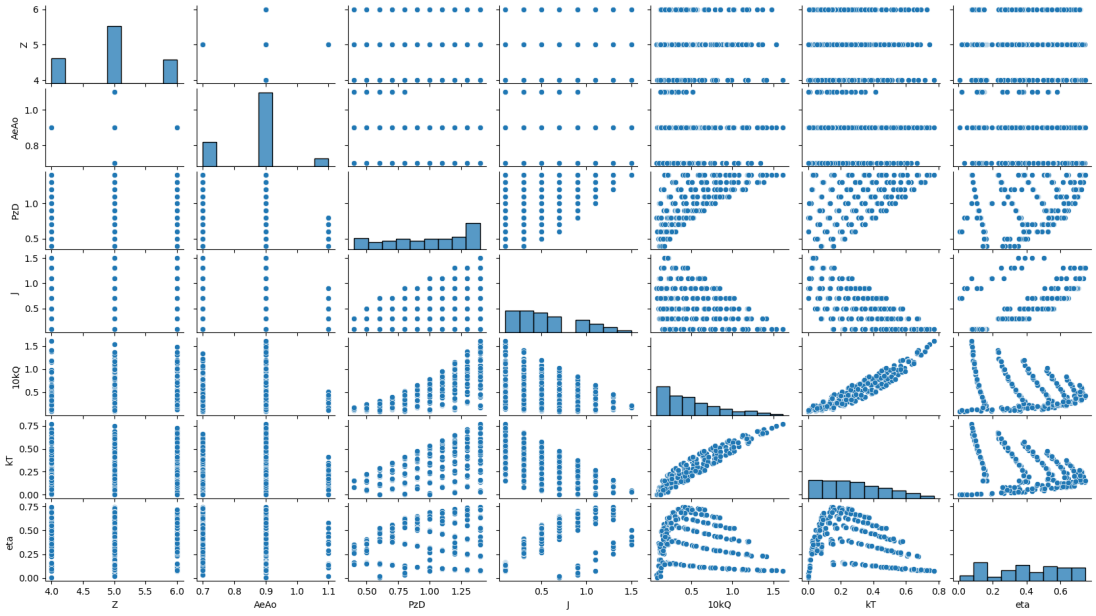

Abbildung 13: Grafische Darstellung der Datenverteilung und der Korrelationen der ProMarin-Daten für die Betriebspunktbestimmung

Nun werden die Abhängigkeiten der 461 Datenpunkte bewertet. Es ergibt sich eine weitere Korrelationsmatrix in Tabelle 7, bei dieser Matrix sind vereinfachend nur die Spalten mit sich um mehr als 2 % ändernden Werte im Vergleich zu Tabelle 6 eingetragen. Die gesamte Matrix findet sich im Anhang A in Tabelle 33. In dieser sind dieselben

Tabelle 7: Korrelationstabelle für die Daten zur Geometriebestimmung der ProMarin-Daten

	P/D	cps	F02	F06	F10	J	$10k_Q$	Kav	η
P/D	1	0	0	0	0	0,4425	0,6830	0,6630	0,3218
cps	0	1	0	0	0	0	0	0	0
F02	0	0	1	1	1	0	0	0,3133	0
F06	0	0	1	1	1	0	0	0,3133	0
F10	0	0	1	1	1	0	0	0,3133	0
J	0,4425	0	0	0	0	1	-0,3172	0,4310	0,7266
$10k_Q$	0,6830	0	0	0	0	-0,3172	1	0,4059	0
Kav	0,6630	0	0,3133	0,3133	0,3133	0,4310	0,4059	1	0
η	0,3218	0	0	0	0	0,7266	0	0	1

Korrelationen aus Schub- und Momentenbeiwert, Fortschrittsgrad und Steigung wie zuvor zu erkennen. Zusätzlich dazu korreliert nun auch die prozentuale Kavitationsfläche eindeutig mit der Steigung wie es zu erwarten ist, da sie einen hohen Einfluss auf das Kavitationsverhalten am Flügel hat. Der Wirkungsgrad korreliert ebenfalls wieder eindeutig mit dem Fortschrittsgrad. Die Wölbungsparameter der Flügel korrelieren untereinander perfekt mit dem Wert 1, da sie immer gemeinsam verändert werden. Allerdings korrelieren sie ansonsten mit keinen weiteren Werten, ebenso wie die anderen Werte Flügelzahl, Flächenverhältnis und in diesem Fall noch die Spitzensteigung.

Auch die Tabelle ist in Abbildung 14 grafisch dargestellt.

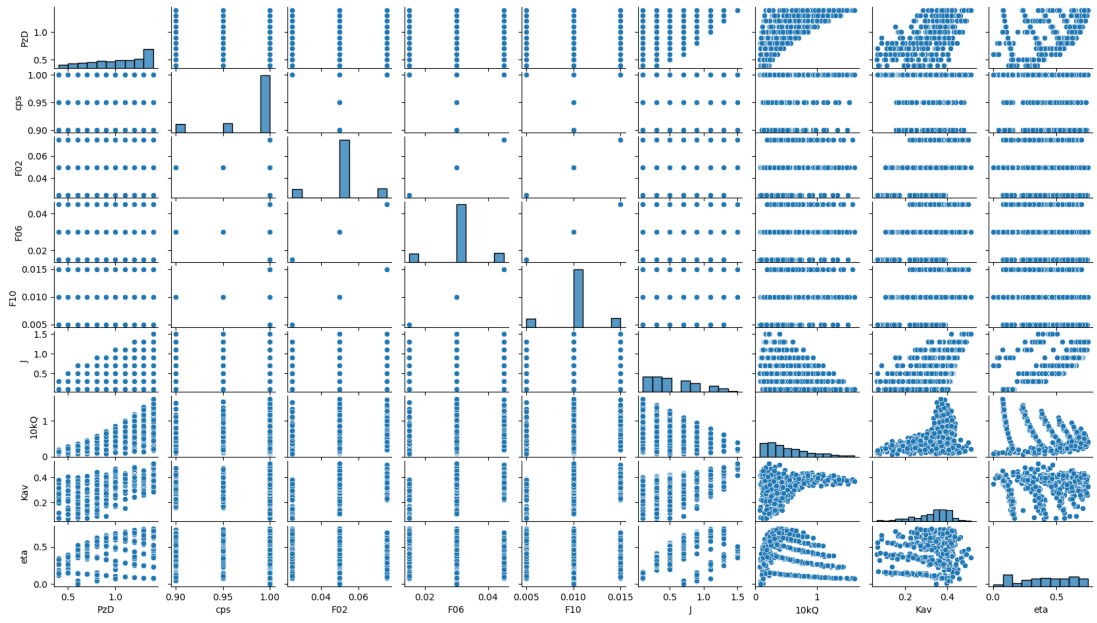

Abbildung 14: Grafische Darstellung der Datenverteilung und der Korrelationen der ProMarin-Daten zur Geometriebestimmung

5 Entwicklung verschiedener Rechenmodelle

Um die Aufgabe des KI-gestützten Entwurfs zu lösen, werden nun zwei verschiedene Algorithmen entwickelt. Zum einen wird eine Regression im Bereich des maschinellen Lernens durchgeführt sowie zum anderen ein neuronales Netz entwickelt. Beide Ansätze werden mit den zwei zuvor in Kapitel 4 beschriebenen Datensätzen trainiert und optimiert, sowie im Anschluss einander gegenübergestellt. Ein Vergleich aus einem neuronalen Netz und der Regression zu ermöglichen ist wichtig, da neuronale Netze wie von *Phanindra et al.*[11] beschrieben nicht immer die beste Lösung eines Problems sind (vergleiche auch Kapitel 3).

5.1 Entwurfsziel

In diesem Abschnitt soll mithilfe des in Abschnitt 2.7 skizzierten Entwurfsprozesses das Ziel der Rechenmodelle ermittelt werden. Im Propellerentwurf soll mit den vorgegebenen Schiffsgrößen der optimale Propeller aus den bekannten Serien gefunden werden. Dazu sind typischerweise entweder die Werte zur Berechnung von k_T^* oder von k_Q^* bekannt. In Tabelle 8 sind die jeweils benötigten Parameter gezeigt. Die meisten davon werden für beide Varianten benötigt. Anhand von Gleichung 38 oder Gleichung 40 kann nun die schiffsseitige Belastungskurve berechnet werden.

Tabelle 8: Benötigte Eingangsdaten für den Entwurfsprozess des Propellers

k_T^*	k_Q^*
Flügelzahl	
Durchmesser	
Schiffsgeschwindigkeit	
Nachstromziffer	
Mindest-Flächenverhältnis	
Schiffswiderstand	Motorleistung
Sogziffer	Gütegrad der Anordnung

In den vorhandenen Daten sind mehrere Freifahrtdiagramme hinterlegt, sodass innerhalb dieser Daten eine Optimierung vorgenommen werden muss. Dabei soll der maximale Wirkungsgrad erreicht werden, wie dies zum Beispiel auch von *Vardhan et al.*[24] als Ziel für die Propellerauslegung gesetzt worden ist.

Berechnung über Schubbeiwert

In diesem Fall sind die nötigen Größen gegeben, um k_T^* zu berechnen. Außerdem soll anhand des Burill-Diagramms bereits ein Mindest-Flächenverhältnis vorliegen. Die Flügelzahl soll ebenfalls aufgrund von Vibration oder anderen Einschränkungen bekannt sein. Damit eine Optimierung für den Wirkungsgrad vorgenommen werden kann, wird k_Q als Eingabeparameter verändert und in einem gewissen Parameterraum variiert. So kann jeweils eine Vorhersage der Ausgabeparameter η, P/D, J und k_T hinsichtlich des maximalen Wirkungsgrades verglichen werden. In Abbildung 15 ist dieser Vorgang skizziert.

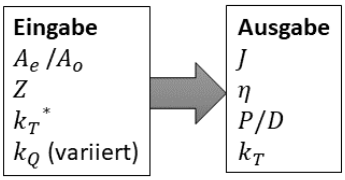

Abbildung 15: Skizze der Parameter des maschinellen Propellerentwurfs über den Schub mit k_T^*

Berechnung über Momentenbeiwert

Der zweite Fall ist von der Rechnung her ähnlich, hier kann k_Q^* berechnet werden. Auch liegt wieder ein Mindest-Flächenverhältnis und eine Flügelzahl vor. Damit die Optimierung für den Wirkungsgrad vorgenommen werden kann, wird k_T in einem gewissen Parameterraum variiert. So kann die Vorhersage der Ausgabeparameter η, P/D, J und k_Q wieder hinsichtlich des maximalen Wirkungsgrades verglichen werden. In Abbildung 16 ist der zweite mögliche Vorgang skizziert.

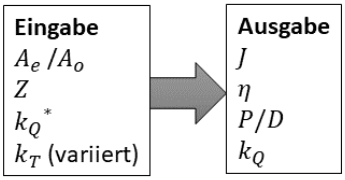

Abbildung 16: Skizze der Parameter des maschinellen Propellerentwurfs über das Moment mit k_Q^*

Berechnung der Geometrie

Die beiden zuvor genannten Methoden dienen zuerst der Ermittlung des Betriebspunktes. In den Daten von ProMarin sind wie in Abschnitt 4.2 beschrieben auch die Geometriedaten der Propeller vorhanden. Diese Daten sollen anhand des Betriebspunktes erzeugt werden, dazu werden die Parameter η, P/D, J, k_Q, k_T, A_e/A_o und Z verwendet um cps, F02, F06, F10 und die Kavitationsfläche zu bestimmen, wie in Abbildung 17 gezeigt. Hierbei wird keine weitere Optimierung vorgenommen, da es zu den optimierten Eingangsdaten bereits die zugehörige Geometrie geben sollte, die diesen Betriebspunkt erzeugen kann.

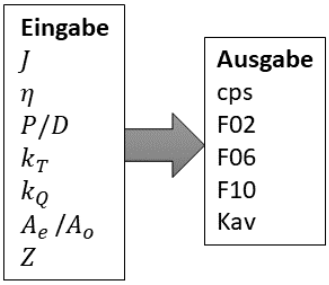

Abbildung 17: Skizze der Parameter des maschinellen Propellerentwurfs für die Propellergeometrie

5.2 Maschinelles Lernen - Regression

Für die Regression wird eine polynomiale Mehrfachregression wie in Abschnitt 2.2 beschrieben durchgeführt. Dazu werden die zuvor beschriebenen Daten von ProMarin aus dem `panda Dataframe` eingelesen und mit `scikit-learn` verarbeitet. Da nur eine abhängige Ausgangsvariable möglich ist, werden mehrere Regressionen benötigt. Der vollständige Quellcode einer Regression ist im Anhang C zu finden, hier werden nur einige Ausschnitte betrachtet.

Vergleich mit einer bekannten Regression

Für die Daten von ProMarin gibt es bereits eine Regressionsgleichung. Diese soll mit einer Regression aus `scikit-learn` verglichen werden. In der vorhandenen Regression wird anhand von Flügelzahl, Fortschrittsgrad, Steigung und Flächenverhältnis der Schubbeiwert berechnet. Die Koeffizienten sollen erneut mit den Modulen `Linear-Regression` und `PolynomialFeatures` aus `scikit-learn` berechnet werden. Dazu werden zuerst die Daten importiert und dann anschließend die Regression durchgeführt. Zur Bewertung werden der MSE der Trainings- und Testdaten sowie Varianz, Verzerrung und Bestimmtheitsmaß ausgegeben. Der Teil des Codes für die Regression ist im folgenden Abschnitt gezeigt.

```
def regression(a, b, c, d, e, degree):
  #Vorbereitung der Daten und Split
  x = pd.DataFrame(np.c_[data[a], data[b], data[c], data[d]], columns =
                                    [a, b, c, d])
  y = data[e]
  x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
                                    0.3, random_state=5)
  polynomial_features= PolynomialFeatures(degree = degree)
  x_train_poly = polynomial_features.fit_transform(x_train)

  #Die eigentliche Regression
  model = LinearRegression()
  model.fit(x_train_poly, y_train)
  x_test_poly = polynomial_features.fit_transform(x_test)

  #Ausgabe Train- und Test-MSE
  pred_test = model.predict(x_test_poly)
  MSE_test = ((y_test-pred_test)**2).sum() / len(y_test)
  pred_train = model.predict(x_train_poly)
  MSE_train = ((y_train-pred_train)**2).sum() / len(y_train)
  print("Testfehler", MSE_test, "Trainfehler", MSE_train)

  #Ausgabe von Varianz, Verzerrung, und durchschnittlichem Fehler
  y_train = y_train.values
  y_test = y_test.values
  avg_expected_loss, avg_bias, avg_var = bias_variance_decomp(model,
                            x_train_poly, y_train,
                            x_test_poly, y_test,
                            loss='mse',
                            num_rounds=200,
                            random_seed=20)
  print('Loss:', avg_expected_loss, 'bias:', avg_bias, 'variance',
                                avg_var, 'R^2', model.score(
                                x_test_poly,y_test))
  return model
```

Um die optimale Regression zu finden, sollen der Polynomgrad variiert und verschiedene Aufteilungs-Techniken für die Daten benutzt werden. Dabei können die Daten wie in dem obigen Beispiel zufällig in einem bestimmten Prozentbereich aufgeteilt werden. Alternativ können sie auch direkt in Trainings- und Testdaten aufgeteilt werden, sodass beispielsweise eine Flügelzahl-Flächenverhältnis-Kombination nicht für Training sondern ausschließlich zum Testen verwendet wird. Dies ist je nach Datensatz manchmal nötig, wie auch im *Noisenet* von *Wang et al.*[1]. Auch k-fache Kreuzvalidierung, bei der alle Daten sowohl für Trainings- als auch Testzwecke genutzt werden, soll getestet werden. Die Regressionen sollen mit den bereits bekannten Werten aus der Regression von ProMarin für fünf Flügel und einem Flächenverhältnis von 0,7 und mit den Originaldaten aus den Datensätzen verglichen werden. Zur Bewertung werden Verzerrung, Varianz, MSE und das Bestimmtheitsmaß R^2 herangezogen.

Die Ergebnisse für die Bewertungsparameter sind in Tabelle 9 gezeigt. Dabei steigen die Polynomgrade an, bis die Werte des nächsthöheren Grades nur noch Werte größer als fünf für den Fehler ausgeben. Zu erkennen ist, dass sowohl mit k-facher Kreuzvalidierung mit zehn Teilen als auch mit einer zufälligen Verteilung Polynomgrade bis sieben bzw. acht verwendet werden können. Bei der harten Aufteilung werden ausschließlich die Daten der Fünfflügler mit Flächenverhältnis 1,1 als Testdaten verwendet. So lässt sich mit einer harten Aufteilung der Daten nur der Polynomgrad eins realisieren und es ist kaum eine Regression mehr möglich. Dasselbe passiert, wenn die k-fache Kreuzvalidierung ihre Pakete nicht zufällig aus den Daten zieht, sondern linear durch die Daten durchgeht und immer genau die benötigte Anzahl für die Testdaten herausnimmt. Für die Daten und die Regression sind harte Aufteilungen also nicht geeignet. Für die anderen Regressionen lässt sich erkennen, dass die Fehler sinken bis der Testfehler wieder ansteigt. Auch die Varianz steigt, während die Verzerrung sinkt. Dieses Verhalten ist für die Regression mit zufälliger Datenaufteilung in Abbildung 18 grafisch dargestellt und verläuft wie in Abschnitt 2.4 beschrieben.

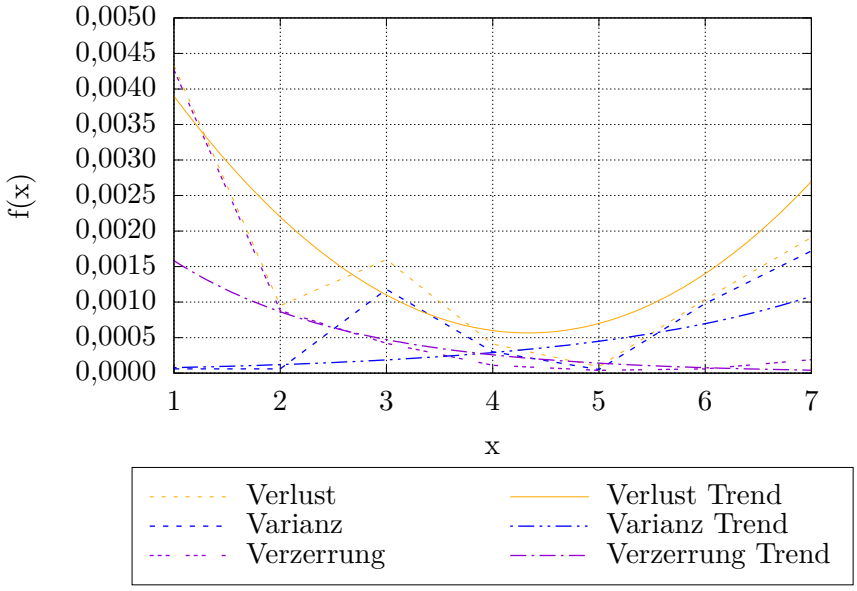

Abbildung 18: Verlauf von Verzerrung und Varianz einer Regression für k_T über polynomischen Grad

Im Vergleich mit der Regression aus den Koeffizienten von ProMarin zeigt sich, dass die erstellte Regression mit zufälliger Datenaufteilung bei Polynomgrad fünf genauso gute Ergebnisse erzielt. Damit ist diese Methode grundsätzlich auf die Daten anwendbar.

Tabelle 9: Übersicht über die Bewertungsparameter für Regressionen über Z, P/D, Ae/Ao und J zur Bestimmung von k_T

Daten-aufteilung	Polynom-grad	MSE-Test	MSE-Train	Varianz	Verzerrung	Bestimmt-heitsmaß
Zufällig	1	0,00428	0,00389	0,00006	0,00427	0,96833
Zufällig	2	0,00088	0,00085	0,00006	0,00088	0,99269
Zufällig	3	0,00049	0,00032	0,00118	0,00042	0,99443
Zufällig	4	0,00028	0,00019	0,00030	0,00011	0,99877
Zufällig	5	0,00004	0,00001	0,00005	0,00004	0,99966
Zufällig	6	0,00006	0,00001	0,00098	0,00006	0,99917
Zufällig	7	0,00006	0,00000	0,00172	0,00019	0,99853
Zufällig	8	0,00012	0,00000	0,33964	0,00725	0,80387
K-fach	1	0,00422	0,00407	0,00007	0,00412	0,96711
K-fach	2	0,00102	0,00090	0,00005	0,00094	0,99196
K-fach	3	0,00073	0,00071	0,00039	0,00039	0,99433
K-fach	4	0,00037	0,00026	0,00013	0,00014	0,99677
K-fach	5	0,00007	0,00003	0,00004	0,00004	0,99942
K-fach	6	0,00011	0,00006	0,00010	0,00006	0,99909
K-fach	7	0,00080	0,00049	0,01634	0,00042	0,99393
Hart	1	0,00568	0,00382	0,00014	0,00561	0,95569

Regression für die Auslegung

Mit derselben Methode wie zuvor sollen jetzt Regressionen erstellt werden, sodass der in Abschnitt 5.1 skizzierte Entwurfsprozess durchgeführt werden kann. Dazu werden mehrere aufeinander aufbauende Regressionen benötigt. Im ersten Schritt wird k_T aus k_T^* oder k_Q aus k_Q^* mithilfe von Flügelzahl, Flächenverhältnis und dem jeweils anderen Beiwert berechnet. Dann kann aus k_T, k_Q, Ae/Ao und Z der zugehörige Fortschrittsgrad bestimmt werden. Im Anschluss werden mit den bisher bestimmten Parametern P/D und η bestimmt. Dann ist der Betriebspunkt bestimmt und aus diesem können die Geometrieparameter und die Kavitationsfläche abgeleitet werden. In Tabelle 10 sind die jeweiligen Ausgangsparameter mit zugehörigem optimierten Polynomgrad und den jeweiligen Bewertungsparametern gezeigt. Die gezeigten Ergebnisse basieren auf einer Regression mit den bereinigten 242 Daten, da dies geringere Fehler ermöglicht. Insbesondere bei η sind ansonsten sehr hohe Fehlerwerte aufgetreten, da dieser Wert in dem nicht bereinigten Datensatz stark schwankt (vergleiche Abschnitt 4.2). Dadurch sind allerdings nur Polynomgrade von bis zu drei in einem Bereich mit Fehler unter fünf möglich, es kann also keine gute Darstellung des Fehlerverlaufs gezeigt werden. Insbesondere die Regression für die Spitzensteigung cps hat ein niedriges Bestimmtheitsmaß. Für alle anderen Parameter sind die Fehler und die Bestimmtheitsmaße in einem guten Rahmen.

