Fakultät für Ingenieurwissenschaften
Ruprecht-Karls-Universität Heidelberg

Projektarbeit

Evaluierung und Auswahl eines geeigneten KI-Frameworks zur Portierung neuronaler Netzwerke auf Mikrocontroller für die Modellierung von Permanentmagnet-Synchronmaschinen

Name:	Kevin Keppler
Matrikelnummer:	3727346
Betreuer:	Prof. Dr. Dirk Koch
Datum der Abgabe:	11. Dezember 2024

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADAM	Adaptive-Moment-Estimation.
ADC	Analog-Digital-Wandler.
BEMF	Back-Electromotive-Force.
DAC	Digital-Analaog-Converter.
DSP	Digital-Signal-Processing-Unit.
FLOPs	Floating-Point-Operations.
FOC	feldorientierte Regelung.
HIL	Hardware-in-the-Loop.
KI	Künstliche Intelligenz.
LSTM	Long-Short-Term-Memory.
MSE	Mean-Squared-Error.
PMSM	Permanentmagnet-Synchronmaschine.
PWM	Pulsweitenmodulation.
RNN	Recurrent-Neural-Network.

1 Einleitung

Um den Entwicklungsaufwand und die damit verbundenen Kosten zu reduzieren, werden Hardware-in-the-Loop (HIL)-Systeme in der Entwicklung eingesetzt. Sie ermöglichen, automatisierte Softwaretests zur Sicherstellung der Softwarequalität durchzuführen. Dabei werden die Signale eines eingebetteten Systems nicht aufwendig durch die reale Hardware bereitgestellt, sondern durch ein weiteres eingebettetes-System simuliert. Der Aufbau eines solchen Systems ist in Abbildung 1.1 dargestellt.

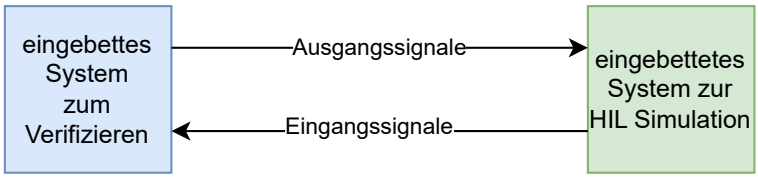

Abbildung 1.1: HIL-System

Im Folgenden wird eine Permanentmagnet-Synchronmaschine (PMSM) als Strecke betrachtet. Ausgangssignale sind beispielsweise durch Pulsweitenmodulation (PWM) generierte Spannungen, welche vom HIL-System über *Capture Timer* erfasst werden. Eingangssignale sind Spannungswerte, welche über einen Analog-Digital-Wandler (ADC) eingelesen werden. Diese werden vom HIL-System mittels eines Digital-Analaog-Converter (DAC) erzeugt. Neben diesen Signalen gibt es noch weitere, welche vom HIL-System simuliert werden. Diese Systeme bieten folgende Vorteile:

- **Kostenersparnis**: Keine teuren Hardware-Prüfstände der Strecke notwendig.
- **Gefahrenreduktio**n: Keine Unfallgefahr durch bewegliche Teile oder hohe Spannungen der Strecke.
- **Flexibilität**: Einfache Änderungen der Strecke möglich.

1.1 Motivation

Für die HIL-Systeme wird die reale Hardware durch Modelle bereitgestellt, deren Erstellung sehr aufwendig ist und ein tiefes Wissen voraussetzt. Eine besondere Herausforderung ist das Einbringen von Sättigungseffekten, Kopplungen und nichtlinearem dynamischem Verhalten [1]. Dabei wird das System mit verschiedenen Sollwertsprüngen stimuliert und Fehlersimulationen betrachtet. Beispiele sind ein Phasenausfall, Blockade oder Überhitzung. Um für ähnliche Strecken die Verwendung des gleichen Modells zu ermöglichen, sind diese parametrierbar. Die Parameter werden aus dem realen System ermittelt und sind durch Messungenauigkeiten eine weitere Fehlerquelle.

1.2 Ziele der Arbeit

Um den Aufwand der Modellierung zu reduzieren, wird in dieser Arbeit untersucht, ob ein neuronales Netzwerk eine alternative zu einem mathematischen Modell darstellt. Dabei wird neben der Genauigkeit ein besonderes Augenmerk auf den Ressourcenverbrauch des neuronalen Netzwerks gelegt. Somit kann zukünftig für das HIL-System ein Mikrocontroller mit geringen Ressourcen verwendet werden. Neben der Untersuchung des Netzwerks ist ein weiteres Ziel der Arbeit, ein geeignetes Framework zu ermitteln, welches die Portierung des gewählten Netzwerks auf einen Mikrocontroller ermöglicht.

1.3 Aufbau der Arbeit

In dieser Arbeit werden im Kapitel 2 die wichtigsten Grundlagen beschrieben. Dazu gehören die neuronalen Netzwerke sowie die Betrachtung von verwandten Arbeiten.

Im darauffolgenden Kapitel 3 wird ein geeignetes neuronales Netzwerk bestimmt. Dazu ist eine geeignete Datenaufbereitung sowie die Bestimmung der Hyperparameter notwendig.

Darauf folgt in Kapitel 4 die Auswahl eines geeigneten Künstliche Intelligenz (KI)-Frameworks für eingebettete-Systeme.

Zum Schluss gibt das Kapitel 5 noch eine Zusammenfassung.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel werden die für diese Arbeit wichtigsten Grundlagen über neuronale Netzwerke betrachtet. Darunter die für die sequentiellen Abläufe in dieser Arbeit gewählten Recurrent-Neural-Network (RNN)-Netzwerke, sowie die wichtigsten Hyperparameter. Des Weiteren werden verwandte Arbeiten untersucht.

2.1 Neuronales Netzwerke

Für neuronale Netze existieren viele verschiedene Architekturen, deren Design Vorteile für bestimmte Aufgaben bietet. Ein *feed forward*-Netzwerk, welches für Regressionsaufgaben eingesetzt werden kann, hat einen klar definierten Signalfluss der Daten. Diese werden von der Input-Schicht durch eine Hidden-Schicht hin zur Output-Schicht geleitet. Hierbei sind nur Verbindungen von einer zur nachfolgenden Schicht erlaubt [2]. *Feed forward* Netze bieten dank ihrer Struktur ein sehr allgemeines Framework zur Approximation jeglicher endlichen n-dimensionalen Funktion.

In *feed forward*-Netzen sind die Eingangsdaten nicht chronologisch geordnet, somit sind diese nicht geeignet um sequentielle Abläufe ressourcenschonenden abzubilden. Es sind viele Eingangsparameter und somit Gewichte notwendig, um einen sequentiellen Ablauf nachzubilden. Eine Alternative sind RNNs, bei welchem der Signalfluss teilweise wieder zurückgeführt wird [3]. Dadurch ist die Anzahl der Gewichte und damit verbundene Speicherbedarf unabhängig von der Länge einer Sequenz, da diese gemeinsam genutzt werden. Wie in Abbildung 2.1 zu erkennen ist, sind die Rückführungen ausschließlich in Form von Schleifen in der Hidden-Schicht vorzufinden.

