Institut für Wärmetechnik

TU Graz

Michael EIBL, BSc

Modellierung von Festoxidbrennstoffzellen mittels künstlicher neuronaler Netze

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

Masterstudium Maschinenbau

Eingereicht an der

Technischen Universität Graz

im April 2019

Betreuerin

Ass.Prof. Dipl.-Ing. Dr.techn. Vanja Subotić

Institut für Wärmetechnik

Beurteiler

Univ.-Prof. Dipl.-Ing. Dr.techn. Christoph Hochenauer

Institut für Wärmetechnik

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Datum

Unterschrift

Vorwort

Die vorliegende Masterarbeit entstand im Zeitraum von Februar 2018 bis März 2019 am Institut für Wärmetechnik der Technischen Universität Graz. Die durchgeführten Berechnungen entstanden zum Großteil auf einem Simulationsrechner, der mir an meinem Arbeitsplatz am Institut zur Verfügung gestellt wurde.

Zuerst möchte ich mich bei meiner Betreuerin Frau Ass.Prof. Dr.techn. Vanja Subotić dafür bedanken, dass sie mir diese – für ein Maschinenbauinstitut doch eher ungewöhnliche – Masterarbeit ermöglicht hat. Besonders bedanken möchte ich mich auch für die ausgezeichnete Unterstützung und stets äußerst angenehme Gesprächsatmosphäre im Labor und bei Besprechungen. Großer Dank gilt auch Herrn Univ.-Prof. Dr.techn. Christoph Hochenauer für die Übernahme der Beurteilung dieser Arbeit.

Ebenfalls möchte ich mich bei Herrn Dipl.-Ing. Thomas Thaller für die gute Zusammenarbeit bedanken und dafür, dass er mir Teile seiner Mess- und Simulationsdaten zur Verfügung gestellt hat. Weiterer Dank gebührt auch den zahlreichen Personen, die unentgeltlich ihr Wissen auf Stack Overflow und ähnlichen Webseiten weitergeben. Eine Programmierarbeit dieser Größenordnung wäre für mich als Maschinenbaustudent ansonsten wohl nicht möglich gewesen.

Abschließend möchte ich mich noch bei meiner Familie und meinen Freunden bedanken: Ihr habt mich auf meinem gesamten Lebensweg immer unterstützt und für umfangreiche Abwechslung vom Studium gesorgt. Darauf werde ich immer gerne zurückblicken!

Abstract

Title: Modelling of solid oxide fuell cells using artificial neural networks

Author: Michael Eibl

1st keyword: polarisation model, impedance model SOFC
2nd keyword: artifical neural networks
3rd keyword: single cell

In order to cover the global energy demand, the commonly used fossil fuels and their combustion lead to high greenhouse gas and pollutant emissions. Solid oxide fuel cells could make a significant contribution to reducing such undesired emissions in the future. They convert the chemical energy of fuels directly into electrical energy, with low emissions and high efficiencies. In order to develop and optimize systems based on fuel cells, as well as to ensure reliable and safe operation, mathematical models of the cells are required. A possible approach is to model fuel cells via artificial neural networks. Models with artificial neural networks are not based on physical equations, but on statistical dependencies of underlying data. Their generation process thus can be vastly automated. Calculations with the models are also very time-efficient, which is why they are also suitable for control-oriented applications. In this master thesis artificial neural networks were used to model essential electrochemical properties of solid oxide fuel cells.

For this purpose programs for automatic model generation were developed and implemented in Python with the software library TensorFlow. Subsequently, several models for polarisation and impedance prediction were generated using these programs. The polarisation models were created with simulation data from a physical single cell model, while the impedance models were created with measurement data from a single cell test bench. The models were then thoroughly reviewed with regard to their predictive quality.

Both polarisation curves and cell impedance, while varying several input parameters, showed good accordance with the reference data. In comparison with the simulated and measured data respectively, the models correctly depicted the physical characteristics of the data. With a few exceptions, the models also achieved a high level of prediction accuracy.

Kurzfassung

Titel: Modellierung von Festoxidbrennstoffzellen mittels künstlicher neuronaler Netze

Autor: Michael Eibl

1. Stichwort: Polarisationsmodell, Impedanzmodell SOFC
2. Stichwort: Künstliche neuronale Netze
3. Stichwort: Einzelzelle

Die Verbrennung fossiler Brennstoffe zur Deckung des weltweiten Energieverbrauchs, führt zu hohen Treibhausgas- und Schadstoffemissionen. Festoxidbrennstoffzellen könnten in Zukunft einen wichtigen Beitrag zur Senkung dieser Emissionen leisten: Sie erlauben die direkte Umwandlung der chemisch gebundenen Energie eines Brennstoffs in elektrische Energie, wobei die Umwandlung höchst effizient und weitgehend schadstofffrei erfolgt. Um Systeme mit Brennstoffzellen zu entwickeln und zu optimieren, sowie einen sicheren Betrieb der Zellen zu gewährleisten, werden mathematische Modelle der Zellen benötigt. Ein möglicher Ansatz ist die Modellierung der Zellen durch künstliche neuronale Netze. Modelle mit künstlichen neuronalen Netzen bilden dabei keine physikalischen Gesetzmäßigkeiten ab, sondern statistische Zusammenhänge von Daten und können deshalb weitgehend automatisiert erstellt werden. Berechnungen mit den Modellen sind zudem sehr zeiteffizient, weshalb sie sich auch für regelungstechnische Anwendungen eignen. In dieser Masterarbeit wurden künstliche neuronale Netze eingesetzt, um wesentliche elektrochemische Eigenschaften von Festoxidbrennstoffzellen zu modellieren.

Für diesen Zweck wurden Programme zur automatisierten Modellerstellung entwickelt und in Python mit der Programmbibliothek TensorFlow implementiert. Anschließend wurden mit diesen Programmen verschiedene Polarisations- und Impedanzmodelle von Festoxidbrennstoffzellen erstellt. Die Polarisationsmodelle wurden mit Simulationsdaten von einem physikalischen Zellmodell erstellt, die Impedanzmodelle hingegen mit Messdaten von einem Einzelzellenprüfstand. Die Modelle wurden danach hinsichtlich ihrer Vorhersagequalität detailliert untersucht.

Sowohl die Polarisationskurven, als auch die Zellimpedanz konnten abhängig von mehreren Eingabeparametern erfolgreich modelliert werden. Im Vergleich mit den Simulations- bzw. Messdaten gaben die erstellten Modelle die wesentlichen physikalischen Merkmale der Daten korrekt wieder. Bis auf wenige Ausnahmen erreichten die Modelle zudem eine hohe Vorhersagegenauigkeit.

Inhaltsverzeichnis

1 Einleitung

Der weltweite Energieverbrauch wird in erster Linie durch fossile Brennstoffe gedeckt [12], was hohe Treibhausgas- und Schadstoffemissionen zur Folge hat. Eine mögliche Schlüsseltechnologie zur Senkung dieser Emissionen ist die Brennstoffzellentechnologie. Brennstoffzellen wandeln die chemische Energie des gasförmigen Brennstoffs direkt in elektrische Energie um und erreichen dabei hohe elektrische Wirkungsgrade. Eine Variante von Brennstoffzellen sind die sogenannten Festoxidbrennstoffzellen (engl. solid oxide fuel cells, SOFCs). Sie tragen ihre Bezeichnung aufgrund ihres keramischen Feststoffelektrolyts. Dieser wird erst bei hohen Temperaturen ionenleitfähig, weshalb SOFCs üblicherweise bei Temperaturen von 600 °C bis 1000 °C betrieben werden. Die entstehende Abwärme weist somit ein hohes Temperaturniveau auf, weshalb ein hoher Anteil der Wärme für weitere Anwendungen genutzt werden kann. Im Gegensatz zu anderen Brennstoffzellentypen, die ausschließlich hochreinen Wasserstoff benötigen, erlauben SOFCs die Umsetzung verschiedenster Brennstoffe – möglich sind zum Beispiel Erdgas, Biogas, Ammoniak oder Dieselreformate. [28]

Es gibt, aus den oben genannten Gründen, vielversprechende Anwendungsmöglichkeiten für SOFCs. In größeren Stückzahlen werden sie seit einigen Jahren zur dezentralen Kraft-Wärme-Kopplung in Einfamilienhäusern eingesetzt. Die gleichzeitige Nutzung der produzierten elektrischen Energie und der Abwärme, erlaubt Effizienzsteigerungen gegenüber der konventionellen Energiebereitstellung. [18] Denkbar ist auch ein Einsatz im Automobilbereich, zum Beispiel als Range Extender für Elektroautos. SOFCs erreichen bessere Wirkungsgrade als Verbrennungsmotoren, emittieren weniger Schadstoffe und sind dennoch hinsichtlich der Brennstoffwahl flexibel [11]. Zur Energiebereitstellung können SOFCs auch mit Gasturbinen in Kraftwerken kombiniert werden (SOFC-GT). Solche SOFC-GT Systeme können die besten bisher erreichten Wirkungsgrade von Gas-und-Dampf-Kraftwerken deutlich übertreffen. [38, 52] Die genannten Anwendungen stellen unterschiedliche Leistungsanforderungen an die Zellen: Zur Kraft-Wärme-Kopplung in Einfamilienhäusern werden Zellen mit elektrischen Leistungen in der Größenordnung von einem Kilowatt verbaut [18], bei SOFC-GT Systemen hingegen können die Leistungen hunderte Kilowatt bis mehrere Megawatt betragen [38, 52]. Um die benötigten Leistungen abdecken zu können, werden einzelne Zellen, meist in Serie und/oder parallel, zu sogenannten Stacks zusammengeschlossen.

Um Systeme mit Brennstoffzellen – beispielsweise die zuvor genannten – entwickeln und optimieren zu können, oder die Funktion der Zellen im Betrieb zu überwachen, werden

meist mathematische Modelle der Zellen benötigt. In den Ingenieurswissenschaften werden üblicherweise Modelle verwendet, die auf physikalischen Gesetzmäßigkeiten basieren. Die Parameter dieser Modelle, z.B. Materialparameter wie die Porosität, haben eine physikalische Bedeutung – ihre Bestimmung ist aber oft schwierig. Komplizierte Modelle, zum Beispiel numerische CFD-Simulationen (computational fluid dynamics), benötigen außerdem hohe Rechenzeiten. Ein alternativer Ansatz ist die Nachbildung von Zellen durch Blackboxmodelle. Diese bilden keinerlei physikalische Prozesse ab, sondern geben statistische Zusammenhänge bzw. das Muster von Daten wieder. Viele Blackboxmodelle sind selbstlernend, d.h. die Bestimmung ihrer Parameter ist weitgehend automatisiert. Berechnungen mit Blackboxmodellen sind darüber hinaus sehr schnell, weshalb sie sich besonders zur Echtzeitüberwachung und Regelung von im Einsatz befindlichen Zellen eignen. [59] Blackboxmodelle können auch eingesetzt werden, um die Betriebsparameter von Zellen schnell und zeiteffizient zu optimieren: In einigen ausgewählten Messpunkten wird das Betriebsverhalten der Zelle experimentell untersucht und als Modell abgebildet. Mit dem Modell werden dann verschiedene Betriebsparameter der Zelle simuliert, die Zellleistung bestimmt und dabei auftretende Verluste quantifiziert. Der experimentelle Messaufwand kann durch diese Vorgehensweise, im Vergleich zu einer vollständigen Vermessung des Betriebsverhaltens der Zelle, erheblich reduziert werden.

Vertreter von Blackboxmodellen sind zum Beispiel Support Vector Machines (SVM) oder künstliche neuronale Netze. Gerade Letztere sind enorm leistungsfähig und können, mit speziellen Netzarchitekturen, auch zeitliche Zusammenhänge in den Daten erfassen und abbilden. Die wohl prominentesten Anwendungen für künstliche neuronale Netze sind die Bilderkennung und die Spracherkennung, sie werden aber auch für ingenieurstechnische Anwendungen eingesetzt. Es wurden zudem bereits Anwendungen für SOFCs demonstriert. So gelang z.B. die Modellierung der Zellspannung einer SOFC, abhängig von verschiedenen Material- und Brennstoffparametern [39, 63]. Auch die im Betrieb einer SOFC auftretende Degradation, eine mit der Zeit zunehmende Verschlechterung der Zelleigenschaften, konnte erfolgreich durch ein künstliches neuronales Netz abgebildet werden [37].

Die vorliegende Arbeit ist die erste Masterarbeit am Institut für Wärmetechnik, die das Thema neuronale Netze behandelt. Ziel der Arbeit war es deshalb, wichtige Grundlagen zur Modellierung von SOFCs durch neuronale Netze zu schaffen. Es sollte untersucht werden, inwiefern sich neuronale Netze zur Modellierung von SOFCs eignen. Im Verlauf der Arbeit sollten Netzmodelle für verschiedene SOFC-Anwendungen erstellt und hinsichtlich ihrer Vorhersagequalität untersucht werden. Um das zu erreichen, beginnt die Arbeit mit theoretischen Grundlagen zu SOFCs und neuronalen Netzen. Die Funktionsweise der neuronalen Netze, deren Erstellung und dabei auftretende Problemstellungen werden im Detail betrachtet. Danach werden drei Prozeduren zur Netzerstellung, die im Laufe der Arbeit entwickelt und implementiert wurden, beschrieben. Diese ermöglichten es, mit Daten neuronale Netzmodelle weitgehend automatisiert zu erstellen. Im Anschluss werden mehrere mit diesen Prozeduren generierte Modelle vorgestellt. Die Vorhersagequalität der

Modelle wird untersucht und die dabei auftretenden Schwierigkeiten werden beschrieben. Es werden verschiedene Methoden angewandt um kritische Modellparameter, die die Vorhersagequalität der Modelle beeinflussen, zu identifizieren.

2 Grundlagen von Festoxidbrennstoffzellen

Dieses Kapitel bietet eine Zusammenfassung der Grundlagen von SOFCs, die zum Verständnis dieser Masterarbeit wichtig sind. Zuerst werden die Funktionsweise und die thermodynamischen Grundlagen erklärt. Danach folgt ein Überblick über die im Betrieb von SOFCs auftretenden Verluste. Abschließend werden zwei elektrochemische Charakterisierungsmethoden beschrieben, die Aussagen über das Betriebsverhalten von Brennstoffzellen ermöglichen.

2.1 Funktionsweise

SOFCs bestehen im Wesentlichen aus einem keramischen Feststoffelektrolyt, zwei Elektroden (Anode und Kathode) und Verbindungsleitungen. Abbildung 2.1 zeigt ein Schema einer SOFC, mit den auftretenden Reaktionen, bei Verwendung von Wasserstoff als Brennstoff. [43]

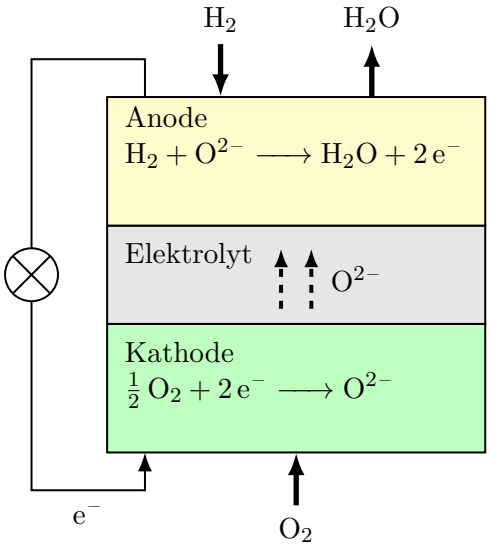


Abbildung 2.1: Schema einer SOFC, nach [43].

In einer SOFC laufen zwei räumlich voneinander getrennte Halbreaktionen ab. An der Kathodenseite, in Abbildung 2.1 unten dargestellt, strömt Sauerstoff ein. Die Sauerstoffmoleküle nehmen Elektronen auf und es werden zweifach negativ geladene Sauerstoffionen (Anionen) gebildet:

$$\frac{1}{2}\,\mathrm{O_2} + 2\,\mathrm{e^-} \longrightarrow \mathrm{O^{2-}} \tag{2.1}$$

Diese Sauerstoffionen werden durch den Elektrolyten geleitet und reagieren mit dem an der Anodenseite, in Abbildung 2.1 oben dargestellt, einströmenden Wasserstoff. Dabei wird unter Elektronenabgabe Wasserdampf gebildet:

$$\mathrm{H_2} + \mathrm{O^{2-}} \longrightarrow \mathrm{H_2O} + 2\,\mathrm{e^-} \tag{2.2}$$

Damit die beiden räumlich voneinander getrennten Halbreaktionen ablaufen können, muss ein Elektronenaustausch zwischen Anode und Kathode erfolgen. Deshalb werden Anode und Kathode über einen Verbraucher elektrisch leitend verbunden. Die über die Verbindung fließenden Elektronen können als Strom genutzt werden. Die Summenreaktion der beiden Halbreaktionen aus Gleichungen (2.1) und (2.2) lautet:

$$\mathrm{H_2} + \frac{1}{2}\,\mathrm{O_2} \longrightarrow \mathrm{H_2O} \tag{2.3}$$

2.2 Thermodynamische Grundlagen

Der folgende Abschnitt basiert zum Großteil auf [43, 32]. Betrachtet werden soll eine allgemeine Reaktion der Form:

$$a\,\mathrm{A} + b\,\mathrm{B} + ... \longrightarrow d\,\mathrm{D} + e\,\mathrm{E} + ... \tag{2.4}$$

Darin sind $\mathrm{A}, \mathrm{B}, \ldots$ die Edukte und $\mathrm{D}, \mathrm{E}, \ldots$ die Produkte der Reaktion. $\mathrm{a}, \mathrm{b}, \ldots$ und $\mathrm{d}, \mathrm{e}, \ldots$ sind die zu den Reaktionspartnern zugehörigen stöchiometrischen Koeffizienten. Die molare Standardreaktionsenthalpie $\Delta_R H_m^0$ der allgemeinen Reaktion in Gleichung (2.4), wird über die Summe der molaren Standardbildungsenthalpien $\Delta_B H_{m,i}^0(T^0)$ (bei Standardbedingungen $T^0 = 298.15\,\mathrm{K}$ und $p^0 = 1\,\mathrm{bar}$) der einzelnen Reaktionspartner berechnet:

$$\Delta_R H_m^0(T^0) = \sum_i \nu_{st,i} \cdot \Delta_B H_{m,i}^0(T^0) \tag{2.5}$$

$\nu_{st,i}$ bezeichnet die stöchiometrischen Koeffizienten. Stöchiometrische Koeffizienten der Produkte werden positiv gezählt, die der Edukte negativ. Analog wird die molare Standardreaktionsentropie $\Delta_R S_m^0(T^0, p^0)$ der Reaktion bestimmt durch:

$$\Delta_R S_m^0(T^0, p^0) = \sum_i \nu_{st,i} \cdot \Delta_B S_{m,i}^0(T^0, p^0) \tag{2.6}$$

$\Delta_R S_m^0(T^0, p^0)$ wird durch die Summe der Standardbildungsenthalpien $\Delta_B S_{m,i}^0(T^0, p^0)$ für einen Zustand ermittelt, in dem die einzelnen Spezies getrennt und mit Partialdruck $p_i = p^0 = 1\,\text{bar}$ vorliegen. Die real auftretenden Druckverhältnisse sind daraus folgend noch nicht berücksichtigt. Findet die Reaktion bei einer von T^0 abweichenden Temperatur T statt, ändert sich die Reaktionsenthalpie. Für ideales Gas wird die molare Reaktionsenthalpie $\Delta_R H_m(T)$, ausgehend von der molaren Standardreaktionsenthalpie $\Delta_R H_m^0(T^0)$, über Integration der molaren Wärmekapazität $C_{mp}(T)$ ermittelt:

$$\Delta_R H_m(T) = \Delta_R H_m^0(T^0) + \int_{T^0}^{T} C_{mp}(T)\,\mathrm{d}T \tag{2.7}$$

Die molare Reaktionsentropie $\Delta_R S_m(T, p)$ hängt zusätzlich zur Temperatur auch vom Druck ab. Läuft die Reaktion bei einer Temperatur T und einem Druck p ab, wird die molare Reaktionsentropie berechnet mit:

$$\Delta_R S_m(T, p) = \Delta_R S_m^0(T^0, p^0) + \int_{T^0}^{T} \frac{C_{mp}(T)}{T}\,\mathrm{d}T - R_m \sum_i \nu_{st,i} \ln \frac{p_i}{p^0} \tag{2.8}$$