Tabelle 10: Übersicht über die Bewertungsparameter der Regressionen für den Propellerentwurf

Ausgang	Polynomgrad	MSE-Test	MSE-Train	Varianz	Verzerrung	Bestimmtheitsmaß
k_Q	2	0,00365	0,00601	0,00081	0,00363	0,96663
k_T	2	0,00086	0,00109	0,00022	0,00089	0,96958
J	2	0,00850	0,00839	0,00113	0,00837	0,94211
P/D	3	0,00003	0,00001	0,00010	0,00004	0,99895
η	3	0,00129	0,00062	0,00379	0,00134	0,86155
cps	1	0,00125	0,00111	0,00002	0,00125	0,01477
F02	3	0,00001	0,00000	0,00001	0,00001	0,94730
F06	3	0,00000	0,00000	0,00000	0,00000	0,94730
F10	3	0,00000	0,00000	0,00000	0,00000	0,94731
Kav	3	0,00093	0,00046	0,00180	0,00091	0,88938

In Abbildung 19 und 20 sind jeweils zwei Darstellungen der Ergebnisse der Regressionen über ihre Trainings- und Testdaten gezeigt. Dazu sind auf der x- und y-Achse jeweils gegeneinander die realen Werte aufgetragen, sodass sich eine schwarze Gerade ergibt. Auf der y-Achse variierend sind zu derselben x-Achse die vorhergesagten Werte aus der Regression aufgetragen. So lassen sich Abweichungen schnell erkennen. Außerdem sind in Tabelle 11 die Fehler einiger einzelner Parameter bezogen auf den ganzen Datensatz zu sehen. Es lässt sich erkennen, dass die Regression sowohl über das Moment als auch den Schub für die Bestimmung der Betriebspunktparameter geeignet ist. Allerdings sind die Fehler für die Bestimmung der Geometrie höher.

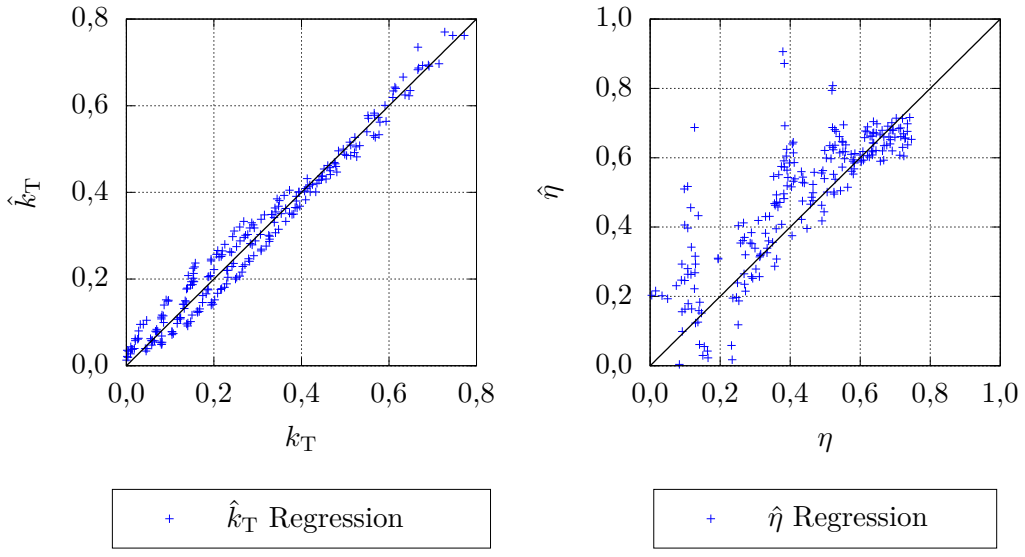

Abbildung 19: Vergleich der Ergebnisse von k_T und η mit der Regression über den Schub mit den ProMarin-Daten

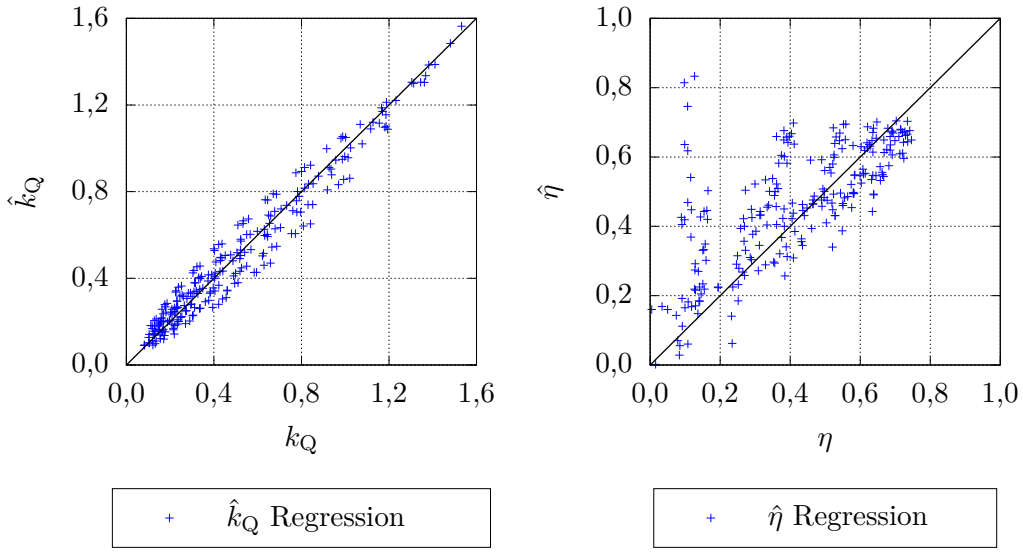

Abbildung 20: Vergleich der Ergebnisse von k_T und η mit der Regression über das Drehmoment mit den ProMarin-Daten

Tabelle 11: Übersicht der Fehler für die einzelnen Parameter in den Regressionen für den Propellerentwurf

	J	P/D	k_T/k_Q	η	cps	F02
Über k_T^*	0,05	0,03	0,001	0,02	0,007	0,94
Über k_Q^*	0,06	0,05	0,005	0,03	0,006	0,62

5.3 Neuronale Netze

Für die Bestimmung der Ausgabeparameter mithilfe von neuronalen Netzen muss eine Netzstruktur aufgebaut werden. Dabei soll grundsätzlich mit `nn.linear` aus PyTorch[28] gearbeitet werden. Es soll zwischen einer und drei verdeckten Schichten geben, wobei das Optimum gefunden werden soll. Ebenso soll die Knotenanzahl in diesen Schichten variiert werden. Als Aktivierungsfunktionen sollen ReLU, LeakyReLU, hyperbolischer Tangens und Softplus miteinander verglichen werden, sowie die beiden Löser SGD und Adam mit der zugehörigen Anfangslernrate. Wie zuvor soll der MSE für die Bewertung verwendet werden. Beim Zusammenstellen der Trainingsdaten soll auch die Losgröße variiert werden. Die Datensätze werden in 30 % Testdaten und 70 % Trainingsdaten aufgeteilt. Üblicherweise werden Daten in 60 % Trainingsdaten, 20 % Validierungsdaten und 20 % Testdaten aufgeteilt.[20, S. 157] Da zum Teil wenige Daten zur Verfügung stehen und die Modelle abschließend über einen zusätzlichen Test bewertet werden sollen, wird hier eine abweichende Verteilung gewählt. In Tabelle 12 sind die Parameter und ihr Variationsraum aufgeführt. Für die Optimierung während des Trainings wird `RayTune`[23] verwendet und vorerst in 50 Epochen trainiert. Dabei wählt `RayTune` die Parameterkombinationen zufällig aus. Danach wird der MSE der Optimierungen verglichen und mit der besten Parameterkombination noch eine Rechnung über 200 Epochen

durchgeführt. Es muss dabei für jeden Datensatz eine Optimierung durchgeführt werden, da sich bereits bei geringfügig abweichenden Daten andere optimale Parameter ergeben. Darauf ist bereits in der Literaturrecherche in Kapitel 3 mit der Veröffentlichung von *Šegota et al.*[4] hingewiesen worden.

Tabelle 12: Variationsraum der Optimierungsparameter für die neuronalen Netze

Parameter	Variationsraum
Anzahl der verdeckten Schichten	1 bis 3
Knotenanzahl pro Schicht	8, 16, 32
Aktivierungsfunktion	ReLU, LeakyReLU, Softplus, Tanh
Löser	SGD, Adam
Anfangslernrate	0,0001 bis 0,01
Losgröße	1, 2, 4

Für den Datenimport wird aus den txt-Tabellen mit den in Kapitel 4 erläuterten Daten ein `pandas Dataframe` erstellt. Für die Darstellung der Flügelzahl wird eine OneHot-Codierung verwendet, da es keine halben Flügelzahlen gibt. Das bedeutet, dass beispielsweise bei Flügelzahlen vier, fünf und sechs die Zahlen als [1, 0, 0], [0, 1, 0] und [0, 0, 1] dargestellt werden und sich damit jeweils aus drei binären x-Werten anstelle von einem dezimalen zusammensetzen.[15, S. 25 f.]

Auf den nächsten Seiten wird der grundsätzliche Aufbau der neuronalen Netze gezeigt. Ein Beispiel für ein vollständiges Programm ist im Anhang B gezeigt. Dabei wird im ersten Code-Abschnitt das Modell, in diesem Fall ein Multilayer-Perzeptron, initialisiert. Dort werden die Schichtenzahl, ihre jeweilige Knotenzahl und die Aktivierungsfunktion bestimmt.

Implementierung des Multilayer-Perzeptrons in Python

```
class MLP(nn.Module):
  '''
  Multilayer Perzeptron
  '''
  def __init__(self, l1=64, l2=32):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(6, l1),
      nn.ReLU(),   ## nn.LeakyReLU(), nn.Tanh(), nn.Softplus() ##
      nn.Linear(l1, l2),
      nn.ReLU(),
      nn.Linear(l2, 4)
      )
  def forward(self, x):
    return self.layers(x)
```

In dem nächsten Abschnitt werden die Daten mit der gewählten Losgröße importiert und das Training aufgerufen. Dazu wird das Modell und der Löser mit seiner Anfangslernrate definiert, und dann über eine Wahrheitsschleife ausgeführt. `RayTune` beendet

die Optimierung selbst.

Definition des Trainingsaufrufs in Python

```
def train_mydata(config):
    random.seed(config["seed"])
  np.random.seed(config["seed"])
  torch.manual_seed(0)
  train_data, test_data = dataimp("Name_of_File.txt", batch_size=
                                  config["batch"])
  model = MLP(config["l1"],config["l2"])
  optimizer = torch.optim.Adam(model.parameters(), lr=config["lr"])
  epochs = config(["epochs"])
  print("Start Training")
  while True:
    train(model, optimizer, train_data, epochs)
    loss = test(model, test_data)
    # Tune aufrufen
    tune.report(mean_loss=loss)
```

Der dritte Abschnitt zeigt die zuvor aufgerufene Trainingsschleife. Dabei wird mit dem Modell und den Trainingsdaten eine Schleife für die definierte Epochenanzahl ausgeführt, in der eine Ausgabe auf Basis der Daten gemacht und mit dem MSE zu den realen Daten bewertet wird. Mit dem MSE werden dann über den Löser die Gewichte des Netzes verändert.

Definition der Trainingsschleife in Python

```
def train(model, optimizer, train_data, epochs):
  # Verlustfunktion
  loss_function = nn.MSELoss(reduction='mean')
  # Trainingsschleife
  for epoch in range(0, epochs):
    # Iteration ueber Trainingsdaten
    for i, data in enumerate(train_data, 0):
      inputs, targets = data
      inputs, targets = inputs.float(), targets.float()
      targets = targets.reshape((targets.shape[0], 4))
      # Gradienten nullen
      optimizer.zero_grad()
      outputs = model(inputs)
      # Verlust berechnen
      loss = loss_function(outputs, targets)
      # Rueckgabe des Verlusts
      loss.backward()
      # Optimieren
      optimizer.step()
```

Danach wird aus der Aufruffunktion `train_mydata` die Testschleife aufgerufen, in der mit den Testdaten auch der MSE zu den Vorhersagen berechnet wird. Anhand dieses Fehlers bewertet `RayTune` dann die Netze.

Definition der Testschleife in Python

```
def test(model, test_data):
  model.eval()
  loss_total = 0
  loss_function = nn.MSELoss(reduction='mean')
  with torch.no_grad():
    # Iterieren ueber die Testdaten
    for i, data in enumerate(test_data, 0):
      inputs, targets = data
      inputs, targets = inputs.float(), targets.float()
      targets = targets.reshape((targets.shape[0], 4))
      outputs = model(inputs)
      loss = loss_function(outputs, targets)
      loss_total += loss.item()
  return loss_total / len(test_data)
```

Der Aufruf des gesamten Programms ist im letzten Abschnitt gezeigt. Dort wird mit den Trainingsparametern in der Variable `config` das Training mit `RayTune` aufgerufen.

Definition der Main-Datei in Python

```
if __name__ == '__main__':
  import argparse
  parser = argparse.ArgumentParser()
  args, _ = parser.parse_known_args()
  # zum Abbrechen
  sched = AsyncHyperBandScheduler()
  analysis = tune.run(
    train_mydata,
    metric="mean_loss",
    mode="min",
    name="exp",
    scheduler=sched,
    stop={
      "mean_loss": 0.001,
      "training_iteration": 1
    },
    resources_per_trial={
      "cpu": 4,
    },
    num_samples=200,
    config={
      "lr": tune.loguniform(1e-4, 1e-2),
      "batch": tune.choice([1, 2, 4]),
      "l1": tune.choice([8, 16, 32]),
      "l2": tune.choice([8, 16,32]),
      "epochs": (50),
      "seed": tune.randint(0,10000)
    })
  print("Best config is:", analysis.best_config)
```

5.3.1 Wageningen

Zunächst sollen anhand der künstlich erzeugten Wageningen-Daten einige grundsätzliche Verhaltensweisen der neuronalen Netze für diesen Anwendungsfall überprüft werden. Dabei werden neben verschiedenen Datenmengen auch verschiedene Regularisierungsmethoden, Aktivierungsfunktionen und Löser untersucht. Zusätzlich zu der Training-Test-Aufteilung können bei den Wageningen-Daten noch vollständig unbekannte Datensätze erzeugt werden. Mit einem solchen unbekannten Datensatz sollen die optimierten Netze zusätzlich überprüft werden.

Optimierung der Aktivierungsfunktion sowie Löser und Anzahl der verdeckten Schichten

Im ersten Schritt werden die Aktivierungsfunktionen und Löser variiert. Dabei werden die in Abschnitt 2.2 genannten Aktivierungsfunktionen Softplus, ReLU, LeakyReLU und Tanh eingesetzt. Als Löser werden Adam und SGD verglichen. Mit den zuvor in Tabelle 12 genannten Variationen liegen so erste Optimierungsergebnisse vor. Die Aktivierungsfunktionen und Löser sind dafür in separaten Läufen bewertet worden. Die jeweils besten Kombinationen der Optimierungsläufe sind in Tabelle 13 dargestellt, dabei ist der MSE nach 50 Epochen gezeigt. Im weiteren Verlauf wird mit Softplus als Aktivierungsfunktion und Adam als Löser gerechnet. Mit dieser optimalen Kombination wird dann im nächsten Schritt die Schichtenanzahl ermittelt. Das Ergebnis ist in den letzten beiden Zeilen zu erkennen. In beiden Fällen tritt eine Verschlechterung des MSE ein, daher wird im weiteren Verlauf mit zwei verdeckten Schichten gerechnet.

Tabelle 13: Optimierte Netzparameter und Fehler für verschiedene Aktivierungsfunktionen, Löser und Schichtenanzahlen mit den Wageningen-Daten

Verdeckte Schichten	Knotenanzahl	Aktivierungsfunktion	Löser	Anfangslernrate	Losgröße	MSE
2	32, 16	Softplus	Adam	0,001950	2	0,0004
2	16, 16	Softplus	SGD	0,007602	2	0,0298
2	8, 32	ReLU	Adam	0,001121	1	0,0010
2	16, 32	ReLU	SGD	0,008888	1	0,0138
2	32, 32	LeakyReLU	Adam	0,002213	2	0,0010
2	16, 32	LeakyReLU	SGD	0,008888	1	0,0108
2	8, 32	Tanh	Adam	0,001633	4	0,0010
2	16, 32	Tanh	SGD	0,008888	1	0,0094
1	32	Softplus	Adam	0,008693	1	0,0173
3	16, 32, 32	Softplus	Adam	0,009894	2	0,0301

In Abbildung 21 sind die zuvor in der Tabelle genannten Ergebnisse grafisch dargestellt. Dabei ist der Test- und der Trainingsfehler über den Trainingsepochen aufgetragen. Zu erkennen ist ein geringer Einfluss der Aktivierungsfunktion auf das Ergebnis. Der Löser beeinflusst den MSE deutlich stärker, sodass alle Optimierungen mit dem Löser SGD einen deutlich höheren MSE aufweisen und nur beispielhaft ein Fehlerverlauf für den Löser SGD gezeigt ist. Die Kurven sind mit Bézier geglättet. Dabei beruht die

Bézier-Anpassung auf den Bernsteinpolynomen, welche für Grad p und eine beliebige Funktion $f(t)$ wie folgt aussehen:

$$B_p[f(t)] = \sum_{i=0}^{p} f(\frac{i}{p})\Phi_{ip}(t), 0 \leq t \leq 1 \tag{44}$$

$$\Phi_{ip}(t) = \binom{p}{i} t^i (1-t)^{p-i}, i \in [0, p] \tag{45}$$

Das Bernstein-Bézier-Polynom geht daraus hervor:

$$P(t) = \sum_{i=0}^{p} \Phi_{ip}(t) V_i, 0 \leq t \leq 1 \tag{46}$$

Dabei handelt es sich grundlegend um die Bernsteinpolynome, allerdings stellen die Punkte in V_i ein Polygon mit p Seiten dar, welches auch Bézier-Polygon genannt wird. Über eine Veränderung der Punkte in V_i wird die Form des Polygon verändert und damit die Kurve die es beschreibt. Daher gehen Bézier-Kurven nicht exakt durch die Punkte, aber werden in der Form über das Polygon von den Punkten beeinflusst. So liegen die Kurven in dem von ihren zugehörigen Punkten bestimmten Korridor.[18, S. 5 ff.] Dies ist für die Kurven der Trainings- und Testfehler wichtig, da eine exakte Abbildung der Punkte aufgrund des starken Schwingens nicht erwünscht ist.

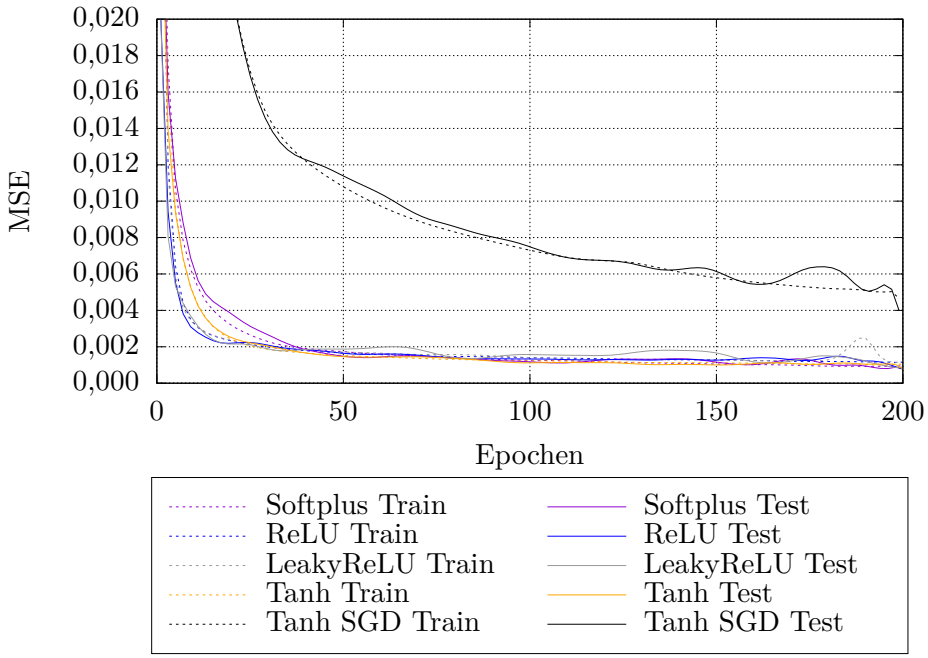

Abbildung 21: Einfluss der Aktivierungsfunktion auf Trainings- und Testfehler

Wie am Anfang des Kapitels erwähnt können mit den Wageningen-Polynomen noch weitere Datensätze erzeugt werden. In Abbildung 22 ist eine Vorhersage mit einem so erzeugten unbekannten Datensatz aus dem zuvor ermittelten optimalen Netz gezeigt. Dazu sind wieder auf der x- und y-Achse jeweils gegeneinander die realen Werte aufgetragen. Eine interessante Erkenntnis ist, dass k_T ein direkter Ausgabewert aus dem

neuronalen Netz sein muss. Es ist beispielhaft der Vergleich aus einem über k_T^* und J^2 berechneten k_T gezeigt und dem aus dem Netz direkt ausgegeben. Bei der Berechnung von k_T bringt sich der Fehler von J quadratisch in k_T ein. Aus dem Grund soll k_T in der Ausgabeschicht mit ausgegeben werden. Dasselbe gilt für k_Q, da es berechnet über k_Q^* von J^3 abhängt.