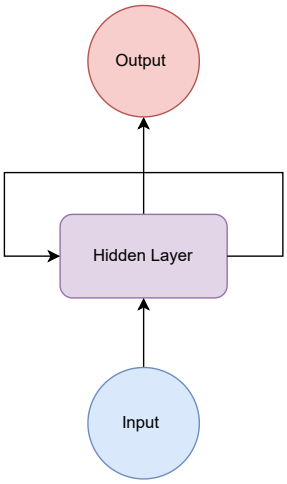

Abbildung 2.1: Aufbau RNN

Aufgrund dieser rekurrenten Verbindungen kann das neuronale Netzwerk bedingt Informationen abhängig von vorangegangenen Daten speichern [4][5]. Um RNNs zu trainieren, können diese abgerollt werden, was die Tiefe des Modells vergrößert. Ein tiefes Modell führt zu verschwindenden oder explodierten Gradienten, wodurch das Training von großen RNNs schwierig ist.

Bei der Nutzung eines RNNs gibt es verschiedene Varianten, welche in Abbildung 2.2 zu sehen sind und im Folgenden erläutert werden. [6]

- One to One: Verhält sich wie ein *feed forward* Netz.
- Many to One: Bei der Many to One Variante bestehen die Inputdaten aus einer Sequenz und generieren einen Output. Für diese Arbeit ist diese Variante hervorragend geeignet, da aus einer Inputfolge der benötigte Output bestimmt wird.
- Many to Many: Bei dieser Variante folgt auf eine Inputsequenz eine eigene Outputsequenz. Diese Variante ist gut für die Übersetzung von Sprachen geeignet.

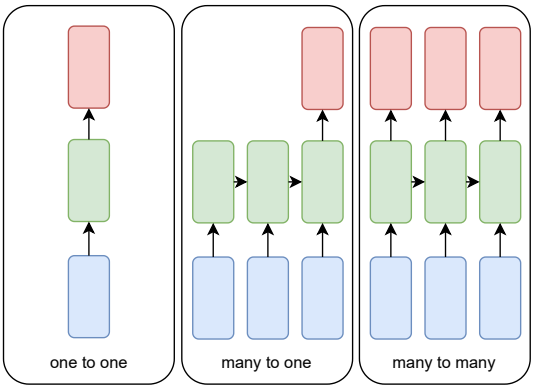

Abbildung 2.2: RNN Types (angelehnt an [6])

Durch Long-Short-Term-Memory (LSTM) Netzwerke wird das Problem mit den verschwindenden oder explodierten Gradienten gelöst. In Abbildung 2.3 ist der Aufbau einer LSTM-Zelle dargestellt. Neben den in den Hidden-States für kurze Zeit gespeicherten Informationen, besitzt die Architektur noch Cell-States in welchen Informationen länger gespeichert werden. Ob die Information ins Langzeitgedächtnis übertragen wird, entscheidet das *Forgot Gate*. [7, Kapitel 15]

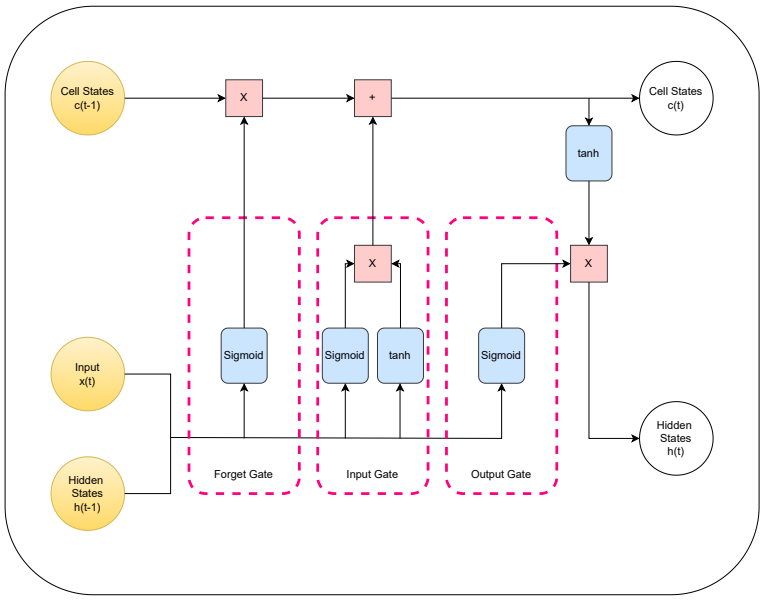

Abbildung 2.3: Keras LSTM Aufbau (angelehnt an [7, Kapitel 15])

2.2 Hyperparameter

Neben der automatischen Optimierung der Gewichtungen durch den Lernalgorithmus sind auch Konfigurationen vorzunehmen, welche ebenfalls einen entscheidenden Einfluss auf den Trainingserfolg haben. Es handelt sich hierbei um sogenannte Hyperparameter. Diese werden zu Beginn eines Trainingszyklus festgelegt und währenddessen nicht verändert. Neben dem Aufbau des Modells existieren eine ganze Bandbreite weiterer Hyperparameter, welche je nach Ausprägung unterschiedliche Auswirkungen auf das neuronale Netz besitzen. [4][8][9, Kapitel 8]

2.2.1 Hyperparameter Training

Im Folgende werden die wichtigsten Hyperparameter für das Training vorgestellt.

Lernrate

Bei der Lernrate handelt es sich um einen positiven Wert, welcher einen Wertebereich [0;1,0] annehmen kann. Während mit einer kleinen Lernrate eine Menge Zeit benötigt

wird, bis die optimalen Gewichtungen gefunden sind, führt eine große Lernrate dazu, dass eventuell keine optimalen Werte ermittelt werden können.

Batchsize

Durch die Batchsize wird die Anzahl der Trainingsbeispiele definiert, welche während einer Iteration des Trainings durchlaufen werden und ein Update der Gewichte und Biase zufolge haben. Die Batchsize hat einen Einfluss auf den Trainingserfolg.

Optimierer

Als Optimierer wird der Algorithmus bezeichnet, welcher verwendet wird, um die Gewichte und Biase während des Trainings zu optimieren. Dabei hat sich der Adaptive-Moment-Estimation (ADAM) Optimierer für einen Großteil der Problemstellungen bewährt. Dabei wird der exponentiell abfallende Durchschnitt der vergangenen Gradienten und deren Quadrate berechnet.

Verlustfunktion

Die Verlustfunktion, welche auch Kostenfunktion genannt wird, berechnet, wie gut oder schlecht ein Modell die Trainingsdaten vorhersagt. Hierzu wird die Differenz zwischen den vorhergesagten und den tatsächlichen Werten berechnet. Das Ziel des Trainingsprozesses ist es, die Verlustfunktion zu minimieren. Für Regressionsprobleme wird häufig die Verlustfunktion Mean-Squared-Error (MSE) verwendet. Sie berechnet den Durchschnitt der quadrierten Differenzen zwischen den tatsächlichen Werten y und den vorhergesagten Werten $\hat{y}$

Epochen

Eine Epoche bezeichnet einen vollständigen Durchlauf des gesamten Trainingsdatensatzes.

2.2.2 Hyperparameter LSTM-Netzwerk

Zur wichtigsten Konfiguration eines LSTM Netzwerks gehören die Anzahl der Units, die Timesteps, sowie die Inputfeatures, welche im Folgenden erklärt werden. [10]

- **Units**: Dimension der Hidden-States und Cell-States
- **Timesteps**: Länge einer Sequenz
- **Inputfeatures**: Dimenson des Inputvectors
- **Stateful**: Bestimmt, ob Cell- und Hidden-States über die Zellen hinweg weitergegeben werden

Die Hyperparameter, Units und Inputfeatures bestimmen die Anzahl der Neuronen und damit verbundenen Parameter. Dies spielt eine entscheidende Rolle bei der Effizienz des Trainings und der Problemlösung. Durch die Länge der Sequenz wird beim Training

die Tiefe des Netzwerks beeinflusst. Je tiefer ein Netzwerk ist, desto komplexere Probleme können gelöst werden. Jedoch führt eine zu hohe Anzahl an Parametern nicht nur zu einem verlangsamten Lernen, sondern birgt auch die Gefahr einer Überanpassung. Das heißt, dass das neuronale Netzwerk beim Training zu stark an die bereits bekannten Daten angepasst wird und bei unbekannten Daten eine schlechtere Leistung erzielt.