Der letzte Term der Gleichung beschreibt die Abweichung von den Druckverhältnissen ($p_i = p^0 = 1\,\text{bar}$ für jede Spezies) des Standardzustands. Darin ist R_m die allgemeine Gaskonstante und p_i der Partialdruck der betrachteten Komponente. Die freie Enthalpie G ist definiert als:

$$G = H - TS \tag{2.9}$$

Gleichung (2.9), angeschrieben mit den bisher verwendeten molaren Größen in Gleichungen (2.7) und (2.8), erlaubt die Bestimmung der molaren freien Reaktionsenthalpie $\Delta_R G_m(T, p)$:

$$\Delta_R G_m(T, p) = \Delta_R H_m(T) - T\Delta_R S_m(T, p) \tag{2.10}$$

Für einen isotherm-isobaren Prozess in einem System, entspricht die freie Reaktionsenthalpie der maximalen elektrischen Arbeit, die dieses System verrichten kann. Das erlaubt eine Verknüpfung der thermodynamischen mit den elektrischen Größen einer Brennstoffzelle. Der zwischen Anode und Kathode fließende Strom ist proportional der pro Mol Brennstoff an der Anode freiwerdenden Anzahl an Elektronen: [61]

$$I = -z_e F \tag{2.11}$$

Dabei ist z_e die Elektronenzahl und F die Faraday-Konstante. Wird beispielsweise Wasserstoff als Brennstoff verwendet, ist $z_e = 2$, vgl. Gleichung (2.2). Pro Mol H_2 werden an der Anode $z_e = 2\,\text{mol}$ Elektronen frei. Die Leistung P der Brennstoffzelle kann nun sowohl durch thermodynamische, als auch elektrische Größen angegeben werden: [61]

$$P = E_{rev} I = \dot{n}_{\mathrm{H_2}} \Delta_R G_m(T, p) \tag{2.12}$$

E ist darin die reversible Zellspannung, I der Zellstrom, $\dot{n}_{\mathrm{H_2}}$ der Molenstrom an umgesetztem Brennstoff und $\Delta_R G_m(T, p)$ die, auf ein Mol Brennstoff bezogene, molare freie Reaktionsenthalpie. Die reversible Zellspannung tritt nur im Leerlauf der Zelle auf. Durch Einsetzen von Gleichung (2.11) in Gleichung (2.12) und anschließendem Umformen, kann die reversible Zellspannung berechnet werden: [61]

$$E_{rev} = -\frac{\Delta_R G_m(T, p)}{zF} \tag{2.13}$$

Wird die reversible Zellspannung mit der freien Standardreaktionsenthalpie $\Delta_R G_m(T^0, p^0)$ berechnet, erhält man die sogenannte Standardzellspannung E^0:

$$E^0 = -\frac{\Delta_R G_m(T^0, p^0)}{zF} \tag{2.14}$$

Bei von Standardbedingungen abweichender Temperatur T, d.h. bei Berechnung mit der freien Reaktionsenthalpie $\Delta_R G_m(T, p^0)$, erhält man die reversible Zellspannung mit Temperaturabhängigkeit E_T:

$$E_T = -\frac{\Delta_R G_m(T, p^0)}{zF} \tag{2.15}$$

Gleichung (2.13) kann, durch Einsetzen von Gleichungen (2.7), (2.8), (2.10) und (2.15), zur sogenannten Nernstgleichung umgeformt werden:

$$E_{rev} = E_N = E_T - \frac{R_m T}{zF} \ln \prod_i \left(\frac{p_i}{p^0} \right)^{\nu_{st,i}} \tag{2.16}$$

Die Nernstgleichung beschreibt in der angegebenen Form die Abhängigkeit der reversiblen Zellspannung von Temperatur und Druck, wobei sie den Einfluss der Partialdrücke von Edukten und Produkten mitberücksichtigt. [43]

2.3 Auftretende Verluste

Die durch die Nernstgleichung, siehe Gleichung (2.16), berechnete reversible Zellspannung tritt nur auf, wenn sich die Brennstoffzelle im thermodynamischen Gleichgewicht befindet. Im realen Betrieb einer Zelle treten unvermeidliche, irreversible Verluste auf. Diese können in drei Hauptverlustarten eingeteilt werden: in Aktivierungsverluste V_{Akt}, ohm'sche Verluste V_{Ohm} und Konzentrationsverluste V_{Konz}. Diese können in Form von sogenannten Überspannungen angegeben werden, d.h. als durch die Verluste verursachte Spannungsabfälle. [43] Die im realen Betrieb tatsächlich auftretende Zellspannung E kann dann berechnet werden, indem die Überspannungsverluste von der reversiblen Zellspannung abgezogen werden: [43]

$$E = E_N - V_{Akt} - V_{Ohm} - V_{Konz} \tag{2.17}$$

Darin ist E_N die reversible Zellspannung. V_{Akt} , V_{Ohm} bzw. V_{Konz} sind die durch die Aktivierungsverluste, ohm'sche Verluste bzw. Konzentrationsverluste verursachten Überspannungen. Diese drei Verlustarten und ihre Entstehung werden nachfolgend beschrieben.

2.3.1 Aktivierungsverluste

Befindet sich eine Brennstoffzelle im Leerlauf, sind die Geschwindigkeiten der Hin- und Rückreaktionen gleich. Der Grund dafür ist, dass die freie Reaktionsenthalpie $\Delta_R G(T, p)$ der Reaktion durch Aufbau einer elektrischen Potentialdifferenz, der sogenannten Galvani-Spannung, kompensiert wird. Das gleicht die Aktivierungsenergien der Hin- und Rückreaktion an und die elektrochemische Reaktion befindet sich im elektrochemischen (oder thermodynamischen) Gleichgewicht. Im Gleichgewicht entspricht die Galvani-Spannung der reversiblen Zellspannung. Die Reaktionsgeschwindigkeiten können bei Brennstoffzellen in Form von Stromdichten angegeben werden. Stromdichten sind auf die aktive Fläche der Zelle bezogene Ströme in $\mathrm{mA\,cm^{-2}}$. [43] Im Gleichgewicht gilt: [43]

$$i_1 = i_2 = i_0 \tag{2.18}$$

Die Stromdichte der Hinreaktion i_1 ist gleich der Stromdichte der Rückreaktion i_2. i_0 wird als Austauschstromdichte bezeichnet. Der zwischen den Elektroden der Zelle fließende Nettostrom bzw. die Nettostromdichte i, die Differenz zwischen i_1 und i_2, ist null: [43]

$$i = i_1 - i_2 \tag{2.19}$$

Damit eine Nettostromdichte $i > 0$ fließen kann, muss die Galvani-Spannung reduziert und damit die Zelle aus dem Gleichgewichtszustand gebracht werden. Die Stromdichte der Hinreaktion i_1 wird dann größer als die Stromdichte der Rückreaktion i_2, was analog für die Reaktionsgeschwindigkeiten gilt. Die für die Nettostromdichte $i > 0$ nötige Differenz zwischen reversibler Zellspannung und reduzierter Galvani-Spannung ist die Überspannung der Aktivierungsverluste V_{Akt}. Die Stromdichten der Hin- und Rückreaktion in Gleichung (2.19) können mit V_{Akt} in Zusammenhang gebracht werden. [43] Das Ergebnis ist die Butler-Volmer Gleichung (ohne Herleitung): [43]

$$i = i_0 \left(e^{\alpha z F V_{Akt}/(R_m T)} - e^{-(1-\alpha) z F V_{Akt}/(R_m T)} \right) \tag{2.20}$$

Darin ist i_0 die Austauschstromdichte, α der Transferkoeffizient, z die Elektronenzahl und F die Faraday-Konstante. Weiters ist R_m die allgemeine Gaskonstante und T ist die Temperatur. Die Austauschstromdichte i_0 ist abhängig von den Konzentrationen der Edukte und Produkte an den Elektrodenoberflächen. Dies wird hier aber nicht mehr näher ausgeführt. Der Transferkoeffizient α beschreibt, wie sich die Aktivierungsenergien der Hin- und Rückreaktion zueinander ändern, wenn die Galvani-Spannung reduziert wird. Für die meisten elektrochemische Reaktionen ist $\alpha = 0.2$ bis 0.5. Ist die Reaktion symmetrisch, ist $\alpha = 0.5$. Wird die Butler-Volmer Gleichung nach der Überspannung V_{Akt} aufgelöst, kann V_{Akt} abhängig von der Nettostromdichte i berechnet werden. [43]

2.3.2 Ohm'sche Verluste

Die ohm'schen Verluste in Brennstoffzellen werden durch den Ladungstransport verursacht. Der Ladungstransport erfolgt einerseits durch Elektronenleitung in den Elektroden (sowie Verbindungsleitungen) und andererseits durch Ionenleitung im Elektrolyten. Sowohl Elektroden als auch Elektrolyt setzen den Elektronen bzw. Ionen einen Widerstand entgegen. Die Widerstände verursachen einen zur Stromdichte i proportionalen Spannungsverlust, die Überspannung der ohm'schen Verluste V_{Ohm}. [43] Diese folgt, wie die Bezeichnung schon verrät, dem ohm'schen Gesetz:

$$V_{Ohm} = i(R_{El} + R_{Io}) \tag{2.21}$$

R_{El} ist der Elektronenleitwiderstand und R_{Io} der Ionenleitwiderstand. Im Allgemeinen überwiegt der Einfluss des Ionenleitwiderstands bei Brennstoffzellen ($R_{Io} \gg R_{El}$) [43].

2.3.3 Konzentrationsverluste

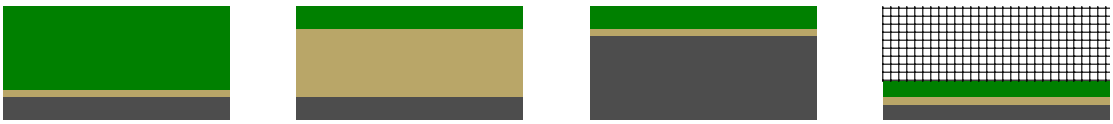
Brennstoffzellen werden mit Gasgemischen versorgt, die gewisse Konzentrationen an Einzelkomponenten aufweisen. Im realen Betrieb weichen die Gaskonzentrationen, die an den katalytisch aktiven Schichten der Elektroden anliegen, davon ab. Ein Grund dafür ist, dass entlang der Strömungskanäle in einer Brennstoffzelle Edukte verbraucht und Produkte gebildet werden. Dies führt zu Verdünnungseffekten: Beispielsweise entsteht bei einer mit Wasserstoff betriebenen SOFC an der Anode Wasserdampf als Produkt, wodurch der Wasserstoff verdünnt wird. Ein weiterer Grund ist, dass der Massentransport durch die Elektroden, hin zu den katalytisch aktiven Schichten, hauptsächlich durch Diffusion bestimmt ist. Dabei ändern sich die Gaskonzentrationen in Richtung der Schichtdicke der Elektroden. Dies führt, im Vergleich zu den Gaskonzentrationen in den Strömungskanälen, zu noch weiter verringerten Edukt- bzw. erhöhten Produktkonzentrationen an den katalytischen Schichten. Wird der Zellstrom erhöht, steigt auch die Reaktionsrate und die beschriebenen Effekte werden verstärkt. [43]

Die Produkte können bei hohen Zellströmen nicht mehr schnell genug von den aktiven Schichten der Elektroden abtransportiert werden. Dadurch wird der Transport der Edukte zu den Schichten hin behindert und die Zelle beginnt zu „verhungern“. Dies verursacht Spannungsverluste, die als Überspannung der Konzentrationsverluste V_{Konz} bezeichnet werden. Besonders bei hohen Stromdichten wird V_{Konz} groß, was dazu führt, dass die Zellspannung bei einer bestimmten Grenzstromdichte bis auf null absinkt. [43]

2.3.4 Aufbau und Materialien

Es gibt verschiedene Möglichkeiten, eine SOFC aufzubauen. Entweder sorgt eine zelleigene Komponente für die mechanische Stabilität (Anoden-, Kathoden- oder Elektrolytschicht) oder eine externe Komponente (zum Beispiel ein Substrat). Wird eine zelleigene Schicht zur Stützung verwendet, muss diese meist dicker ausgeführt werden als die anderen Schichten. [40] Abbildung 2.2 zeigt verschiedene Varianten des Zellaufbaus.

Die Hauptanforderung an den Elektrolyten ist eine hohe Ionenleitfähigkeit bei Betriebstemperatur der Zelle. Gleichzeitig soll die Elektronenleitfähigkeit möglichst gering sein. Weitere Anforderungen sind eine hohe chemische und mechanische Beständigkeit und



(a) ASC (b) ESC (c) CSC (d) SSC

Abbildung 2.2: Verschiedene Varianten des Zellaufbaus: (a) anodengestützte Zelle (ASC), (b) elektrolytgestützte Zelle (ESC), (c) kathodengestützte Zelle (CSC) und (d) substratgestützte Zelle (SSC) – bestehend aus Anode, Elektrolyt, Kathode und Substrat. Entnommen aus [58].

niedrige Material- und Fertigungskosten. Ein Material, dass diese Anforderungen weitgehend erfüllt, ist yttriumstabilisiertes Zirconiumoxid (YSZ). YSZ ist das für SOFCs derzeit am häufigsten verwendete Elektrolytmaterial. Um YSZ herzustellen, wird Zirconiumoxid mit Yttriumoxid dotiert, was die Ionenleitfähigkeit erhöht. [43]

Die Elektroden sind als, mit dem Elektrolyten verbundene, poröse Schichten ausgeführt. Sie sollen eine hohe Porosität, eine hohe katalytische Aktivität und eine hohe elektrische Leitfähigkeit aufweisen. Häufig verwendete Materialien sind Keramiken oder Cermets (ein Verbund aus keramischen und metallischen Werkstoffen). Elektroden können mit einem Zweischicht-Aufbau ausgeführt sein: Die erste, sehr dünne und fein poröse Schicht, ist direkt mit dem Elektrolyten verbunden. Sie ist katalytisch höchst aktiv und sowohl ionen- als auch elektronenleitfähig. Der Zweck dieser Schicht ist es, die Dreiphasengrenze zu vergrößern, an der alle für eine Halbreaktion benötigten Phasen aufeinandertreffen: Gasphase, Elektrolyt (ionenleitfähig) und Elektrode (elektronen- und ionenleitfähig). Die zweite, dickere und grob poröse Schicht, sorgt für mechanische Stabilität und weist eine gute Elektronenleitfähigkeit auf. Durch die grob poröse Struktur ist gewährleistet, dass das Gas leicht zur dünnen, katalytisch aktiven Schicht strömen bzw. diffundieren kann. [43]

Ein Beispiel für ein häufig verwendetes Anodenmaterial, bei Verwendung eines YSZ-Elektrolyts, ist Ni-YSZ Cermet. Ni ist elektronenleitfähig und katalytisch aktiv. YSZ ist ionenleitfähig, vergrößert dadurch die Dreiphasengrenze und sorgt für die benötigte mechanische Stabilität. [43] Ein Beispiel für ein Kathodenmaterial ist $(La, Sr)(Co, Fe)O_3$ (LSCF), das elektronen- und ionenleitfähig ist. Solche Materialien werden als MIEC (mixed ion-electron conductor) bezeichnet. [27]

2.4 Charakterisierungsmethoden

Um die charakteristischen Eigenschaften von Brennstoffzellen zu ermitteln, zum Beispiel die Leistung der Zelle oder die Porosität der Elektroden, gibt es verschiedene Methoden. Diese werden als Charakterisierungsmethoden bezeichnet und lassen sich in sogenannte In-situ und Ex-situ Methoden unterteilen. In-situ bedeutet, dass die Brennstoffzelle zusammengebaut und unter Betriebsbedingungen untersucht wird. Das Betriebsverhalten der gesamten Zelle kann so ermittelt werden. Ex-situ bedeutet hingegen, dass die Brennstoffzelle oder ihre Komponenten in nicht betriebsfähigem Zustand untersucht werden. Meist muss dazu die Brennstoffzelle zerlegt werden. Der Zustand oder das Verhalten der Einzelkomponenten kann so genau untersucht werden, nicht jedoch das Betriebsverhalten der gesamten Zelle. [43]

In diesem Abschnitt sollen zwei für diese Masterarbeit relevante Charakterisierungsmethoden vorgestellt werden: die Messung von Polarisationskurven und die elektrochemische Impedanzspektroskopie. Es handelt sich dabei um zerstörungsfreie, elektrochemische In-situ Methoden.

2.4.1 Messung von Polarisationskurven

Eine Polarisationskurve ist nichts anderes als eine Strom-Spannungs-Kennlinie. Für die Messung einer Polarisationskurve wird der der Zelle entnommene Strom, ausgehend vom Leerlauf, schrittweise erhöht. Die sich in jedem Schritt bzw. Betriebspunkt einstellende Zellspannung wird aufgezeichnet und deren Verlauf ermittelt. Die Spannungsmessung soll dabei nur erfolgen, wenn sich die Zelle im stationären Zustand befindet, d.h. sich Strom und Spannung mit der Zeit nicht ändern. Es muss daher, in jedem Betriebspunkt, mit der Spannungsmessung so lange gewartet werden, bis der stationäre Zustand erreicht ist. [43]

Polarisationskurven erlauben die Bestimmung des stationären Verhaltens und der Leistungsmerkmale einer Zelle. Die Anteile der Verlustmechanismen (Aktivierungs-, Konzentrations- und ohm'sche Verluste, siehe Abschnitt 2.3) an den Gesamtverlusten, können anhand von Polarisationskurven im Allgemeinen nur grob abgeschätzt werden. [43] Abbildung 2.3 zeigt eine gemessene Polarisationskurve einer anodengestützten SOFC. Die Zellspannung E (——) ist auf der linken vertikalen Achse aufgetragen und die Stromdichte i auf der horizontalen Achse. Die Stromdichte ist der auf die chemisch aktive Fläche der Zelle bezogene Strom in $\mathrm{mA\,cm^{-2}}$. Zusätzlich ist die berechnete Leistungsdichte $p = Ei$ (- - -) auf der rechten vertikalen Achse aufgetragen. Die SOFC wurde mit 45 % H_2 / 55 % N_2 (brenngasseitig), 21 % O_2 / 79 % N_2 (sauerstoffseitig) und bei einer Temperatur von 800 °C betrieben.

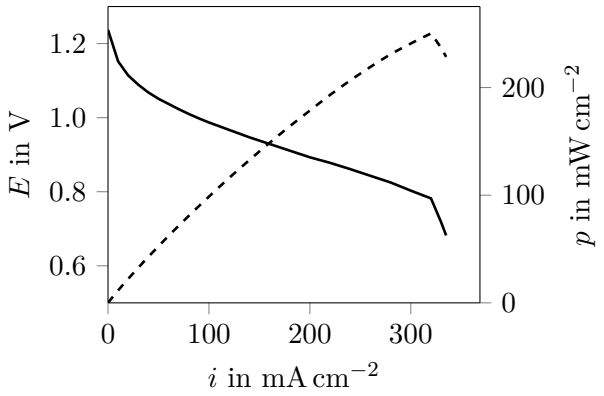


Abbildung 2.3: Polarisationskurve einer SOFC. Die —— Zellspannung E (linke vertikale Achse) ist in Abhängigkeit der Stromdichte i aufgetragen. Zusätzlich ist die - - - Leistungsdichte p der Zelle (rechte vertikale Achse) dargestellt.

Mit steigender Stromdichte, ausgehend vom Leerlauf, fällt die Spannung E zunächst nichtlinear ab. Dieser nichtlineare Spannungsabfall wird in erster Linie durch die Aktivierungsverluste der elektrochemischen Reaktionen an den Zellelektroden verursacht. Mit weiter steigender Stromdichte nehmen die Aktivierungsverluste kaum mehr zu und der Spannungsabfall wird zunehmend linear. Der Verlauf der Kurve ist in diesem Bereich durch die ohm'schen Verluste geprägt, die hauptsächlich durch den Ionenleitwiderstand des Elektrolyten entstehen. Bei hohen Stromdichten kann der Zelle durch die zugrundeliegenden konvektiven und diffusiven Transportvorgänge, die nötige Menge an Produkten nicht mehr zugeführt werden. Die Konzentrationsverluste steigen stark und die Spannung sinkt abrupt. [3] Die Messung in Abbildung 2.3 wurde bei einer Stromdichte i von $335\,\mathrm{mA\,cm^{-2}}$ beendet. Bei weiterer Erhöhung der Stromdichte i, wäre die Zellspannung E weiter gegen null gesunken. Der unstetige Knick im Verlauf von E entstand durch die Messauflösung. Die Leistungsdichte $p = Ei$ weist, aufgrund der mit steigender Stromdichte stetig abfallenden Zellspannung, ein Maximum auf.