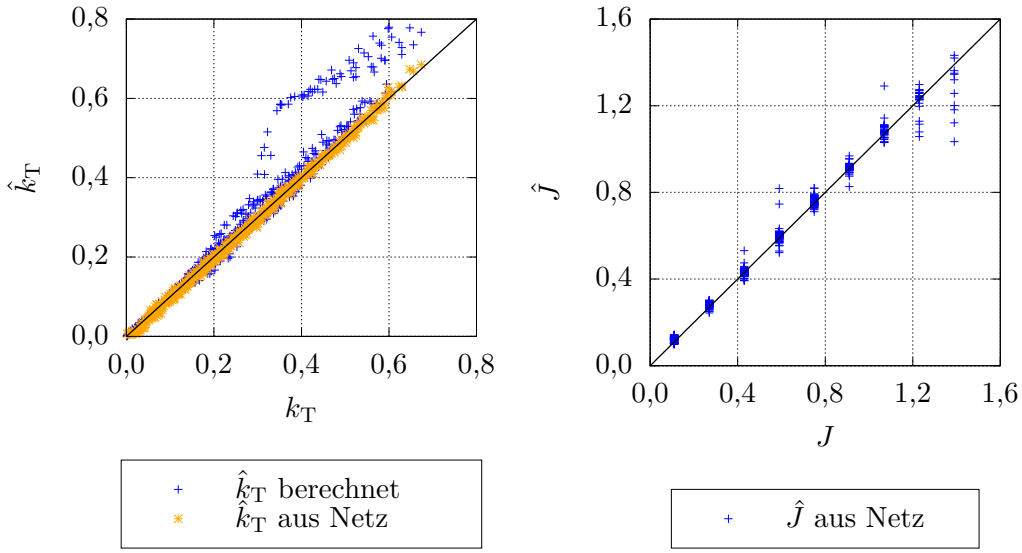

Abbildung 22: Vergleich der Ergebnisse der optimierten Netze von k_T und J bei 4496 Trainingsdaten

Datenmenge

Da bei den Wageningen-Daten beliebig große Datenmengen erzeugt werden können, wird weiterhin anhand eines neuronalen Netzes ausprobiert, mit wie vielen Daten eine gute Bewertung möglich ist. Dazu werden nun drei Datensätze verwendet mit je 247, 891 und 1484 Datenpunkten. Die Verteilungen der Daten sind in Abbildung 23, 24 und 25 gezeigt. Dabei ist zu erkennen, dass keine weiteren Flügelzahlen hinzugefügt worden sind. Auch die Flächenverhältnisse sind nur für den Datensatz mit 1484 Daten um zwei Werte erweitert worden. Bei der Steigung und dem Fortschrittsgrad sind weitere Stützstellen aufgrund der höheren Variationsmenge hinzugekommen.

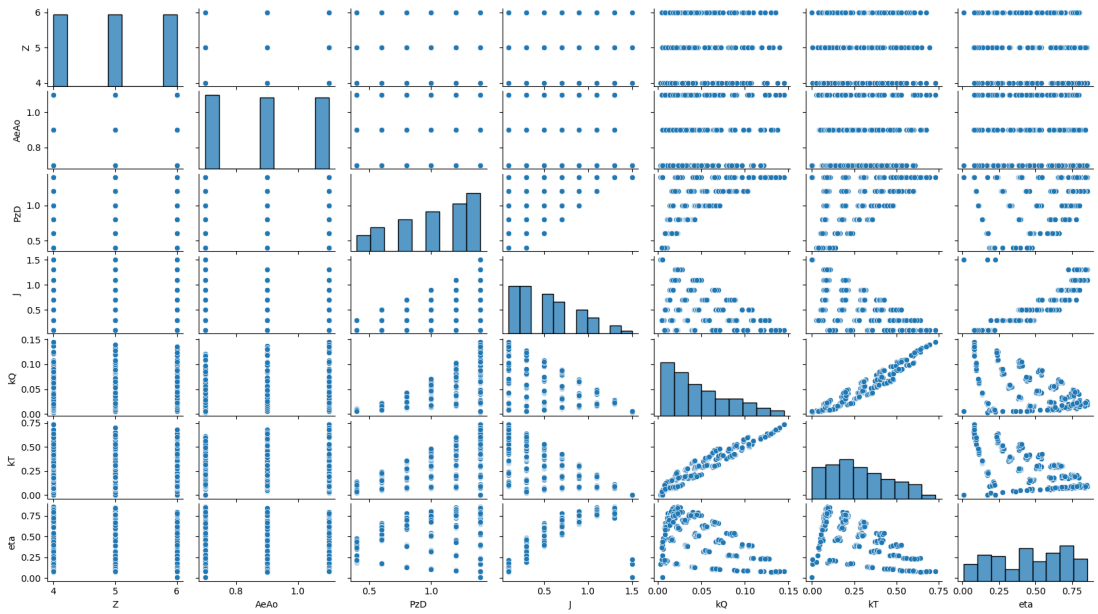

Abbildung 23: Grafische Darstellung der Datenverteilung und Korrelationen der 247 Wageningen-Daten

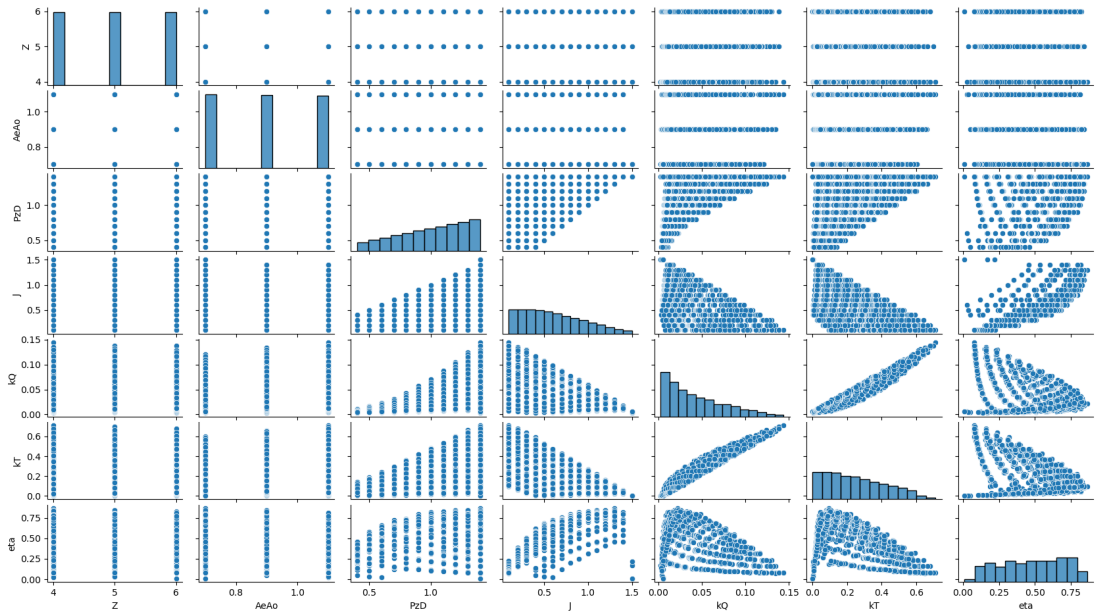

Abbildung 24: Grafische Darstellung der Datenverteilung und Korrelationen der 891 Wageningen-Daten

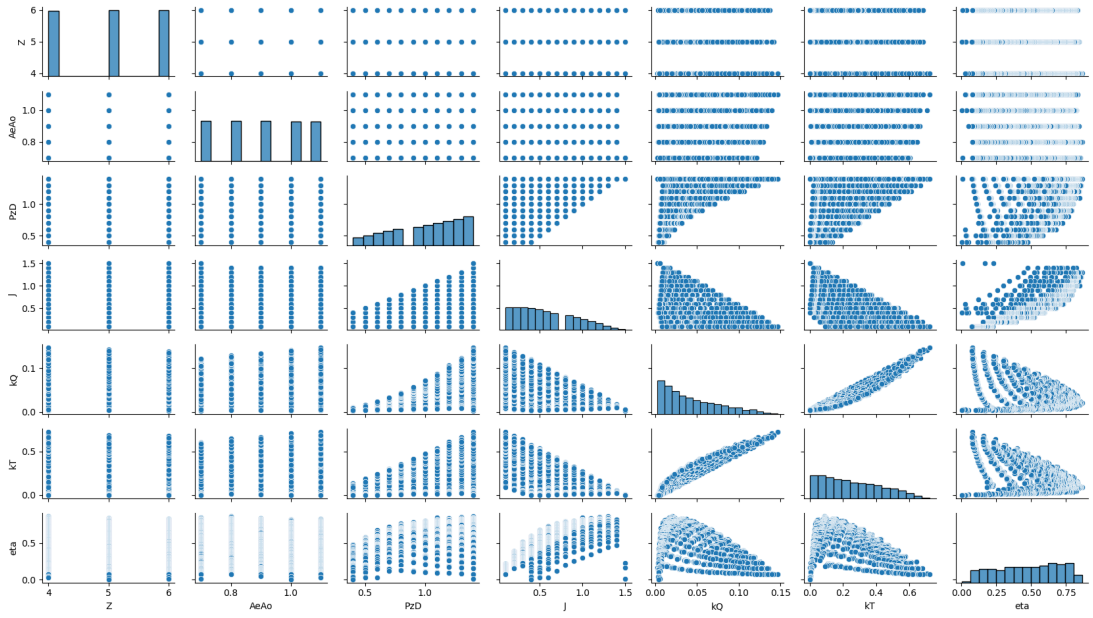

Abbildung 25: Grafische Darstellung der Datenverteilung und Korrelationen der 1484 Wageningen-Daten

Für jeden Datensatz wird eine eigene Optimierung durchgeführt, da bereits kleine Abweichungen zu anderen optimalen Trainingsparametern führen (vergleiche Kapitel 3). Alle Daten sind mit zwei verdeckten Schichten und der Aktivierungsfunktion Softplus trainiert worden. Die Ergebnisse der Optimierung sind in Tabelle 14 gezeigt. Es ist ein geringer Einfluss der Datenmenge auf den MSE zu erkennen, wobei dieser von 247 zu 891 Daten zunächst sinkt und dann von 891 zu 1484 Daten etwa gleich bleibt.

Tabelle 14: Optimierte Netzparameter und Fehler für die Netze der verschiedenen Datenmengen

Datenmenge	Knotenanzahl	Lernrate	Losgröße	MSE
247	8, 16	0,009748	1	0,0030
891	32, 8	0,004969	1	0,0017
1484	32, 16	0,004946	1	0,0018

In Abbildung 26 sind die Verläufe der Fehler der Trainings- und Testdatensätze gezeigt. Es ist kaum ein Einfluss zu erkennen und für alle drei Datensätze liegen die Testabweichungen am Ende des Trainings auf den Trainingsabweichungen. Das bedeutet, dass die Vorhersage für die Testdaten so gut ist wie für die Trainingsdaten. Einzig bei den ersten fünf Epochen sind die Trainingsabweichungen höher, je kleiner der Datensatz ist. Anhand dieser Graphen ist also noch keine Bewertung bezüglich der nötigen Datenmenge möglich.

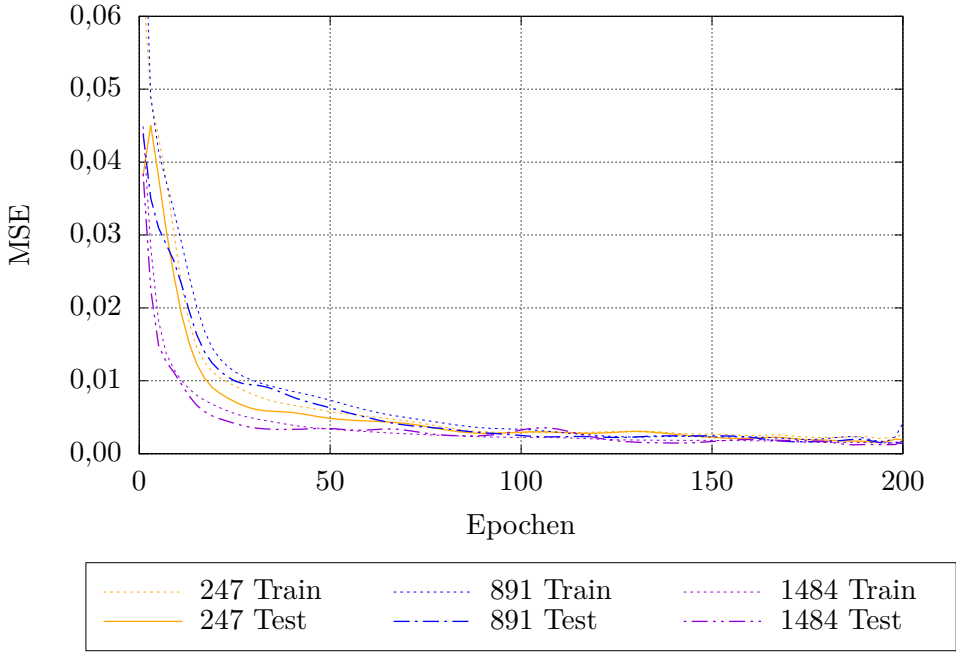

Abbildung 26: Einfluss der Datenmenge auf Trainings- und Testfehler bei den Wageningen-Daten

Zusätzlich soll nun die Vorhersage der Daten mit dem unbekannten Testdatensatz verglichen werden, um darüber eine Bewertung zu ermöglichen. In Tabelle 15 sind dazu die Testfehler der einzelnen Größen mit diesem unbekannten Datensatz aufgeführt. Dabei ist zu erkennen, dass für größere Trainingsdatensätze alle Fehler bei den unbekannten Daten sinken. Am niedrigsten ist der Fehler für den Datensatz mit den 4496 Trainingspunkten.

Tabelle 15: Fehler für die einzelnen Ausgabeparameter der Netze des Datenmengenvergleichs mit einem unbekannten Testdatensatz

Datenmenge	P/D	J	η	k_T
247	0,0104	0,0090	0,0076	0,0004
891	0,0095	0,0078	0,0059	0,0003
1484	0,0030	0,0030	0,0020	0,0002
4496	0,0013	0,0014	0,0019	0,0001

Für die Steigung und den Schubbeiwert sind in Abbildung 27 wieder Vergleichsdarstellungen zu den realen Daten aufgetragen. Die Gerade gibt dabei wieder den realen Wert dar, die Abweichung in y-Richtung den Fehler. Dort ist zu erkennen, dass alle drei Vorhersagen den Sollwerten folgen. Es gibt jedoch Ausreißer insbesondere bei der Steigung, diese sind nahezu identisch für 247 und 446 Daten und verringern sich etwas für den Datensatz mit 1484 Daten. Grundsätzlich zeigen diese Versuche mit den unbekannten Daten, dass ein größerer Trainingsdatensatz für dieses Problem bessere Ergebnisse erzielt.

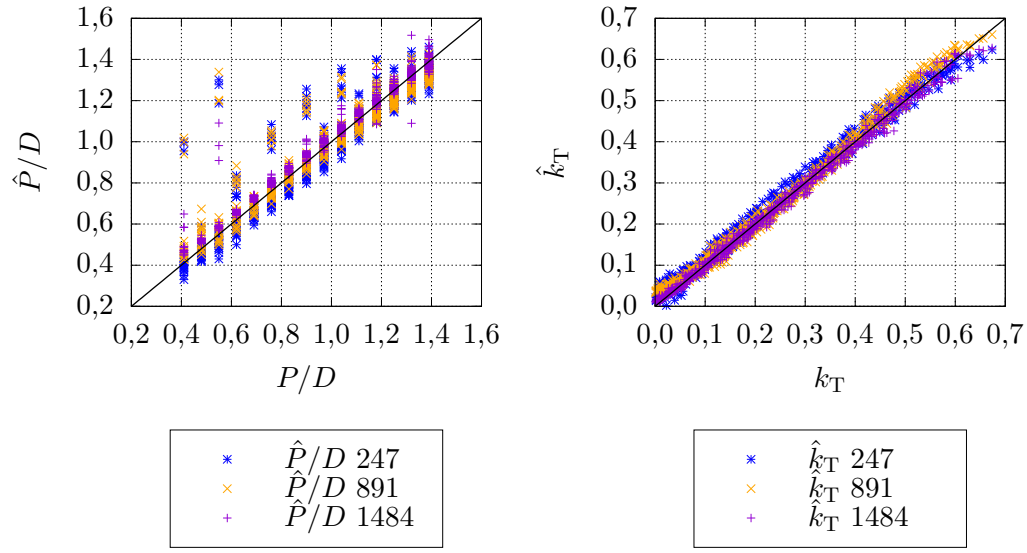

Abbildung 27: Vergleich der Ergebnisse der optimierten Netze von k_T und P/D bei verschiedenen Datenmengen

Regularisierung und Datenaufteilung

Da bei geringen Datenmengen Überanpassung zu befürchten ist, vergleiche *Šegota et al.*[5] in Kapitel 3, sollen im folgenden Abschnitt die zuvor in Abschnitt 2.5.3 behandelten Regularisierungsmöglichkeiten auf den großen mit 4496 und den kleinen Datensatz mit 247 Datenpunkten angewendet werden. Dabei werden Dropout-Wahrscheinlichkeiten für verschiedene Schichten, L1- und L2-Regularisierung, sowie Stochastic Weight Averaging und k-fache Kreuzvalidierung verwendet. Mit Dropout werden zufällig einzelne Knoten ausgeschaltet, sodass zu starke Überanpassung vermieden wird. L1- und L2-Regularisierung bestrafen hohe Gewichte mit zusätzlichen Termen im MSE. SWA vergrößert das Plateau auf dem sich die Gewichte einpendeln. K-fache Kreuzvalidierung ist besonders für kleine Datensätze interessant, da dort alle Daten sowohl als Test- als auch als Trainingsdaten verwendet werden. In Tabelle 16 sind die Optimierungsparameter gezeigt.

Tabelle 16: Variationsraum der Optimierungsparameter für die Regularisierung neuronaler Netze

Parameter	Variationsraum
Dropout-Wahrscheinlichkeit	0,01, 0,05, 0,1, 0,2
λ für L1- und L2-Regularisierung	0,001, 0,01, 0,1
Anfangslernrate für SWA	0,01, 0,05, 0,1
k der k-fachen Kreuzvalidierung	5

Für die k-fache Kreuzvalidierung wird eine Aufteilung in fünf Datensätze gewählt, sodass immer jeweils 80 % der Daten als Trainingsdaten und 20 % als Testdaten verwendet werden. Als Aktivierungsfunktion wird weiterhin Softplus und als Löser Adam

verwendet. In Tabelle 17 sind die jeweiligen mit `RayTune` optimierten Läufe gezeigt.

Tabelle 17: Optimierte Netzparameter und Fehler für die Netze mit verschiedenen Regularisierungsmethoden mit den Wageningen-Daten

Datenmenge	Regularisierung	Faktor	Knotenanzahl	Lernrate	Losgröße	MSE
4496	Dropout beide Schichten	p=0,01	32, 16	0,001992	2	0,0015
4496	Dropout Schicht 1	p=0,01	32, 16	0,003583	4	0,0011
4496	Dropout Schicht 2	p=0,05	32, 8	0,004885	4	0,0012
4496	L1-Regularisierung	λ=0,001	8, 8	0,000108	1	0,0313
4496	L2-Regularisierung	λ=0,001	16, 8	0,004096	4	0,0307
4496	SWA	L=0,01	16, 8	0,008693	4	0,0011
4496	k-fache Kreuzvalidierung	k=5	16, 8	0,001040	1	0,0014
247	Dropout beide Schichten	p=0,05	16, 16	0,003323	1	0,0051
247	Dropout Schicht 1	p=0,01	16, 16	0,007420	4	0,0037
247	Dropout Schicht 2	p=0,01	16, 16	0,007420	4	0,0039
247	L1-Regularisierung	λ=0,001	32, 8	0,001974	2	0,0267
247	L2-Regularisierung	λ=0,001	32, 32	0,003323	2	0,0245
247	SWA	L=0,01	8, 8	0,008894	1	0,0057
247	k-fache Kreuzvalidierung	k=5	32, 32	0,000174	1	0,0361

Es ist zu erkennen, dass anhand des MSE keine Verbesserung im Vergleich zu der Optimierung ohne Regularisierung möglich ist. Dieser liegt für die 4496 Daten ohne Regularisierung bei 0,0004 und für die 247 Daten bei 0,003. Das ist jedoch auch nicht zu erwarten, da die Regularisierungsmethoden nicht zwingend zu einer Verringerung des Fehlers, sondern zu einer besseren Anpassung an unbekannte Datensätze führen sollen. In Abbildung 28 sind die Verläufe der Trainings- und Testfehler für SWA, L1- und L2-Regularisierung sowie zum Vergleich ohne Regularisierung für 4496 Daten gezeigt. Deutlich zu erkennen ist der höher liegende Trainfehler bei der L1- und L2-Regularisierung. Dies liegt daran, dass in der Ausgabe des MSE ebenfalls die dazu addierten Terme enthalten sind. Der Testfehler ist ebenfalls höher, weil die Gewichte nicht mehr so gut zu den Daten passen. Für SWA ergibt sich ein interessantes Bild, es wird nach Epoche 140 zugeschaltet und bringt den Datensatz zu deutlich schlechterer Konvergenz. Das zeugt davon, dass der Datensatz vermutlich bereits groß genug ist, um die Überanpassung rein durch die Größe der Test- und Trainingsdatensätze zu verhindern.