2.3 Verwandte Arbeiten

Es gibt nur wenige Bereiche, in welchen das Thema KI heutzutage keine Anwendung findet. Dementsprechend hoch ist der Umfang an Arbeiten, welche sich mit neuronalen Netzen und dynamischen Systemen befassen. Dabei steht meistens nicht das Nachbilden der Strecke im Fokus, sondern eine Identifikation, um diese zu regeln.

In der Arbeit [11] ist die Umsetzung der Identifikation und Regelung von nichtlinearen dynamischen Systemen durch neuronale Netzwerke beschrieben.

In der Arbeit [12] wird beschrieben, wie die Drehzahl einer PMSM durch neuronale Netzwerke identifiziert wird. Das Schätzen der Motortemperatur, welche in der Arbeit [13] beschrieben wird, ist ein weiterer Anwendungsfall.

Die Anomalieerkennung während des Betriebs von PMSMs ist ein weiterer Bereich. Eine Möglichkeit zur Detektierung eines Phasenausfalls wird beispielsweise in Arbeit [14] beschrieben. Um die Leistung des neuronalen Netzwerks zu erhöhen, werden nicht die RAW-Werte der Inputsignale verwendet, sondern diese durch eine FFT aufbereitet.

In der Arbeit [15] wird ebenfalls eine PMSM modelliert. Dort werden weitere Daten modelliert und ein deutlich komplexeres Netzwerk angewendet.

Aufgrund der Strecke und dem damit verbundenen sequentiellen Abläufe stechen in den vorangegangenen Arbeiten drei Implementierungen für neuronale Netzwerke heraus.

- Neuronale Netzwerke mit Time Delays
- RNNs
- Transformer

Durch den geringeren Ressourcenverbrauch von RNNs wird der Fokus in dieser Arbeit auf diese Netzwerke gelegt.

3 Neuronales Netzwerk

In diesem Kapitel wird ein geeignetes neuronales Netzwerk ermittelt. In Abschnitt 2.3 wird gezeigt, dass RNNs häufig Anwendung in der für diese Arbeit fokussierte Strecke, einer PMSM, finden. Somit werden diese Netzwerke für die Untersuchung betrachtet. Es wird zunächst die Aufbereitung der Daten, sowie der Aufbau des Netzwerks beschrieben. Anschließend werden verschiedene Netzwerke und die dazugehörigen Hyperparameter validiert.

3.1 Daten Generierung

Für die Datengenerierung und damit verbundene Modellierung wird ein Simulationstool verwendet. Das Modell besteht im Wesentlichen aus einem Inverter und Motor.

Der Inverter Block, welcher die Spannung und Ströme der Motorphasen stellt und erfasst, hat als Inputsignal die Tastverhältnis der drei Motorphasen U, V und W. Der Ausgang dieses Blocks ist die Spannungen der Motorphasen. Das Verhalten des Inverters ist von der Zwischenkreisspannung abhängig. Dies ist die Spannung, welche bei 100% Tastverhältnis an den Motorwicklungen anliegt. Um den Inverter mit einer realen Hardware zu verwenden, müssen noch weitere Eigenschaften berücksichtigt werden [16]:

- Maximalstrom
- Shunt Widerstand für Strommessung
- Drain-to-Source-On Widerstand
- Maximal messbarer Strom

Für diese Arbeit wurde ein Inverter mit folgenden Eigenschaften gewählt (Tabelle 3.1):

Tabelle 3.1: Eigenschaften Inverter

Eigenschaft	**Wert**
Zwischenkreisspannung	24V
Maximalstrom	9.7A
Shunt Widerstand	0.08Ω
Drain-to-Source-On Widerstand	0.01Ω
Maximal messbarer Strom	8.65A

Der Motorblock, welcher die physikalischen Eigenschaften eines Motors repräsentiert, hat als Inputsignal die drei Phasenspannungen U, V und W, sowie eine Last. Ausgangssignale sind die Phasenströme, sowie die Drehzahl des Motors. Außerdem können

weitere Zwischenergebnisse des Models angezeigt werden, welche für diese Arbeit nicht relevant sind. Um verschiedene Motoren über das gleiche Modell abzubilden, können Motor spezifische Eigenschaften parametrisiert werden. Dazu gehören:

- Polpaaranzahl
- Motorphasen Widerstand
- Induktivität
- Back-Electromotive-Force (BEMF)-Konstante
- Trägheit und viskose Dämpfung

Für diese Arbeit wird folgender Motor gewählt: (Tabelle 3.2):

Tabelle 3.2: Eigenschaften Motor

Eigenschaft	**Wert**
Polpaarzahl	4
Motorphasen Widerstand	0.75 (Ω)
Induktivität	1.05 (mH)
BEMF-Konstante	3.8 ($\mathrm{mV/min^{-1}}$)
Trägheit	2.4×10^{-6} ($\mathrm{kg\,m^2}$)
viskose Dämpfung	11.6×10^{-6} ($\mathrm{kg\,m^2\,s^{-1}}$)

Um die Daten der Strecke zu generieren wird diese mittels einer feldorientierte Regelung (FOC) kommutiert. Die Regelung besteht aus einem Stromregler, sowie einem übergeordneten Drehzahlregler über welchen der Sollwert vorgegeben wird.

Da die Hardware des Mikrocontrollers die Spannungen nicht kontinuierlich stellen und die Ströme nicht erfassen kann, handelt es sich um zeit-diskrete Werte. Typischerweise werden diese Werte in einer Frequenz von 20kHz erfasst. Diese ist die Schwelle, an welcher die durch eine Stromänderung hervorgerufene Geräusche des Motors für einen Menschen hörbar sind. Außerdem ist diese Frequenz ausreichend, um die nötige Stromänderung für gängige Motorauslegungen zu erzeugen. Ausnahmen sind dabei Motoren mit sehr hohen Nenndrehzahlen. [17]

Um ein dynamisches Verhalten der Strecke zu betrachten wird der Sollwert, sowie die Belastung des Motors variiert. Zusätzlich wird gezeigt, dass durch die Änderung der Werte eine Drehzahländerung folgt, dessen Verlauf von den Eigenschaften des Modells abhängig ist. Ein Ausschnitt dieser Verläufe ist in Abbildung 3.1 dargestellt. Der gesamte Verlauf der Trainingsdaten und Testdaten beträgt jeweils 10s.