2.4.2 Elektrochemische Impedanzspektroskopie

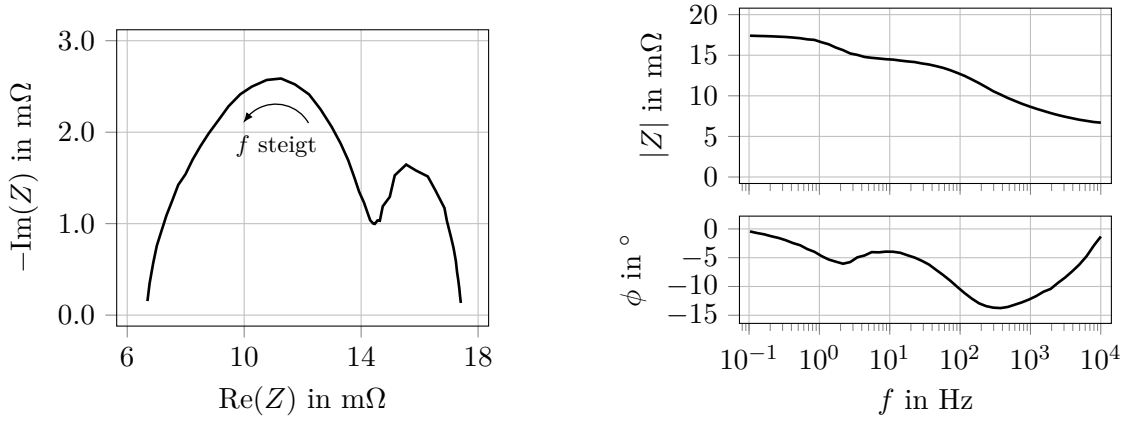
Bei Anwendung der elektrochemischen Impedanzspektroskopie (EIS) wird die Impedanz (der Wechselstromwiderstand) der Brennstoffzelle bestimmt. Ausgangspunkt ist ein stationärer Betriebspunkt mit konstanter Stromdichte i und Zellspannung E. Die Zelle wird, durch Überlagerung einer harmonischen Wechselstromdichte Δi oder einer harmonischen Wechselspannung ΔE, angeregt. Je nach Art der Anregung reagiert die Zelle mit einer Wechselspannungsantwort ΔE (bei Anregung durch Δi), oder mit einer Wechselstromantwort Δi (bei Anregung durch ΔE). [26] Beide Größen werden gemessen

und die Impedanz Z ermittelt: [26]

$$Z(j\omega) = \frac{\Delta E(j\omega)}{\Delta i(j\omega)} \tag{2.22}$$

Die Impedanz wird auf diese Weise für verschiedene Anregungsfrequenzen ermittelt ($\omega = 2\pi f$ mit der Frequenz f) und die Messungen ergeben, zusammengefasst, ein Impedanzspektrum. Eine nach Gleichung (2.22) durchgeführte Impedanzberechnung ist nur gültig, wenn ein linearer Zusammenhang zwischen Anregung und Antwort des untersuchten Systems besteht. Bei Brennstoffzellen ist der Zusammenhang, aufgrund zahlreicher Transport- und elektrochemischer Prozesse, nichtlinear. Die Amplitude der Anregung muss deshalb so klein gewählt werden, dass die Zelle innerhalb eines näherungsweise linearen Bereichs agiert. [26]

Die EIS erlaubt die Untersuchung und Quantifizierung der einzelnen Verlustmechanismen (Aktivierungs-, Konzentrations- und ohm'sche Verluste, siehe Abschnitt 2.3). Dazu wird die Impedanz durch geeignete Ersatzschaltbilder modelliert, was die Zuteilung der Verluste zu den einzelnen Mechanismen erleichtert. Ohne Einsatz einer Referenzelektrode bei den Messungen, ist die Aufteilung der Verluste auf Prozesse an Anode und Kathode jedoch aufwendig. Die EIS eignet sich auch zur Online-Überwachung von Brennstoffzellen, um Degradation (eine Verschlechterung der Zellleistung durch verschiedene Ablagerungs- und Alterungsprozesse) vorzeitig zu erkennen. [43, 56]



(a) Nyquist-Diagramm

(b) Bode-Diagramm

Abbildung 2.4: Darstellung eines Impedanzspektrums als Nyquist- und Bode-Diagramm.

Zwei häufig anzutreffende Varianten um Impedanzspektren darzustellen, sind das Nyquist- und das Bode-Diagramm. Abbildung 2.4 zeigt ein Impedanzspektrum einer SOFC in

beiden Varianten. Die SOFC wurde mit 60 % H_2 / 40 % N_2 (brenngasseitig), 21 % O_2 / 79 % N_2 (sauerstoffseitig) und bei einer Temperatur von 700 °C betrieben. Die Gleichstromdichte im statischen Betriebspunkt betrug $100\,\mathrm{mA\,cm^{-2}}$. Die stromgesteuerte, sinusförmige Anregung erfolgte mit einer Amplitude von 4 % des Gleichstroms, d.h. mit $4\,\mathrm{mA\,cm^{-2}}$, die Anregungsfrequenz f wurde dabei von 0.1 Hz bis 10 000 Hz moduliert. Zur Darstellung des Nyquist-Diagramms, siehe Abbildung 2.4a, wird der negative Imaginärteil der Impedanz $-\mathrm{Im}(Z)$, für verschiedene Anregungsfrequenzen f, über dem Realteil der Impedanz $\mathrm{Re}(Z)$ aufgetragen. Der direkte Zusammenhang der Kurve mit der Frequenz f geht dabei verloren. Die Frequenz f nimmt jedoch stetig in der Richtung des eingezeichneten, gebogenen Pfeils zu. Die zweite Variante, das Bode-Diagramm in Abbildung 2.4b, zeigt den Betrag der Impedanz $|Z|$ über der logarithmisch dargestellten Frequenz f. Meist wird auch die Phasenverschiebung ϕ, wie hier im unteren Teil des Diagramms, mit dargestellt. ϕ beschreibt in diesem Fall die auftretende Phasenverschiebung zwischen Stromanregung und Spannungsantwort.

3 Grundlagen künstlicher neuronaler Netze

Künstliche neuronale Netze bilden das Verhalten der Neuronen im menschlichen Gehirn vereinfacht nach. Sie sind neben Entscheidungsbäumen und Support Vector Machines eine von vielen Methoden des sogenannten maschinellen Lernens. Nach [49] verleiht maschinelles Lernen Computersystemen die Fähigkeit zu lernen, ohne explizit programmiert zu werden[1]. „Lernen“ bedeutet in diesem Kontext, dass ein Computersystem eine Aufgabe durch Erfahrung lernt, wenn sich dessen Leistung in der Aufgabe mit steigender Erfahrung verbessert [41].

Maschinelles Lernen weist einige Vorteile gegenüber der Verwendung von herkömmlichen Algorithmen auf. Maschinenlernmethoden eignen sich besonders zur Lösung komplizierter und komplexer Probleme. Ändert sich die Problemstellung geringfügig, können Maschinenlernmethoden, meist mit geringem Aufwand, darauf angepasst werden. Nicht zuletzt können im Einsatz befindliche Maschinenlernmethoden selbstständig weiterlernen, falls sie mit neuen Daten konfrontiert werden. Herkömmliche Algorithmen hingegen stoßen, mit steigender Kompliziertheit und Komplexität des Problems, schnell an ihre Grenzen: der Entwicklungsaufwand wächst enorm. Bereits kleine Änderungen der Problemstellung können eine Neuentwicklung des Algorithmus notwendig machen. Schließlich muss die Anpassung herkömmlicher, im Einsatz befindlicher Algorithmen, durch manuelles Eingreifen erfolgen. [19] Als Beispiel für oben angeführte Punkte sei die Erkennung handschriftlicher Zahlen genannt. Einen Algorithmus zu programmieren, der dies zu leisten vermag, ist per se schon keine triviale Aufgabe. Soll der Algorithmus dann noch so abgeändert werden, dass er auch mit japanischen Schriftzeichen geschriebene Zahlen erkennt, werden voraussichtlich weitreichende Änderungen im Programmcode notwendig. Im Vergleich dazu sind, aus der Sicht des maschinellen Lernens, beide Problemstellungen weitgehend identisch.

Die Motivation, gerade künstliche neuronale Netze für maschinelles Lernen zu verwenden, findet sich in der Leistungsfähigkeit des menschlichen Gehirns: Menschen lernen durch Erfahrung und können ihr so gewonnenes Wissen hochgradig abstrahieren und auf neue, unbekannte Problemstellungen anwenden. Da das menschliche Gehirn aber aufgrund seiner Komplexität derzeit nur in Bruchstücken modelliert werden kann, erreichen künstliche neuronale Netze keine vergleichbar hohen Abstraktionsgrade – ein Netz wird daher

[1] Für die A. Samuel oft zugeschriebene Definition „Machine learning: field of study that gives computers the ability to learn without being explicitly programmed.“ wurde keine belastbare Quelle gefunden, die Aussage ist aber sinngemäß in der angegebenen Quelle enthalten.

zumeist für eine bestimmte, im Voraus bekannte Aufgabe erstellt. Beispiele für Aufgaben bzw. Problemstellungen sind: [22]

- **Regression**, d.h. die Vorhersage eines numerischen Wertes (oder mehrerer Werte) abhängig von den Eingaben. Ein Beispiel ist die Vorhersage eines Aktienkurses abhängig von den Börsendaten.
- **Klassifizierung**, d.h. die Einteilung der Eingaben in Klassen. Ein Beispiel ist die Objekterkennung anhand von Fotos.
- **Rauschunterdrückung**, d.h. die Ausgabe eines rauschfreien Datensatzes bei Eingabe eines rauschbehafteten Datensatzes. Ein Beispiel ist die Rauschunterdrückung von Digitalkameras bei Nachtaufnahmen.
- **Anomalie- bzw. Ausreißererkennung**, d.h. das Erkennen unüblicher Eingaben. Ein Beispiel ist die Erkennung von unüblichen Zahlungsvorgängen bei Kreditkartenbetrug.

Der Ablauf bei der Erstellung ist dabei meistens wie folgt: Zuerst wird die Topologie des neuronalen Netzes festgelegt, danach folgt der Lern- bzw. Trainingsvorgang: Anhand von Trainingsbeispielen (in einem Datensatz), lernt das Netz dabei, ein Verhalten oder ein Muster zu erkennen. Abschließend wird getestet, ob das Netzwerk auch für neue und unbekannte Daten zufriedenstellende Vorhersagen trifft.

Den großen, öffentlichkeitswirksamen Durchbruch erzielten neuronale Netze erst in den Jahren von 2009 bis 2012. Ein Team um Schmidhuber J. gewann in dieser Zeit insgesamt 8 bekannte, internationale Maschinenlernwettbewerbe. Kategorien waren zum Beispiel die Erkennung zusammenhängender arabischer Handschriften (2009), die optische Erkennung von Verkehrsschildern (2011) und die Mitosedetektion in histologischen Bildern zur Unterstützung von Brustkrebsprognosen (2012). Die bis dahin meistverwendeten, konkurrierenden Methoden wurden in vielen Fällen deutlich übertroffen. In einigen Fällen erreichten die neuronalen Netze sogar eine höhere Vorhersagegenauigkeit als die besten menschlichen Probanden. Führende, amerikanische IT-Konzerne zeigten daraufhin großes Interesse an der Technologie und begannen sie einzusetzen. [4]

Dieses Kapitel widmet sich den Grundlagen künstlicher neuronaler Netze. Aufgrund der Aufgabenstellung der Masterarbeit liegt der Schwerpunkt dabei auf neuronalen Netzen für Regression. Zu Beginn werden kurz die biologischen Grundlagen behandelt. Danach wird, ausgehend von einem einzelnen künstlichen Neuron, die Funktionsweise künstlicher neuronaler Netze und deren Training beschrieben. Abschließend geht es um die äußeren Parameter künstlicher neuronaler Netze, die sogenannten Hyperparameter und deren Optimierung.

3.1 Biologische Neuronen als Vorbild

Biologische Neuronen sind erregbare Nervenzellen, die Signale weiterleiten können und im Nervensystem komplexe Netzwerke bilden [53]. Im menschlichen Gehirn gibt es viele verschiedene Arten von Neuronen mit unterschiedlichem Verhalten, deren grundlegende Funktionsweise jedoch durch ein vereinfachtes Standardneuron abgebildet werden kann. Abbildung 3.1 zeigt eine mögliche Variante eines Standardneurons mit dessen Hauptbestandteilen. [34]

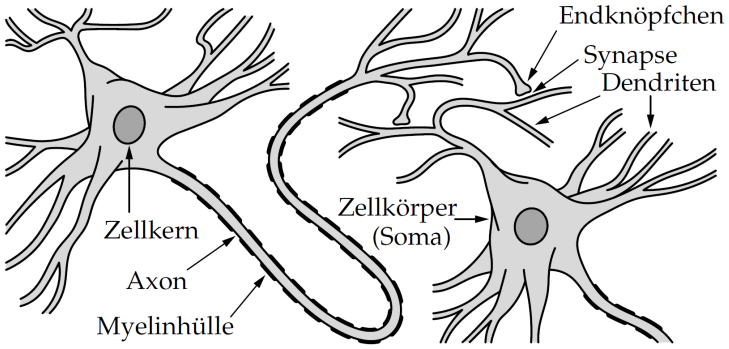


Abbildung 3.1: Vereinfachte biologische Standardneuronen, aus [34].

Das hier beschriebene Standardneuron besitzt einen Zellkörper, der einen Zellkern enthält. Am Zellkörper befinden sich zudem Fortsätze, die Dendriten und das Axon. Der Zellkörper und die Fortsätze werden außerdem von einer Zellmembran umspannt. Das Axon verästelt sich an seinem Ende zu einer Vielzahl an Endknöpfchen, die an verschiedenen Orten Schnittstellen zu den Dendriten anderer Neuronen bilden. Diese Schnittstellen werden Synapsen genannt. [34]

Ein Neuron erhält ankommende Signale an den Dendriten. Die Übertragung erfolgt dabei über chemische Botenstoffe, sogenannte Neurotransmitter, die von den Endknöpfchen anderer Neuronen an Rezeptoren an den Dendriten diffundieren und dort Ladungsverschiebungen verursachen. Abhängig von der Art der Neurotransmitter und der Rezeptoren wird dadurch die Potentialdifferenz zwischen Innenseite und Außenseite der Zellmembran erhöht oder gesenkt[1]. Eine Verschiebung der Potentialdifferenz zu weniger negativen Werten entspricht dabei einer Erregung des Neurons. Überschreitet die Potentialdifferenz am Ansatz des Axons einen Schwellenwert, wird ein Signal an das Axon weitergeleitet: das sogenannte Aktionspotential. Dieses hat zur Folge, dass an den Endknöpfchen des Axons

[1] Im Ruhezustand beträgt das sogenannte Ruhepotential $-50\,\mathrm{mV}$ bis $-100\,\mathrm{mV}$, wobei das Zellinnere gegenüber dem Äußeren negativ geladen ist.

wiederum Neurotransmitter an nachgeschaltete Neuronen ausgeschüttet werden. [34, 53]

Nach Auslösen des Aktionspotentials wird die Potentialdifferenz der Zellmembran meist innerhalb weniger Millisekunden wieder auf ihren Ruhewert abgebaut. Das Neuron kann aber während dieser Zeit nicht oder nur sehr schwer dazu gebracht werden, ein weiteres Aktionspotential auszulösen – das bedeutet, dass die Weiterleitung der Signale durch neuronale Netzwerke impulsartig erfolgt. An dieser Stelle sei angemerkt, dass künstliche neuronale Netze das Impulsverhalten ihres biologischen Vorbilds nicht abbilden. Zwei zentrale Eigenschaften der Synapsen sind, dass sie Signale im Allgemeinen nur in eine Richtung übertragen und in ihren Übertragungseigenschaften modifiziert werden können. [34, 53]

In einem menschlichen Gehirn befinden sich nach aktuellen Schätzungen circa 86 Milliarden[1] Neuronen [6], die miteinander vernetzt sind. Ein einzelnes Neuron hat dabei oft mehrere tausend Verbindungen zu anderen Neuronen [53]. Ein Neuron im Neocortex hat zum Beispiel durchschnittlich etwa 7000 Synapsen [44].

3.2 Vorwärtsverknüpfte künstliche neuronale Netze

Vorwärtsverknüpfte neuronale Netze (engl. feedforward neural networks) sind künstliche neuronale Netze, deren Neuronen in Schichten eingeteilt sind. Die erste Schicht des Netzwerkes wird als Eingabeschicht bezeichnet, danach folgt eine beliebige Anzahl verdeckter Schichten und abschließend die Ausgabeschicht. Vorwärtsverknüpft bedeutet in diesem Zusammenhang, dass ausgehende Verbindungen von Neuronen nur an Neuronen einer nachfolgenden Schicht (in Richtung Ausgabeschicht) anknüpfen dürfen. [33] Neuronale Netze, die aus zwei oder mehr Schichten bestehen, also zumindest aus Eingabe- und Ausgabeschicht, werden dabei als „mehrlagig“ bezeichnet. Ein Beispiel für den Schichtaufbau eines vorwärtsverknüpften Netzwerkes mit vier Schichten, d.h. mit zwei verdeckten Schichten, zeigt Abbildung 3.2.

3.2.1 Das künstliche Neuron

Die Grundbausteine vorwärtsverknüpfter neuronaler Netze sind künstliche Neuronen. Bereits 1958 stellte Rosenblatt das Perzeptron vor [48], ein Schwellenwertelement, das in seinen wesentlichen Eigenschaften den in modernen neuronalen Netzen verwendeten Neuronen gleicht. Im weiteren Sinne wird der Begriff Perzeptron auch für verschiedene Varianten von ein- oder mehrlagigen neuronalen Netzen verwendet, an dieser Stelle sei

[1] Nach [5] hat das menschliche Gehirn davon abweichend nur circa 67 Milliarden Neuronen

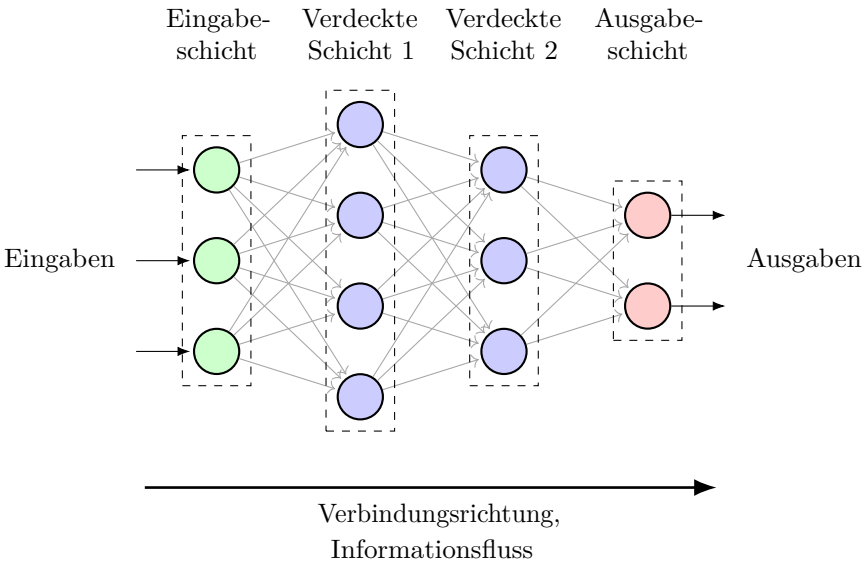


Abbildung 3.2: Schichtaufbau eines mehrlagigen vorwärtsverknüpften Netzwerkes mit vier Schichten.

damit aber das einzelne Neuron gemeint. Dargestellt ist das Perzeptron in Abbildung 3.3. [19]

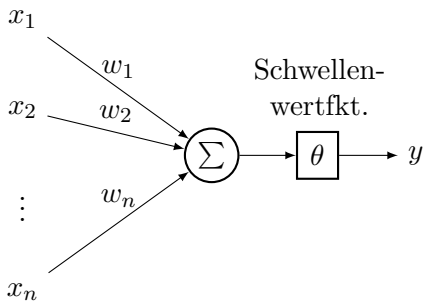


Abbildung 3.3: Das Perzeptron als einzelnes Neuron, nach [19].

Die Ausgabe eines Perzeptrons mit n binären Eingabegrößen berechnet sich wie folgt [42]:

$$y(x) = \begin{cases} 1, & \text{wenn } \sum_{j=1}^{n} w_j x_j > \mathit{Schwellenwert} \\ 0, & \text{sonst} \end{cases} \tag{3.1}$$

Jede Eingabegröße x_j des Perzeptrons hat einen zugehörigen Gewichtungsfaktor w_j. Überschreitet die gewichtete Summe der Eingabegrößen $\sum_{j=1}^{n} w_j x_j$ einen Schwellenwert,

wird die Ausgabe y des Perzeptrons gleich 1, sonst ist die Ausgabe gleich 0. Wird das Perzeptron zum Beispiel als binärer Klassifikator eingesetzt, kann es basierend auf n Eingabegrößen eine Einteilung in 2 Klassen vornehmen (Klasse 0 oder 1). Es kann von Vorteil sein, den Schwellenwert durch Umformen auf die andere Seite der Gleichung zu bringen [42]:

$$y(x) = \begin{cases} 1, & \text{wenn } \sum_{j=1}^{n} w_j x_j + b > 0 \\ 0, & \text{sonst} \end{cases} \tag{3.2}$$

$$\text{mit } b \equiv -\mathit{Schwellenwert}$$

b wird dann als Bias bezeichnet und ist ein Maß dafür, wie leicht das Perzeptron „feuert“, d.h. den Wert 1 ausgibt [42]. Der Einfluss des Bias auf die Ausgabe des Perzeptrons ist in Abbildung 3.4 dargestellt: Wird der Bias zum Beispiel von 0 (——) auf 0.3 (-·-·-) erhöht, feuert das Neuron bereits bei einer niedrigeren gewichteten Summe $\sum_{j=1}^{n} w_j x_j + b$.

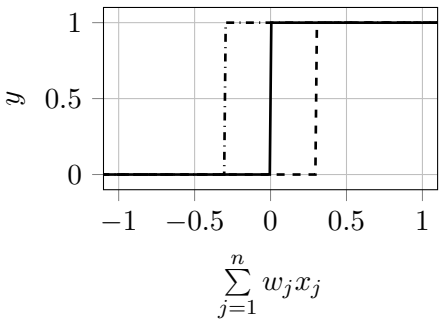
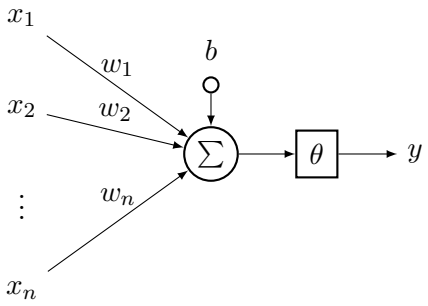


Abbildung 3.4: Die Ausgabe y eines Perzeptrons dargestellt über der gewichteten Summe der Eingangsgrößen bei variiertem Bias - - - $b = -0.3$, —— $b = 0$ und -·-·- $b = 0.3$.