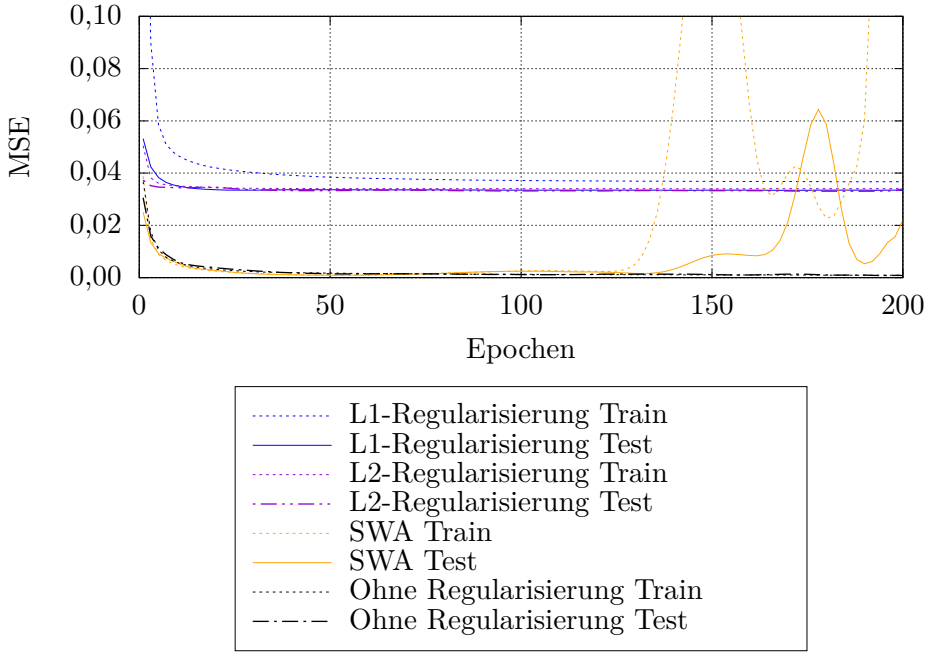

Abbildung 28: Einfluss der Regularisierungsmethoden SWA, L1- und L2-Regularisierung auf Trainings- und Testfehler bei 4496 Daten

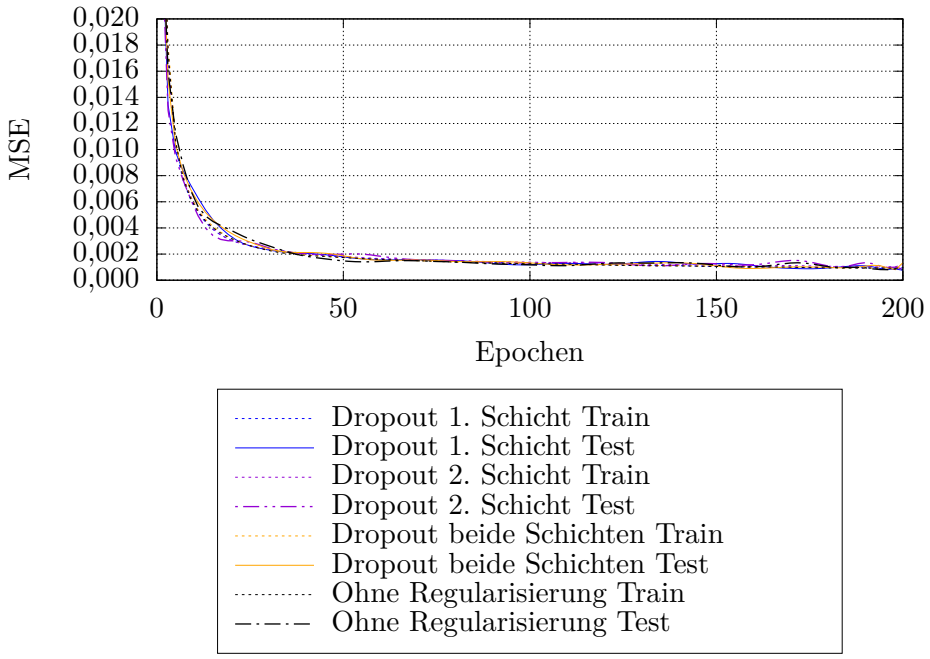

Abbildung 29: Einfluss der Regularisiungsmethode Dropout auf Trainings- und Testfehler bei 4496 Daten

In Abbildung 29 ist zu erkennen, dass der Verlauf der Trainings- und Testfehler mit und ohne Dropout gleich verläuft. Bei der Größe des Datensatzes ist keine Verbesse-

rung oder Verschlechterung des MSE durch Dropout möglich. In Abbildung 30 sind die Trainings- und Testfehler für den kleineren Datensatz und L1-, L2-Regularisierung sowie SWA gezeigt. Es ist wie für den großen Datensatz zu erkennen, dass der Trainingsfehler für die L1- und L2-Regularisierung höher liegt als für die Vergleichskurve. Der Testfehler liegt ebenfalls höher, was mit der schlechteren Anpassung der Gewichte an den Datensatz zu erklären ist. L1 liegt auch über L2, was ebenfalls Sinn ergibt, da dort ein absoluter Term und kein quadratischer hinzuaddiert wird. Bei SWA ist hier gut zu erkennen, dass die Rechnung anders als für den großen Datensatz trotzdem konvergiert. Dabei steigt der Trainings- und Testfehler ab Epoche 140 wieder leicht an, da die Gewichte ab dort gemittelt werden. Insgesamt konvergiert die Rechnung aber.

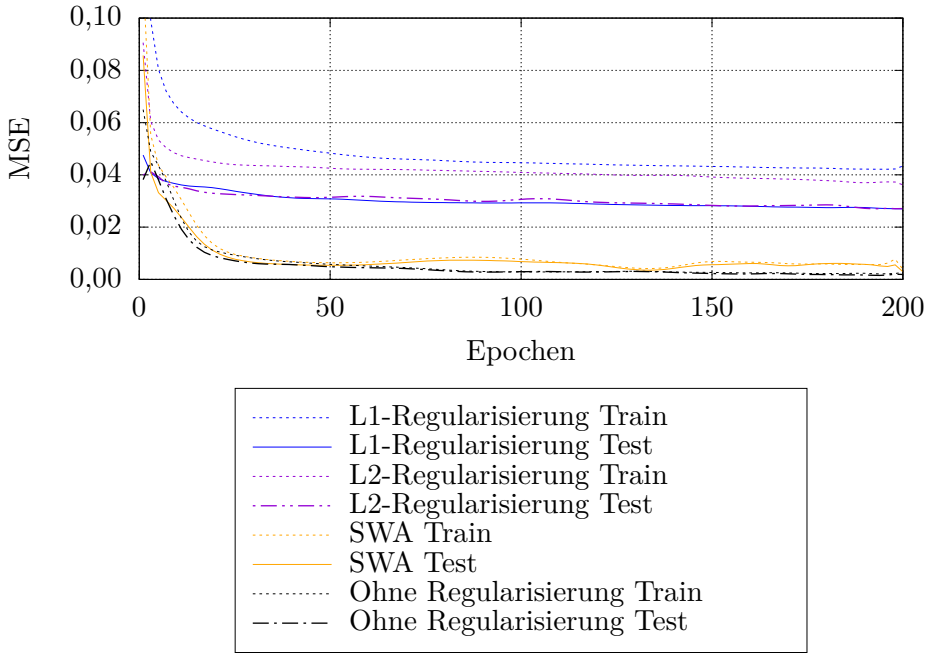

Abbildung 30: Einfluss der Regularisierungsmethoden SWA, L1- und L2-Regularisierung auf Trainings- und Testfehler bei 247 Daten

Bei den Dropout-Schichten ändert sich ähnlich wie bei dem großen Datensatz wenig, vergleiche Abbildung 31. Nur in den ersten Epochen liegen die Netze mit Dropout-Schichten mit dem MSE etwas über der Vergleichskurve.

In Abbildung 32 ist bezüglich der k-fachen Kreuzvalidierung keine Verbesserung oder Verschlechterung für den großen Datensatz zu erkennen. Bei dem kleinen Datensatz liegen Trainings- und Testfehler höher und sinken auch nach 500 Epochen nur auf 0,0059. K-fache Kreuzvalidierung soll jedoch eine für den gesamten Datensatz optimale Netzstruktur finden. Daher wird noch ein Vergleich der Vorhersage zu den unbekannten Daten herangezogen.

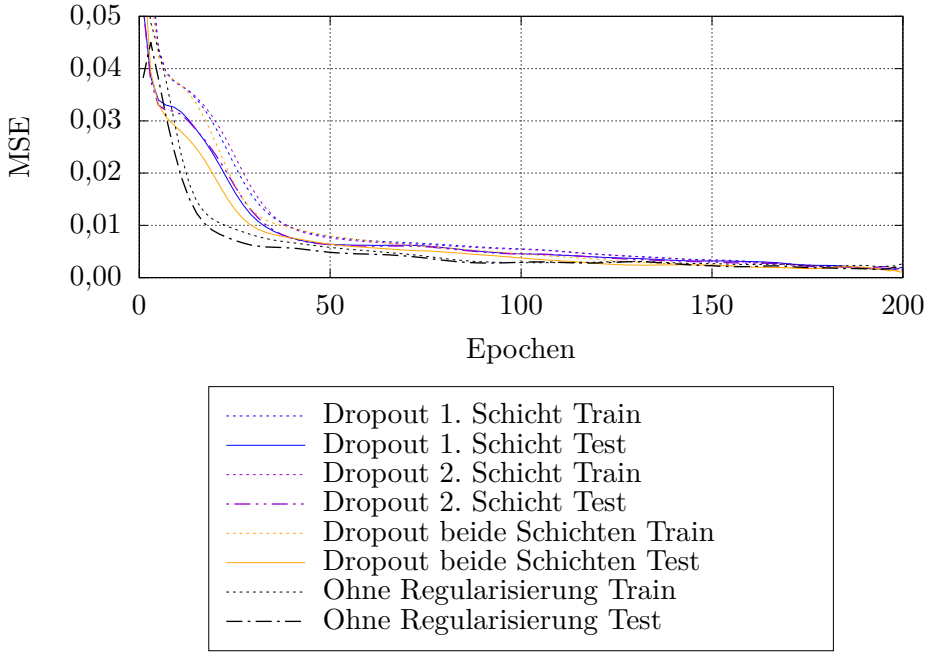

Abbildung 31: Einfluss der Regularisiungsmethode Dropout auf Trainings- und Testfehler bei 247 Daten

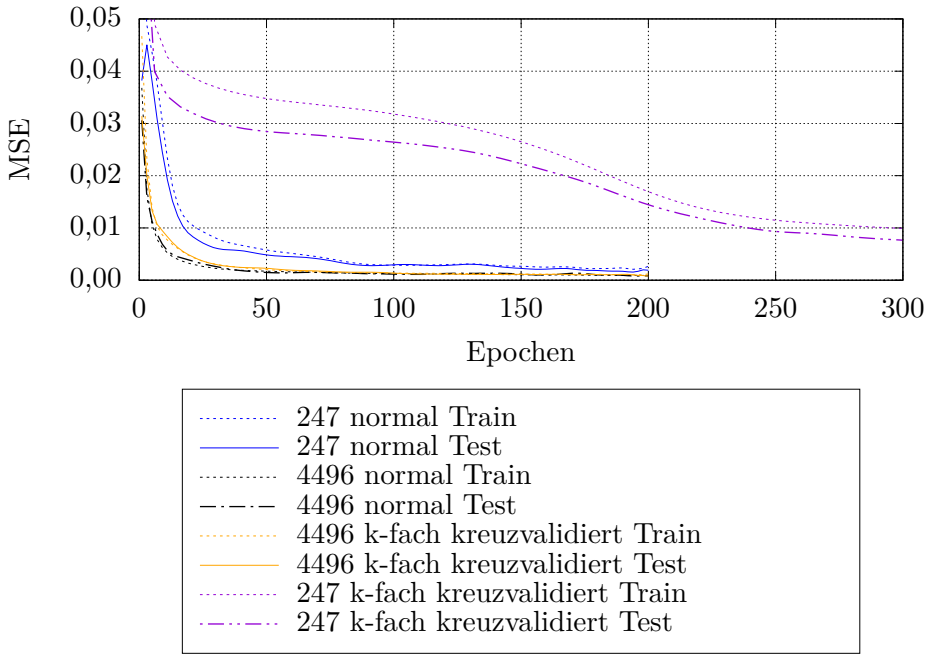

Abbildung 32: Einfluss von k-facher Kreuzvalidierung auf Trainings- und Testfehler bei verschiedenen Datenmengen

Da die größten Unterschiede zwischen den einzelnen Regularisierungsmethoden bei dem kleinen Datensatz aufgetreten sind und es dort aufgrund der geringen Datenmenge

eher zur Überanpassung kommen kann, sollen die Vorhersagen anhand des unbekannten Datensatzes mit den Netzen aus dem kleinen Datensatz überprüft werden. Die Ergebnisse für k_T sind in Abbildung 33 gezeigt. Insbesondere bei der L1- und L2-Regularisierung sind große Abweichungen von der Soll-Geraden zu erkennen. Das erklärt ebenfalls den hohen Testfehler. Mit SWA gehen die Vorhersagen für k_T wie eine Schere um den Sollwert auseinander. Dasselbe passiert für die k-fache Kreuzvalidierung, allerdings deutlich abgeschwächt. Für Dropout sind nur die Werte für eine Dropout-Wahrscheinlichkeit in beiden Schichten angegeben, diese streuen allerdings ebenfalls etwas weiter um die Soll-Gerade als die Werte ohne Regularisierung. Daher scheint eine Regularisierungsmethode selbst für den kleinen Datensatz nicht nötig zu sein.

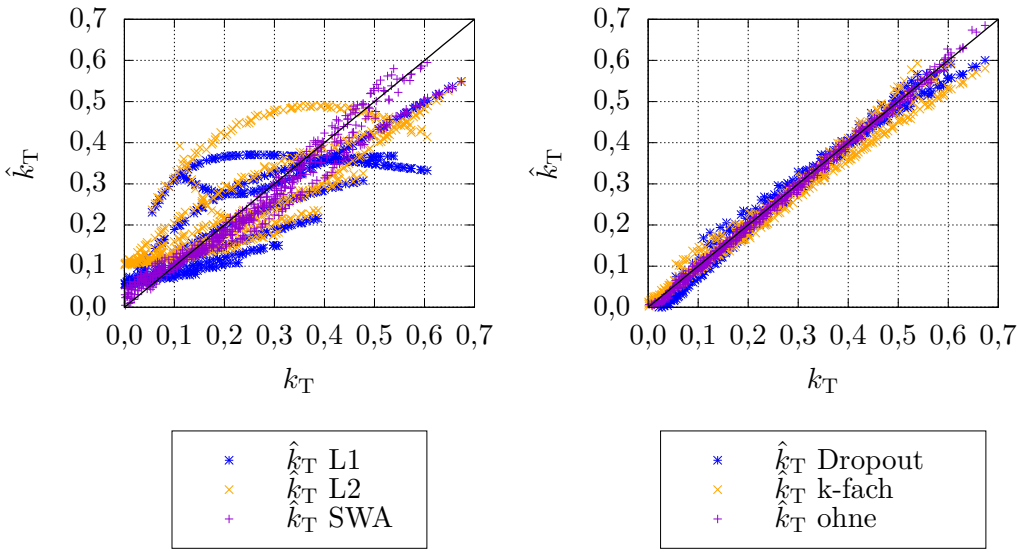

Abbildung 33: Vergleich der Ergebnisse der regulierten Netze von k_T bei 247 Daten

Skalieren

Um ebenfalls noch den Einfluss einer Skalierung auf die Datensätze zu prüfen, werden die Daten wie folgt auf den Bereich von 0-1 gebracht:

$$x_{skaliert} = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{47}$$

Das Ergebnis für diese Optimierungsläufe ist in Tabelle 18 gezeigt. In beiden Fällen ist eine Verschlechterung des MSE zu erkennen.

Tabelle 18: Optimierte Netzparameter und Fehler für die Netze mit skalierten Daten

Datenmenge	Knotenanzahl	Lernrate	Losgröße	MSE
4496	32, 16	0,005082	2	0,0015
247	32, 8	0,006554	2	0,0116

Dies ist ebenfalls in Abbildung 34 zu erkennen, die Abweichung im Trainings- und Testfehler ist für den großen Datensatz sehr gering, wobei für den Datensatz mit 247

Datenpunkten eine deutliche Abweichung erkennbar ist. Dies liegt vermutlich daran, dass alle Daten bereits in einem beinahe skalierten Bereich liegen, vergleiche Tabelle 3. Zudem geht durch das Skalieren auch das Verhältnis aus beispielsweise Schub- und Momentenkoeffizienten verloren.

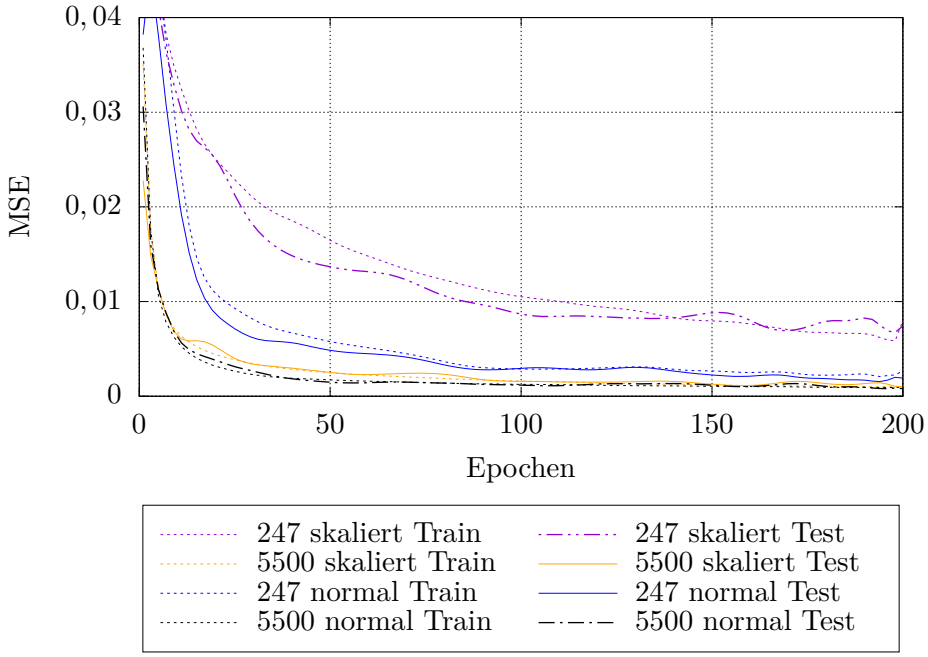

Abbildung 34: Einfluss von Skalieren auf Trainings- und Testfehler

5.3.2 ProMarin

Bei den neuronalen Netzen für das Training mit den ProMarin-Daten muss wie zuvor erwähnt in zwei Netze unterschieden werden. Dabei soll nun zuerst die Optimierung für das erste Netz zur Betriebspunkt-Bestimmung betrachtet werden, da diese auch vergleichbar zu den Schritten bei den Wageningen-Daten ist. Dazu ist in Tabelle 19 eine Übersicht gezeigt, wie der Einfluss der Aktivierungsfunktionen und Löser sich für diese Daten verhält. Anhand der Daten in Tabelle 19 werden für die weitere Optimierung die Aktivierungsfunktion Tanh und der Löser Adam verwendet.

Im nächsten Schritt soll die optimale Schichtenanzahl ermittelt werden, dies ist in den unteren beiden Zeilen in Tabelle 23 zu erkennen. Anhand des MSE nach den 50 Trainingsepochen sind weder drei noch eine Schicht besser als zwei. Daher wird im weiteren Verlauf mit zwei Schichten gerechnet. In Abbildung 35 ist nochmal der beste Lauf mit seinen Trainings- und Testfehlern gezeigt.