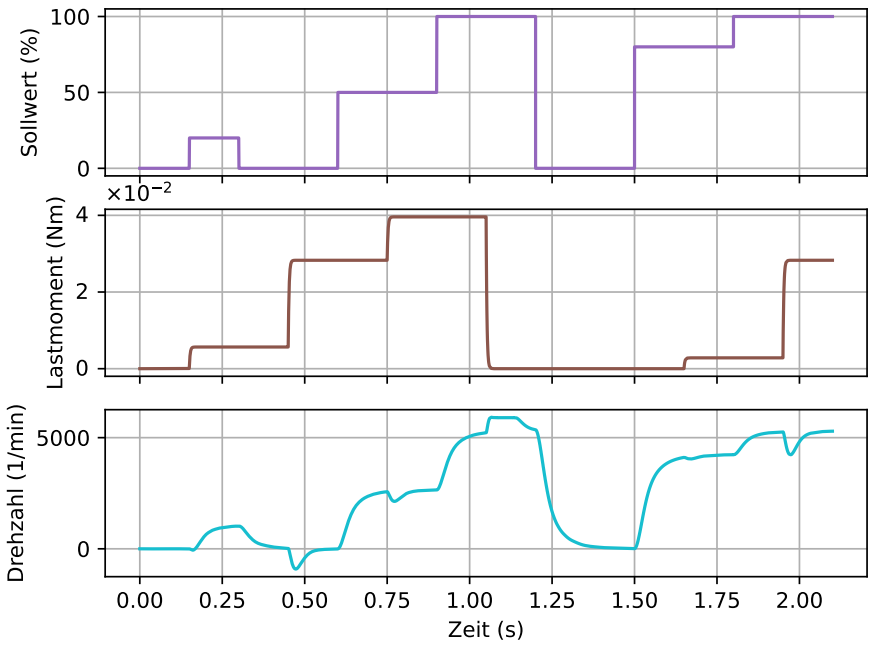

Abbildung 3.1: Sollwert und Drehzahl

In Abbildung 3.2 ist ein Ausschnitt des Verlaufs der Ströme und Spannungen der Phase U dargestellt. Der Sollwert ändert sich von 0% auf 100% worauf ein Start des Motors und sinusförmigen Strom- und Spannungsverlauf folgt. Die Frequenz steigt bis die vorgegebenen Drehzahl erreicht ist.

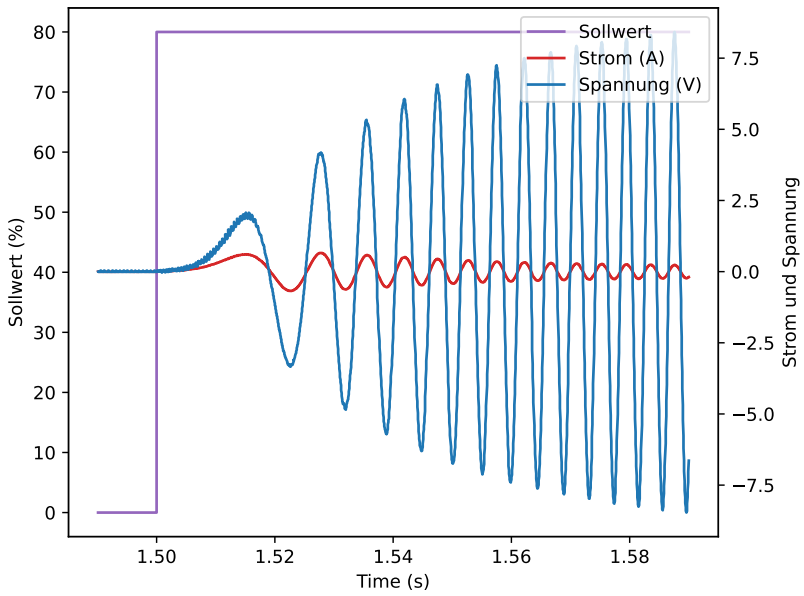

Abbildung 3.2: Strom und Spannungsverlauf beim Starten

In Abbildung 3.3 ist der Strom und Spannungsverlauf bei der Änderung des Lastmomentes dargestellt. Durch die Erhöhung der Last ist ein kurzzeitiger Einbruch der Strom- und Spannungsfrequenz zu erkennen. Durch die FOC wird die vorgegebene Frequenz erneut erreicht, wodurch die Amplitude des Stromes ansteigt.

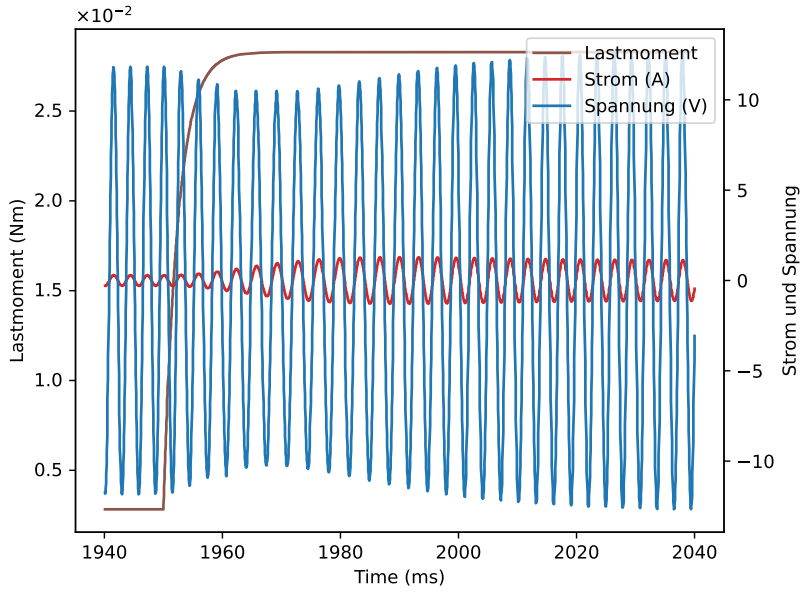

Abbildung 3.3: Strom und Spannungsverlauf bei Laständerung

3.2 Aufbau neuronales Netzwerk

Bei der Aufgabenstellung handelt es sich um sequentielle Abläufe, wofür sich RNNs bewährt haben. Dabei werden die Vorteile des geringeren Speicherverbrauchs gegenüber neuronaler Netzwerke mit Time Delays genutzt. Daraus ergibt sich das in Abbildung 3.4 dargestellte Modell. Dieses besteht aus einer Input-Schicht, der RNN-Schicht, sowie einer nachgelagerten *fully-connected*-Schicht und der Output-Schicht.

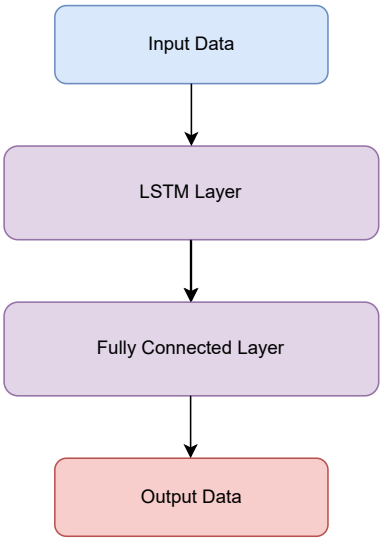

Abbildung 3.4: Neuronales Netz

Die Input-Schicht besteht aus 4 Features, sowie der Sequenzlänge, welche als Hyperparameter untersucht wird. Die RNN-Schicht besteht aus mehreren Units, welche als weiterer Hyperparamter ermittelt werden. Die *fully-connected*-Schicht ist das Bindeglied zur Output-Schicht, welche 3 Featuers umfasst.

3.3 Datenaufbereitung

Um bei jedem Berechnungsschritt auf eine Inputfolge einen Output zur Verfügung zu stellen wird ein Many-to-One RNN verwendet. Für diesen Typ von Netzwerk ist es notwendig die Inputdaten als Sequenzen aufzubereiten.