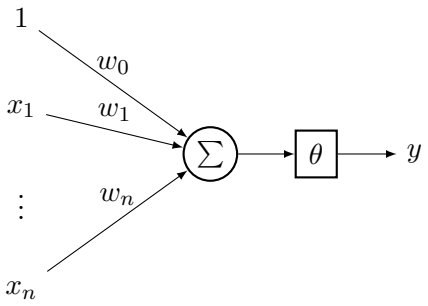
Da neuronale Netze durch Anpassung ihrer Gewichte trainiert werden, worauf später noch näher eingegangen wird, wird der Bias für Berechnungen oft als Gewicht behandelt [33]: dann ist $b = w_0 x_0$, wobei x_0 konstant gesetzt wird (zum Beispiel auf 1):

$$\sum_{j=1}^{n} w_j x_j + \underbrace{b}_{w_0 x_0} = \sum_{j=0}^{n} w_j x_j$$

Diese Umwandlung des Bias in ein Gewicht ist zur Verdeutlichung grafisch in Abbildung 3.5 dargestellt. Abbildung 3.5a zeigt das Perzeptron darin mit separatem Bias b und Abbildung 3.5b mit einem dem Bias rechnerisch äquivalentem Gewicht w_0.



(a) Perzeptron mit dem Bias b, nach [19].

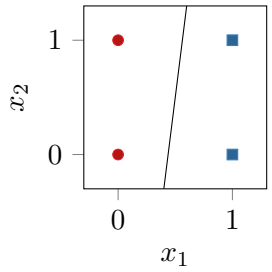


(b) Perzeptron mit dem Bias rechnerisch äquivalentem Gewicht w_0, nach [41].

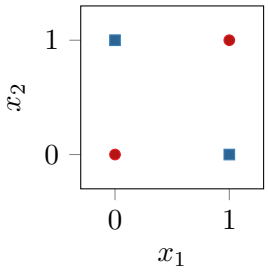
Abbildung 3.5: Umwandlung des Bias b in ein Gewicht w_0 bei einem Perzeptron.

3.2.2 Zusammengesetzte Neuronen als neuronales Netz

Die Anwendungsmöglichkeiten von einzelnen Perzeptronen sind begrenzt. Ein einzelnes Perzeptron kann nur linear separable, boolsche Funktionen berechnen. Im Sinne der Klassifikation bedeutet das, dass das Perzeptron die Punkte zweier Klassen 0 und 1 im Vektorraum seiner Eingaben nur dann richtig einteilen kann, wenn diese durch eine lineare Funktion trennbar sind. Grafisch veranschaulicht ist der Begriff der linearen Separierbarkeit in Abbildung 3.6. [34]



(a) Linear separabel.



(b) Linear nicht separabel.

Abbildung 3.6: Grafische Darstellung des Begriffs der linearen Separierbarkeit für eine Einteilung in ● Klasse 0 und ■ Klasse 1 im $\Re^2$, nach [34].

In Abbildung 3.6a lassen sich die Punkte der Klassen 0 (●) und 1 (■) durch eine lineare Funktion, d.h. im $\Re^2$ durch eine Gerade, trennen. In Abbildung 3.6b ist dies nicht möglich. Mit zunehmender Anzahl der binären Eingaben, also steigender Dimensionalität des Vektorraums, sinkt der Anteil der linear separablen boolschen Funktionen stark. Das Perzeptron ist dann als Approximator allgemeiner boolscher Funktionen unbrauchbar. [34]

Um diese Einschränkung zu umgehen, werden Perzeptronen zu Netzwerken verbunden. Mehrschichtige Perzeptronennetzwerke können, bei genügend hoher Neuronenanzahl, alle für eine beliebige Anzahl an Eingängen existierenden boolschen Funktionen berechnen [34]. Dargestellt ist solch ein Netzwerk in Abbildung 3.7. Die Eingaben x_1,...,x_E des Netzwerks werden von der Eingabeschicht an die erste verdeckte Schicht weitergeleitet. Die Information fließt durch die Schichten, bis die Ausgabeschicht als abschließende Schicht die Ausgaben y_1,...,y_A berechnet. Angemerkt sei, dass das dargestellte Netzwerk dicht verbunden ist. Das bedeutet, dass jedes Neuron einer Schicht alle gewichteten Ausgaben der Neuronen der vorgeschalteten Schicht als Eingaben erhält. Nicht dicht verbundene Netztypen, beispielsweise Convolutional Neural Networks (CNN), werden in erster Linie zur Bilderkennung eingesetzt und deshalb in dieser Arbeit nicht behandelt.

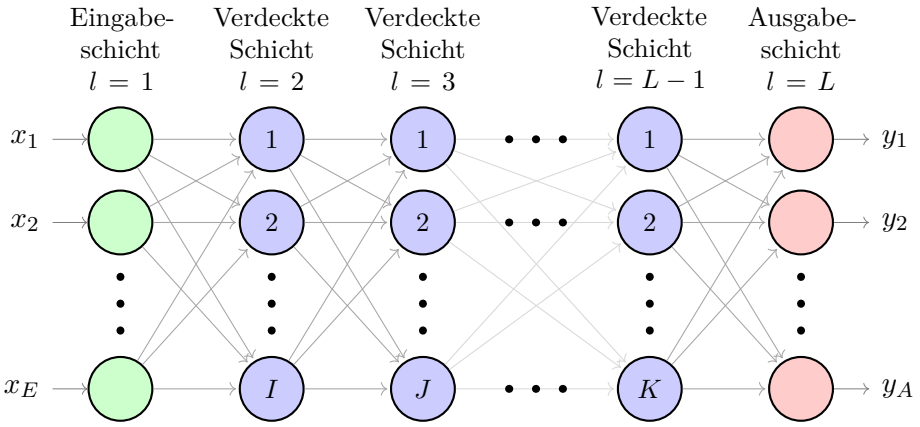


Abbildung 3.7: Darstellung eines dicht verbundenen, mehrschichtigen künstlichen neuronalen Netzes mit E Eingaben, A Ausgaben und den Neuronenanzahlen $I, J, ..., K$ in $L - 2$ verdeckten Schichten, nach [34].

Nachfolgend wird die Berechnung eines Neurons, das Teil einer Schicht des Netzwerks in Abbildung 3.7 ist, näher betrachtet. Die Ausgabe eines Neurons wird dabei als Aktivierung bezeichnet. Die Aktivierung a_k^l des k-ten Neurons in Schicht l berechnet sich für J Neuronen in Schicht $l - 1$ wie folgt [42]:

$$a_k^l = \theta \left(\sum_{j=1}^{J} w_{kj}^l a_j^{l-1} + b_k^l \right) \tag{3.3}$$

Dabei ist θ die Schwellenwertfunktion, w_{kj}^l das Gewicht vom j-ten Neuron in Schicht $l - 1$ zum k-ten Neuron in Schicht l, a_j^{l-1} die Aktivierung des j-ten Neurons in Schicht $l - 1$ und b_k^l der Bias des k-ten Neurons in Schicht l.

Anstatt die Berechnung für einzelne Neuronen anzuschreiben, kann die Berechnung auch für alle Neuronen einer Schicht unter Verwendung einer Matrix-Vektor-Multiplikation

zusammengefasst werden:

$$\boldsymbol{a}^l = \boldsymbol{\theta}\left(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l\right) \tag{3.4}$$

$$\text{mit } \boldsymbol{W}^l = \begin{pmatrix} w_{11}^l & w_{12}^l & \cdots & w_{1J}^l \\ w_{21}^l & w_{22}^l & \cdots & w_{2J}^l \\ \vdots & \vdots & \ddots & \vdots \\ w_{K1}^l & w_{K2}^l & \cdots & w_{KJ}^l \end{pmatrix}, \boldsymbol{a}^{l-1} = \begin{pmatrix} a_1^{l-1} \\ a_2^{l-1} \\ \vdots \\ a_J^{l-1} \end{pmatrix} \text{ und } \boldsymbol{b}^l = \begin{pmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_K^l \end{pmatrix}$$

Dann ist $\boldsymbol{\theta}$ die (vektorisierte) Schwellenwertfunktion, $\boldsymbol{W}^l$ die Gewichtsmatrix eingehend in Schicht l, $\boldsymbol{a}^{l-1}$ Aktivierungsvektor der Neuronen in Schicht $l-1$ und $\boldsymbol{b}^l$ der Biasvektor in Schicht l. Dazu seien noch einige Anmerkungen angeführt: Die Neuronen der Eingabeschicht geben in der gewählten Darstellung die Eingabewerte unverändert weiter, d.h. die Berechnung nach Gleichung (3.4) wird für jede Schicht $l = 2, ..., L$ ausgeführt, aber nicht für $l = 1$. Der Aktivierungsvektor $\boldsymbol{a}^l$ der Ausgabeschicht $l = L$ entspricht außerdem dem Ausgabevektor $\boldsymbol{y}$ des Netzwerks.

Die bisher beschriebenen neuronalen Netzwerke können nur boolsche Funktionen berechnen [34]. Um die Netzwerke zur Approximation reellwertiger Funktionen einsetzen zu können, sind zwei Änderungen notwendig: [34]

- Die Eingabegrößen x_1,...,x_E werden reell.
- Die Schwellenwertfunktion der Ausgabeneuronen wird durch die Identität ersetzt.

Ein Netzwerk mit diesem Aufbau und einer verdeckten Schicht kann laut dem universellen Approximationstheorem (UAT) jede beliebige, mehrdimensionale stetige Funktion beliebig genau annähern, vorausgesetzt es besitzt eine genügend hohe Anzahl an verdeckten Neuronen [34, 15]. Das ist im Falle mehrerer Ausgabeneuronen auch für vektorwertige Funktionen gültig [42]. Obwohl diese Eigenschaft bemerkenswert ist, da sie die Existenz einer beliebig genauen Näherung garantiert, besagt sie nicht, dass diese Näherung mit den verfügbaren Methoden auch gefunden werden kann. Weiters ist Folgendes zu beachten: Das UAT gilt bereits für Netzwerke mit nur einer verdeckten Schicht. Trotzdem kann die Verwendung mehrerer verdeckter Schichten Vorteile hinsichtlich des Trainings bieten, oder weniger Neuronen in den verdeckten Schichten erforderlich machen. [34]

Nun soll wieder das zuvor beschriebene Netzwerk betrachtet werden: dieses nähert eine zu approximierende Funktion stufenförmig an. Um das zu zeigen, wurde eine beliebige Funktion $y = x^2 - 0.18x^3$ gewählt. Die Gewichte des Ausgabeneurons des Netzwerks wurden dazu passend festgelegt. Das Ausgabeneuron und die damit berechnete Näherung der Funktion sind in Abbildung 3.8 dargestellt. Die Anzahl der Eingänge, die Gewichte und der Bias des Ausgabeneurons sind dabei fest vorgegeben. An den Eingängen des Ausgabeneurons können nur die Werte 0 oder 1 anliegen. Feuert ein, sich vor dem Ausgabeneuron befindliches, verdecktes Neuron, dann erhöht es die gewichtete Summe des Ausgabeneurons. Dies verursacht einen Sprung in der Ausgabe des Netzwerks. [34]

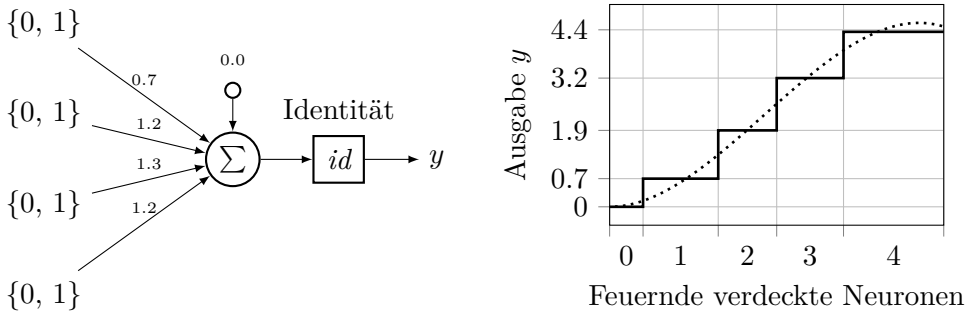


Abbildung 3.8: Exemplarisches Ausgabeneuron eines Netzwerks mit Eingaben aus vorgeschalteten Schwellenwertelementen und dessen —— stufenförmige Näherung der Funktion ····· $y = x^2 - 0.18x^3$, nach [34].

Auch bei Anpassung der Gewichte und des Bias kann diese Näherung nicht wesentlich genauer werden, da die Anzahl der Stufen mit der Anzahl der verdeckten Neuronen vorgegeben ist. Abhängig von der Form der zu nähernden Funktion kann deshalb eine große Neuronenanzahl erforderlich sein, um eine gute Näherung zu erhalten. Eine mögliche Abhilfe ist es, die Schwellenwertfunktionen der verdeckten Schichten durch andere Funktionen zu ersetzen. [34] Die Schwellenwertfunktion $\boldsymbol{\theta}$ einer Schicht wird dann durch eine (allgemeine) Aktivierungsfunktion $\boldsymbol{\sigma}^l$ ersetzt und Gleichung (3.4) wird zu:

$$\boldsymbol{a}^l = \boldsymbol{\sigma}^l \left(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \right) \tag{3.5}$$

Darin ist $\boldsymbol{\sigma}^l$ die (vektorisierte) allgemeine Aktivierungsfunktion der Schicht l. Das universelle Approximationstheorem behält bei Verwendung von Aktivierungsfunktionen, die stetig und keine Polynome (also auch nicht konstant oder linear) sind, seine Gültigkeit [34, 45]. Es gibt also eine große Anzahl möglicher Aktivierungsfunktionen, die in neuronalen Netzwerken verwendet werden können. Auf die gängigen Varianten und ihre Vor- und Nachteile wird in Abschnitt 3.3.2 näher eingegangen. Ein Beispiel sei an dieser Stelle die Aktivierungsfunktion der sogenannten Rectified Linear Unit (ReLU) $\sigma(z) = \max(0, z)$, dargestellt in Abbildung 3.9. Angemerkt sei, dass diese Funktion trotz ihrer irreführenden Bezeichnung im Punkt $z = 0$ eine Nichtlinearität aufweist.

Zur Verdeutlichung des Vorteils, der mit einem Tausch der Aktivierungsfunktion gewonnen werden kann, wird ein einzelnes Neuron mit ReLU-Aktivierungsfunktion mit genau einem Eingang x, einem zugehörigen Gewicht w und einem Bias b betrachtet. Abbildung 3.10 zeigt, wie sich die Aktivierung des Neurons bei Variation von w und b ändert. Wird das Gewicht vergrößert, siehe Abbildung 3.10a, erhöht sich die Steigung der Aktivierung (und umgekehrt). Bei negativen Gewichten fällt die Gerade ab, anstatt anzusteigen. Wird der Bias erhöht, siehe Abbildung 3.10b, verschiebt sich die Gerade zu kleineren Werten

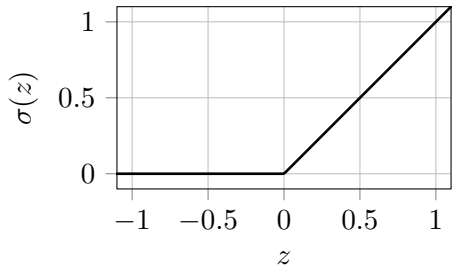
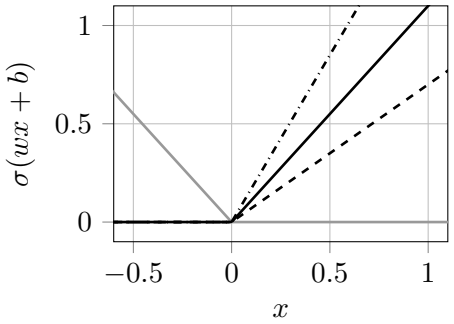
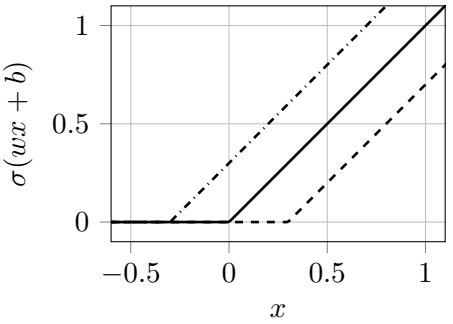


Abbildung 3.9: ReLU-Aktivierungsfunktion $\sigma(z) = \max(0, z)$

von x (und umgekehrt). Netzwerke aus ReLUs lassen sich somit stückweise linear an eine vorgegebene Funktion anpassen.



(a) Variation des Gewichts mit - - - $w = 0.7$, —— $w = 1.1$, -·-·- $w = 1.7$ und —— $w = -1.1$ bei konstantem Bias $b = 0.0$.



(b) Variation des Bias mit - - - $b = -0.3$, —— $b = 0.0$ und -·-·- $b = 0.3$ bei konstantem Gewicht $w = 1.0$.

Abbildung 3.10: Einfluss des Gewichts w und des Bias b auf die Aktivierung eines ReLU-Neurons.

Ein Netzwerk mit einer verdeckten Schicht, die 2 ReLUs enthält, könnte die zuvor betrachtete Funktion $y = x^2 - 0.18x^3$ beispielsweise wie in Abbildung 3.11 dargestellt approximieren. Ein Netzwerk aus Schwellenwertelementen würde eine ungleich höhere Neuronenanzahl benötigen, um eine ähnlich genaue Näherung zu ermöglichen. Damit erklärt sich die Motivation, hinsichtlich der Aktivierungsfunktion vom biologischen Vorbild abzuweichen – menschliche Neuronen sind im Allgemeinen Schwellenwertelemente. Die Wahl der Aktivierungsfunktionen hat außerdem einen großen Einfluss darauf, ob es gelingt, das Netzwerk erfolgreich zu trainieren.

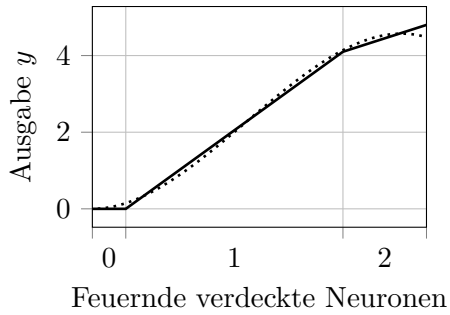


Abbildung 3.11: Darstellung einer —— exemplarischen Näherung der Funktion ····· $y = x^2 - 0.18x^3$ durch ein Netzwerk mit 2 ReLUs in der verdeckten Schicht, nach [34].

3.2.3 Training neuronaler Netze

Grundsätzlich lernen neuronale Netze durch Anpassung ihrer Topologie und ihrer Parameter. Das kann zum Beispiel durch

- Hinzufügen neuer oder Entfernen bestehender Verbindungen
- Hinzufügen oder Entfernen von Neuronen
- Anpassen der Aktivierungsfunktionen
- Anpassen der Gewichte
- Anpassen der Bias

geschehen. [33] Parameter wie die Gewichte und die Bias werden auch als innere Parameter eines Netzwerks bezeichnet. Die Optimierung der inneren Parameter lässt sich einfach mathematisch beschreiben und als Algorithmus umsetzen – sie werden deswegen während des eigentlichen Trainingsvorgangs angepasst. Parameter wie die Neuronenanzahl oder die Art der Aktivierungsfunktion, werden auch als äußere Parameter (oder Hyperparameter) bezeichnet. Ihre Optimierung erfolgt meistens außerhalb des Trainingsvorganges in einer Schleife, worauf in Abschnitt 3.3 näher eingegangen wird. Im Folgenden soll die Optimierung der inneren Netzparameter näher behandelt werden.

Ein neuronales Netzwerk wird mit einer Menge an Trainingsbeispielen (einem Trainingsdatensatz) trainiert. Jedes Trainingsbeispiel besteht dabei aus einer Eingabe $\boldsymbol{x}$. Abhängig vom gewählten Lernparadigma kann ein Trainingsbeispiel auch eine zugehörige Soll-Ausgabe $\boldsymbol{g}$ beinhalten. [33] Drei wichtige Lernparadigmen für neuronale Netze werden nachfolgend aufgezählt: [33]

- **Unüberwachtes Lernen:** Das Netzwerk erhält nur Eingaben $\boldsymbol{x}$ und versucht diese in Klassen einzuteilen.
- **Bestärkendes Lernen:** Das Netzwerk erhält Eingaben $\boldsymbol{x}$ und erfährt ob seine berechneten Ausgaben wahr oder falsch sind.

- **Überwachtes Lernen:** Das Netzwerk erhält Eingaben $\boldsymbol{x}$ und die zugehörigen Soll-Ausgaben $\boldsymbol{g}$. Somit kann für jede Ausgabe ein genauer Fehlerwert berechnet werden.

Hier soll das überwachte Lernen betrachtet werden: Jedes Trainingsbeispiel besteht daher aus einer Eingabe $\boldsymbol{x}$ und einer zugehörigen Soll-Ausgabe $\boldsymbol{g}$ [33]. Der sich mit jedem Beispiel des Datensatzes wiederholende Trainingsablauf ist folgender: [33]

1. Berechnung der Netzausgabe $\boldsymbol{y}$ resultierend aus der Eingabe $\boldsymbol{x}$
2. Berechnung der Abweichung der Netzausgabe $\boldsymbol{y}$ vom Sollwert $\boldsymbol{g}$
3. Ermittlung des Einflusses der Netzparameter auf die Abweichung
4. Anpassen der Parameter

Wurde jedes Trainingsbeispiel des Trainingsdatensatzes genau einmal verwendet, hat das Netzwerk eine Trainingsepoche durchlaufen. Die Anzahl der Epochen beschreibt also, wie oft der Trainingsdatensatz im Training verwendet wurde. Neuronale Netze werden im Allgemeinen mehrere Epochen lang trainiert.