Tabelle 19: Optimierte Netzparameter und Fehler für verschiedene Aktivierungsfunktionen, Löser und Schichtenanzahlen mit den ProMarin-Daten für die Propellerauslegung über den Schubbeiwert

Verdeckte Schichten	Knotenanzahl	Aktivierungsfunktion	Löser	Anfangslernrate	Losgröße	MSE
2	8, 16	Softplus	Adam	0,007069	2	0,0030
2	32, 32	Softplus	SGD	0,006026	1	0,0288
2	32, 32	ReLU	Adam	0,007360	4	0,0017
2	16, 32	ReLU	SGD	0,008888	1	0,0110
2	32, 16	LeakyReLU	Adam	0,003161	4	0,0017
2	16, 32	LeakyReLU	SGD	0,008888	1	0,0111
2	32, 16	Tanh	Adam	0,003161	4	0,0013
2	8,32	Tanh	SGD	0,008536	1	0,0122
1	16	Tanh	Adam	0,008888	1	0,0139
3	32, 16, 32	Tanh	Adam	0,009648	1	0,0153

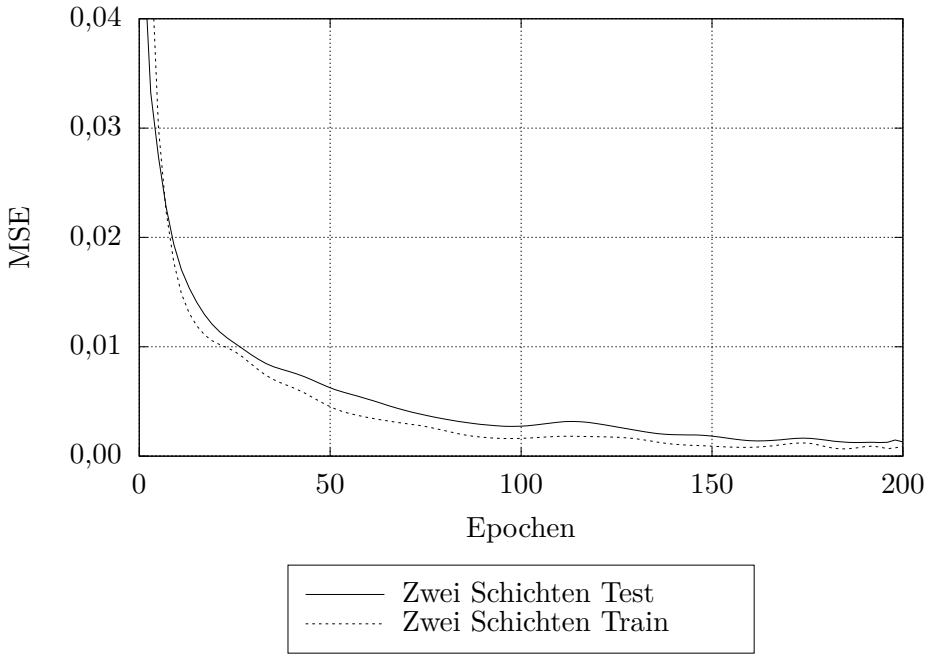

Abbildung 35: Verlauf von Trainings- und Testfehler für das optimierte Netz mit ProMarin-Daten

Regularisierung und Datenaufteilung

Da auch hier wenige Daten für das Training zur Verfügung stehen, sollen zur Vermeidung von möglicher Überanpassung die Regularisierungsmethoden aus Abschnitt 2.5.3 angewendet werden. Dazu werden die Parameter für L1, L2-Regularisierung, Dropout, SWA und k-fache Kreuzvalidierung wie zuvor bei den Wageningen-Daten in Tabelle 16

variiert. Zusätzlich sollen über eine harte Aufteilung der Trainings- und Testdaten verhindert werden, dass für das Training des Netzes bereits alle Datentypen verwendet werden. Für die harte Aufteilung werden wie bei der Regression die Fünfflügler mit Flächenverhältnis 1,1 als Testdaten benutzt. Die Ergebnisse für die optimalen Läufe sind in Tabelle 20 gezeigt. Es ist keine Verbesserung zum vorherigen MSE von 0,0013 erkennen.

Tabelle 20: Optimierte Netzparameter und Fehler für verschiedene Regularisierungsmethoden mit den ProMarin-Daten

Regularisierung	Faktor	Knoten-anzahl	Lernrate	Los-größe	MSE
Dropout beide Schichten	p=0,01	32, 16	0,003995	1	0,0024
Dropout Schicht 1	p=0,05	32, 16	0,009894	4	0,0018
Dropout Schicht 2	p=0,01	16, 32	0,003980	1	0,0024
L1-Regularisierung	λ=0,001	16, 32	0,003161	1	0,0046
L2-Regularisierung	λ=0,001	16, 32	0,003161	1	0,0032
SWA	L=0,01	8, 8	0,007942	4	0,0035
k-fache Kreuzvalidierung	k=5	8, 8	0,000123	1	0,0420
Harte Datenaufteilung	-	16, 16	0,003607	1	0,0023

In Abbildung 36 sind für Dropout und die harte Aufteilung die Verläufe der Trainings- und Testfehler gezeigt. Da alle Trainings- und Testfehler vergleichbar verlaufen, sollen diese nochmal anhand einer Ausgabe geprüft werden. Da im Falle der ProMarin-Daten keine neuen Daten erzeugt werden können, wird der gesamte Datensatz genommen und für alle Punkte, unabhängig ob sie vorher im Trainings- oder im Testdatensatz waren, eine Ausgabe erzeugt.

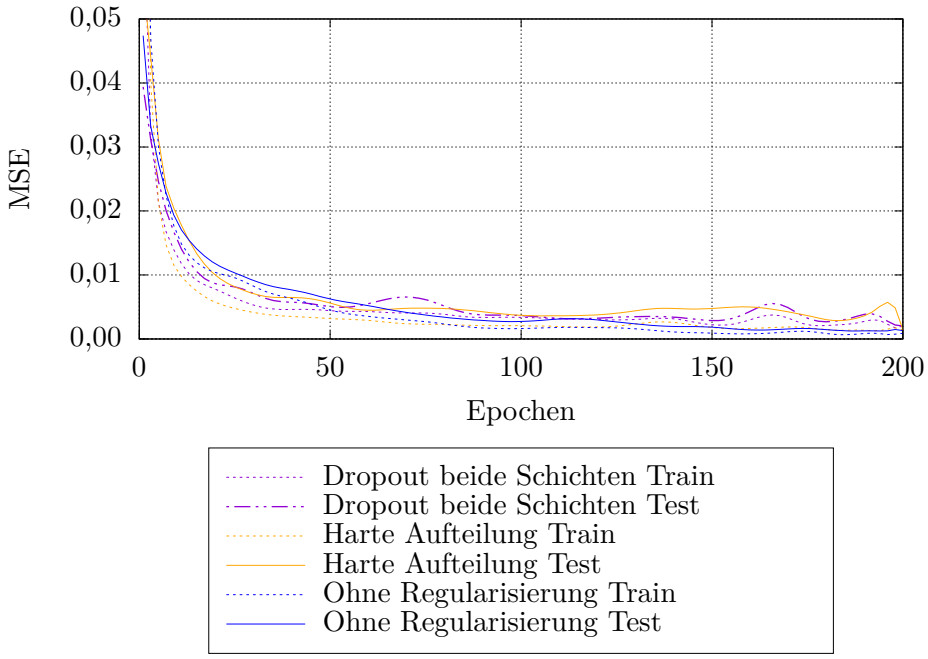

Abbildung 36: Einfluss von Dropout und harter Aufteilung auf Trainings- und Testfehler

In Abbildung 37 sind die ausgegebenen Daten zu der Soll-Geraden aufgetragen. Dabei ist zu erkennen, dass die Ausgabewerte der regulierten Netze für k_T schlechter, aber für P/D besser ist. Dies ist auch bei den Einzelfehlern für alle vorhergesagten Größen in Tabelle 21 zu erkennen. Da der Gesamtfehler jedoch bei dem nicht regulierten Netz niedriger ist, wird dieses weiterhin als optimal betrachtet. Dasselbe Verhalten ist bereits bei den Wageningen-Daten aufgetreten, wo trotz der geringeren Datenmenge Regularisierung keine Verbesserung gebracht hat.

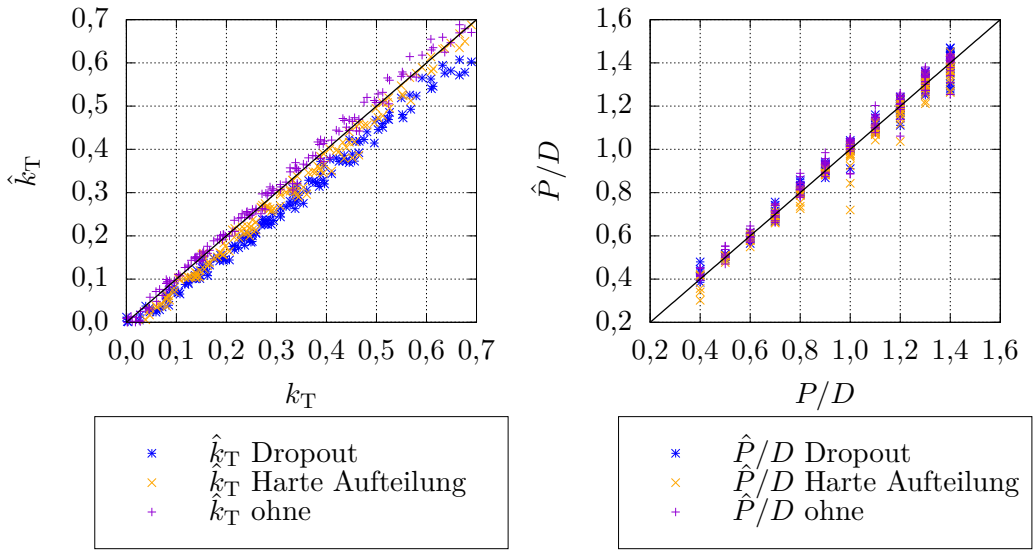

Abbildung 37: Vergleich der Ergebnisse der regulierten Netze von k_T bei 247 Daten

Tabelle 21: Fehler für die einzelnen Ausgabeparameter der Netze mit allen Trainingsdaten

Optimierung	P/D	J	η	k_T
ohne Regularisierung	0,0018	0,0007	0,0003	0,0003
Dropout	0,0012	0,0019	0,0007	0,0024
Harte Aufteilung	0,0019	0,0017	0,0008	0,0008

Skalieren

Für diesen Datensatz ist ebenfalls eine Rechnung mit Skalieren der Daten wie bei den Wageningen-Daten ausprobiert worden. Mit einem MSE von 0,0086 bringt Skalieren jedoch auch bei diesem Datensatz keine Verbesserung. Dies konnte ebenfalls bei den Wageningen-Daten beobachtet werden.

Netz für die Auslegung über das Drehmoment

Die Auswahl des Propellers soll wie in Abschnitt 2.7 beschrieben sowohl über den Schubbeiwert, als auch über den Momentenbeiwert möglich sein. Da die Daten für beide Netze identisch sind, soll an dieser Stelle nur eine kurze Übersicht über die Optimierungsläufe

gegeben werden. Insgesamt hat sich in den vorigen Kapiteln gezeigt, dass der MSE ein ausreichendes Maß an Genauigkeit aufweist. In Tabelle 22 sind diese Optimierungen gezeigt. Der geringste MSE liegt auch hier bei einer Optimierung ohne Regularisierung mit dem Löser Adam aber mit der Aktivierungsfunktion ReLU. Im Anhang ist ein Verlauf der Trainings- und Testfehler zu finden.

Tabelle 22: Optimierte Netzparameter und Fehler für verschiedene Aktivierungsfunktionen, Löser und Schichtenanzahlen mit den ProMarin-Daten für die Propellerauslegung über den Momentenbeiwert

Verdeckte Schichten	Knotenanzahl	Aktivierungsfunktion	Löser	Anfangslernrate	Losgröße	MSE
2	8, 8	Softplus	Adam	0,003626	1	0,0058
2	8, 16	ReLU	Adam	0,002477	1	0,0035
2	16, 16	LeakyReLU	Adam	0,008528	2	0,0048
2	16, 8	Tanh	Adam	0,001764	2	0,0096
2	16, 16	ReLU und Dropout	Adam	0,006300	1	0,0065

Netz für die Geometrie

Zuvor ist die Optimierung der Netze für den Betriebspunkt erläutert worden. Mit denselben Schritten soll nun ein Netz für die Geometriedaten erstellt werden. Dazu werden die in Abschnitt 4.2 erläuterten Daten mit 461 Datenpunkten verwendet. In Tabelle 23 sind die Ergebnisse für die Optimierung der Aktivierungsfunktionen zu sehen. Zu erkennen ist, dass es zwischen den Aktivierungsfunktionen mit dem Löser Adam kaum einen Unterschied im Ergebnis gibt. Dennoch ist die Aktivierungsfunktion LeakyReLU minimal besser als die anderen. Deshalb wird noch der Löser SGD mit LeakyReLU getestet, um auszuschließen, dass bei diesem Datensatz SGD geringere Fehler liefert als Adam. Dies ist jedoch nicht der Fall. Daher wird mit LeakyReLU und Adam weiter gerechnet.

Tabelle 23: Optimierte Netzparameter und Fehler für verschiedene Aktivierungsfunktionen, Löser und Schichtenanzahlen mit den ProMarin-Daten für die Bestimmung der Geometriedaten

Verdeckte Schichten	Knotenanzahl	Aktivierungsfunktion	Löser	Anfangslernrate	Losgröße	MSE
2	16, 32	Softplus	Adam	0,008888	1	0,0005
2	32, 16	ReLU	Adam	0,003583	4	0,0003
2	32, 16	LeakyReLU	Adam	0,001633	2	0,0003
2	32, 8	Tanh	Adam	0,006554	2	0,0006
2	32,32	LeakyReLU	SGD	0,008706	1	0,0012

Regularisierung

Im nächsten Schritt sollen auch hier die Regularisierungsmethoden ausprobiert werden.

Die Ergebnisse sind in Tabelle 24 gezeigt. Jedoch ist auch hier keine Verbesserung des MSE zu erkennen.

Tabelle 24: Optimierte Netzparameter und Fehler für verschiedene Regularisierungsmethoden mit den ProMarin-Daten für die Bestimmung der Geometriedaten

Regularisierung	Faktor	Knotenanzahl	Lernrate	Losgröße	MSE
Dropout beide Schichten	p=0,05	32, 16	0,003583	4	0,0004
L1-Regularisierung	λ=0,01	32, 8	0,002993	1	0,0015
L2-Regularisierung	λ=0,001	8, 16	0,004930	4	0,0008
SWA	L=0,01	8, 16	0,003287	2	0,0005
k-fache Kreuzvalidierung	k=5	16, 16	0,000174	1	0,0004

Skalieren

Mit einem MSE von 0,0265 bringt Skalieren auch bei diesem Datensatz keine Verbesserung. Daher wird für die Geometriebestimmung das neuronale Netz mit LeakyReLU und Adam verwendet. Im Anhang D ist eine Abbildung des Verlaufs aus Trainings- und Testfehler für das optimierte Netz zu finden.

Netz für die Freifahrtdiagramme

Mit denselben Schritten wie zuvor soll nun noch ein Netz für die Erstellung von Freifahrtdiagrammen trainiert werden. Dazu werden auch die in Abschnitt 4.2 erläuterten Daten mit 461 Datenpunkten verwendet. Die Eingabe besteht aus dem Fortschrittsgrad, der Steigung und den Geometrieparametern Ae/Ao, cps, F02, F06, F10 und der Flügelzahl. Ausgegeben werden der Wirkungsgrad sowie der Schub- und Momentenbeiwert, damit kann dann ein Freifahrtdiagramm erstellt werden.

Tabelle 25: Optimierte Netzparameter und Fehler für verschiedene Aktivierungsfunktionen, Löser und Schichtenanzahlen mit den ProMarin-Daten für die Propellerauslegung über den Momentenbeiwert

Verdeckte Schichten	Knotenanzahl	Aktivierungsfunktion	Löser	Anfangslernrate	Losgröße	MSE
2	8, 16	Softplus	Adam	0,007843	1	0,0009
2	32, 32	ReLU	Adam	0,002985	1	0,0007
2	16, 8	LeakyReLU	Adam	0,002831	4	0,0008
2	32, 8	Tanh	Adam	0,006978	2	0,0010
2	8, 32	ReLU und Dropout	Adam	0,004935	2	0,0011

In Tabelle 25 sind die Ergebnisse für die Optimierung der Aktivierungsfunktionen zu sehen. Zu erkennen ist, dass es zwischen den Aktivierungsfunktionen kaum einen Unterschied im Ergebnis gibt. Dennoch hat die Aktivierungsfunktion ReLU einen minimal geringeren MSE als die anderen. Auch die Regularisierung mit Dropout bringt

keine Verbesserung des MSE. Daher wird der Löser Adam mit der Aktivierungsfunktion ReLU für dieses Netz verwendet.

5.4 Gegenüberstellung der jeweiligen Modelle

In diesem Abschnitt sollen die Modelle miteinander verglichen werden. Dazu soll die Komplexität der Erstellung sowie ihre Fehlerrate verglichen werden, um ein Ergebnis nach Aufwand und Nutzen zu erhalten. Ein Vergleich der Auslegung der neuronalen Netze mit einer CFD-Rechnung wird im nächsten Kapitel erfolgen.

Grundsätzlich sind die Regressionen sehr einfach anzuwenden. Die Programmierung ist einfach und auch die Berechnung von Fehlern und anderen Bewertungsmaßen wird zum Großteil standardmäßig dafür geliefert. Außerdem kann man sich die Polynome und ihre Berechnung über die entsprechenden Funktionen leicht vorstellen und daher viele Verhaltensweisen nachvollziehen. Jedoch ist es vergleichsweise aufwendig, eine Regression für mehrere Parameter zu erstellen, da jede Regression einen Ausgabeparameter hat. Um nichtlineares Verhalten abbilden zu können muss eine Umwandlung der Werte in Polynome erfolgen. Die Fehler sind je nach Abhängigkeit der Daten untereinander hoch. Durch die einzelnen Regressionen für jeden Ausgabeparameter können aber auch Parameter, die im neuronalen Netz aufgrund von Ausreißern bei einzelnen Werten im gesamten Netz für Fehler sorgen, mit der Regression berechnet werden, da die einzelnen Ausgabeparameter nichts miteinander zu tun haben.

Neuronale Netze hingegen lassen eine beliebige Anzahl an Ein- und Ausgabewerten zu. Dafür sind sie deutlich schwieriger nachzuvollziehen und die Optimierung der Modelle benötigt vergleichsweise viel Zeit. Wenn die Modelle hinreichend optimiert und trainiert sind, liefern sie jedoch gute Werte. Die Beziehungen der Variablen untereinander fließen bereits mit in das Training ein. Dadurch muss die Qualität der Daten jedoch besser sein als für Regressionen. Neuronale Netze bieten direkt ein nichtlineares Lernverhalten über eine nichtlineare Aktivierungsfunktion an.

Insgesamt lässt sich festhalten, dass neuronale Netze für den Fall der Propellerserienauslegung die besseren Regressionsergebnisse erzielen können und in der späteren Anwendung im Auslegungsprogramm genauso einfach zu handhaben sind, wie Regressionen und ihre Koeffizienten. Im Training sind sie jedoch deutlich zeitaufwändiger als Regressionen.

6 Bewertung der Modelle

In diesem Kapitel werden die Ergebnisse der Regressionen und der optimierten neuronalen Netze einem Vergleich mit einer Auslegung anhand eines beispielhaften Schiffes von ProMarin unterzogen. So soll die Anwendungsmöglichkeit validiert werden.

6.1 Beispielhafter Betriebspunkt

In den Modellen kann die Auslegung sowohl über das Drehmoment als auch den Schub erfolgen. Deshalb werden hier zwei verschiedene Auslegungen durchgeführt. Zusätzlich sollen die beiden Auslegungen zu zwei unterschiedlichen Schiffen durchgeführt werden. In Tabelle 26 sind die für das Rechenmodell nötigen Eingabeparameter der beiden Schiffe genannt. Es handelt sich bei beiden Betriebspunkten um eine Flächenverhältnis-Flügelzahl-Kombination, welche so in den Daten nicht abgebildet ist. Daher können die Modelle anhand ihrer Ausgabe bewertet werden, wie gut sie unbekannte Daten annähern können.

Tabelle 26: Schiffsdaten für die Bestimmung des Betriebspunktes im Auslegungsvergleich für "Fast Vessel" und "Coaster"

Parameter	"Fast Vessel"	"Coaster"
Flügelzahl	5	4
Durchmesser	1,7 m	3 m
Schiffsgeschwindigkeit	14,4 $\frac{\text{m}}{\text{s}}$	6,69 $\frac{\text{m}}{\text{s}}$
Nachstromziffer	0,089	0,25
Mindest-Flächenverhältnis	1	0,7
Schiffswiderstand	98,02 kN	93,47 kN
Sogziffer	0	0
Schaftleistung	1920 kW	720 kW
Gütegrad der Anordnung	1	1

Für die neuronalen Netze gibt es ein Auslegungsprogramm, die Regressionen hingegen sind direkt in dem zuvor gezeigten Quellcode verwendet worden, da dort das Training kaum Zeit in Anspruch nimmt. In dem Ordner des Auslegungsprogramms sind in einem Unterordner die nötigen neuronalen Netze hinterlegt und in einem weiteren Unterordner werden das Freifahrtdiagramm und die Daten zum Betriebspunkt und der Geometrie mit aktuellem Zeitstempel gespeichert. Die Eingabe der Daten erfolgt über die in Abbildung 38 gezeigte Eingabemaske, in der sich zwischen der Eingabe zum Moment und Schub wechseln lässt.

In dem Programm wird dann mit der ausgewählten Eingabeart zuerst der Betriebspunkt bestimmt. Dabei wird eine Maximierung des Wirkungsgrades vorgenommen, indem bei bekanntem Moment k_T und bei bekanntem Schub k_Q variiert wird. Um die Werte in einem physikalisch sinnvollen Rahmen zu begrenzen, werden nur Steigungen zwischen 0,3 und 1,5 zugelassen. In den Datensätzen sind Steigungen von 0,4 bis 1,4 trainiert worden. Damit der ausgegebene Betriebspunkt auch die Schub- bzw. Momentenanforderungen erfüllt, wird auch das aus dem Betriebspunkt errechnete k_T^* oder k_Q^* auf eine Abweichung von maximal 15 % begrenzt. 15 % sind nötig, damit für jede Kom-

bination der Eingabeparameter ein Betriebspunkt gefunden wird. Sobald der Betriebspunkt mit dem maximalen Wirkungsgrad ermittelt ist, liegen alle nötigen Daten vor, um mit dem weiteren gespeicherten Modell die Geometrie zu bestimmen. Anschließend kann mit den Geometriedaten das Freifahrtdiagramm über variierende Fortschrittsgrade erstellt werden. Der Code für das Auslegungsprogramm ist im Anhang E zu finden. Für die Regressionen ist die Ausgabe mit den Begrenzungen genauso aufgebaut, dabei werden jedoch alle gesuchten Werte nacheinander ermittelt und es gibt keine bedienbare Eingabemaske.