In Abbildung 3.5 ist die Aufbereitung der Sequenzen dargestellt. In diesem Beispiel werden fünf Zeitschritte betrachtet, wodurch der erste Output nach fünf Zeitschritten berechnet wird. Für vorherige Outputs sind nicht alle Inputdaten bekannt, weshalb für fehlende Werte Initialwerte betrachtet werden. In dieser Strecke sind Initialwerte von 0 passend, da kein Sollwert anliegt und somit kein Strom durch die PMSM fließt.

Eine Eingangssequenz besteht aus folgenden Signalen:

- Spannungen (Uu, Uv, Uw) zum Zeitpunkt $t = 0$
- Lastmoment zum Zeitpunkt $t = 0$

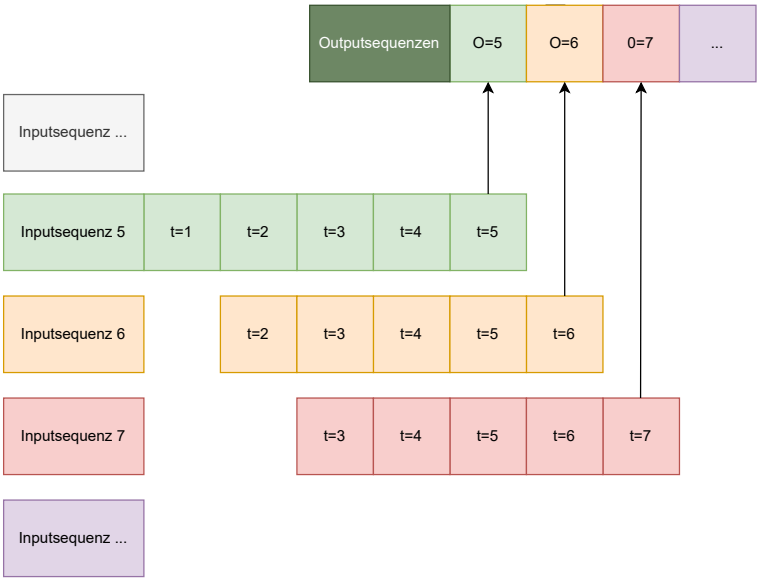

Abbildung 3.5: Aufbereitung Sequenzen

Der Output einer Sequenz besteht aus folgenden Signalen:

- Ströme (Iu,Iv,Iw) zum Zeitpunkt $t = 0$

3.4 Ermittlung der Hyperparameter

In diesem Abschnitt werden die wichtigsten Hyperparameter des Netzwerks betrachtet. Als Initial-werte wird dabei die Arbeit [13] berücksichtigt, welche die Temperatur einer PMSM mittels RNNs bestimmt. Durch die Ähnlichkeit der Strecken ist ein gleiches Verhalten zu erwarten. Zu den wichtigsten Parametern gehört die Sequenzlänge, welche im Bereich von 1-128 als Zweierpotenzfolge betrachtet wird, sowie die Anzahl der Units als Zweierpotenzfolge im Bereich von 2-256. Neben dem MSE der Trainings- und Testdaten werden die benötigten Rechenressourcen beleuchtet. Des Weiteren wird für die RNN-Schicht ein LSTM-Netzwerk, sowie ein einfaches RNN-Netzwerk betrachtet.

Der Hyperparameter Epochen wurde durch Validierung des Trainingsverlaufs ermittelt. Für die weiteren Trainings Hyperparameter wurden Standardwerte verwendet. Diese sind in Tabelle 3.3 dargestellt. [10]

Tabelle 3.3: Hyperparameter

Parameter	**Wert**
Batch size	32
Learningrate	0.001
Optimizer	Adam
Loss Function	Mean Square Error
Epochs	300

3.4.1 Hyperparameter Sequenzen

Um die benötigten Rechenressourcen so gering wie möglich zu halten, sind kurze Inputsequenzen notwendig, da diese maßgeblich die benötigten Rechenoperationen beeinflussen. Kurze Sequenzen haben den Nachteil, dass keine komplette Strom- und Spannungsperiode der PMSM ausgewertet wird. Bei einem dynamischen Verhalten, welches beispielsweise durch eine Änderung der Drehzahlvorgabe gegeben ist, sind diese Verläufe unterschiedlich. Dies bedeutet, der gleiche Spannungsverlauf führt zu unterschiedlichen Stromverläufen.

In der folgenden Abbildung (Abbildung 3.6) ist der Verlauf des MSE über die Zweierpotenzfolge von Sequenzen von 1 bis 128 dargestellt, welcher beim Training mit 256 Units erzeugt wurde. Dabei ist zu erkennen, dass das LSTM-Netzwerk deutliche bessere Werte als das RNN-Netzwerk erzielt. Das RNN-Netzwerk erzeugt mit zunehmenden Sequenzlängen größere Fehler, wodurch die Vorteile des Gedächtnisses des LSTM-Netzwerkes verdeutlicht werden. Des Weiteren ist zu erkennen, dass bei beiden Netzwerken die Trainingsdaten einen geringeren Fehler als die Testdaten aufweisen.

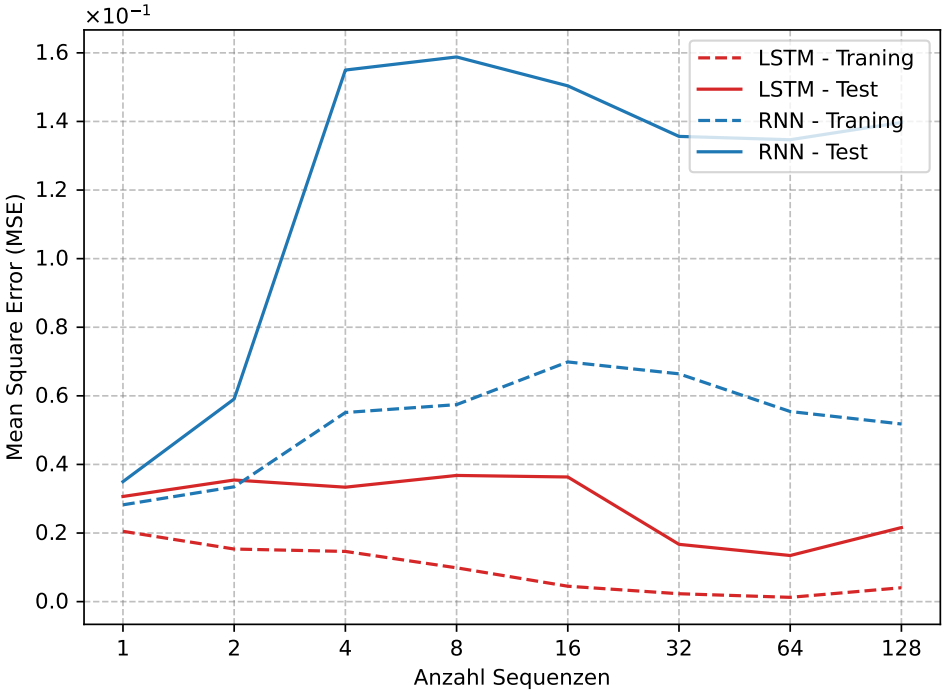

Abbildung 3.6: Sequenzen Mean Square Error

Um den Verlauf des MSE des LSTM-Netzwerkes genauer darzustellen, wurde in Abbildung 3.7) die Sequenz von 32 bis 128 eingeschränkt. Des Weiteren wurde hier das LSTM-Netzwerk alleine betrachtet. Dabei ist zu sehen, dass der sowohl der MSE der Trainings- sowie der Testdaten mit einer Sequenzlänge von 64 am geringsten ist. Anschließend steigt dieser minimal an.