Wie sich die Netzausgabe $\boldsymbol{y}$ aus einer Eingabe $\boldsymbol{x}$ berechnet, wurde bereits in Abschnitt 3.2.2 beschrieben. Weil sich die Information dabei von der Eingabeschicht Richtung Ausgabeschicht durch das Netz fortpflanzt, wird dieser Vorgang auch als Vorwärtspass bezeichnet. Als Maß für die Abweichung der Netzausgabe $\boldsymbol{y}$ vom Sollwert $\boldsymbol{g}$ wird eine sogenannte Kostenfunktion (auch Fehlerfunktion) C gewählt, zum Beispiel die quadratische Abweichung (squared error, SE), geschrieben mit der euklidischen Norm[1]:

$$C(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \|\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{g}(\boldsymbol{x})\|_2^2 \tag{3.6}$$

Die Kostenfunktion C ist dabei von allen Gewichtsmatrizen $\boldsymbol{W}^1$, ..., $\boldsymbol{W}^l$ und Biasvektoren $\boldsymbol{b}^1$, ..., $\boldsymbol{b}^l$ des Netzwerkes (sowie von der Eingabe $\boldsymbol{x}$) abhängig. Den Einfluss der Parameter einer verdeckten Schicht l auf die Kosten, kann über die partiellen Ableitungen $\frac{\partial C}{\partial \boldsymbol{W}^l}$ und $\frac{\partial C}{\partial \boldsymbol{b}^l}$ berechnet werden. Weil die Ableitungen dabei von der Ausgabeschicht Richtung Eingabeschicht zurückgerechnet werden, wird dieser Vorgang auch als Rückwärtspass bezeichnet. Die Anpassung der Parameter erfolgt im einfachsten Fall durch das Verfahren des steilsten Abstiegs mit konstanter Schrittweite für alle Schichten $l = 2, ..., L$:

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \frac{\partial C}{\partial \boldsymbol{W}^l} \tag{3.7}$$

$$\boldsymbol{b}^l \leftarrow \boldsymbol{b}^l - \eta \frac{\partial C}{\partial \boldsymbol{b}^l} \tag{3.8}$$

Durch die Lernrate η kann die Schrittweite des Gradientenabstiegs im Parameterraum angepasst werden. Ein Beispiel für den Verlauf der Kostenfunktion C während des

[1] Die euklidische Norm ist definiert als $\|\boldsymbol{x}\|_2 := \sqrt{\sum_{i=1}^{n} |x_i|^2}$.

Trainings zeigt Abbildung 3.12. Die Kosten fallen zu Beginn des Trainings schnell ab und konvergieren dann gegen einen unbekannten Wert. Zusätzlich schwanken die Kosten im Trainingsverlauf, was bei Verwendung von Gradientenverfahren üblich ist.

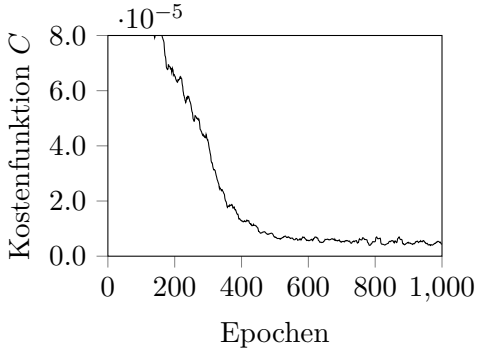


Abbildung 3.12: Exemplarischer Verlauf der —— Kostenfunktion C während des Trainings eines neuronalen Netzwerks, aufgetragen über der Anzahl der Epochen.

Angemerkt sei, dass das Verfahren des steilsten Abstiegs hier nur einfachheitshalber als Optimierungsverfahren gewählt wurde. Es gibt leistungsfähigere Verfahren, die in Abschnitt 3.3.5 beschrieben werden. Die Basis für die Optimierung ist aber bei all diesen Verfahren der Gradient, ausgedrückt durch die partiellen Ableitungen $\frac{\partial C}{\partial \boldsymbol{W}^l}$ und $\frac{\partial C}{\partial \boldsymbol{b}^l}$. Verfahren höherer Ordnung werden in dieser Arbeit nicht behandelt.

Die Kostenfunktion und danach die Parameteranpassungen können, wie im betrachteten Fall, für jedes Trainingsbeispiel einzeln berechnet werden. Sie können aber auch für Batches an Trainingsbeispielen berechnet werden: Ein Batch enthält mehrere Trainingsbeispiele und die Parameteranpassungen werden pro Batch nur einmal durchgeführt. Angenommen, der Trainingsdatensatz enthält n_{train} Trainingsbeispiele, dann werden abhängig von der Batchgröße n_{batch} drei Verfahren unterschieden: [19]

- **Stochastischer Gradientenabstieg**: Die Parameter werden für jedes Trainingsbeispiel einzeln angepasst ($n_{batch} = 1$). Die Anzahl der Iterationen pro Epoche entspricht der Anzahl der Trainingsbeispiele. Der Gradient wird stochastisch genähert.
- **Minibatch-Gradientenabstieg**: Die Parameter werden für mehrere Trainingsbeispiele zusammengefasst angepasst ($1 < n_{batch} < n_{train}$). Die Anzahl der Iterationen pro Epoche ist abhängig von der Batchgröße. Der Gradient wird genähert.
- **Batch-Gradientenabstieg**: Die Parameter werden für alle Trainingsbeispiele des Trainingsdatensatzes zusammengefasst angepasst ($n_{batch} = n_{train}$). Es gibt eine Iteration pro Epoche. Der wahre Gradient wird berechnet.

Die Kostenfunktion der quadratischen Abweichung, Gleichung (3.6), wird bei Verwendung

von Batches der Größe n_{batch} gemittelt und wird dadurch zur mittleren quadratischen Abweichung (mean squared error, MSE):

$$C_{MSE}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \|\boldsymbol{y}_i(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{g}_i(\boldsymbol{x})\|_2^2 \tag{3.9}$$

Darin ist $\boldsymbol{y}_i$ die für ein Trainingsbeispiel des Batches berechnete Netzausgabe und $\boldsymbol{g}_i$ der zugehörige Sollwert. Zur Parameteranpassung wird dann der Gradient der Kostenfunktion in Gleichung (3.9) herangezogen. Die Anzahl der Iterationen pro Epoche errechnet sich nach:

$$n_{it} = \left\lceil \frac{n_{train}}{n_{batch}} \right\rceil \tag{3.10}$$

Dabei ist $\lceil\ \rceil$ die Aufrundungsfunktion und n_{train} die Anzahl der Trainingsbeispiele im Trainingsdatensatz. Sind zum Beispiel $n_{train} = 1050$ und $n_{batch} = 200$ vorgegeben, werden pro Epoche $n_{it} = 6$ Iterationen berechnet. Der letzte Batch einer Epoche enthält dann nur 50 Beispiele.

Die Berechnung des Gradienten

Moderne Frameworks für neuronale Netze berechnen den Gradienten durch automatische Differentiation (AD). Dabei handelt es sich um eine Gruppe von Verfahren, die eine effiziente und automatisierte Berechnung des Gradienten ermöglichen. Die Verfahren berechnen numerische Werte für den Gradienten, sie berechnen diese aber durch wiederholte Anwendung der Kettenregel mathematisch exakt. Sie sind darüber hinaus sehr flexibel in der Anwendung, da sie zum Beispiel auch bei Auftreten von Verzweigungen, Schleifen oder Rekursion im Programmcode funktionieren. Automatische Differentiation gibt es im Vorwärts- und Rückwärtsmodus. In dieser Masterarbeit wird nur der Rückwärtsmodus behandelt, da er für Funktionen $f : \Re^n \to \Re^m$ mit $n \gg m$, was bei neuronalen Netzwerken i.A. der Fall ist, effizienter ist als der Vorwärtsmodus[1]. [7]

Ein Sonderfall der automatischen Differentiation im Rückwärtsmodus ist die Backpropagation[2] (of errors). Seit circa 1986 ist sie der meistverwendete Algorithmus zur Bestimmung des Gradienten beim Training neuronaler Netze. Hinsichtlich des Programmablaufs ist die Backpropagation wenig flexibel, in vielen Anwendungsfällen aber der automatischen Differentiation im Rückwärtsmodus äquivalent[3]. [7] Die Backpropagation lässt sich mathematisch übersichtlich darstellen, deshalb wird ihre Funktionsweise zuerst beschrieben

[1] Gradienten von Funktionen $f : \Re^n \to \Re$ kann der Rückwärtsmodus sogar in nur einem einzigen Berechnungsdurchlauf ermitteln [7].

[2] Der Begriff wird oft mehrdeutig verwendet. Hier sei die Berechnung der Gradienten durch Backpropagation **ohne Anpassen der Parameter** gemeint.

[3] Jedenfalls in der Anwendung auf vorwärtsverknüpfte, mehrschichtige neuronale Netze.

– alle wichtigen Zusammenhänge für das Training neuronaler Netze lassen sich davon ableiten. Danach werden die Grundlagen der automatischen Differentiation erklärt und anhand eines Beispiels verdeutlicht.

Der Backpropagation-Algorithmus

Bevor die Gleichungen behandelt werden, werden zwei neue Größen eingeführt: Erstens, die gewichtete Eingabe (plus Bias) einer Schicht l, $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$. Der Aktivierungsvektor der Schicht wird dann zu $\boldsymbol{a}^l = \boldsymbol{\sigma}^{\boldsymbol{l}}(\boldsymbol{z}^l)$, vgl. Gleichung (3.5). Zweitens, ein Maß für den Fehler einer Schicht l, der Fehler $\boldsymbol{\delta}^l$. Der Algorithmus kann dadurch für ein Netzwerk mit L Schichten durch 4 Gleichungen beschrieben werden. [42] Diese sind: [42]

1. Die Berechnung des Fehlers $\boldsymbol{\delta}^L$ der Ausgabeschicht L:
$$\boldsymbol{\delta}^L = \boldsymbol{\Sigma}'(\boldsymbol{z}^L)\frac{\partial C}{\partial \boldsymbol{a}^L} \tag{3.11}$$
2. Die Berechnung des Fehlers $\boldsymbol{\delta}^l$ einer Schicht l abhängig vom Fehler $\boldsymbol{\delta}^{l+1}$ der Schicht $l+1$:
$$\boldsymbol{\delta}^l = \boldsymbol{\Sigma}'(\boldsymbol{z}^l)(\boldsymbol{W}^{l+1})^T \boldsymbol{\delta}^{l+1} \tag{3.12}$$
3. Die Berechnung der partiellen Ableitungen der Kostenfunktion C nach den Biasvektoren $\boldsymbol{b}^l$:
$$\frac{\partial C}{\partial \boldsymbol{b}^l} = \boldsymbol{\delta}^l \tag{3.13}$$
4. Die Berechnung der partiellen Ableitungen der Kostenfunktion C nach den Gewichtsmatrizen $\boldsymbol{W}^l$:
$$\frac{\partial C}{\partial \boldsymbol{W}^l} = \boldsymbol{\delta}^l (\boldsymbol{a}^{l-1})^T \tag{3.14}$$

$\boldsymbol{\Sigma}'(\boldsymbol{z}^l)$ beschreibt für J Neuronen in Schicht l eine $J \times J$ Diagonalmatrix mit den Einträgen $\sigma_j'^l = \frac{\partial \sigma_j^l}{\partial z_j^l}$ für $j = 1, ..., J$. Der Term $\boldsymbol{\delta}^l (\boldsymbol{a}^{l-1})^T$ in Gleichung (3.14) entspricht dem äußeren Produkt $\boldsymbol{\delta}^l \otimes (\boldsymbol{a}^{l-1})$. Bevor die Backpropagation ausgeführt werden kann, ist ein Vorwärtspass erforderlich, in dem die Netzausgabe $\boldsymbol{y}$ aus einer Eingabe $\boldsymbol{x}$ berechnet wird. Das ermöglicht es, den Fehler $\boldsymbol{\delta}^L$ der Ausgabeschicht L mit Gleichung (3.11) zu berechnen, da $\frac{\partial C}{\partial \boldsymbol{a}^L} = \boldsymbol{f}(\boldsymbol{a}^L)$ und $\boldsymbol{a}^L = \boldsymbol{y}$. Der Fehler wird dann mit Gleichung (3.12) für alle $l = L-1, ..., 2$ durch das Netzwerk bis zur Eingabeschicht rückgeführt. Der gesuchte Gradient wird abschließend durch die Gleichungen (3.13) und (3.14) bestimmt. [42]

Um zu zeigen, dass es sich bei der Backpropagation um eine Form der Anwendung der Kettenregel handelt, wird ein Ausschnitt eines einfachen neuronalen Netzwerks, dargestellt in Abbildung 3.13, betrachtet.

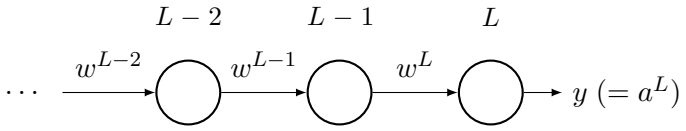


Abbildung 3.13: Ausschnitt eines L-schichtigen Netzwerks mit je einem Neuron pro Schicht

Die Aktivierung einer Schicht l ist in diesem Fall $a^l = \sigma^l(z^l)$ mit $z^l = w^l a^{l-1} + b^l$. Die partielle Ableitung der Kostenfunktion C nach einem Gewicht, zum Beispiel w^{L-2}, kann durch Anwendung der Kettenregel wie folgt berechnet werden:

$$\frac{\partial C}{\partial w^{L-2}} = \frac{\partial C}{\partial a^L}\frac{\partial a^L}{\partial z^L}\underbrace{\frac{\partial z^L}{\partial a^{L-1}}}_{w^L}\frac{\partial a^{L-1}}{\partial z^{L-1}}\underbrace{\frac{\partial z^{L-1}}{\partial a^{L-2}}}_{w^{L-1}}\frac{\partial a^{L-2}}{\partial z^{L-2}}\underbrace{\frac{\partial z^{L-2}}{\partial w^{L-2}}}_{a^{L-2}} \tag{3.15}$$

Unter Verwendung der Backpropagation erhält man, wenn man die Gleichungen (3.12) und (3.13) in (3.11) für den einfachen Fall einsetzt, das gleiche Ergebnis:

$$\begin{aligned}
\frac{\partial C}{\partial w^{L-2}} &= \delta^{L-2}\, a^{L-2} \\
&= \sigma'^{L-2}\, w^{L-1}\, \delta^{L-1}\, a^{L-2} \\
&= \sigma'^{L-2}\, w^{L-1}\, \sigma'^{L-1}\, w^{L}\, \delta^{L}\, a^{L-2} \\
&= \sigma'^{L-2}\, w^{L-1}\, \sigma'^{L-1}\, w^{L}\, \sigma'^{L}\, \frac{\delta C}{\delta a^L}\, a^{L-2} \\
&= \frac{\delta C}{\delta a^L}\, \sigma'^{L}\, w^{L}\, \sigma'^{L-1}\, w^{L-1}\, \sigma'^{L-2}\, a^{L-2}
\end{aligned} \tag{3.16}$$

Auf einen Beweis für den allgemeinen Fall wird an dieser Stelle verzichtet, findet sich aber in [42]. Die Backpropagation entspricht also mathematisch der Kettenregel. Es werden aber, anstatt die Berechnung analytisch durchzuführen, an geeigneter Stelle numerische, bereits im Vorwärtsdurchlauf berechnete Werte eingesetzt. Die Rückübertragung der Fehler erfolgt ebenso mit numerischen Werten. Im Gegensatz zur numerischen Differentiation, z.B. durch Differenzenquotienten, ist die Backpropagation jedoch frei von Näherungsfehlern und arbeitet mit Maschinengenauigkeit [7].

Automatische Differentiation im Rückwärtsmodus

Nachfolgend wird die automatische Differentiation (AD) im Rückwärtsmodus, wie sie in TensorFlow umgesetzt ist, beschrieben. TensorFlow ist eine Programmbibliothek für numerische Berechnungen, mit der die Berechnungen dieser Masterarbeit durchgeführt wurden. Ein Berechnungsverlauf in TensorFlow wird durch einen gerichteten Graphen beschrieben. Der Graph besteht aus Knoten (nodes), die Operationen darstellen und aus

Kanten (edges), die Ein- und Ausgänge der Knoten miteinander verbinden. Die Kanten werden als Pfeile dargestellt. Größen, die über die Kanten weitergegeben werden, werden als Tensoren (tensors) bezeichnet und sind beliebig dimensionale Arrays, deren Datentyp (z.B. integer) entweder definiert oder bei der Grapherstellung abgeleitet wird. Es gibt zusätzlich auch Kontrollstrukturen für Verzweigungen, Schleifen und Ähnliches, die aber im weiteren Verlauf nicht näher beschrieben werden. Ein Graph wird zuerst definiert und dann i.A. für mehrere Berechnungsdurchläufe verwendet. [1]

Der gewünschte Vorwärtspfad einer Berechnung wird zu Beginn vom Benutzer vorgegeben und ein Berechnungsgraph erstellt. Wird nun der Gradient eines Tensors T bezüglich eines Tensors F benötigt, ermittelt TensorFlow zuerst den Pfad von T nach F. Anschließend wird dieser Pfad von F nach T zurückverfolgt und der Berechnungsgraph für jede gefundene Operation um eine zugehörige Gradientenoperation erweitert. Diese Gradientenoperationen bilden den Rückwärtspfad. Sie können neben den entlang des Rückwärtspfades berechneten partiellen Gradienten auch Tensoren des Vorwärtspfades als Eingangsgrößen erhalten. [1] Anders formuliert wird eine Berechnung in ihre Elementaroperationen zerlegt, deren Ableitungen i.A. bekannt sind. Durch geeignete Kombination dieser Ableitungen, entsprechend der Kettenregel, lassen sich die Ableitungen der gesamten Berechnung bestimmen. [7]

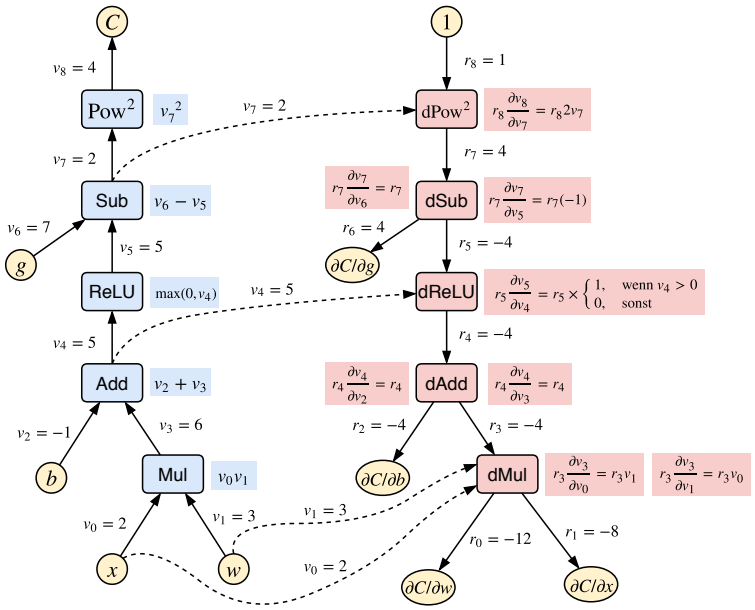


Abbildung 3.14: Darstellung eines TensorFlow Berechnungsgraphen mit Beispielrechnung. Adaptiert und erweitert aus [1].

Ein Beispiel für einen Berechnungsgraphen in TensorFlow zeigt Abbildung 3.14. Die linke Hälfte beschreibt den Vorwärtspfad der Berechnung, die rechte Hälfte den Rückwärtspfad. Entlang der gestrichelten Pfeile werden Tensoren vom Vorwärtspfad an den Rückwärtspfad weitergegeben, die zur Berechnung der Gradienten benötigt werden. Die von einem Knoten durchgeführten Operationen sind jeweils daneben aufgeführt.