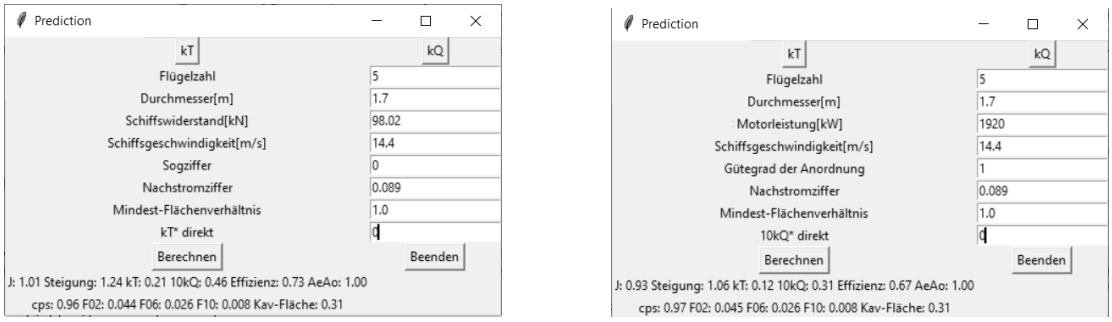

Abbildung 38: Ansicht der Eingabemaske des Auslegungsprogramms mit neuronalen Netzen

6.2 Vergleich mit Cfd-Optimierung

In diesem Abschnitt werden die Ergebnisse aus den neuronalen Netzen und der Regression mit den Soll-Werten aus der CFD-Rechnung verglichen. Diese sind mit den Ergebnissen der verschiedenen Rechnungen in Tabelle 27 und 28 gezeigt. Dabei bedeutet Netz in der Überschrift die Ermittlung der Ausgabeparameter über das neuronale Netz und Regression über die entsprechende Regression. Die Geometriedaten sind auch aus dem vorgegebenen Betriebspunkt für beide Schiffe ermittelt worden, die Ergebnisse sind in den Spalten mit dem Zusatz Geometrie zu finden.

In Tabelle 27 ist zu erkennen, dass für das schnelle Schiff "Fast Vessel" die Ausgabe der Regressionen bis auf die Kavitationsfläche nah am Sollwert liegen, aber dennoch die höheren Fehler im Vergleich zu den Netzen aufweisen. Die Auslegung mit geringstem Fehler liefert das neuronale Netz über das Moment mit einem MSE von 0,004. Das neuronale Netz über den Schub ist etwas schlechter bei einem MSE von 0,014. Die kavitierende Fläche wird von beiden Netzen mit einer Abweichung von 10 % ermittelt.

Zusätzlich zu den vollständigen Auslegungen von Betriebspunkt und Geometrie sind mit der Regression und dem Netz aus dem vorgegebenen Betriebspunkt die Geometrien und die Kavitationsfläche bestimmt worden. Dabei liegt insbesondere der MSE für das neuronale Netz bei nahezu null, sodass davon auszugehen ist, dass mit einer Verbesserung der Bestimmung des Betriebspunktes durch beispielsweise noch weitere Trainingsdaten auch eine Verbesserung in der Geometriebestimmung möglich ist. Die Prognose der Kavitationsfläche hat dann einen Fehler von unter 4 %.

Tabelle 27: Vergleich der Auslegungen für die Betriebspunkte des "Fast Vessel"

Ausgabewert	Netz k_Q^*	Netz k_T^*	Netz Geometrie	Regression k_Q^*	Regression k_T^*	Regression Geometrie	Soll
J	0,93	1,01	–	0,72	0,70	–	0,77
P/D	1,06	1,24	–	0,75	0,75	–	0,95
k_T	0,12	0,21	–	0,09	0,11	–	0,11
k_Q	0,03	0,05	–	0,02	0,02	–	0,02
$AeAo$	1,00	1,00	–	1,00	1,00	–	1,00
η	0,67	0,73	–	0,52	0,56	–	0,67
cps	0,97	0,96	0,98	0,98	0,98	0,98	1,00
F02	0,045	0,044	0,044	0,053	0,057	0,004	0,05
F06	0,026	0,026	0,025	0,032	0,034	0,002	0,03
F10	0,008	0,008	0,008	0,011	0,011	0,001	0,01
Kav	0,31	0,31	0,29	0,75	0,75	0,09	0,28
MSE gesamt	0,004	0,014	0,0001	0,026	0,025	0,008	–

Die Ergebnisse für das langsamere Schiff "Coaster" sind in Tabelle 28 gezeigt. Dabei ist die Auslegung über das neuronale Netz über den Schub die mit dem geringsten Fehler von 0,008. Für das Netz über das Moment liegt der MSE bei 0,02. Die Regressionen versagen bei dieser Auslegung völlig. Wird die Geometrie mit dem korrekten Betriebspunkt bestimmt, liegen die Fehler sowohl für das Netz als auch die Regression bei 0,0001. Für die Betrachtung der Ergebnisse ist wichtig, dass dieser Propeller im Soll nicht kavitiert, die Netze und Regressionen allerdings ausschließlich mit kavitierenden Propellerdaten angelernt worden sind. Daher kann die Bestimmung der Kavitationsfläche für diesen Fall nicht korrekt ausgeführt werden und sie wird im MSE nicht berücksichtigt.

Tabelle 28: Vergleich der Auslegungen für die Betriebspunkte des "Coaster"

Ausgabewert	Netz k_Q^*	Netz k_T^*	Netz Geometrie	Regression k_Q^*	Regression k_T^*	Regression Geometrie	Soll
J	0,57	0,85	–	-4E+09	-5E+09	–	0,89
P/D	0,69	1,23	–	2E+28	2E+28	–	0,97
k_T	0,08	0,29	–	0,38	0,01	–	0,18
k_Q	0,02	0,05	–	0,17	0,01	–	0,04
$AeAo$	0,70	0,70	–	0,70	0,70	–	0,70
η	0,56	0,65	–	4E+28	4E+28	–	0,65
cps	0,98	1,00	0,98	1E+27	1E+27	0,98	1,00
F02	0,046	0,037	0,04	-5E+84	-5E+84	0,058	0,05
F06	0,028	0,026	0,027	-3E+84	-3E+84	0,034	0,03
F10	0,008	0,010	0,008	-10E+83	-10E+83	0,011	0,01
Kav	0,25	0,37	0,30	-8E+85	-8E+85	0,33	0,001
MSE gesamt	0,02	0,008	0,0001	4E+168	4E+168	0,0001	–

Zu den Auslegungen werden für die neuronalen Netze auch Freifahrtdiagramme ausgegeben. Diese sind für die beiden Schiffe in Abbildung 39 und 40 gezeigt. Dabei ist zu erkennen, dass die Auslegung über das Moment für das "Fast Vessel" gut den Wirkungsgrad trifft mit einem Fehler von 0,003 und leicht über den Kurven der Beiwerte liegt. Für die Auslegung über den Schub liegen die Kurven der Beiwerte deutlich über den realen Kurven, die Fehler liegen hier bei 0,03 für k_T und 0,17 für k_Q, aber der Wirkungsgrad wird noch gut ausgegeben mit einem Fehler von 0,007. Bei dem "Coaster" ist der Verlauf der Auslegung über den Schub ähnlich. Allerdings sind die Kurven der Beiwerte der Auslegung über das Moment deutlich unter den realen Kurven mit Fehlern von 0,02 für k_T und 0,06 für k_Q. Auch die Kurve des Wirkungsgrades erreicht null bereits bei sehr niedrigen Werten des Fortschrittsgrades. Der Fehler liegt für den Wirkungsgrad bei 0,2. Alle Fehler sind in Tabelle 29 und 30 gezeigt.

Tabelle 29: Vergleich der Fehler der Freifahrtdiagramme des "Fast Vessel"

Netz	MSE η	MSE k_T	MSE k_Q
Über k_Q^*	0,003	0,004	0,02
Über k_T^*	0,007	0,027	0,17
Aus korrekten Eingabedaten	0,001	0,0003	0,0005

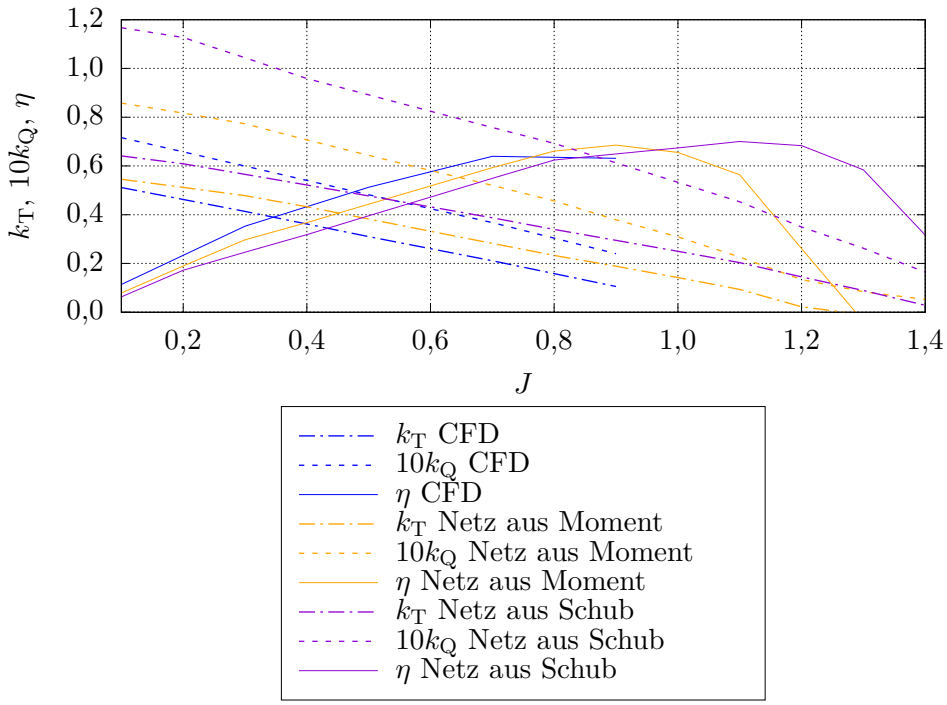

Abbildung 39: Freifahrtdiagramme aus der Auslegung der neuronalen Netze für das "Fast Vessel"

Tabelle 30: Vergleich der Fehler der Freifahrtdiagramme des ”Coaster”

Netz	MSE η	MSE k_T	MSE k_Q
Über k_Q^*	0,21	0,02	0,06
Über k_T^*	0,008	0,02	0,01
Aus korrekten Eingabedaten	0,004	0,0003	0,0008

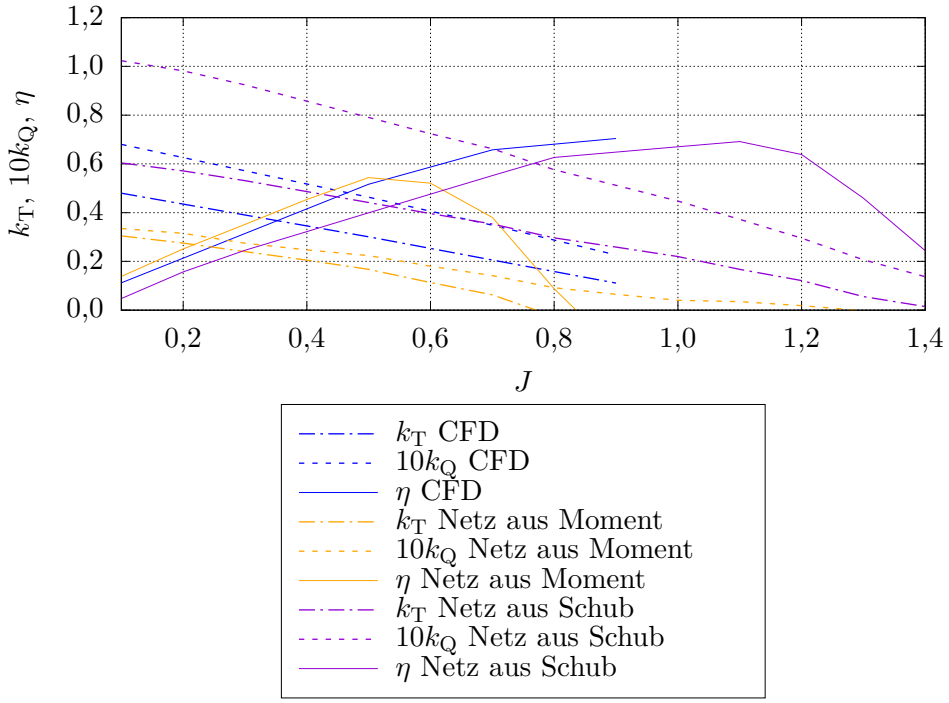

Abbildung 40: Freifahrtdiagramme aus der Auslegung der neuronalen Netze für den ”Coaster”

Damit beurteilt werden kann, ob mit den Netzen grundsätzlich Freifahrtdiagramme zu unbekannten Eingangsparametern erstellt werden können, sind die beiden Freifahrtdiagramme für ”Fast Vessel” und ”Coaster” nochmals mit dem korrekten Betriebspunkt und korrekter Geometrie bestimmt worden. Diese sind in Abbildung 41 gezeigt. Es ist zu sehen, dass für beide Schiffe sehr gute Freifahrtdiagramme erzeugt werden können, obwohl diese Flächenverhältnis-Flügelzahl-Kombination so nicht in den Daten abgebildet ist. Die Abweichungen in den vorherigen Freifahrtdiagrammen liegen also an der fehlerhaften Bestimmung des Betriebspunktes, da sowohl die Bestimmung der Geometrie, als auch die des Freifahrtdiagrammes bei korrektem Betriebspunkt deutlich geringere Fehler aufweist. Die Fehler für die direkte Bestimmung der Freifahrtdiagramme sind in Tabelle 29 und 30 ebenfalls aufgeführt. Sie liegen dabei besonders für k_T und k_Q von der Größenordnung her etwa um den Faktor zehn bis 100 niedriger, für den Wirkungsgrad ist die Größenordnung vergleichbar.

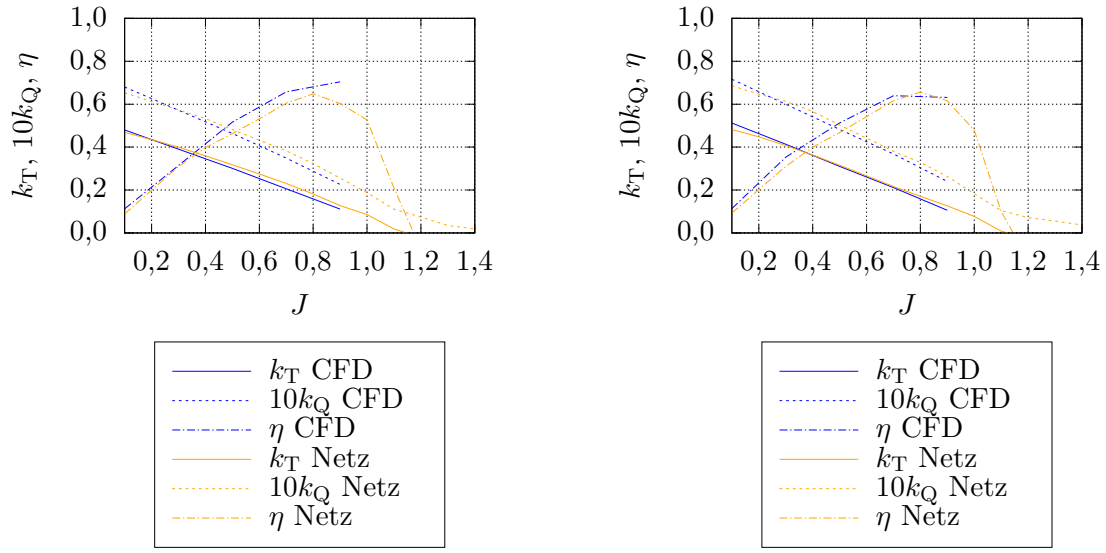

Abbildung 41: Vergleich der Soll-Freifahrtdiagramme und aus den Netzen mit korrekter Geometrie ermittelten Freifahrtdiagramme

Da nun entscheidend ist, ob der Betriebspunkt über eine Erhöhung der Datenmenge noch verbessert werden kann, werden die Betriebspunkte mit den Netzen aus den Wageningen-Daten bestimmt. Dazu wird als Sollwert die Ausgabe eines Skripts benutzt, welches innerhalb der Polynome den optimalen Propeller interpoliert. Die Ausgabe aus den Netzen erfolgt in einem weiteren Programm ähnlich dem Auslegungsprogramm mit den ProMarin-Daten, aber ohne grafische Oberfläche. In Tabelle 31 und 32 sind die Sollwerte aus der Interpolation den Werten aus den Modellen gegenüber gestellt. Dabei sind jeweils die mit 247 und 4496 Daten trainierten Modelle einander gegenüber gestellt. Zu erkennen ist bei beiden Schiffen, dass die Fehler für die größeren Netze mindestens um den Faktor zehn niedriger liegen. Mit einer höheren Datenmenge und ebenfalls über den Parameterraum gleichmäßig gestreuten Trainingsdaten lässt sich also die Ermittlung des Betriebspunktes verbessern.

Tabelle 31: Vergleich der Auslegungen für die Betriebspunkte des "Fast Vessel" mit den Wageningen-Daten

Ausgabewert	k_Q^* 4496	k_T^* 4496	k_Q^* 247	k_T^* 247	Soll
J	0,97	0,91	0,52	1,11	0,94
P/D	1,28	1,15	0,54	1,48	1,20
k_T	0,20	0,16	0,01	0,23	0,17
k_Q	0,04	0,03	0,006	0,05	0,03
$AeAo$	1,00	1,00	1,10	1,00	1,00
η	0,77	0,77	0,38	0,71	0,73
MSE gesamt	0,003	0,001	0,128	0,019	–

Tabelle 32: Vergleich der Auslegungen für die Betriebspunkte des "Coaster" mit den Wageningen-Daten

Ausgabewert	k_Q^* 4496	k_T^* 4496	k_Q^* 247	k_T^* 247	Soll
J	0,69	0,59	0,75	0,59	0,63
P/D	1,03	0,86	1,16	0,83	0,90
k_T	0,19	0,14	0,25	0,14	0,16
k_Q	0,03	0,02	0,05	0,02	0,02
$AeAo$	0,80	0,80	1,10	1,00	0,70
η	0,68	0,67	0,65	0,66	0,67
MSE gesamt	0,005	0,002	0,042	0,016	–

6.3 Bewertung der Vergleichsauslegung

Die Auslegung für einen vollständig unbekannten Betriebspunkt, welcher aber mit den Daten grundsätzlich beschrieben werden kann, hat gezeigt, dass eine Regression nicht geeignet ist, um diese Auslegung durchzuführen. Obwohl sie für die Trainings- und Testdaten gute Ergebnisse geliefert hat, sind bei vier Regressionen für eine beispielhafte Auslegung nur zwei mit plausiblen Werten gewesen. Diese wiesen jedoch auch einen höheren Fehler aus als die Ergebnisse der Netze, was vermutlich daran liegt, dass die Fehler sich für die einzelnen Regressionen potenzieren.

Bei den Netzen funktioniert die Auslegung über den Schub trotz der geringen Daten robust, weicht jedoch bei beiden Auslegungen um etwa denselben Fehler vom Freifahrtdiagramm ab. Die Auslegung über das Moment scheint anfälliger für abweichende Eingangsdaten. Zu einem der Schiffe findet das Netz von allen Modellen den besten Betriebspunkt, bei dem anderen Schiff hat es einen höheren MSE als das Netz über den Schub. Eine Berechnung der unbekannten Freifahrtdiagramme bei korrektem Betriebspunkt funktioniert sehr gut über Netze, daher sollte die Bestimmung des Betriebspunktes noch optimiert werden.

Insgesamt sollte für zukünftige Gestaltung von KI-gestützen Rechenmodellen im Propellerentwurf eher an den neuronalen Netzen weitergearbeitet werden, da diese noch mehr Möglichkeiten zur Optimierung bieten. Zudem liefern sie im Vergleich zu Regressionen bereits jetzt ein gutes Ergebnis.

7 Diskussion und Fazit

Im letzten Kapitel der Arbeit werden die Ergebnisse zusammengetragen. Außerdem wird die Arbeit bewertet und ein Ausblick auf weitere Forschungsmöglichkeiten gegeben.

7.1 Zusammenfassung

Zu Beginn dieser Arbeit wurde mit einer Literaturrecherche der Stand der Forschung in Bezug auf KI-gestützten Propellerentwurf ermittelt. Dazu wurden einige aktuelle Veröffentlichungen beschrieben und auf deren Einfluss auf den KI-gestützten Propellerentwurf untersucht. Es gibt bereits Veröffentlichungen, welche sich mit der Bestimmung von dem durch Propeller abgestrahlten Lärm beschäftigen. Zum Propellerentwurf wurde eine Veröffentlichung gefunden, die über eine Random Forest Regression Geometrieparameter des Propellers bestimmt. Dort ist die Herkunft und damit Qualität der Trainingsdaten jedoch unklar. Eine weitere Veröffentlichung beschäftigt sich mit einer Kombination aus evolutionären Algorithmen mit maschinellem Lernen, um den Aufwand der menschlichen Bewertung zu reduzieren. Insgesamt sind bereits gute Hinweise bezüglich der Optimierung von neuronalen Netzen enthalten, wie zum Beispiel, dass kleine Änderungen im Datensatz bereits eine neue Optimierung erfordern oder Regularisierung zur Vermeidung von Überanpassung bei komplexen Netzen mit kleinen Datensätzen nötig ist.

Danach wurden die in dieser Arbeit verwendeten Trainingsdaten erläutert. Es wurden dabei zum einen über Polynome der Wageningen-B-Serie erzeugte Daten verwendet, wobei die Größe der Datensätze variiert wurde. Die Parameter sind der Fortschrittsgrad, die Steigung, die Schub- und Momentenbeiwerte, das Flächenverhältnis, die Flügelzahl sowie der Wirkungsgrad. Dabei entstammen die Polynome Freifahrtdiagrammen, welche in Modellversuchen ermittelt wurden. Zum anderen wurden Seriendaten der Firma ProMarin für den F115-Propeller verwendet. Diese wurden über CFD-Rechnungen erstellt und bestehen aus denselben Parametern wie die der Wageningen-Daten, sodass der Betriebspunkt des Propellers bestimmt ist. Zusätzlich enthalten sie die Geometrie des Propellers, dabei handelt es sich um die Spitzensteigung und die Wölbung des Profils an drei verschiedenen Radien. Es lässt sich aus den CFD-Daten auch die prozentuale Kavitationsfläche ermitteln, wobei dort vereinfachend alle Punkte, die den Dampfdruck unterschreiten, als kavitierend angenommen werden.