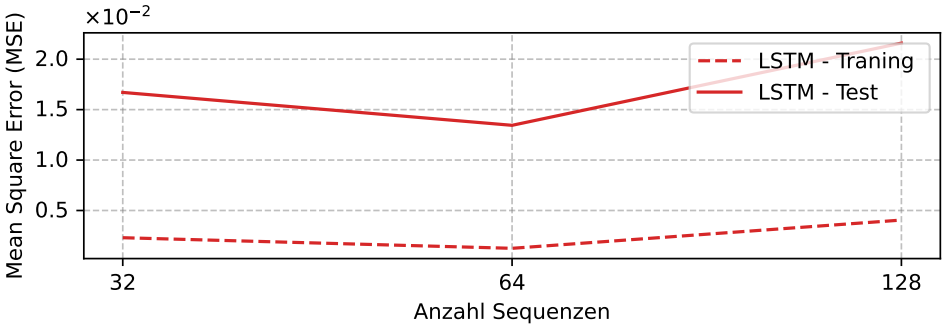

Abbildung 3.7: Sequenzen LSTM Mean Square Error

In Abbildung 3.8 ist durch die Anzahl an Floating-Point-Operations (FLOPs) die benötigte Rechenleistung dargestellt. Es ist zu sehen, dass die FLOPs nahezu proportional zur Anzahl der Sequenzen verlaufen. Das LSTM-Netzwerk benötigt ungefähr die vierfache Anzahl an FLOPs gegenüber des RNN-Netzwerks.

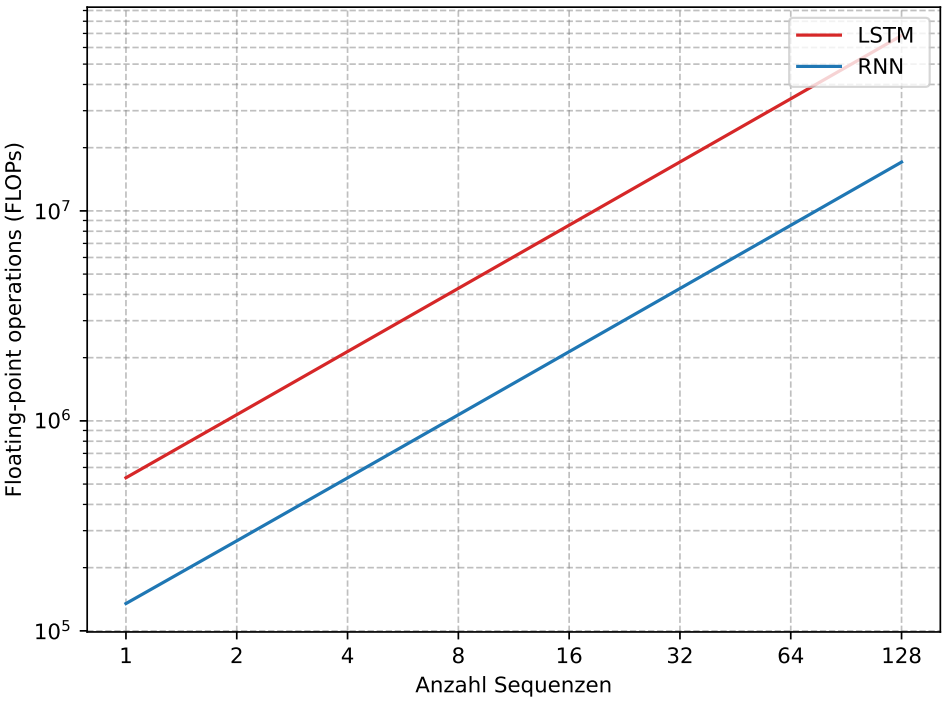

Abbildung 3.8: Sequenzen Flops

Die Anzahl an trainierbaren Parameter ist unabhängig von der Sequenzlänge und wird im nächste Unterkaptiel 3.4.2 betrachtet.

3.4.2 Hyperparameter Units

In der folgenden Abbildung (Abbildung 3.9) sind die Verläufe des MSE in Abhängigkeit von 2 bis 256 Units dargestellt, welche beim Training mit 64 Sequenzen erzeugt wurden. Dabei ist zu erkennen, dass das LSTM-Netzwerk ein ähnliches Verhalten wie das RNN-Netzwerk aufweist. Mit steigender Anzahl an Units sinkt der MSE. Beim LSTM-Netzwerk ist ein kleiner Anstieg der Testdaten ab 64 Units zu erkennen. Bei dem RNN-Netzwerk ist ein Anstieg ab 128 Units zu erkennen.

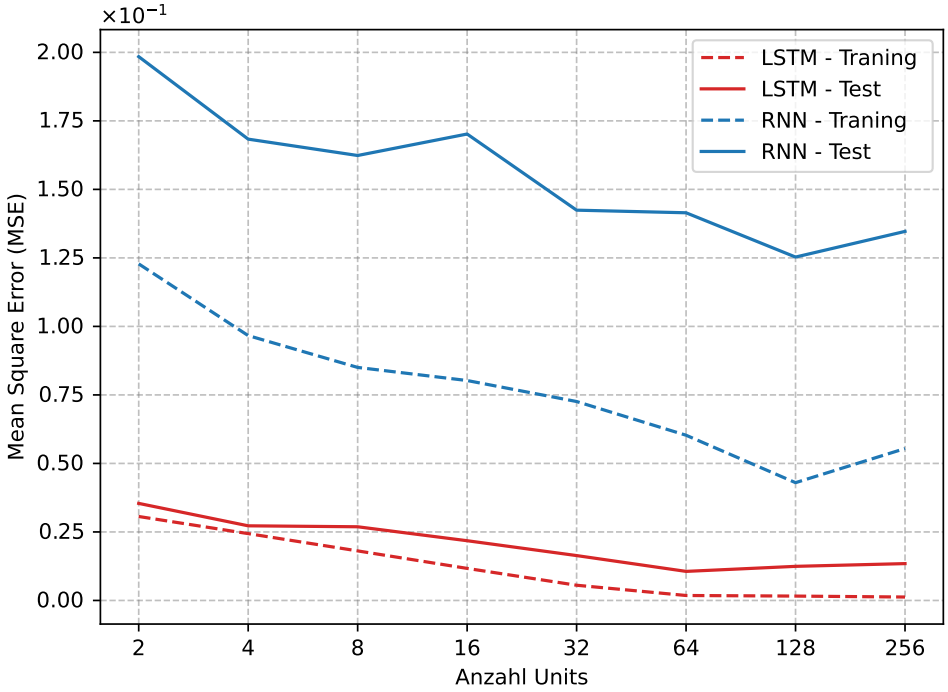

Abbildung 3.9: Units Mean Square Error

Um den Verlauf des MSE des LSTM-Netzwerkes genauer darzustellen, wurde in Abbildung 3.10) die Units von 32 bis 256 eingeschränkt. Des Weiteren wurde hier das LSTM-Netzwerk alleine betrachtet. Dabei ist zu erkennen, dass der Fehler der Trainingsdaten kontinuierlich abnimmt, wobei die Testdaten bei größeren Werten leicht ansteigen. Dies sind Anzeichen für eine Überanpassung.