3.2.4 Erwartungen an Maschinenlern-Modelle

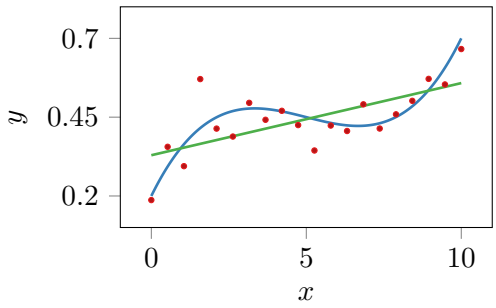
Eine Hauptanforderung an Maschinenlern-Modelle ist, dass diese auch für unbekannte Daten genaue Vorhersagen treffen. Ein Modell, das nur Vorhersagen für bereits bekannte Datenpunkte wiedergeben kann, ist schließlich obsolet – die Datenpunkte sind ja bereits verfügbar. Die Fähigkeit eines Modells, diese Anforderung zu erfüllen, wird als Generalisierung bezeichnet. Um untersuchen zu können, ob ein Modell gut generalisiert oder nicht, werden zusätzlich zu den Trainingsdaten weitere Daten benötigt: die Testdaten. Mit den Trainingsdaten wird der sogenannte Trainingsfehler (bzw. die Trainingskosten) des Modells ermittelt. Der Trainingsfehler wird dann durch Anpassung der Modellparameter minimiert (das Modell wird trainiert). Ist das Training beendet, wird mit den Testdaten der sogenannte Testfehler berechnet. Der Testfehler ist im Allgemeinen größer als der Trainingsfehler. Er entspricht dem zu erwartenden Fehler des Modells für unbekannte Daten, zum Beispiel im späteren Einsatz für industrielle Anwendungen. All dies ergibt zusammengefasst zwei wichtige Ziele im Maschinenlernen: [22]

1. Ein möglichst kleiner Trainingsfehler
2. Eine möglichst kleine Differenz zwischen Trainings- und Testfehler

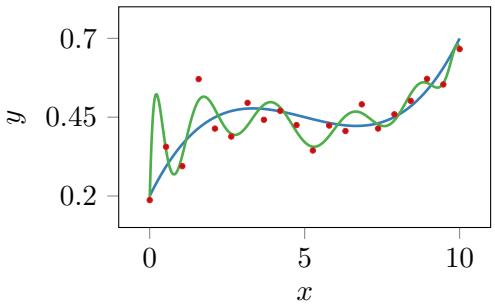
Diese zwei Ziele führen direkt zu den Begriffen Unteranpassung (underfitting) und Überanpassung (overfitting). Unteranpassung beschreibt den Fall, dass der Trainingsfehler eines Modells zu groß ist. Überanpassung beschreibt hingegen den Fall, dass die Differenz zwischen Trainings- und Testfehler zu groß ist. [22]

Es soll nun je ein Beispiel für Unter- bzw. Überanpassung gezeigt werden. Dazu wurden, durch die beliebig gewählte „wahre“ Funktion $y(x) = 0.2 + 0.2x - 0.045x^2 + 0.03x^3$, Trainingsdaten generiert und mit normalverteiltem Rauschen versehen. Danach wurden polynomiale Regressionsfunktionen 1. und 9. Ordnung an die Trainingsdaten angepasst. Abbildung 3.15a zeigt Unteranpassung durch die Polynomfunktion 1. Ordnung: Der Großteil der Trainingsdaten (•) liegt weit entfernt von der Regressionsfunktion (——). Die Regressionsfunktion kann die Besonderheiten der Daten, d.h. die Form der wahren Funktion (——), nicht darstellen. Abbildung 3.15b zeigt Überanpassung durch die Polynomfunktion 9. Ordnung: Der Großteil der Trainingsdaten (•) liegt in der Nähe oder auf der Regressionsfunktion (——). Die Regressionsfunktion folgt in diesem Beispiel zwar der

Form der wahren Funktion (——), stellt aber auch die Besonderheiten des Rauschens der Daten dar.



(a) Unteranpassung durch Polynom 1.Ordnung.



(b) Überanpassung durch Polynom 9.Ordnung.

Abbildung 3.15: Beispiel für Überanpassung und Unteranpassung durch polynomiale Regression. Die —— Regressionsfunktion ist angepasst auf durch die —— wahre Funktion erzeugte • Trainingsdaten mit simuliertem Rauschen.

Der Bias–variance-tradeoff

Eng verknüpft mit den Begriffen der Unter- und Überanpassung ist der sogenannte Bias-variance-tradeoff. Zur weiteren Ausführung sind zunächst einige Definitionen notwendig: Sei $F(\boldsymbol{x})$ die wahre, rauschbehaftete und unbekannte Funktion die genähert werden soll. D sei ein durch $F(\boldsymbol{x})$ generierter Datensatz aus n Beispielen und $y(\boldsymbol{x}, D)$ die auf D basierende Näherung von $F(\boldsymbol{x})$. Es wird untersucht, inwiefern $y(\boldsymbol{x}, D)$ von D abhängt. Als Maß für die Güte der Näherung wird ihre mittlere quadratische Abweichung zur wahren Funktion gewählt. Gemittelt über alle Datensätze D, ausgedrückt durch den Erwartungswert $\mathcal{E}_D$ erhält man: [17]

$$\mathcal{E}_D\left[(y(\boldsymbol{x}, D) - F(\boldsymbol{x}))^2\right] = \underbrace{\left(\mathcal{E}_D\left[y(\boldsymbol{x}, D) - F(\boldsymbol{x})\right]\right)^2}_{Bias^2} + \underbrace{\mathcal{E}_D\left[(y(\boldsymbol{x}, D) - \mathcal{E}_D\left[y(\boldsymbol{x}, D)\right])^2\right]}_{Varianz} \quad (3.17)$$

Der linke Term auf der rechten Seite der Gleichung (3.17) beschreibt den quadrierten Bias, der rechte Term die Varianz. Ein geringer Bias sagt aus, dass $F(\boldsymbol{x})$ durch $y(\boldsymbol{x}, D)$ im Durchschnitt genau genähert wird. Eine niedrige Varianz sagt aus, dass sich die Näherung $y(\boldsymbol{x}, D)$ mit variiertem Datensatz D nicht stark ändert. [17]

Die mittlere quadratische Abweichung der Näherung kann nur gering sein, wenn sowohl

der Bias als auch die Varianz gering sind. Im Maschinenlernen ist dies zumeist ein Zielkonflikt, der als Bias-variance-tradeoff bezeichnet wird: [17]

- **Modelle mit hoher Flexibilität** (mit vielen Parametern) passen sich i.A. gut an die Daten an, sind aber empfindlich gegenüber sich ändernden Datensätzen. Sie haben daher einen niedrigen Bias, aber eine hohe Varianz. Sie neigen zu Überanpassung.
- **Modelle mit geringer Flexibilität** (mit wenigen Parametern) passen sich i.A. schlecht an die Daten an, sind aber wenig empfindlich gegenüber sich ändernden Datensätzen. Sie haben daher einen hohen Bias, aber eine geringe Varianz. Sie neigen zu Unteranpassung.

Ockhams Rasiermesser

Ockhams Rasiermesser (Occam's razor) ist ein allgemeines Prinzip, das auch im Sinne des Maschinenlernens interpretiert wird: Es besagt, dass der Einsatz von Methoden, die komplizierter sind als nötig, vermieden werden sollte[1]. [17] Ockhams Rasiermesser kann in zwei Teilen formuliert werden: [16]

1. Aus zwei Methoden mit demselben Testfehler sollte die einfachere gewählt werden, da Einfachheit erstrebenswert ist.
2. Aus zwei Methoden mit demselben Trainingsfehler sollte die einfachere gewählt werden, da sie voraussichtlich einen geringeren Testfehler aufweist.

Der erste Teil hat praktische Gründe: Einfachere Modelle sind für den Menschen besser zu verstehen und benötigen im Allgemeinen weniger Rechenressourcen. Der zweite Teil kann als Vermeidung von Überanpassung gesehen werden. Gegen ihn gibt es aber theoretische Einwände und empirische Untersuchungen deuten darauf hin, dass er nicht gültig ist [16]. Dennoch findet Ockhams Rasiermesser weite Verbreitung im Maschinenlernen.

Die No Free Lunch Theoreme oder „nichts ist umsonst“

Die No Free Lunch Theoreme besagen, dass der Erwartungswert des Fehlers aller Maschinenlern-Methoden[2], gemittelt über alle möglichen Zielfunktionen, gleich ist. Erreicht eine Methode A in einer bestimmten Klasse von Problemen einen geringeren Fehler als eine Methode B, ist das für die Klasse aller anderen Probleme umgekehrt. Es

[1] „Kompliziert“ bezieht sich bei neuronalen Netzen zum Beispiel auf die Anzahl an Neuronen und „nötig“ auf den gewünschten Ausgabefehler.

[2] Mit dem Begriff Methoden sind sowohl unterschiedliche Verfahren (z.B. Entscheidungsbäume, neuronale Netze) als auch verschiedene Modelle innerhalb eines Verfahrens gemeint.

gibt daraus folgend keine von der Problemstellung unabhängigen Gründe, bestimmte Methoden gegenüber anderen zu bevorzugen. [17, 62]

Resultierend aus den Theoremen sind generelle Aussagen darüber, ob eine Methode „besser“ ist als andere, nicht möglich. Bei Anwendung auf bestimmte Problemstellungen können aber sehr wohl Aussagen darüber gemacht werden, ob eine Methode (bezüglich der Problemstellungen) „besser“ ist als andere. Im praktischen Alltag sind oft die Eigenheiten einer Problemstellung bekannt und Erfahrungswerte dazu vorhanden – es wird versucht, ungeeignete Methoden noch vor etwaiger Untersuchungen auszuschließen. Die No Free Lunch Theoreme sprechen nicht grundsätzlich gegen diese Vorgehensweise, mahnen aber zu Vorsicht vor zu schnell getroffenen Annahmen.

3.3 Hyperparameter von neuronalen Netzen

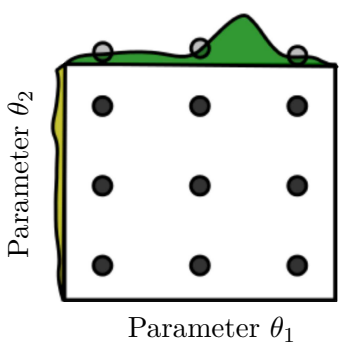
Wie bereits in Abschnitt 3.2.3 kurz angeführt, werden die Parameter neuronaler Netze, die nicht durch den Gradientenabstieg im Training angepasst werden können, als Hyperparameter bezeichnet. Im weiteren Sinne können alle Parameter, sogar Programmeinstellungen und Entscheidungen, die die Netzerstellung beeinflussen, als Hyperparameter aufgefasst werden: Es sind beispielsweise auch die Wahl des im Training verwendeten Gradientenabstiegverfahrens und dessen Einstellungen Hyperparameter. Das lässt erahnen, dass Neuronale Netze i.A. eine sehr hohe Anzahl an Hyperparametern aufweisen, was deren Optimierung schwierig macht. Nachfolgend sollen zuerst drei häufig verwendete Optimierungsstrategien für Hyperparameter beschrieben werden. Im Anschluss daran wird ein Überblick über die wichtigsten Hyperparameter neuronaler Netze geboten.

3.3.1 Optimierungsstrategien für Hyperparameter

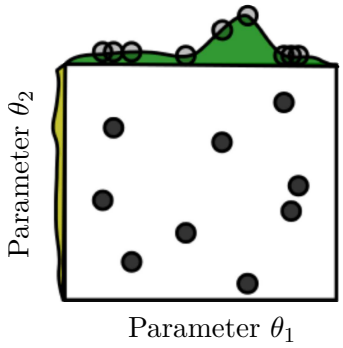
Ziel aller Optimierungsstrategien für Hyperparameter ist die Minimierung des Testfehlers (oder die Maximierung der Testgenauigkeit) eines neuronalen Netzes. Wie der Verlauf des Testfehlers von den Hyperparametern abhängt, ist dabei unbekannt. Der Wert des Testfehlers lässt sich aber stellenweise ermitteln, worauf die Optimierungsstrategien aufbauen. Manche Optimierungsstrategien basieren auf dem Prinzip von „trial and error“: Es werden mehrere neuronale Netze mit unterschiedlichen Parameterkombinationen trainiert und getestet – das beste Netzwerk, d.h. das Netzwerk mit dem geringsten Testfehler oder der höchsten Genauigkeit, definiert die optimale Parameterkombination. Strategien, die auf diesem Prinzip basieren, sind zum Beispiel die Rastersuche und die Zufallssuche. Andere Strategien versuchen den Verlauf des Testfehlers, abhängig von den Hyperparametern, zu modellieren. Ein Beispiel ist die Bayes-Optimierung. Die drei genannten Strategien werden in diesem Abschnitt genauer beschrieben. Zuvor jedoch

noch eine wichtige Anmerkung: Die in der Hyperparameteroptimierung verwendeten Testdaten sollen nicht für den Test des finalen Modells verwendet werden, da das Modell mit diesen Daten optimiert wurde und es sie bereits „kennt“. Die Datenverwendung wird in Kapitel 4 aber noch genauer behandelt.

Bei der Rastersuche werden für jeden Parameter die zu untersuchenden Werte vorgegeben. Es werden dann alle möglichen Kombinationen der Parameterwerte gebildet. Das entspricht der Erstellung eines Punktrasters im Parameterraum, wobei jeder Punkt einer Parameterkombination entspricht. Der Wert des Testfehlers wird in jedem der Punkte ermittelt. Der Punkt mit dem kleinsten Testfehler ist das Optimum. Bei der Zufallssuche wird hingegen jeder Parameter, innerhalb eines vorgegebenen Wertebereichs, durch Zufallszahlen aus einer Wahrscheinlichkeitsverteilung bestimmt – meist aus einer Gleichverteilung. Das entspricht der zufälligen Wahl eines Punktes im Parameterraum. Auf diese Weise werden mehrere Punkte bestimmt und der Wert des Testfehlers in diesen Punkten ermittelt. Gleich wie bei der Rastersuche, ist der Punkt mit dem kleinsten Testfehler das Optimum.



(a) Rastersuche.



(b) Zufallssuche.

Abbildung 3.16: Exemplarische Darstellung von Rastersuche und Zufallssuche mit 9 untersuchten Punkten, für den zweidimensionalen Fall. Optimiert wird die Zielfunktion $f(\theta_1, \theta_2) = g(\theta_1) + h(\theta_2) \approx g(\theta_1)$. $g(\theta_1)$ ist grün dargestellt, $h(\theta_2)$ gelb. θ_1 hat großen und θ_2 kleinen Einfluss auf die Zielfunktion. Die Rastersuche untersucht den „wichtigen“ Parameter θ_1 nur an 3 Stellen, die Zufallssuche hingegen an 9 Stellen. Neu beschriftet und übernommen aus [9].

Sowohl Rastersuche als auch die Zufallssuche sind vom Prinzip her einfach verständlich. Sie sind beide programmiertechnisch einfach umzusetzen und können im Ablauf leicht parallelisiert werden, da die einzelnen Berechnungen in den Punkten nicht voneinander abhängen. Die Rastersuche ist für die hochdimensionale Parameteroptimierung weniger gut geeignet: Die benötigte Rechenleistung steigt mit der Parameteranzahl exponentiell. Sollen beispielsweise 10 Parameter mit je 3 Werten variiert werden, müssen bereits

$3^{10} \approx 59000$ Kombinationen untersucht werden. Außerdem weisen Funktionen in hochdimensionalen Parameterräumen, bei neuronalen Netzen z.B. der Verlauf des Testfehlers in Abhängigkeit der Hyperparameter, im Allgemeinen eine geringe effektive Dimensionalität auf. Eine Funktion mit geringer effektiver Dimensionalität hängt näherungsweise nur von wenigen Parametern ab, das heißt die restlichen Parameter haben wenig Einfluss auf die Funktion. Ein Beispiel für eine Funktion mit geringer effektiver Dimensionalität im zweidimensionalen Fall ist $f(\theta_1, \theta_2) = g(\theta_1) + h(\theta_2) \approx g(\theta_1)$. Die Optimierung dieser Funktion durch die Raster- und die Zufallssuche ist exemplarisch in Abbildung 3.16 dargestellt: Mit der Rastersuche wird viel Rechenzeit für die Optimierung „unwichtiger" Parameter verschwendet. [9]

Die Bayes'sche Optimierung soll an dieser Stelle nur sehr vereinfacht beschrieben werden. Der Wert des Testfehlers wird in einigen Punkten im Parameterraum ermittelt und eine Ersatzfunktion für dessen Verlauf bestimmt. Mit der Ersatzfunktion wird dann nach Minima gesucht. Wissen über bereits durchgeführte Berechnungen wird dabei laufend zur Anpassung der Ersatzfunktion verwendet. Die Bayes'sche Optimierung ist sehr effizient, es gibt aber Gründe, die gegen ihre Verwendung sprechen: Die Bayes'sche Optimierung skaliert schlecht mit steigender Parameteranzahl, das bedeutet die benötigte Rechenleistung steigt stark. Die Bayes'sche Optimierung hat außerdem wiederum eigene Hyperparameter, deren richtige Wahl schwierig, aber Voraussetzung für eine erfolgreiche Optimierung ist. Sie ist außerdem schwer zu parallelisieren, da die Berechnungen voneinander abhängen und sequenziell ausgeführt werden müssen. [2]

3.3.2 Wahl der Aktivierungsfunktionen

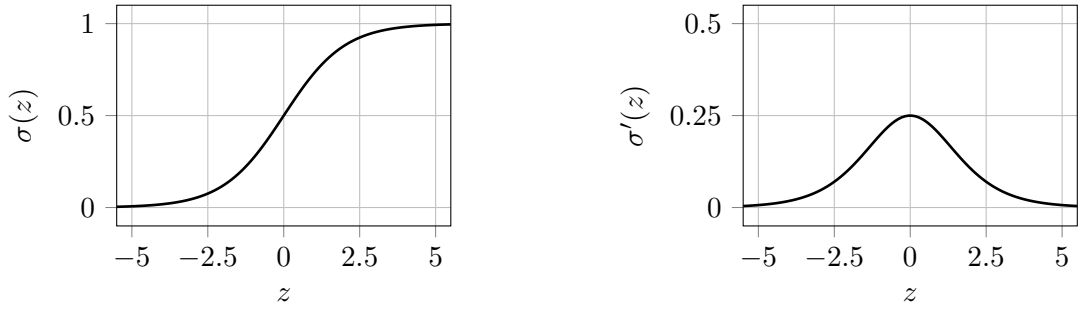
Aktivierungsfunktionen in neuronalen Netzwerken beeinflussen nicht nur die Berechnung der Ausgaben, vgl. Gleichung (3.5), sondern auch das Training. Die Wahl der richtigen Aktivierungsfunktion ist von großer Bedeutung, da sonst das Training nicht oder nur sehr langsam abläuft. Um dies zu zeigen soll hier noch einmal die Backpropagation, angewandt auf den vereinfachten Fall, betrachtet werden, vgl. Gleichung (3.16):

$$\frac{\partial C}{\partial w^{L-2}} = \frac{\delta C}{\delta a^L}\, \sigma'^L\, w^L\, \sigma'^{L-1}\, w^{L-1}\, \sigma'^{L-2}\, a^{L-2}$$

Wird auf der rechten Seite dieser Gleichung ein Term $\sigma'^l \approx 0$ (und nicht durch andere, sehr große Terme kompensiert) lernt das Gewicht w^{L-2} sehr langsam. Das zu σ'^l zugehörige Neuron ist dann gesättigt. Als Beispiel für eine Aktivierungsfunktion soll die Sigmoid-Funktion[1] $\sigma(z) = \frac{1}{(1+e^{-z})}$ herangezogen werden, die mit ihrer Ableitung in Abbildung

[1] Der Begriff Sigmoid-Funktion wird in dieser Arbeit für die sogenannte logistische Funktion verwendet. Genau genommen gibt es mehrere Sigmoidfunktionen und die logistische Funktion ist nur eine Variante davon.

3.17 dargestellt ist. Ein Neuron mit dieser Aktivierungsfunktion ist gesättigt, wenn die gewichtete Eingabe z viel größer oder kleiner als 0 wird, siehe Abbildung 3.17b: Die Ableitung $\sigma(z)'$ geht dann gegen 0. [42]



(a) Funktionsverlauf.

(b) Ableitungsverlauf.

Abbildung 3.17: Verlauf der Sigmoid-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z.

Besonders bei vielschichtigen Netzen kann auch noch ein anderer Effekt dazu führen, dass ein Gewicht langsam lernt: das Problem der verschwindenden Gradienten (vanishing gradients problem). Bei Berechnung der Gradienten durch Backpropagation werden Terme der Art $\sigma'^l w^l$ multiplikativ aneinander gereiht. In der betrachteten Gleichung sind dies nur zwei: $\sigma'^L w^L$ und $\sigma'^{L-1} w^{L-1}$. Weist das neuronale Netz jedoch eine hohe Anzahl an Schichten auf, werden, bei Berechnung der Gradienten der vorderen Schichten, sehr viele dieser Terme miteinander multipliziert. Sind diese Terme $\sigma'^l w^l < 1$, sinkt ihr Produkt exponentiell mit der Schichtanzahl und Gewichte in den vorderen Schichten lernen langsam. Es ist jedoch auch der umgekehrte Fall möglich: sind die Terme $\sigma'^l w^l > 1$, steigt ihr Produkt exponentiell und man spricht vom Problem der explodierenden Gradienten (exploding gradients problem). Ein Beispiel, indem $\sigma'^l w^l < 1$ auftritt, soll mithilfe der Sigmoid-Funktion konstruiert werden: Das Maximum der Steigung der Sigmoid-Funktion ist $\sigma(0)' = 0.25$, siehe Abbildung 3.17b. Das bedeutet, dass $\sigma(z)' \leq 0.25$. Sind die Gewichte w^l zu Beginn des Trainings mit Mittelwert 0 und Standardabweichung 1 normalverteilt, ist i.A. $w^l < 1$ und daher $\sigma'^l w^l < 0.25$. Es ist kaum möglich, dies mit großen Gewichten w^l auszugleichen: Wird w^l groß, wird nämlich wiederum σ'^l klein. [42] Eine ausführlichere und allgemeinere Erklärung des Problems der verschwindenden Gradienten findet sich in [25][1].