Mit diesen beiden unterschiedlichen Datensätzen wurden dann Modelle für den KI-gestützten Entwurf entwickelt. Dazu sind polynomiale Regressionen und neuronale Netze zum Einsatz gekommen. Als Eingangsparameter wurden die Flügelzahl, das Flächenverhältnis und entweder die Kombination aus k_T^* und k_Q oder aus k_Q^* und k_T verwendet, je nachdem ob der Entwurf über den Schub oder das Moment des Propellers durchgeführt werden soll. Die Ausgabeparameter sind im ersten Schritt dementsprechend k_T oder k_Q, der Fortschrittsgrad, der Wirkungsgrad und die Steigung. Damit ist der Betriebspunkt des Propellers beschrieben. Im zweiten Schritt wurde dann mit den Netzen aus den ProMarin-Daten die Geometrie der Propeller sowie ihre prozentuale Kavitationsfläche ausgegeben.

Die Regressionen lassen sich direkt mit `scikit-learn` in Python erstellen und einfach trainieren. Jedoch ist eine Regression pro Ausgabeparameter nötig, da zwar die Menge der Eingangsparameter erhöht werden kann, nicht jedoch die der Ausgabeparameter. Für die neuronalen Netze ist zuerst eine Struktur in `PyTorch` aufgebaut und dann mit

`RayTune` optimiert worden. Bei neuronalen Netzen ist insbesondere die Optimierung der Hyperparameter wichtig. Dabei handelt es sich um die Anzahl der verdeckten Schichten, ihre Knotenanzahl, die Aktivierungsfunktionen und Löser, sowie Anfangslernraten. Auch sind verschiedene Regularisierungsmethoden und Datenaufteilungen für Test- und Trainingsdaten miteinander verglichen worden. Dabei wurde mit den Wageningen-Daten auch ermittelt, wie viele Trainingsdaten für eine gute Ermittlung des Betriebspunktes aus den neuronalen Netzen nötig sind.

Mit den so trainierten Netzen und Regressionen ist dann ein Vergleich zu einer CFD-Auslegung gemacht worden. Die Auslegung liegt außerhalb des trainierten Flächenverhältnis- und Flügelzahl-Bereiches. Dabei ist in einem eingeschränkten Parameterbereich der Wirkungsgrad optimiert und der Betriebspunkt mit dem besten Wirkungsgrad als optimal ausgewählt worden. Es hat sich herausgestellt, dass die Regressionen Ergebnisse mit hohen Fehlern liefern und zum Teil gar keine physikalisch sinnvollen Betriebspunkte auswählen. Für die Netze ist die Ermittlung des Betriebspunktes gut. Es zeigt sich jedoch im Vergleich der Datenmenge bei den Ausgaben der Wageningen-Daten, dass die Netze der ProMarin-Daten mit mehr Datenpunkten noch verbessert werden können. Die richtige Ermittlung des Betriebspunktes ist wichtig, um für die Geometrie und die Freifahrtdiagramme sehr gute Werte mit Fehlern im Bereich 10^{-3} oder kleiner zu erhalten. Auch die Kavitation wird bei korrekter Geometrie bei ähnlichen Betriebspunkten zu den Trainingsdaten mit einem Fehler von weniger als 4 % ausgegeben.

7.2 Schlussfolgerung

Der KI-gestützte Propellerentwurf hat sich in dieser Arbeit als anwendbar erwiesen. Insbesondere neuronale Netze zeigen hier gute Ergebnisse, sofern ausreichend Trainingsdaten zur Verfügung stehen. Sie sind jedoch aufwendig zu optimieren und trainieren, dafür können sie aber beliebig viele Eingangs- und Ausgabeparameter verwenden und sind in der späteren Anwendung schnell. Die Regressionen zeigen sich als nicht ausreichend geeignet, da sie in unbekannten Parameterbereichen physikalisch nicht mehr sinnvolle Ergebnisse ermitteln, obwohl sie zuvor geringe Trainingsfehler aufweisen. Das Training der Regressionen ist deutlich einfacher als das der neuronalen Netze, aber es muss für jeden Ausgabeparameter eine eigene Regression erstellt werden.

7.3 Ausblick

Insgesamt haben neuronale Netze für diese Anwendung ein hohes Potenzial gezeigt und können noch verbessert werden. Zum einen kann in einem nächsten Schritt mit erhöhter Datenmenge noch die Ermittlung des Betriebspunktes mit den ProMarin-Daten verbessert werden. Weitergehend ist auch denkbar, zusätzlich zu den Seriendaten noch direkt die Druck- und Geschwindigkeitsdaten aus den CFD-Rechnungen zu verwenden, um anstelle von Kavitationsflächen die Druckverteilungen auf den Flügeln ausgeben zu können. Gerade eine gute Vorhersage von Kavitation ist mit maschinellem Lernen bisher schwierig. Es können auch hybride Modelle eingesetzt werden, sodass für bekannte Zusammenhänge Formeln als physikalische Modelle in die neuronalen Netze eingegeben werden und dann weitere unbekannte Beziehungen aus den Daten erlernt werden. Die Kombination von evolutionären Algorithmen und neuronalen Netzen ist ebenfalls denkbar. Das Feld des KI-gestützten Propellerentwurfes steht gerade in den Anfängen und bietet damit noch einiges an Forschungsmöglichkeiten.

Literaturverzeichnis

[1] Youjiang Wang; Keqi Wang; Moustafa Abdel-Maksoud. noiseNet: A neural network to predict marine propellers' underwater radiated noise. *Ocean Engineering*, 236:109542, 2021.

[2] Ioli Gypa; Marcus Jansson; Krister Wolff; Rickard Bensow. Propeller optimization by interactive genetic algorithms and machine learning. *Ship Technology Research*, pages 1–16, 2021.

[3] Christopher M. Bishop. *Pattern recognition and machine learning*. Computer science. Springer, New York, NY, 2006.

[4] Sandi Baressi Šegota; D. Štifanić; K. Ohkura; Z. Car. Use of Artificial Neural Network for Estimation of Propeller Torque Values in a CODLAG Propulsion System. In Institute of Electrical and Electronics Engineers, editor, *International Joint Conference on Neural Networks (IJCNN)*, 2020.

[5] Sandi Baressi Šegota; Nikola Anđelić; Jan Kudláček; Robert Čep. Artificial neural network for predicting values of residuary resistance per unit weight of displacement. *Journal of Maritime & Transportation Science*, 57(1):9–22, 2019.

[6] Uwe Lämmel; Jürgen Cleve. *Künstliche Intelligenz: Wissensverarbeitung - neuronale Netze*. Hanser, München, 5., überarbeitete auflage edition, 2020.

[7] Luca Oneto; Francesca Cipollini; Leonardo Miglianti; Giorgio Tani; Stefano Gaggero; Andrea Coraddu. Deep Learning for Cavitating Marine Propeller Noise Prediction at Design Stage. In Institute of Electrical and Electronics Engineers, editor, *International Joint Conference on Neural Networks (IJCNN)*, 2020.

[8] Ian Goodfellow; Yoshua Bengio; Aaron Courville. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts and London, England, 2016.

[9] Fabian Pedregosa; Gaël Varoquaux; Alexandre Gramfort; Vincent Michel; Bertrand Thirion; Olivier Grisel; Mathieu Blondel; Peter Prettenhofer; Ron Weiss; Vincent Dubourg; Jake Vanderplas; Alexandre Passos; David Cournapeau; Matthieu Brucher; Matthieu Perrot; Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011.

[10] Dudenredaktion. "Intelligenz" auf Duden online. `https://www.duden.de/rechtschreibung/Intelligenz`, o.J. zuletzt geprüft am 29.07.2022.

[11] Bhavana Ram Phanindra; et al. Machine Learning Based Classification of Ducted and Non-Ducted Propeller Type Quadcopter. In Institute of Electrical and Electronics Engineers, editor, *International Conference on Advanced Computing & Communication Systems (ICACCS)*, 2020.

[12] John Henry Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. Complex adaptive systems. MIT Press, Cambridge, Mass, 1st mit press ed. edition, 1992.

[13] M. M. Barnitsas; D. Ray; P. Kinley. Kt, Kq and Efficiency Curves for the Wageningen B-Series Propellers. https://deepblue.lib.umich.edu/handle/2027.42/91702.

[14] Werner Kinnebrock. *Neuronale Netze: Grundlagen, Anwendungen, Beispiele*. Oldenbourg, München and Wien, 2., verb. aufl. edition, 1994.

[15] Steven W. Knox. *Machine learning: A concise introduction*. Wiley series in probability and statistics. Wiley, Hoboken, NJ, 2018.

[16] Mahiout Thomas; Fillatre Lionel; Deruaz-Pepin Laurent. Propeller Noise Detection with Deep Learning. In Institute of Electrical and Electronics Engineers, editor, *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 306–310. IEEE, 04.05.2020 - 08.05.2020.

[17] Wei-Meng Lee. *Python machine learning*. Wiley, Indianapolis, IN, 2019.

[18] Sambhunath Biswas; Brian C. Lovell. *Bézier and Splines in Image Processing and Machine Vision*. Springer London, London, 2008.

[19] Umberto Michelucci. *Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks*. Springer eBook Collection. Apress, Berkeley, CA, 2018.

[20] Nikhil Ketkar; Jojo Moolayil. *Deep learning with Python: Learn best practices of deep learning models with PyTorch*. Apress, New York, second edition edition, 2021.

[21] Stefan Krüger. Schiffspropeller. https://www.tuhh.de/t3resources/ssi/layout01INSTITUTE/Lehre/Schiffspropeller/Vorlesungsunterlagen/Propellertheorie.pdf. zuletzt geprüft am 31.05.2022.

[22] Stefan Krüger. Grundlagen der Propulsion, 2017.

[23] Richard Liaw; Eric Liang; Robert Nishihara; Philipp Moritz; Joseph E. Gonzalez; Ion Stoica. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*, 2018.

[24] Harsh Vardhan; Peter Volgyesi; Janos Sztipanovits. Machine learning assisted propeller design. In Martina Maggio; James Weimer; Mohammad Al Faruque; Meeko Oishi, editor, *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, pages 227–228, New York, NY, USA, 05192021. ACM.

[25] the pandas developtment team. https://pandas.pydata.org/pandas-docs/stable/reference/index.html#api, o.J. zuletzt geprüft am 30.07.2022.

[26] Peter Soukup; Reinhard Postlep; Thomas Rung. Propellerfreifahrt- und Kavitationsversuch: Schiffbaulabor SoSe2007, 2009.

[27] Torch Contributors. https://pytorch.org/docs/stable/generated/torch.nn.Softplus.html#torch.nn.Softplus, 2019. zuletzt geprüft am 29.05.2022.

[28] Torch Contributors. https://pytorch.org/docs/stable/index.html, o.J. zuletzt geprüft am 29.07.2022.

[29] Pavel Izmailov; Dmitrii Podoprikhin; Timur Garipov; Dmitry Vetrov; Andrew Gordon Wilson. Averaging Weights Leads to Wider Optima and Better Generalization. https://arxiv.org/pdf/1803.05407, 14.03.2018. Appears at the Conference on Uncertainty in Artificial Intelligence (UAI), 2018.

Appendix

A. Korrelationstabelle für die Daten von ProMarin mit Geometrie

Tabelle 33: Vollständige Korrelationstabelle für die Daten zur Geometriebestimmung der ProMarin-Daten

	Z	$AeAo$	PzD	cps	F02	F06	F10	J	$10k_Q$	k_T	Kav	η	$10k_Q^*$	k_T^*
Z	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$AeAo$	0	1	0	0	0	0	0	0	0	0	0	0	0	0
PzD	0	0	1	0	0	0	0	0,4425	0,6830	0,4757	0,6630	0,3218	0	0
cps	0	0	0	1	0	0	0	0	0	0	0	0	0	0
F02	0	0	0	0	1	1	1	0	0	0	0,3133	0	0	0
F06	0	0	0	0	1	1	1	0	0	0	0,3133	0	0	0
F10	0	0	0	0	1	1	1	0	0	0	0,3133	0	0	0
J	0	0	0,4425	0	0	0	0	1	-0,3172	-0,5701	0,4310	0,7266	-0,5216	-0,6041
$10k_Q$	0	0	0,6830	0	0	0	0	-0,3172	1	0,9433	0,4059	0	0,5116	0,4752
k_T	0	0	0,4757	0	0	0	0	-0,5701	0,9433	1	0	-0,3922	0,5888	0,5997
Kav	0	0	0,6630	0	0,3133	0,3133	0,3133	0,4310	0,4059	0	1	0	0	0
η	0	0	0,3218	0	0	0	0	0,7266	0	-0,3922	0	1	-0,6142	-0,6848
$10k_Q^*$	0	0	0	0	0	0	0	-0,5216	0,5116	0,5888	0	-0,6142	1	0,9791
k_T^*	0	0	0	0	0	0	0	-0,6041	0,4752	0,5997	0	-0,6848	0,9791	1

B. Code des neuronalen Netzes

```
import numpy as np
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import random_split
from torch.optim.swa_utils import SWALR
import ray
from ray import tune
from ray.tune import CLIReporter
from ray.tune.schedulers import ASHAScheduler
from ray.tune.schedulers import AsyncHyperBandScheduler
from torch.utils.data import DataLoader
import pandas as pd
import random

random.seed(42)
np.random.seed(42)
torch.manual_seed(0)

class MyDataset(torch.utils.data.Dataset):
  def __init__(self, X, y):
    if not torch.is_tensor(X) and not torch.is_tensor(y):
      self.X = torch.from_numpy(X)
      self.y = torch.from_numpy(y)
    else :
      self.X = X
      self.y = y

  def __len__(self):
      return len(self.X)

  def __getitem__(self, i):
      return self.X[i], self.y[i]

class MLP(nn.Module):
  '''
    Multilayer Perceptron for regression.
  '''
  def __init__(self, l1=16, l2=16, pp=0):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(6, l1),
      nn.Dropout(p=pp),
      nn.LeakyReLU(negative_slope=0.01),
      nn.Linear(l1, l2),
      nn.Dropout(p=pp),
      nn.LeakyReLU(negative_slope=0.01),
      nn.Linear(l2, 4)
    )

  def forward(self, x):
    '''
      Forward pass
    '''
    return self.layers(x)

def data_prep(data, batchsize):
```

```
    y=np.array(data["PzD"])
    y=y.reshape(-1, 1)
    y=np.column_stack((y, data["J"]))
    y=np.column_stack((y, data["eta"]))
    y=np.column_stack((y, data["kT"]))

    X=np.array([data["kT*"]])
    X=X.reshape(-1, 1)
    X=np.column_stack((X, data["kQ"]))
    X=np.column_stack((X, data["AeAo"]))
    X=np.column_stack((X, data["Z1"]))
    X=np.column_stack((X, data["Z2"]))
    X=np.column_stack((X, data["Z3"]))

    dataset = MyDataset(X,y)
    data_res = torch.utils.data.DataLoader(dataset, batch_size= batchsize, shuffle=True)

    return data_res

def train(mlp, optimizer, train_data, test_data, EPOCH_SIZE):
  loss_function = nn.MSELoss(reduction='mean')
  test_loss = np.zeros((EPOCH_SIZE,1))
  total_loss = np.zeros((EPOCH_SIZE,1))
  epochloss = np.zeros((int(len(train_data)/1),EPOCH_SIZE))
  # Run the training loop
  for epoch in range(0, EPOCH_SIZE):

    # Print epoch
    print(f'Starting epoch {epoch+1}')

    # Iterate over the Traindata for training data
    for i, data in enumerate(train_data, 0):
      # Get and prepare inputs
      inputs, targets = data
      inputs, targets = inputs.float(), targets.float()
      targets = targets.reshape((targets.shape[0], 4))

      # Zero gradients
      optimizer.zero_grad()

      # Perform forward pass
      outputs = mlp(inputs)

      # Compute loss
      loss = loss_function(outputs, targets)

      # Perform backward pass
      loss.backward()

      # Perform optimization
      optimizer.step()

      # Print statistics
      epochloss[i, epoch] = loss.item()#.item()
    test_loss[epoch] = test(mlp, test_data)
  total_loss = np.sum(epochloss,axis=0) / len(train_data)
  # Process is complete.
  print('Training process has finished.')
  return total_loss, test_loss, epochloss
```

```
def test(model, data_loader):
  model.eval()
  loss_total = 0
  loss_function = nn.MSELoss(reduction='mean')
  with torch.no_grad():
      for i, data in enumerate(data_loader, 0):
          inputs, targets = data
          inputs, targets = inputs.float(), targets.float()
          targets = targets.reshape((targets.shape[0], 4))
          outputs = model(inputs)
          loss = loss_function(outputs, targets)
          loss_total += loss.item()
  return loss_total / len(data_loader)

def dataimp(filename, scale_data, batch_size):
    real_path = os.path.realpath(__file__)
    dir_path = os.path.dirname(real_path)
    path_dir = os.sep.join([dir_path, filename])
    data = pd.read_csv(path_dir,sep="\t",decimal=",")
    data['J'] = data['J'].astype(float, errors = 'raise')
    data['kQ'] = data['kQ'].astype(float, errors = 'raise')
    data['kT'] = data['kT'].astype(float, errors = 'raise')
    data['eta'] = data['eta'].astype(float, errors = 'raise')
    data['AeAo'] = data['AeAo'].astype(float, errors = 'raise')
    data['PzD'] = data['PzD'].astype(float, errors = 'raise')
    data['J_squared'] = data['J'] * data['J']
    data['J_three'] = data['J'] * data['J'] * data['J']
    data['kT*'] = data['kT'] / data['J_squared']
    data['kQ*'] = data['kQ'] / data['J_three']
    data = data[data['kQ'] > 0]
    data = data[data['eta'] > 0]
    data.replace([np.inf, -np.inf], np.nan, inplace=True)
    tensor_z=data["Z"] -4
    tensor_z=torch.tensor(tensor_z.tolist())
    one_hot = torch.nn.functional.one_hot(tensor_z, num_classes=- 1)
    target_name = ['Z1','Z2','Z3']
    one_hot= pd.DataFrame(one_hot.numpy(), columns = target_name)
    data = pd.concat([data, one_hot], axis = 1, names = target_name)
    data = data.dropna()

    if scale_data==True:
      scalemaxdat = data.max()
      scalemindat = data.min()
      data = (data-data.min())/ (data.max() - data.min())
    else:
      scalemaxdat = 0
      scalemindat = 0

    test_data = data.sample(frac = 0.3)
    train_data = data.drop(labels=test_data.index)

    testloader = data_prep(test_data, batch_size)
    trainloader = data_prep(data, batch_size)

    return trainloader, testloader

def train_mydata(config):
  train_data, test_data = dataimp("Wageningen_Daten.txt", config["scale"], batch_size=config["batchs"])
```

```
  model = MLP(config["l1"],config["l2"],config["pdrop"])
  optimizer = torch.optim.Adam(model.parameters(), lr=config["lr"])
  print("Start Training")
  total, test, trainloss = train(model, optimizer, train_data, test_data, config["epochs"])
  print('Training process has finished.')
  return total, test, trainloss, model

def save_mydata(path_dir, filename, datatype, datatosave):
  # Specify a path
  PATH = os.sep.join([path_dir, filename])
  # Save
  if datatype == "model":
    torch.save(datatosave, PATH)
    print("Model saved.")
  elif datatype == "txt":
    path_file = os.sep.join([path_dir, filename])
    np.savetxt(path_file, datatosave, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='',
comments='# ', encoding=None)

if __name__ == '__main__':
  config={
      "lr": (0.0036069811947669916),
      "batchs": (1),
      "l1": (16),
      "l2": (16),
      "pdrop": (0.0),
      "epochs": (200),
      "scale": False,
    }

  total_loss, test_loss, trainloss, mlp = train_mydata(config)

  real_path = os.path.realpath(__file__)
  path_dir = os.path.dirname(real_path)
  save_mydata(path_dir, "NeuralNet_OptX_LR_Ad_16_16_0036_kT.pt", "model", mlp)
  save_mydata(path_dir, "Loss-NN_OptX_LR_Ad_16_16_0036_Train_Total.txt", "txt", total_loss)
  save_mydata(path_dir, "Loss-NN_OptX_LR_Ad_16_16_0036_Total.txt", "txt", test_loss)
```