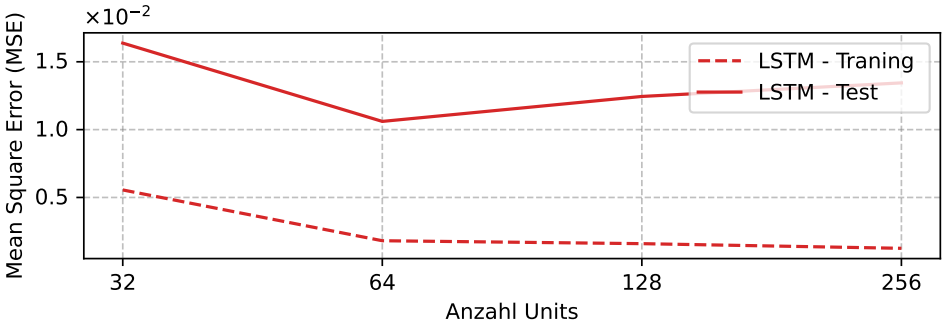

Abbildung 3.10: Units LSTM Mean Square Error

In Abbildung 3.11 und Abbildung 3.12 sind die Anzahl der FLOPs und Parameter in Abhängigkeit der Units dargestellt. Es ist zu erkennen, dass die FLOPs und Parameter quadratisch mit der Anzahl der Units ansteigen.

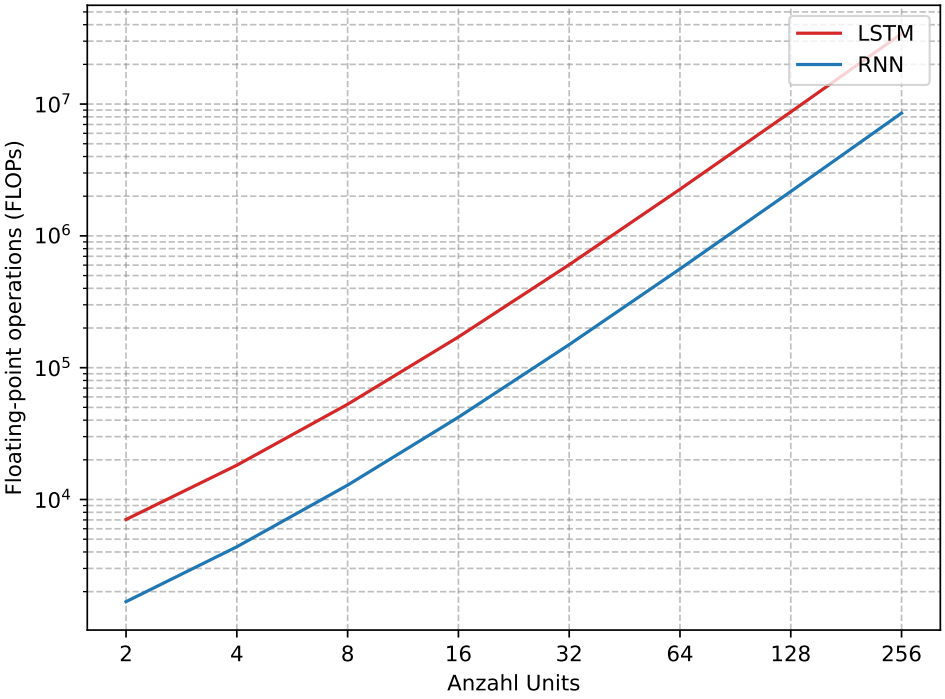

Abbildung 3.11: Units FLOPs

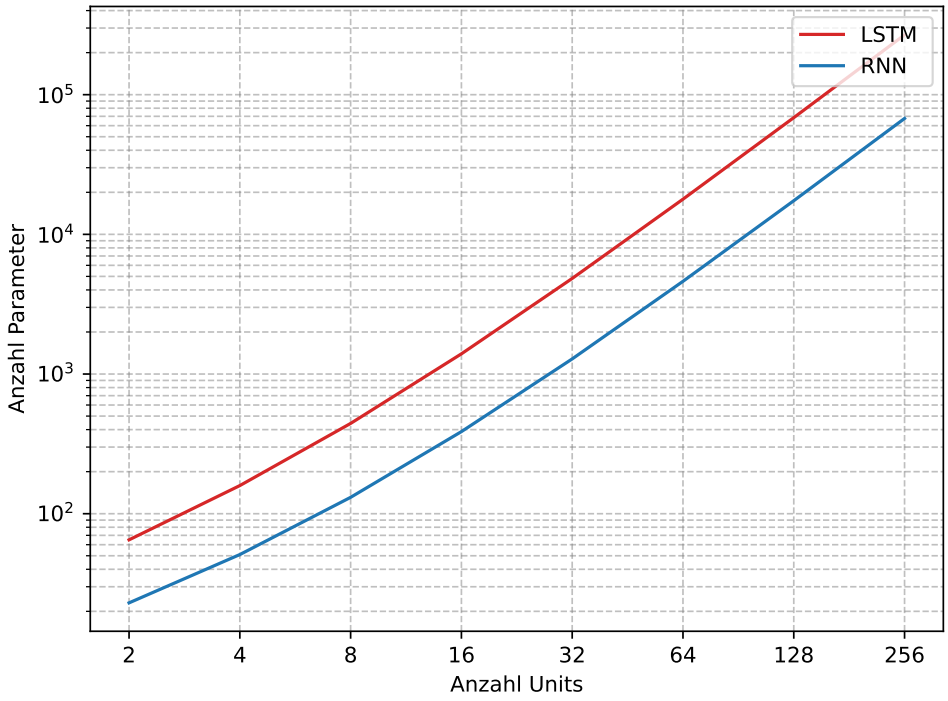

Abbildung 3.12: Units Parameter

3.4.3 Zusammenfassung

Es wurde gezeigt, dass ein LSTM-Netzwerk mit 64 Units und einer Sequenzlänge von 64 die besten Ergebnisse für die Testdaten liefert. Da beim MSE der Trainingsdaten nur der letzte Batch berücksichtigt wird, ist das Ergebnis der Testdaten wichtiger. Somit wird dieses Netzwerk für eine weitere Verwendung empfohlen. Das Netzwerk liefert einen MSE der Testdaten von 1.06×10^{-2} und einen MSE von 1.81×10^{-3} für die Trainingsdaten. Dafür benötigt das Netzwerk 2.26×10^{6} Rechenoperationen und 1.79×10^{4} Parameter.

4 Embedded KI-Framework

Zur Erstellung, dem Training und der Validierung des neuronalen Netzwerks wird ein Framework eingesetzt, das diese Prozesse unterstützt. Neben den zu erstellten Modellen ist in dieser Arbeit die Ausführbarkeit auf einem eingebetteten-System mit geringen Ressourcen im Fokus. Frameworks für diese Anforderungen sind häufig unter dem Begriff TinyML zu finden. [4] [18, Kapitel 13] [18, Kapitel 15]

4.1 Randbedingungen

Die wichtigsten Randbedingungen eines TinyML Frameworks sind:

- **Speicherallokation**: Diese ist meist statisch, jedoch wird häufig eine Mehrfachverwendung des statischen Speichers ermöglicht.
- **Portierungsmöglichkeiten**: Gibt es eine gute Hardware Abstraktion, sodass das Framework von verschiedenen Mikrocontrollern verwendet werden kann. Des Weiteren ist es wichtig, welche Compiler unterstützt werden und welche Laufzeitbibliotheken davon benötigt werden. Häufig ist kein Betriebssystem notwendig, sodass diese als Bare-Metal ausgeführt werden können.
- **Modelloptimierung**: Ist eine unsichere Optimierung wie Quantisierung oder Pruning möglich.
- **Laufzeitsystem**: Ist ein zusätzlicher Interpreter notwendig und können Optimierungen der Operatoren einfach eingesetzt werden, um beispielsweise die Digital-Signal-Processing-Unit (DSP) des Mikrocontrollers zu nutzen.