Zusammengefasst haben somit die Aktivierungsfunktionen einen wesentlichen Einfluss auf die Berechnung der Gradienten in neuronalen Netzen. Das Beispiel zum Problem

[1] Es geht darin zwar um rekurrente neuronale Netze, die Ergebnisse lassen sich aber auf mehrschichtige Netze übertragen.

der verschwindenden Gradienten hat außerdem angedeutet, dass Wechselwirkungen zwischen Aktivierungsfunktionen und der Art und Weise, wie Gewichte zu Beginn des Trainings initialisiert werden, bestehen können. Im Folgenden wird auf in der Literatur häufig anzutreffende Aktivierungsfunktionen näher eingegangen. Der Einfluss der Gewichtsinitialisierung wird in Abschnitt 3.3.3 behandelt.

Sigmoid-Aktivierungsfunktion

Die Sigmoid-Aktivierungsfunktion, dargestellt in Abbildung 3.17, ist eine der bekanntesten Aktivierungsfunktionen im Maschinenlernen. Die Funktion und ihre Ableitung berechnen sich nach:

$$\sigma(z) = \frac{1}{(1 + \mathrm{e}^{-z})} \tag{3.18}$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)) \tag{3.19}$$

Die Sigmoid-Funktion ist immer positiv – ihr Wertebereich liegt zwischen 0 und 1, das heißt sie ist nicht um Null zentriert[1]. Diese Eigenschaft kann die Konvergenz zu einem (lokalen) Minimum verlangsamen. [35] Der Wertebereich der Ableitung der Sigmoidfunktion liegt zwischen 0 und 0.25. Der geringe Maximalwert der Ableitung kann das Problem der verschwindenden Gradienten verstärken [25].

In Untersuchungen für Bilderkennung wurde festgestellt, dass die letzte verdeckte Schicht in Sigmoid-Netzwerken dazu neigt, in Sättigung zu gehen (die Einträge des Aktivierungsvektors gehen gegen null). Das Training der anderen verdeckten Schichten ist dann blockiert. Das Verlassen der Sättigung ist zwar in manchen Fällen möglich, dauert aber mit steigender Schichtanzahl länger. Es wurden außerdem höhere Testfehler als mit der Tanh- oder Softsign-Aktivierungsfunktion erreicht. [20]

Tanh-Aktivierungsfunktion

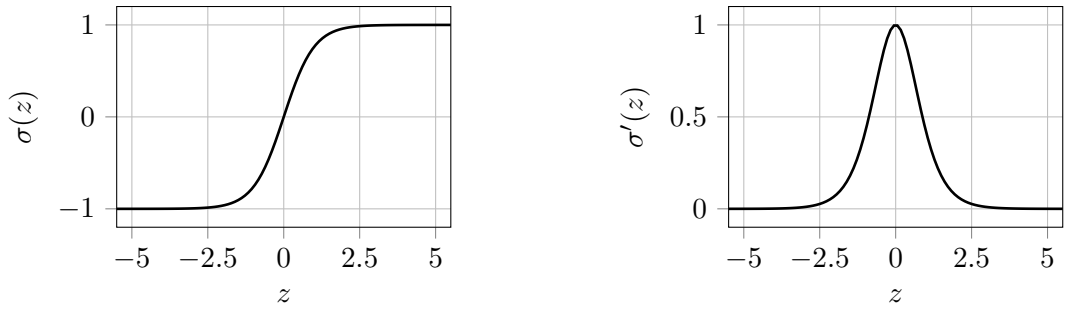
Die Tanh-Funktion und ihre Ableitung, dargestellt in Abbildung 3.18, berechnen sich nach:

$$\sigma(z) = \tanh z \tag{3.20}$$

$$\sigma'(z) = 1 - (\tanh z)^2 \tag{3.21}$$

Die Tanh-Funktion ist punktsymmetrisch zum Koordinatenursprung und kann sowohl positive als auch negative Werte annehmen. Sie ist im Gegensatz zur Sigmoid-Funktion um Null zentriert. Der Wertebereich der Tanh-Funktion liegt zwischen -1 und 1 und

[1] Der Mittelwert der Funktion in einem Intervall $[-a, a]$ mit $a \in \Re$ ist größer als Null.



(a) Funktionsverlauf.

(b) Ableitungsverlauf.

Abbildung 3.18: Verlauf der Tanh-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z.

der ihrer Ableitung zwischen 0 und 1. Der im Vergleich zur Sigmoid-Funktion höhere Maximalwert der Ableitung kann helfen, das Problem der verschwindenden Gradienten abzuschwächen.

Das in den bereits erwähnten Untersuchungen für Bilderkennung bei Sigmoid-Netzwerken festgestellte Sättigungsverhalten der letzten verdeckten Schicht, trat bei Tanh-Netzwerken nicht auf. Darüber hinaus konnte, mit geeigneter Initialisierung der Gewichte zu Beginn des Trainings, das Problem der verschwindenden Gradienten bei Verwendung der Tanh-Funktion sogar weitgehend vermieden werden. Gezeigt wurde das für Netzwerke mit bis zu 5 verdeckten Schichten. [20]

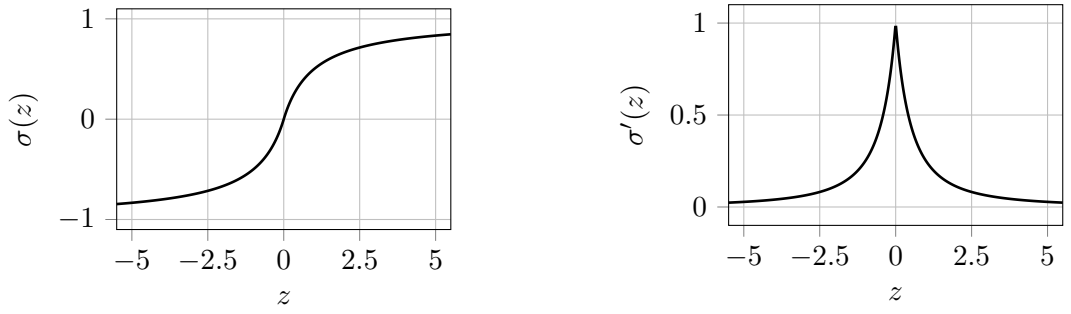
Softsign-Aktivierungsfunktion

Die Softsign-Funktion ist eine Alternative zur Tanh-Funktion. Sie ist mit ihrer Ableitung in Abbildung 3.19 dargestellt. [10]

$$\sigma(z) = \frac{z}{1 + |z|} \tag{3.22}$$

$$\sigma'(z) = \frac{1}{(1 + |z|)^2} \tag{3.23}$$

Die Softsign-Funktion ist punktsymmetrisch zum Koordinatenursprung und kann sowohl positive als auch negative Werte annehmen. Sie ist, ebenso wie die Tanh-Funktion, um Null zentriert. Der Wertebereich der Funktion liegt zwischen -1 und 1 und der ihrer Ableitung zwischen 0 und 1. Die Softsign-Funktion geht, verglichen mit der Tanh-Funktion, langsamer in Sättigung. Ein Grund dafür ist, dass die Ableitung der Softsign-Funktion über einen breiteren Bereich nennenswert größer Null ist. [10]



(a) Funktionsverlauf

(b) Ableitungsverlauf

Abbildung 3.19: Verlauf der Softsign-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z, nach [10].

In Untersuchungen für Bilderkennung wurden mit der Softsign-Funktion in 8 von 9 Fällen bessere Ergebnisse erreicht, als mit der Tanh-Funktion [10]. Es ist hier jedoch anzumerken, dass 6 der 9 Fälle mit Netzwerken mit quadratischen Neuronen[1] berechnet wurden. Die Softsign-Funktion wies, in den bereits erwähnten Untersuchungen, höhere Robustheit als die Tanh-Funktion gegenüber Wahl der Gewichtsinitialisierung auf [20].

ReLU-Aktivierungsfunktion und ihre Varianten

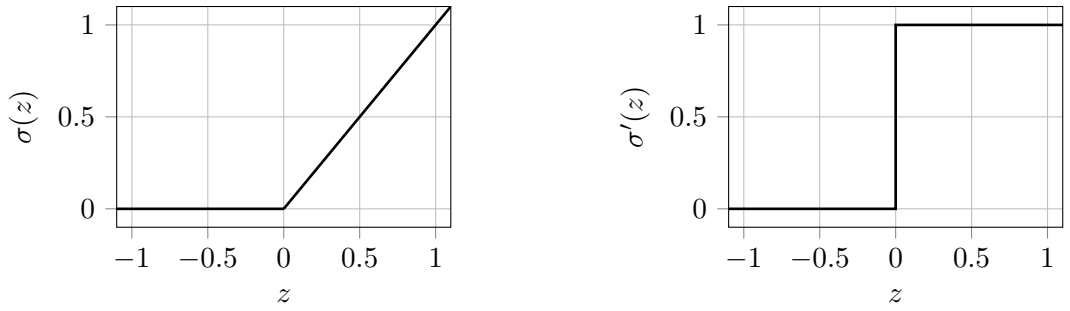
Die ReLU-Aktivierungsfunktion, dargestellt mit ihrer Ableitung in Abbildung 3.20, ist besonders gut für tiefe Netzstrukturen geeignet [21]. Sie hatte einen großen Anteil am Erfolg neuronaler Netze ab 2011 [50]. Sie berechnet sich nach: [21]

$$\sigma(z) = \max(0, z) \tag{3.24}$$

$$\sigma'(z) = \begin{cases} 1, & \text{wenn } z > 0 \\ 0, & \text{sonst} \end{cases} \tag{3.25}$$

Die ReLU-Funktion ist nach oben unbeschränkt. Sie ist, wie die Sigmoid-Funktion, nicht um Null zentriert, was die Konvergenz zu einem Minimum im Training verlangsamen kann [35]. Ihre Ableitung ist eine Stufenfunktion und kann nur die Werte 0 oder 1 annehmen. Das Problem der verschwindenden Gradienten tritt daher nicht auf. Ein ReLU-Neuron kennt zwei Zustände, es ist entweder aktiv (die Ausgabe ist die Identität des Arguments z) oder inaktiv (die Ausgabe ist Null). In einem ReLU-Netzwerk ist, für eine konstant vorgegebene Eingabe, daher meist ein Teil der Neuronen inaktiv und die aktiven Neuronen bilden einen oder mehrere Pfade durch das Netzwerk. Die Ausgabe

[1] Es wurden zusätzliche, quadratische Terme bei der Berechnung der gewichteten Eingabe eingeführt.



(a) Funktionsverlauf.

(b) Ableitungsverlauf.

Abbildung 3.20: Verlauf der ReLU-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z, nach [21].

des Netzwerks ist dabei eine lineare Funktion der Eingabe. Für sich ändernde Eingaben ändert sich auch der Pfadverlauf im Netzwerk – erst dadurch entsteht Nichtlinearität. Eine durch ein ReLU-Netzwerk berechnete Funktion ist stückweise linear. [21]

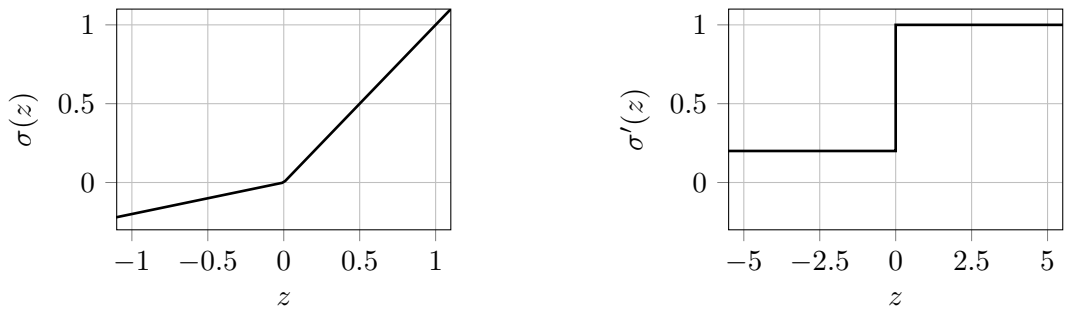
Inaktive ReLU-Neuronen geben exakt den Wert Null aus. Es ist mit ihnen daher möglich, dünnbesetzte Netzwerke (mit dünnbesetzten Matrizen) zu erstellen. Ein möglicher Vorteil dünnbesetzter Netzwerke ist, dass sie Abhängigkeiten der zu nähernden Funktion von den Eingabevariablen besser erkennen und trennen können. [21] Ein Nachteil von ReLU-Netzwerken ist hingegen, dass die Ableitung der Aktivierungsfunktion inaktiver Neuronen (sowie die Aktivierung selbst) null ist. Über diese Neuronen kann somit durch den Backpropagation-Algorithmus kein Fehler rückgeführt werden und sie sind in Sättigung (bekannt als „dying ReLU“ Problem). Solange jedoch nicht alle Neuronen einer Schicht gesättigt sind, können gesättigte Neuronen über alternative Pfade wieder aktiviert werden. [21] In Untersuchungen für Bilderkennung und Spracherkennung wurden mit der ReLU-Funktion geringere Testfehler erzielt als mit der Tanh-Funktion. [21, 36]

Um den Einfluss des „dying ReLU“ Problems zu untersuchen, wurde eine modifizierte Variante vorgeschlagen: die Leaky-ReLU-Funktion, dargestellt in Abbildung 3.21. Die Funktion und ihre Ableitung berechnen sich nach: [36]

$$\sigma(z) = \begin{cases} z, & \text{wenn } z > 0 \\ \alpha z, & \text{sonst} \end{cases} \tag{3.26}$$

$$\sigma'(z) = \begin{cases} 1, & \text{wenn } z > 0 \\ \alpha, & \text{sonst} \end{cases} \tag{3.27}$$

Über den Parameter $\alpha > 0$ kann die Steigung der Funktion im Bereich negativer



(a) Funktionsverlauf.

(b) Ableitungsverlauf.

Abbildung 3.21: Verlauf der Leaky-ReLU-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z, nach [36].

Argumente z vor Beginn des Trainings festgelegt werden[1]. Die Leaky-ReLU-Funktion ist nicht um Null zentriert, abhängig von α wird ihr Mittelwert jedoch Richtung Null verschoben. Wenn $\alpha \ll 1$ ist, dann ist dieser Einfluss aber sehr gering (in [36] ist zum Beispiel $\alpha = 0.01$). Die Ableitung der Leaky-ReLU-Funktion wird niemals null und somit kann die Fehlerrückführung mit dem Backpropagation-Algorithmus nicht durch inaktive Neuronen blockiert werden. Der Vorteil dünnbesetzter Matrizen mit exakten Nullen wird jedoch aufgegeben. In Untersuchungen für Spracherkennung konnten minimale Vorteile der Leaky-ReLU-Funktion gegenüber der ReLU-Funktion festgestellt werden. [36]

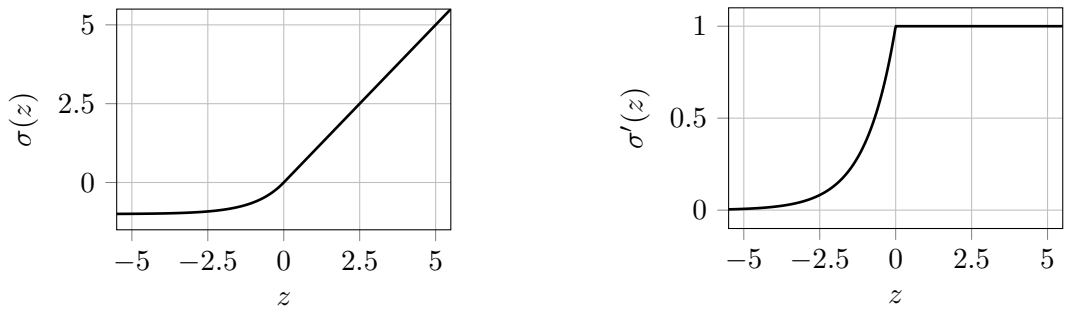
Eine weitere Modifikation der ReLU-Funktion ist die ELU-Funktion (exponential linear unit), dargestellt in Abbildung 3.22. Die Funktion und ihre Ableitung berechnen sich wie folgt: [14]

$$\sigma(z) = \begin{cases} z, & \text{wenn } z \geq 0 \\ \alpha(e^z - 1), & \text{sonst} \end{cases} \tag{3.28}$$

$$\sigma'(z) = \begin{cases} 1, & \text{wenn } z \geq 0 \\ \alpha e^z, & \text{sonst} \end{cases} \tag{3.29}$$

Der Parameter $\alpha > 0$ legt die untere Schranke der ELU-Funktion fest, d.h. den Wert, den sie bei Sättigung für negative Argumente ausgibt. Die ELU-Funktion ist nicht um Null zentriert, ihr Mittelwert wird aber abhängig von α, ebenso wie bei der Leaky-ReLU-Funktion, Richtung Null verschoben. Die einseitige Sättigung erlaubt eine klare Trennung aktiver und inaktiver Neuronen. Das bedeutet, dass nur aktive Neuronen relevante Informationen weitergeben, ähnlich wie bei dünnbesetzten Netzwerken. In

[1] α kann auch ein lernbarer Parameter sein, der während des Trainings durch den Backpropagation-Algorithmus optimiert wird – die Aktivierungsfunktion wird in diesem Fall als PReLU (parametric ReLU) bezeichnet [23].



(a) Funktionsverlauf.

(b) Ableitungsverlauf.

Abbildung 3.22: Verlauf der ELU-Aktivierungsfunktion und ihrer Ableitung über der gewichteten Eingabe z, nach [14].

Untersuchungen für Bilderkennung konnten ELU-Netzwerke geringere Testfehler erreichen als ReLU- oder Leaky-ReLU-Netzwerke. [14]

3.3.3 Initialisierung der Gewichte

Zu Beginn des Trainings müssen die Gewichte eines neuronalen Netzwerks initialisiert werden. Die Art und Weise wie das geschieht, hat in Wechselwirkung mit den verwendeten Aktivierungsfunktionen, einen großen Einfluss auf das Training. Dies wurde vereinfacht bereits in Abschnitt 3.3.2 gezeigt. Generell geschieht die Gewichtsinitialisierung zufällig, um Symmetrien zwischen verdeckten Neuronen einer Schicht aufzubrechen. Symmetrie beschreibt den Fall, dass Neuronen einer Schicht gleiche Eingabe- und Ausgabegewichte haben: Die berechneten Gradienten der Neuronen sind dann gleich und somit auch die Gewichtsanpassungen. Die Neuronen bleiben identisch und Netzwerkkapazität wird verschwendet. [8] Die Standardstrategien zur Gewichtsinitialisierung verwenden typischerweise gleich- oder normalverteilte Zufallszahlen. Im Folgenden sollen einige dieser Standardstrategien und für welche Aktivierungsfunktionen sie sich eignen beschrieben werden.

Gewichtsinitialisierung nach LeCun et al.

Um Sättigung in mehrschichtigen Netzwerken mit punktsymmetrischen Aktivierungsfunktionen (z.B. Tanh, Softsign) möglichst zu vermeiden, sollen diese Funktionen mehrheitlich in ihrem näherungsweise linearen Bereich aktiv sein. Dazu muss die Varianz der Aktivierungen von Schicht zu Schicht gleich bleiben. Dies ist verstärkt der Fall, wenn

die Gewichte jeder Schicht durch Zufallszahlen einer Verteilung mit Mittelwert 0 und Standardabweichung

$$\sigma_w = \frac{1}{\sqrt{fan_{in}}} \tag{3.30}$$

initialisiert werden. fan_{in} ist dabei die Anzahl der in die jeweilige Schicht eingehenden Verbindungen. Zusätzlich sollen die Netzeingaben auf eine Verteilung mit Mittelwert 0 und Standardabweichung 1 transformiert (normalisiert) werden. [35]

Gewichtsinitialisierung nach Glorot und Bengio

Die Bedingung, die zu der Initialisierung nach LeCun et al. führt, lässt sich um eine zweite erweitern: [20]

1. Im Vorwärtspass soll die Varianz der Aktivierungen von Schicht zu Schicht gleich bleiben.
2. Im Rückwärtspass soll die Varianz der zurückgeführten Fehler von Schicht zu Schicht gleich bleiben.

Für Netzwerke mit punktsymmetrischen Aktivierungsfunktionen (z.B. Tanh, Softsign) und der Annahme, dass die Ableitung der Funktionen um den Koordinatenursprung näherungsweise gleich 1 ist, sind beide Bedingungen im Allgemeinen nicht gleichzeitig erfüllbar. Ein Kompromiss zwischen den Bedingungen ist eine Gewichtsinitialisierung durch Zufallszahlen einer Verteilung mit Mittelwert 0 und einer Standardabweichung von: [20]

$$\sigma_w = \frac{\sqrt{2}}{\sqrt{fan_{in} + fan_{out}}} \tag{3.31}$$

fan_{in} ist dabei die Anzahl der in die jeweilige Schicht eingehenden Verbindungen fan_{out} die Anzahl der ausgehenden Verbindungen. Haben die Schichten des Netzwerks die gleiche Neuronenanzahl, ist $fan_{in} = fan_{out}$ und beide zuvor genannten Bedingungen sind erfüllt [20]. Die Initialisierung nach Glorot und Bengio entspricht dann der Initialisierung nach LeCun et al. Es sei noch angemerkt, dass die beiden letztgenannten Initialisierungen auch in Sigmoid-Netzwerken geeignet sind, um Sättigung zumindest zu verringern [42]. Die bereits angeführten Nachteile von Sigmoid-Netzwerken bleiben aber erhalten.

In Versuchen für Bilderkennung, führte die Initialisierung nach Glorot und Bengio bei Tanh-Netzwerken zu deutlich geringeren Testfehlern. Der Testfehler von Softsign-Netzwerken blieb in etwa unverändert. [20]

Gewichtsinitialisierung nach He et al.