C. Code des Regressionsprogramms

```
import os #Dateien einlesen
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from mlxtend.evaluate import bias_variance_decomp

# Funktion zum Einlesen des Files
def dataimp(filename):
    real_path = os.path.realpath(__file__)
    dir_path = os.path.dirname(real_path)
    path_dir = os.sep.join([dir_path, filename])
    data = pd.read_csv(path_dir,sep="\t",decimal=",")
    data['J'] = data['J'].astype(float, errors = 'raise')
    data['kQ'] = data['kQ'].astype(float, errors = 'raise')
    data['kT'] = data['kT'].astype(float, errors = 'raise')
    data['eta'] = data['eta'].astype(float, errors = 'raise')
    data['AeAo'] = data['AeAo'].astype(float, errors = 'raise')
    data['PzD'] = data['PzD'].astype(float, errors = 'raise')
    data['J_squared'] = data['J'] * data['J']
    data['J_three'] = data['J'] * data['J'] * data['J']
    data['kT*'] = data['kT'] / data['J_squared']
    data['kQ*'] = data['kQ'] / data['J_three']
    data.replace([np.inf, -np.inf], np.nan, inplace=True)
    data = data.dropna()
    return data

#Funktion für die lineare Regression aus a b c d zu
def regression(a, b, c, d, e, degree):
    # Die Daten werden mit Index und 4 Spalten in x abgelegt
    x = pd.DataFrame(np.c_[data[a], data[b], data[c], data[d]], columns = [a, b, c, d])
    # y hat nur eine Spalte
    y = data[e]
    # Reines Splitten der Daten in Test und Training DFs
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=5)
    polynomial_features= PolynomialFeatures(degree = degree)
    # multipliziert die Werte nacheinander durch mit sich selbst und allen anderen Werten, führt so zu
mehr Spalten; in der ersten Spalte steht immer 1
    # Bsp: Macht aus einem pandas-DF 313*4 mit 1252 Elementen ein Numpy Array mit 313*5 und 1565
Elementen bei deg = 1
    x_train_poly = polynomial_features.fit_transform(x_train)
    global model

    model = LinearRegression()
    #Führt die regression aus
    model.fit(x_train_poly, y_train)
    x_test_poly = polynomial_features.fit_transform(x_test)
    #Ausgabe Train- und Test-MSE
    pred_test = model.predict(x_test_poly)
    MSE_test = ((y_test-pred_test)**2).sum() / len(y_test)
    pred_train = model.predict(x_train_poly)
    MSE_train = ((y_train-pred_train)**2).sum() / len(y_train)
    print("Testfehler", MSE_test, "Trainfehler", MSE_train)

    #macht aus dem PD-dataframe einen formlosen Vektor
```

```
    y_train = y_train.values
    y_test = y_test.values
    # Hier müssen die poly-Werte eingegeben werden, da die Methode sich nochmals die Vorhersagen  aus
demn model berechnet
    avg_expected_loss, avg_bias, avg_var = bias_variance_decomp(model, x_train_poly,
                                                                y_train, x_test_poly,
                                                                y_test,
                                                                loss='mse',
                                                                num_rounds=50,
                                                                random_seed=20)
    print('Loss:', avg_expected_loss, 'bias:', avg_bias, 'variance', avg_var, 'R^2',
model.score(x_test_poly,y_test))

    real_path = os.path.realpath(__file__)
    path_dir = os.path.dirname(real_path)
    path_file = os.sep.join([path_dir, "Regression_Loss_Bias_Var.txt"])
    file = open(path_file, "a")
    file.write("\n" + repr(degree)+ "\t"+ repr(avg_expected_loss) + "\t"+ repr(avg_var) + "\t"+
repr(avg_bias))
    file.close()

def prediction(one, two, three, four, degree):
    value=pd.DataFrame(np.c_[one, two, three, four], columns = ['a', 'b', 'c', 'd'])
    polynomial_features= PolynomialFeatures(degree = degree)
    value_poly = polynomial_features.fit_transform(value) #das Modell kann nur mit polynom-Werten
rechnen
    pred = model.predict(value_poly)
    return pred

data = dataimp("Wageningen_Daten.txt") # Importiert die gewünscht csv-Datei aus demselben Verzeichnis
in einen Pandas-Dataframe, Index u. 6 Spalten
regression("J","Z","PzD","AeAo", "kT", 4) #Startet eine lineare Regression über die ersten 4 Spalten
mit Ziel der 5. Spalte und dem gewünschten Grad
```

D. Verlauf der Training- und Testfehler des Netzes für die Geometrie

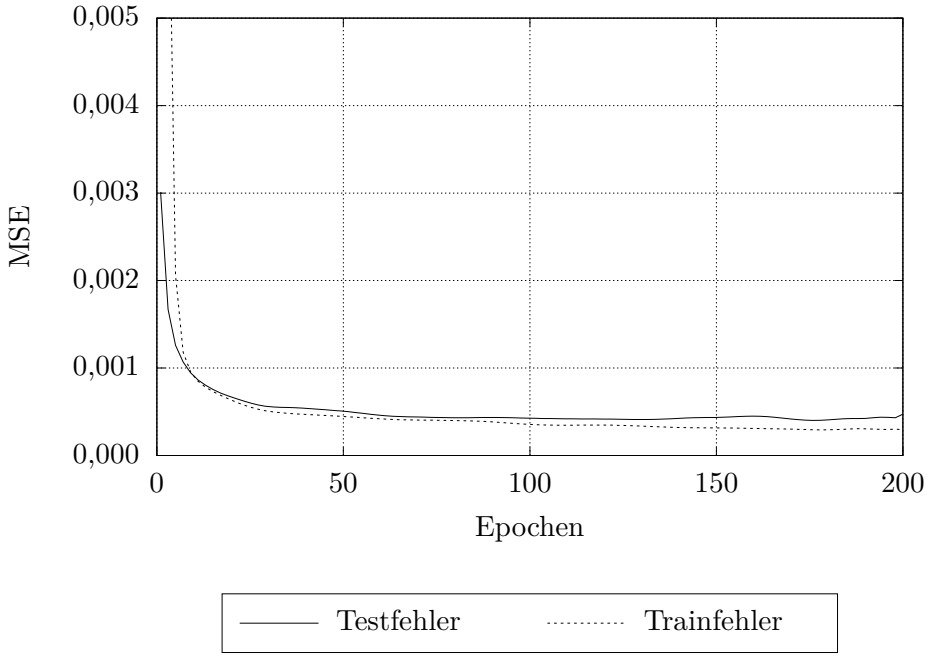

Abbildung 42: Verlauf von Trainings- und Testfehler für das neuronale Netz der Geometrie

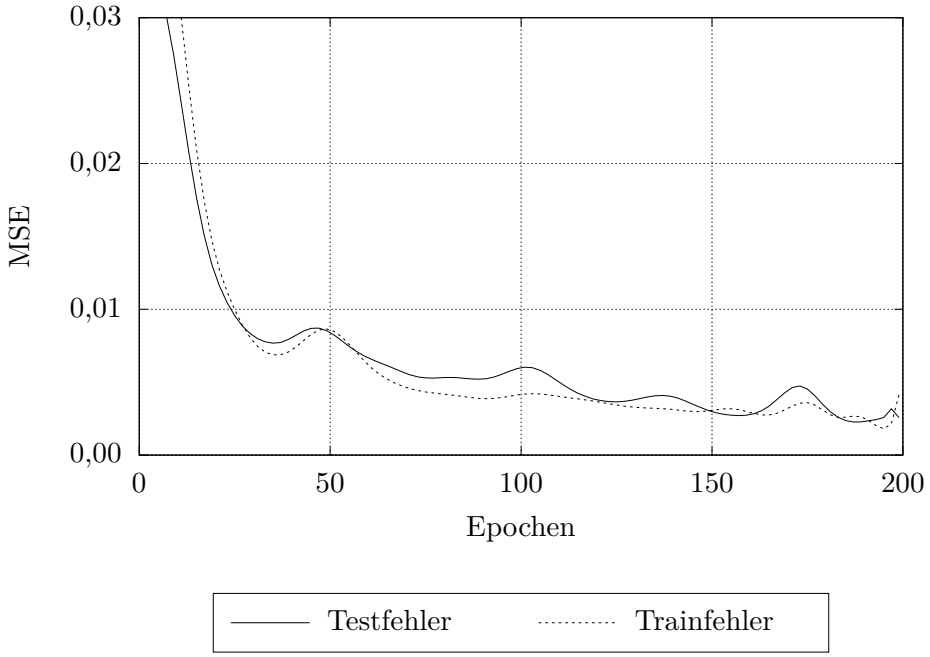

Abbildung 43: Verlauf von Trainings- und Testfehler für das neuronale Netz zur Ermittlung des Betriebspunktes über das Drehmoment

E. Code des Auslegungsprogramms

```
import tkinter as tk
from tkinter import *
from datetime import datetime
import torch
from torch import nn
from torch.utils.data import DataLoader
from sklearn.preprocessing import StandardScaler
import os #Dateien einlesen
import pandas as pd
import numpy as np
import pytorch_lightning as pl
fields_kT = 'Flügelzahl', 'Durchmesser[m]', 'Schiffswiderstand[kN]', 'Schiffsgeschwindigkeit[m/s]',
'              Sogziffer              ', 'Nachstromziffer', 'Mindest-Flächenverhältnis', '   kT*
direkt   '
fields_kQ = 'Flügelzahl', 'Durchmesser[m]', 'Motorleistung[kW]   ', 'Schiffsgeschwindigkeit[m/s]',
'Gütegrad der Anordnung', 'Nachstromziffer', 'Mindest-Flächenverhältnis', '10kQ* direkt'

def f_range(von=0, bis=None, schritt=1):
    if bis is None:
        von, bis = 0, von
    f = von
    if schritt > 0:
        while f < bis:
            yield(f)
            f += schritt
    else:
        while f > bis:
            yield(f)
            f += schritt

def makeform(root, var):
  global entries
  global helpvar
  if var == 0:
    fields = fields_kT
    helpvar = 0
  else:
    fields = fields_kQ
    helpvar = 1
  entries = []
  row = 2
  for field in fields:
      lab = tk.Label(root)
      lab.config(text=field)
      ent = tk.Entry(root)
      lab.grid(row=row,column=0)
      ent.grid(row=row,column=1)
      entries.append((field, ent))
      if row == 2:
        ent.insert(0,"4")
      if row in [5,4,3]:
        ent.insert(0,"1")
      if (row == 6 and var == 0):
        ent.insert(0,"0.1")
      if (row == 6 and var == 1):
        ent.insert(0,"1")
      if row == 7:
        ent.insert(0,"0.25")
      if row == 8:
```

```
        ent.insert(0,"0.7")
      if (row == 9):
        ent.insert(0,"0")
      row = row+1
  return entries

def calculation():
   rho = 1.025
   pi = 3.14
   real_path = os.path.realpath(__file__)
   dir_path = os.path.dirname(real_path)
   path_dir = os.sep.join([dir_path, "Modell"])
   global data
   data = []
   for entry in entries:
      field = entry[0]
      text  = entry[1].get()
      data.append(text)

   Z = int(data[0])
   D = float(data[1])
   v_s = float(data[3])
   w = float(data[5])
   AeAo = float(data[6])

   v_a = (1-w)*v_s

   if Z == 4:
       Z=[1,0,0]
   elif Z == 5:
        Z=[0,1,0]
   elif Z==6:
        Z=[0,0,1]
   else:
       Z=[0,0,0]

   ### Rechnung mit kT oder kT*
   if helpvar == 0:
    R_T = float(data[2])
    t = float(data[4])
    k_T = float(data[7])

    T = R_T / (1-t)
    if k_T == 0:
      k_T = T / (rho*D**2*v_a**2)
    ### Laden des Neuronalen Netzes
    PATH = os.sep.join([path_dir, "NeuralNet_Opt4_LR_Ad_32_16_0031_kT.pt"])
    model = torch.load(PATH)
    model.eval()
    ### Erzeugen der Vorhersage-Parameter
    kQ_all = np.linspace(0.05,1.6, num = 200)
    pred_all = np.zeros((len(kQ_all),4))
    max_pred = np.zeros((len(kQ_all),4))
    max_eta = 0
    ### Schleife für Mindest-Flächenverhältnis
    for Ae in f_range(AeAo, 1.15, 0.1):
      ### Schleife für Vorhersage von PzD, J, eta für kT/J^2, variiertes kQ, Flügelzahl und Auswerten
der Effizienz
      eta_help = 0
```

```
      for i in range(0,len(kQ_all)):
        kQ = kQ_all[i]
        test= [k_T,float(kQ), Ae, float(Z[0]), float(Z[1]), float(Z[2])]
        pred = model(torch.tensor(test))
        pred_all[i] = pred.detach().numpy()
        eta_ber = pred_all[i,1]*pred_all[i,3]/(2*pi*kQ/10)
        k_T_ber = pred_all[i,3]/(pred_all[i,1]*pred_all[i,1])
        #if (delta < eta_help):
        if (eta_ber > eta_help) and (pred_all[i,0]>0.3) and (pred_all[i,0]<1.5)and (0.85*k_T < k_T_ber)
and (k_T_ber < 1.15*k_T):
          eta_help = eta_ber
          i_eta = i
      if (eta_help > max_eta):
        max_eta = eta_help
        max_pred[i_eta] = pred_all[i_eta]
        kT = pred_all[i_eta,3]
        imax = i_eta
        Aemax = Ae
    kQ = kQ_all[imax]
    ### Ausgabe der Ergebnisse
    label = tk.Label(root)
    label.config(text="")
    label.config(text=["J:", "{:.2f}".format(pred_all[i_eta,1]) , "Steigung:",
"{:.2f}".format(pred_all[i_eta,0]) ,"kT:", "{:.2f}".format(kT), "10kQ:",
"{:.2f}".format(kQ_all[i_eta]), "Effizienz:", "{:.2f}".format(pred_all[i_eta,2])])
    label.grid(row=13,column=0)
    label.config(text=["J:", "{:.2f}".format(max_pred[imax,1]) , "Steigung:",
"{:.2f}".format(max_pred[imax,0]) , "kT:", "{:.2f}".format(max_pred[imax,3]), "10kQ:",
"{:.2f}".format(kQ_all[imax]), "Effizienz:", "{:.2f}".format(max_pred[imax,2]), "AeAo:",
"{:.2f}".format(Aemax)])

   ### Rechnung mit kQ oder kQ*
   elif helpvar == 1:
    P_D = float(data[2])
    eta_R = float(data[4])
    k_Q = float(data[7])

    if k_Q == 0:
      k_Q = P_D * eta_R / (2*pi*rho*D**2*v_a**3)*10
    ### Laden des Neuronalen Netzes
    PATH = os.sep.join([path_dir, "NeuralNet_Opt1_LR_Ad_8_16_0024_kQ.pt"])
    model = torch.load(PATH)
    model.eval()
    ### Erzeugen der Vorhersage-Parameter
    kT_all = np.linspace(0.75,0.05, num = 200)
    pred_all = np.zeros((len(kT_all),4))
    max_pred = np.zeros((len(kT_all),4))
    max_eta = 0
    ### Schleife für Mindest-Flächenverhältnis
    for Ae in f_range(AeAo, 1.15, 0.1):
      ### Schleife für Vorhersage von PzD, J, eta für kT/J^2, variiertes kQ, Flügelzahl und Auswerten
der Effizienz
      eta_help = 0
      for i in range(0,len(kT_all)):
        kT = kT_all[i]
        test= [k_Q,float(kT), Ae, float(Z[0]), float(Z[1]), float(Z[2])]
        pred = model(torch.tensor(test))
        pred_all[i] = pred.detach().numpy()
        eta_ber = pred_all[i,1]*kT/(2*pi*pred_all[i,3]/10)
```

```
        k_Q_ber = pred_all[i,3]/(pred_all[i,1]*pred_all[i,1]*pred_all[i,1])
        if (eta_ber > eta_help) and (pred_all[i,0]>0.3) and (pred_all[i,0]<1.5) and (0.85*k_Q <
k_Q_ber) and (k_Q_ber < 1.15*k_Q):
          eta_help = eta_ber
          i_eta = i
      if (eta_help > max_eta):
        max_eta = eta_help
        max_pred[i_eta] = pred_all[i_eta]
        kQ = pred_all[i_eta,3]
        imax = i_eta
        Aemax = Ae
    kT = kT_all[imax]

    ### Ausgabe der Ergebnisse
    label = tk.Label(root)
    label.config(text="")
    label.config(text=["J:", "{:.2f}".format(max_pred[imax,1]) , "Steigung:",
"{:.2f}".format(max_pred[imax,0]) , "kT:", "{:.2f}".format(kT_all[imax]), "10kQ:",
"{:.2f}".format(max_pred[imax,3]), "Effizienz:", "{:.2f}".format(max_pred[imax,2]), "AeAo:",
"{:.2f}".format(Aemax)])

   ### Ausgabe der Geometrie
   PATH = os.sep.join([path_dir, "NeuralNet_Opt1_LR_Ad_32_16_0016_Test_geom.pt"])
   model_geom = torch.load(PATH)
   model_geom.eval()
   geom = [float(kT), float(kQ), float(max_pred[imax,1]), float(max_pred[imax,2]),
float(max_pred[imax,0]), Aemax, float(Z[0]), float(Z[1]), float(Z[2])]
   pred_geom = model_geom(torch.tensor(geom))
   pred_geom = pred_geom.detach().numpy()

   label2 = tk.Label(root)
   label2.config(text="")
   label2.config(text=["cps:", "{:.2f}".format(pred_geom[0]) , "F02:", "{:.3f}".format(pred_geom[1]),
"F06:", "{:.3f}".format(pred_geom[2]), "F10:", "{:.3f}".format(pred_geom[3]), "Kav-Fläche:",
"{:.2f}".format(pred_geom[4])])
   label2.grid(row=17,column=0)

   ##Erzeugen eines Freifahrtdiagramms
   PATH = os.sep.join([path_dir, "NeuralNet_Opt1_LR_Ad_32_32_0029_Test_FF.pt"])
   model_FF = torch.load(PATH)
   model_FF.eval()
   kurve = np.zeros((14,4))
   i=0
   for j in np.arange(0.1,1.5,0.1):
     ff_data = [float(j), float(max_pred[imax,0]), Aemax, float(Z[0]), float(Z[1]),
float(Z[2]),pred_geom[1],pred_geom[2],pred_geom[3],pred_geom[0]]
     pred_FF = model_FF(torch.tensor(ff_data))
     pred_FF = pred_FF.detach().numpy()
     kurve[i,0] = j
     kurve[i,1] =pred_FF[0]
     kurve[i,2] =pred_FF[1]
     kurve[i,3] =pred_FF[2]
     i=i+1
   time_now = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")
   path_dir = os.sep.join([dir_path, "Daten"])
   path_file = os.sep.join([path_dir, time_now+"-Freifahrtdaten.txt"])
   np.savetxt(path_file, kurve, fmt='%.18e', delimiter=' ', newline='\n', header='J 10kQ kT eta',
footer='', comments='# ', encoding=None)
```

```
    if Z==[1,0,0]:
      Z=4
    elif Z == [0,1,0]:
      Z=5
    elif Z==[0,0,1]:
      Z=6
    else:
        Z=0
    path_file = os.sep.join([path_dir, time_now+"-Betriebspunkt.txt"])
    file = open(path_file, "a")
    file.write("kT" + "\t"+ "kQ" + "\t"+ "J" + "\t"+ "PzD" + "\t"+ "AeAo" + "\t"+ "Z")
    file.write("\n" + repr(float(kT))+ "\t"+ repr(float(kQ)) + "\t"+ repr(max_pred[imax,1]) + "\t"+
repr(max_pred[imax,0]) + "\t"+ repr(Aemax) + "\t"+ repr(Z))
    file.close()
    path_file = os.sep.join([path_dir, time_now+"-Geometrie.txt"])
    file = open(path_file, "a")
    file.write("cps" + "\t"+ "F02" + "\t"+ "F06" + "\t"+ "F10" + "\t"+ "Kav")
    file.write("\n" + repr(pred_geom[0])+ "\t"+ repr(pred_geom[1]) + "\t"+ repr(pred_geom[2]) + "\t"+
repr(pred_geom[3]) + "\t"+ repr(pred_geom[4]))
    file.close()


class MLP(nn.Module):
  def __init__(self, l1=32, l2=16, pp=0):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(6, l1),
      nn.Dropout(p=pp),
      nn.Tanh(),   ##nn.Sigmoid(), nn.ReLU(), nn.Tanh(), nn.Softplus(beta=1, threshold=20) ##
      nn.Linear(l1, l2),
      nn.Dropout(p=pp),
      nn.Tanh(),
      nn.Linear(l2, 4)
    )

  def forward(self, x):
    return self.layers(x)

class MLPgeom(nn.Module):
  def __init__(self, l1=32, l2=16, pp=0, spbeta=1):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(9, l1),
      nn.Dropout(p=pp),
      nn.LeakyReLU(),   ##nn.Sigmoid(), nn.ReLU(), nn.Tanh(), nn.Softplus(beta=1, threshold=20),
LeakyReLU(negative_slope=0.01) ##
      nn.Linear(l1, l2),
      nn.Dropout(p=pp),
      nn.LeakyReLU(),
      nn.Linear(l2, 5)
    )

  def forward(self, x):
    return self.layers(x)

class MLPkQ(nn.Module):
  def __init__(self, l1=8, l2=16, pp=0):
    super().__init__()
    self.layers = nn.Sequential(
```

```
      nn.Linear(6, l1),
      nn.Dropout(p=pp),
      nn.ReLU(),   ##nn.Sigmoid(), nn.ReLU(), nn.Tanh(), nn.Softplus(beta=1, threshold=20) ##
      nn.Linear(l1, l2),
      nn.Dropout(p=pp),
      nn.ReLU(),
      nn.Linear(l2, 4)
    )

  def forward(self, x):
    return self.layers(x)

class MLPFF(nn.Module):
  def __init__(self, l1=32, l2=32, pp=0, spbeta=1):
    super().__init__()
    self.layers = nn.Sequential(
      nn.Linear(10, l1),
      nn.Dropout(p=pp),
      nn.LeakyReLU(),   ##nn.Sigmoid(), nn.ReLU(), nn.Tanh(), nn.Softplus(beta=1, threshold=20),
LeakyReLU(negative_slope=0.01) ##
      nn.Linear(l1, l2),
      nn.Dropout(p=pp),
      nn.LeakyReLU(),
      nn.Linear(l2, 3)
    )

  def forward(self, x):
    return self.layers(x)

if __name__ == '__main__':
   root = tk.Tk()
   root.title("Prediction")
   c1 = tk.Button(root, text='kT', command = lambda:makeform(root, 0))
   c2 = tk.Button(root, text='kQ', command = lambda:makeform(root, 1))
   c1.grid(row=0)
   c2.grid(row=0, column = 1)
   b3 = tk.Button(root, text='Berechnen',
          command=calculation)
   b3.grid(row = 11, column=0)
   b2 = tk.Button(root, text='Beenden', command=root.quit)
   b2.grid(row=11,column=1)
   root.mainloop()
```