4.2 Auswahl

Die wichtigsten Anforderungen an das Framework sind, dass es das ermittelte LSTM-Netzwerk unterstützt und eine Integration ohne zusätzliche Laufzeitbibliotheken als Bare-Metal-Applikation in bestehende Softwaresysteme ermöglicht. Außerdem sollte das Framework keine zusätzlichen Kosten verursachen, weshalb eine Open-Source-Lösung angestrebt wird.

In den Arbeiten [19] und [20] werden verschiedene Frameworks aufgeführt, die häufig für TinyML-Anwendungen eingesetzt werden. Dazu gehören unter anderem *TensorFlow Lite Micro*, *µTensor*, *micro TVM* und *Edge ML*. Diese Frameworks sind auf die Anforderungen eingebetteter Systeme zugeschnitten und bieten effiziente Möglichkeiten zur Integration. Da sich diese Frameworks teilweise noch in der Entwicklung befinden, existieren oft nur begrenzte verlässliche Dokumentationen. Um die Kompatibilität mit dem

LSTM-Netzwerk zu überprüfen, ist daher eine detaillierte Analyse des Quellcodes notwendig.

Aufgrund der breiten Unterstützung durch die Community und der häufigen Verwendung in der Literatur wurde *TensorFlow Lite Micro* für diese Arbeit ausgewählt. Die Analyse ergab, dass das Framework alle notwendigen Anforderungen erfüllt und gut in das bestehende System integriert werden kann. In [21] wird beschrieben und evaluiert, dass das Framework entwickelt wurde, um Modelle mit minimalem Speicherverbrauch und geringem Laufzeit-Overhead auszuführen. Insgesamt zeichnet sich das Framwork durch folgenden Eigenschaften aus:

- **Statische Speicherallokation**: Der Speicher wird effizient verwaltet und kann mehrfach verwendet werden, um die Ressourcennutzung zu optimieren.
- **Bare-Metal-Ausführung**: Es wird lediglich ein C++11-Compiler benötigt, ohne zusätzliche Laufzeitbibliotheken.
- **Modelloptimierung**: Durch Quantisierung kann der Speicherbedarf reduziert und die Effizienz auf eingebetteten Systemen verbessert werden.
- **Flexibler Interpreter**: Der Interpreter ermöglicht die einfache Integration neuer Operatoren und die Nutzung von Hardwarebeschleunigungen wie der DSP des Mikrocontrollers.
- **Kompatibilität mit TensorFlow-Modellen**: Modelle können nach einer Konvertierung in das TensorFlow-Lite-Format direkt genutzt werden.

Zusammenfassend wurde *TensorFlow Lite Micro* als geeignetes Framework ausgewählt, da es die spezifischen Anforderungen des LSTM-Netzwerks und der Zielplattform vollständig unterstützt.

5 Zusammenfassung

In dieser Arbeit wurde gezeigt, dass ein neuronales Netzwerk die Strecke einer PMSM mit einem MSE von 1.06×10^{-2} nachgebildet werden kann. Hierfür wird ein LSTM-Netzwerk mit 64 Units und einer Sequenzlänge von 64 ermittelt. Als weitere Untersuchung kann sowohl die Genauigkeit erhöht als auch die benötigten Ressourcen reduziert werden. Ein einfaches RNN-Netzwerk liefert um den Faktor drei schlechtere Ergebnisse der Genauigkeit.

Für die Portierung auf einen Mikrocontroller wird das Framework Tensoflow Lite Micro empfohlen. Dies erfüllt die Anforderungen für die einfache Integration in ein bestehendes Softwaresystem. Des Weiteren wird das LSTM-Netzwerk unterstützt. Ein weiterer Punkt ist die weite Verbreitung des Frameworks, die eine gute Unterstützung durch die Community erwarten lässt.

Literaturverzeichnis

[1] J. Böcker, "Geregelte Drehstromantriebe, Controlled Three-Phase Drives," https://ei.uni-paderborn.de/fileadmin-eim/elektrotechnik/fg/lea/Lehre/GDA/Dokumente/Geregelte_Drehstromantriebe_DE_EN.pdf, 2021.

[2] D. Kriesel, *Ein kleiner Überblick über Neuronale Netze*, 2007.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning - das umfassende Handbuch : Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze.* Bonn: MITP, 2018.

[4] C. François, *Deep Learning mit Python und Keras - Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek.* Heidelberg: MITP-Verlags GmbH & Co. KG, 2018.

[5] A. Burkov, *The Hundred-Page Machine Learning Book.* Andriy Burkov, 2019. [Online]. Available: https://books.google.de/books?id=0jbxwQEACAAJ

[6] A. Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks," http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015, blogpost.

[7] A. Geron, *Hands-on machine learning with scikit-learn, keras, and TensorFlow 3e*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022.

[8] A. Géron, *Praxiseinstieg Machine Learning mit Scikit-Learn und TensorFlow - Konzepte, Tools und Techniken für intelligente Systeme.* Sebastopol: O'Reilly, 2018.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[10] F. Chollet *et al.*, "Keras 3 API - LSTM Layer," https://keras.io/api/layers/recurrent_layers/lstm/, 2024.

[11] K. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[12] R. Kumar, R. A. Gupta, and A. K. Bansal, "Identification and control of pmsm using artificial neural network," in *2007 IEEE International Symposium on Industrial Electronics*, 2007, pp. 30–35.

[13] W. Kirchgässner, O. Wallscheid, and J. Böcker, "Estimating electric motor temperatures with deep residual machine learning," *IEEE Transactions on Power Electronics*, vol. 36, pp. 7480 – 7488, 12 2020.

[14] P. Pietrzak, M. Wolkiewicz, and T. Orlowska-Kowalska, "Pmsm stator winding fault detection and classification based on bispectrum analysis and convolutional neural network," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 5, pp. 5192–5202, 2023.

[15] J. Song and W. Song, "Model Predictive Control of PMSM Drives with Reduced DC-Link Capacitance," in *2023 IEEE International Conference on Predictive Control of Electrical Drives and Power Electronics (PRECEDE)*, 2023, pp. 1–6.

[16] A. Kleimaier, "Grundlagen elektrische Antriebe: Kapitel 11 - Aufbau und Betriebsverhalten der Synchronmaschine," https://www.alexander-kleimaier.de/vorlesung/VorlesungGeA_Kapitel11.pdf, 2024.

[17] ——, "Grundlagen elektrische Antriebe: Kapitel 13 - Ansteuerung und Systemverhalten BLDC-Motor," https://www.alexander-kleimaier.de/vorlesung/VorlesungGeA_Kapitel13.pdf, 2024.

[18] P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019. [Online]. Available: https://books.google.de/books?id=tn3EDwAAQBAJ

[19] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on tinyml," *IEEE Access*, vol. 11, pp. 96 892–96 922, 2023.

[20] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware: A review," *IEEE Sensors Journal*, vol. 22, no. 22, pp. 21 362–21 390, 2022.

[21] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," 2021. [Online]. Available: https://arxiv.org/abs/2010.08678