ReLU-Funktionen können um ihren Koordinatenursprung nicht als linear mit Ableitung gleich 1 betrachtet werden. Die Herleitung von Glorot und Bengio lässt sich aber für ReLU-Netzwerke adaptieren. Daraus ergibt sich eine Gewichtsinitialisierung durch Zufallszahlen einer Verteilung mit Mittelwert 0 und einer Standardabweichung von: [23]

$$\sigma_w = \frac{\sqrt{2}}{\sqrt{fan_{in}}} \tag{3.32}$$

In Versuchen für Bilderkennung führte die Initialisierung nach He et al. bei ReLU-Netzwerken zu schnellerer Konvergenz als die Initialisierung nach Glorot und Bengio. Die Testfehler waren nicht signifikant geringer. Gezeigt wurde das für ein sehr tiefes Netzwerk mit 22 Schichten. [23]

3.3.4 Initialisierung der Bias

Die Bias eines neuronalen Netzwerkes müssen, ebenso wie die Gewichte eines neuronalen Netzwerks, zu Beginn des Trainings initialisiert werden. Sie können mit 0 initialisiert werden, was im Allgemeinen keinen Einfluss auf den Trainingserfolg hat. [8]

3.3.5 Wahl des Optimierers

Es gibt vielfältige Erscheinungen, die die Optimierung der inneren Parameter von neuronalen Netzen, das Training, erschweren. Es handelt sich hierbei um nicht-konvexe Optimierung. Das bedeutet, dass die Kostenfunktion lokale Minima mit, im Vergleich zum globalen Minimum, hohen Kosten aufweisen kann[1]. Da der Gradient in einem lokalen Minimum sehr klein wird, kann es für den Trainingsalgorithmus schwer sein, dieses wieder zu verlassen. Zusätzlich können Sattelpunkte auftreten, in deren Nähe der Gradient ebenso fast verschwindet. Sattelpunkte können jedoch i.A. von einem Trainingsalgorithmus leichter durchschritten werden als lokale Minima. Mit steigender Parameteranzahl (steigender Dimensionalität der Kostenfunktion) treten, im Vergleich zu den lokalen Minima, vermehrt Sattelpunkte auf. Dies könnte eine Erklärung dafür sein, dass die Optimierung neuronaler Netze – trotz ihrer oft enorm hohen Parameteranzahl – in vielen Fällen zufriedenstellend funktioniert. [22]

Um die Optimierung zu beschleunigen und Bereiche mit geringeren Kosten zu erreichen, werden regelmäßig neue Optimierungsalgorithmen entwickelt und vorgeschlagen. Drei weitverbreitete Standardalgorithmen werden nachfolgend angeführt.

[1] Bei konvexer Optimierung ist jedes lokale Minimum garantiert ein globales Minimum.

Gradientenabstieg mit Impuls

Das Verfahren des steilsten Abstiegs, bereits in Gleichung (3.7) bzw. (3.8) angegeben, kann langsam sein. Dies ist zum Beispiel der Fall, wenn die Gradienten über einen Bereich der Kostenfunktion sehr klein sind oder stark rauschen. Ersteres führt zu sehr kleinen Schritten, Letzteres zu zickzack-artigen Sprüngen im Parameterraum. Lokale Minima können unter Umständen so nicht verlassen werden. Um dieses Verhalten zu vermeiden, wird ein gleitender Mittelwert vergangener Gradienten gebildet und damit der Gradientenabstieg durchgeführt. Der Einfluss vergangener Gradienten nimmt dabei Schritt für Schritt exponentiell ab. Die Anpassung der Parameter erfolgt nach: [22, 19]

$$\boldsymbol{m} \leftarrow \alpha \boldsymbol{m} - \eta \nabla_\theta C \tag{3.33}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{m} \tag{3.34}$$

Der Vektor $\boldsymbol{m}$ ist der gleitende Mittelwert der Gradienten und $0 \leq \alpha < 1$ der Impulsparameter. Im Vektor $\boldsymbol{\theta}$ sind alle Gewichte und Bias zusammengefasst. C ist die Kostenfunktion und η die Lernrate. Die Bezeichnung Impuls kommt aus einer Analogie zur Physik: Wenn der Gradient als Kraft betrachtet wird, die auf ein Teilchen mit Einheitsmasse wirkt, kann $\boldsymbol{m}$ als der Impuls (momentum) dieses Teilchens betrachtet werden[1]. Das Verfahren wird deshalb als Gradientenabstieg mit Impuls (gradient descent with momentum) bezeichnet. Um die Funktionsweise des Algorithmus zu verdeutlichen, sei angenommen, dass der Gradient $\nabla_\theta C$ überall konstant gleich $\boldsymbol{k}$ ist. Der Algorithmus beschleunigt dann in Richtung $-\boldsymbol{k}$ auf die Endgeschwindigkeit:

$$\frac{\eta \|\boldsymbol{k}\|_2}{1-\alpha}$$

Ist beispielsweise $\alpha = 0.9$, dann ist ein Schritt zehnmal so groß wie ein Schritt durch den Gradientenabstieg ohne Impuls. [22]

RMSprop (root mean square propagation)

Sehr große Gradienten können dazu führen, dass die Parameteranpassung weit über das Ziel hinaus schießt [22]. Exemplarisch ist das dargestellt in Abbildung 3.23.

Hinzu kommt noch, dass der in einem Punkt im Parameterraum berechnete Gradient meist nicht genau in Richtung des Minimums zeigt. In Abbildung 3.24 ist der Verlauf einer quadratischen Kostenfunktion, die von zwei Parametern abhängt, in der Draufsicht

[1] In der Form $\boldsymbol{m} \leftarrow \alpha \boldsymbol{m} + (1-\alpha)\nabla_\theta C$ ist der Impuls eine Näherung für das erste Moment des Gradienten (first moment of the gradient). Das erste Moment, ein Begriff aus der Stochastik, beschreibt den Erwartungswert einer Zufallsvariablen.

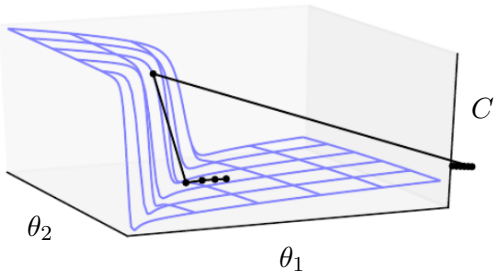


Abbildung 3.23: Zu großer Schritt in der Parameteranpassung durch eine steile Region der Kostenfunktion mit großen Gradienten. Zweidimensionaler Fall mit angepasster Achsenbeschriftung aus [22].

dargestellt. Der in einem Punkt des Parameterraumes berechnete Gradient (gestrichelter Pfeil) zeigt nicht genau in Richtung des Minimums. Der durch eine geeignete Anpassung transformierte Gradient (durchgezogener Pfeil) jedoch schon. Die in Abbildung 3.24 gezeigte Kostenfunktion hat die Form eines elliptischen Paraboloids. Die Form ist auch für die Kostenfunktion von neuronalen Netzen mit vielen Parametern, lokal eine gute Näherung. [24]

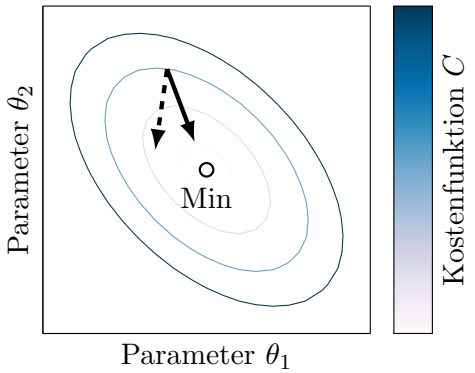


Abbildung 3.24: Der Gradient (gestrichelter Pfeil) zeigt weit am Minimum vorbei. Der durch adaptive Schrittweiten transformierte Gradient (durchgezogener Pfeil) zeigt mehr in Richtung Minimum. Zweidimensionaler Fall, Beträge nicht maßstäblich, nach [24].

Aufgrund der zuvor genannten Phänomene können Lernalgorithmen mit folgendem Verhalten vorteilhaft sein: [24]

1. Ausführung großer Schrittweiten in Richtung kleiner aber konsistenter Gradienten
2. Ausführung kleiner Schrittweiten in Richtung großer, rauschender Gradienten
3. Separate Anpassung der Schrittweite für jeden Parameter

Solche Verfahren werden als Verfahren mit adaptiver Schrittweite bezeichnet. Der RMSprop-Algorithmus ist ein Vertreter dieser Verfahren. Er bildet einen gleitenden Mittelwert des quadrierten Gradienten. Bereits einzelne, hohe Gradienten führen dazu, dass dieser Mittelwert groß wird (vergleichbar mit der mittleren quadratischen Abweichung, die sehr empfindlich auf Ausreißer reagiert). Der aktuelle Gradient wird dann für die Parameteranpassung durch die Wurzel dieses Mittelwerts dividiert. Die Schrittweite wird außerdem für jeden Parameter, d.h. für jedes Gewicht und jeden Bias, einzeln angepasst. Die Anpassungsvorschrift des RMSprop-Algorithmus lautet: [24, 19]

$$\boldsymbol{s} \leftarrow \beta \boldsymbol{s} + (1-\beta)(\nabla_\theta C)^2 \tag{3.35}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_\theta C \oslash \sqrt{\boldsymbol{s} + \boldsymbol{\epsilon}} \tag{3.36}$$

Der Vektor $\boldsymbol{s}$ ist der gleitende Mittelwert des quadrierten Gradienten und ist eine Näherung für dessen zweites Moment[1] (second moment of the gradient). $0 \leq \beta < 1$ ist die Abklingrate. Ein typischer Wert für β ist 0.9. Im Vektor $\boldsymbol{\theta}$ sind wieder alle Gewichte und Bias zusammengefasst. Die Wurzel und das Quadrat sind als elementweise Operation zu verstehen, $\oslash$ ist die elementweise Division. $\boldsymbol{\epsilon}$ verhindert Division durch 0 und hat Einträge $\ll 1$, zum Beispiel $\epsilon_i = 1 \cdot 10^{-8}$. [19]

Adam (adaptive moment estimation)

Der Adam-Algorithmus basiert auf verschiedenen Verfahren [31], vereinfacht beschrieben ist er eine Kombination aus dem Gradientenabstieg mit Impuls und dem RMSprop-Algorithmus [19]. Die Parameteranpassung basiert auf Näherungen des ersten und zweiten Moments des Gradienten. Der Adam-Algorithmus lässt sich in fünf Anpassungsvorschriften zusammenfassen: [31]

$$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1-\beta_1)\nabla_\theta C \tag{3.37}$$

$$\boldsymbol{s} \leftarrow \beta_2 \boldsymbol{s} + (1-\beta_2)(\nabla_\theta C)^2 \tag{3.38}$$

$$\boldsymbol{m} \leftarrow \frac{\boldsymbol{m}}{1-\beta_1} \tag{3.39}$$

$$\boldsymbol{s} \leftarrow \frac{\boldsymbol{s}}{1-\beta_2} \tag{3.40}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{m} \oslash \sqrt{\boldsymbol{s} + \boldsymbol{\epsilon}} \tag{3.41}$$

Gleichungen (3.37) und (3.38) beschreiben die Anpassungsvorschrift für die Näherungen des ersten und des zweiten Moments des Gradienten. Gleichungen (3.39) und (3.40) sind

[1] Das zweite Moment, ein Begriff aus der Stochastik, beschreibt die Varianz einer Zufallsvariablen.

Korrekturen, um die Optimierung, insbesondere zu Beginn des Trainings, zu beschleunigen. Gleichung (3.41) beschreibt schlussendlich die Anpassungsvorschrift der Parameter. $0 \leq \beta_1 < 1$ und $0 \leq \beta_2 < 1$ sind die Abklingraten der Impulse und η ist die Lernrate. Im Vektor $\boldsymbol{\theta}$ sind wieder alle Gewichte und Bias zusammengefasst. Die Wurzel und das Quadrat sind als elementweise Operation zu verstehen, $\oslash$ ist die elementweise Division. $\boldsymbol{\epsilon}$ verhindert Division durch 0 und hat Einträge $\ll 1$, zum Beispiel $\epsilon_i = 1 \cdot 10^{-8}$. [31]

Auf die vielfältigen theoretischen Betrachtungen, denen der Adam-Algorithmus zugrunde liegt, soll an dieser Stelle nicht näher eingegangen werden. In Untersuchungen für Bilderkennung konvergierte der Adam-Algorithmus schneller als verschiedene, konkurrierende Verfahren. Der Algorithmus ist effizient und braucht wenig Speicher. [31]

3.3.6 Parameter des Optimierers

Optimierer für neuronale Netze haben, abhängig vom Algorithmus, einen oder mehrere Einstellungsparameter. Diese Einstellungsparameter können den Hyperparametern der Netzmodelle zugerechnet werden. Im Folgenden wird auf drei wichtige Hyperparameter eingegangen, die alle gängigen Optimierer aufweisen.

Die Lernrate

Die Lernrate ist für viele Maschinenlernmodelle einer der wichtigsten Hyperparameter. Wird die Lernrate zu klein gewählt, dauert die Optimierung zu lange. Wird sie zu groß gewählt, wird die Optimierung instabil oder springt in zu weitem Abstand um das Minimum herum. Verfahren mit adaptiver Schrittweite, wie RMSprop oder Adam, sind etwas robuster gegenüber der Wahl der Lernrate als Verfahren mit konstanter Schrittweite. [8, 19]

Die Batchgröße

Die Batchgröße legt fest, wie viele Trainingsbeispiele in einem Batch zusammengefasst werden, vgl. Abschnitt 3.2.3. Die Berechnung der Trainingskosten des Netzwerkes, sowie das Parameterupdate durch den Gradientenabstieg, werden pro Batch nur einmal durchgeführt.

Theoretisch betrachtet konvergiert der stochastische Gradientenabstieg, bezogen auf die Rechenzeit, am schnellsten [60]. Dieser hat die Batchgröße $n_{batch} = 1$. Mit steigender Batchgröße steigt zwar die Genauigkeit der Näherung des Gradienten mit $\frac{1}{\sqrt{n_{batch}}}$, die benötigte Rechenleistung pro Iteration steigt aber linear mit n_{batch} [22]. Zusätzlich

reduziert eine steigende Batchgröße das Rauschen in der Gradientennäherung, was das Verlassen spitzer lokaler Minima erschweren kann [30]. Große Batches, vor allem der Batch-Gradientenabstieg, sollten aus diesen Gründen vermieden werden.

Praktisch betrachtet, also bei programmtechnischer Umsetzung, ermöglicht die Verwendung moderater Batchgrößen eine Erhöhung der Rechengeschwindigkeit durch Einsatz von Matrix-Matrix-Multiplikationen anstatt Matrix-Vektor-Multiplikationen [8] und kann zudem Mehrkernarchitekturen wie moderne CPUs und insbesondere GPUs höher auslasten [22].

Zusammengefasst deutet all das darauf hin, dass es für viele Problemstellungen eine optimale Batchgröße > 1 gibt, die unter Umständen auch von der genutzten Hardware abhängt. Der Minibatch-Gradientenabstieg findet deshalb weite Verbreitung für Maschinenlern-Anwendungen.

Abschließend sei noch ein Trick erwähnt, der den Ausnutzungsgrad unterforderter Mehrkernarchitekturen vergrößert. Wird die Batchgröße erhöht und dazu proportional die Lernrate, ändert sich das Lernverhalten eines Optimierungsalgorithmus über der Epochenanzahl kaum. Gültig ist das für einen Trainingsdatensatz der Größe n_{train} jedoch nur, solange $n_{batch} \ll n_{train}$[1]. Konvergenz kann so unter Umständen mit einer erheblich geringeren Anzahl an Iterationen erreicht werden. [54]

Der Lernratenverlauf

Bei Verwendung des stochastischen Gradientenabstiegs oder des Minibatch-Gradientenabstiegs wird der Gradient nur genähert, vgl. Abschnitt 3.2.3. Durch das Rauschen der Näherung, verursacht durch die unterschiedlichen Batches, wird der angenäherte Gradient selbst direkt im Minimum niemals 0. Das Erreichen des Minimums wird dadurch erschwert. Eine mögliche Abhilfe dafür ist eine Verringerung der Lernrate im Verlauf der Optimierung. Beispielsweise kann eine Anfangslernrate η_0 im Verlauf der Optimierung verringert werden, bis sie ab Epoche τ der Lernrate η_τ entspricht: [22]

$$\eta = \left(1 - \frac{e}{\tau}\right)\eta_0 + \frac{e}{\tau}\eta_\tau \tag{3.42}$$

e ist darin die aktuelle Epochenzahl. Es gibt viele weitere Möglichkeiten den Verlauf der Lernrate festzulegen, auf die in dieser Arbeit aber nicht weiter eingegangen wird. Anstatt die Lernrate η abklingen zu lassen, kann in vielen Fällen auch die Batchgröße n_{batch} erhöht werden. Das Rauschen der Gradientennäherung nimmt dann ab, was zu einem ähnlichen Verhalten führt wie eine Reduktion der Lernrate. Das Erreichen des Minimums wird erleichtert. Gültig ist das für einen Trainingsdatensatz der Größe n_{train} ungefähr solange $n_{batch} < \frac{n_{train}}{10}$. [54]

[1] Der Trick wurde empirisch bestätigt für $n_{batch} < \frac{n_{train}}{10}$ [54].

3.3.7 Wahl der Kostenfunktion

Die Wahl der Kostenfunktion hängt in erster Linie von der Problemstellung selbst ab. So werden für Regression typischerweise andere Kostenfunktionen verwendet, als für Klassifikation. In diesem Abschnitt wird eine kleine Auswahl an Kostenfunktionen für Regression vorgestellt.

Zuerst wird die mittlere absolute Abweichung (mean absolute error, MAE) als Kostenfunktion behandelt. Für einen Batch der Größe n_{batch} (d.h. einen Batch, der n_{batch} Beispiele enthält) wird sie wie folgt berechnet:

$$C_{MAE}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \|\boldsymbol{y}_i(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{g}_i(\boldsymbol{x})\|_1 \tag{3.43}$$

$\boldsymbol{g}_i(\boldsymbol{x})$ ist darin die gewünschte Ausgabe und $\boldsymbol{y}_i(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b})$ die Netzausgabe des Beispiels i. $\| \ \|_1$ ist die Summennorm[1]. Eine weitere mögliche Kostenfunktion ist die mittlere quadratische Abweichung. Sie wurde bereits in Gleichung (3.9) angegeben, soll hier aber erneut aufgeführt werden:

$$C_{MSE}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \|\boldsymbol{y}_i(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{g}_i(\boldsymbol{x})\|_2^2 \tag{3.44}$$

Die darin vorkommenden Größen sind dieselben wie in Gleichung (3.43). $\| \ \|_2$ ist die euklidische Norm[2]. Die mittlere quadratische Abweichung reagiert empfindlich auf Ausreißer (einzelne Datenpunkte mit großer Abweichung). Grund dafür sind die in der Gleichung auftretenden quadratischen Terme. Manchmal wird auch die Wurzel der mittleren quadratischen Abweichung als Kostenfunktion verwendet (root mean squared error, RMSE).

Abschließend noch ein Hinweis: In manchen Maschinenlern-Bibliotheken wird die Kostenfunktion nicht nur über den Batch, sondern auch über die Einträge des Ausgabevektors gemittelt. Die mittlere quadratische Abweichung, vgl. Gleichung (3.44), wird dann beispielsweise nach

$$C_{MSE}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \frac{1}{n_{batch}} \sum_{i=1}^{n_{batch}} \frac{1}{n_A} \|\boldsymbol{y}_i(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) - \boldsymbol{g}_i(\boldsymbol{x})\|_2^2 \tag{3.45}$$

berechnet. n_A ist dabei die Anzahl der Einträge des Ausgabevektors (für alle $\boldsymbol{g}_i$ bzw. $\boldsymbol{y}_i$ gleich) und entspricht der Anzahl der Ausgabeneuronen des Netzwerks.

[1] Die Summennorm ist definiert als $\|\boldsymbol{x}\|_1 := \sum_{i=1}^{n} |x_i|$.

[2] Die euklidische Norm ist definiert als $\|\boldsymbol{x}\|_2 := \sqrt{\sum_{i=1}^{n} |x_i|^2}$.

3.3.8 Regularisierung

Neuronale Netzwerke weisen, im Vergleich zu anderen Maschinenlernmodellen, meist sehr viele Parameter auf. Deshalb neigen sie zu Überanpassung, vgl. Abschnitt 3.2.4. Methoden, die dazu dienen, Überanpassung zu vermeiden, werden unter dem Begriff Regularisierung zusammengefasst. [19] Anders beschrieben, zielen die Methoden darauf ab, den Testfehler eines Modells zu verringern. Dabei wird in Kauf genommen, dass sich der Trainingsfehler des Modells möglicherweise erhöht. [22] Nachstehend sollen zwei in der Masterarbeit verwendete Regularisierungsmethoden beschrieben werden.

Early Stopping des Trainings

Das vorzeitige Beenden des Trainings (engl. Early Stopping) ist eine Regularisierungsmethode, die bei künstlichen neuronalen Netzen häufig Anwendung findet. Für Early Stopping werden mindestens zwei Datensätze benötigt: die Trainings- und die Stoppdaten. Soll das fertig trainierte Modell auch auf seine Generalisierungsfähigkeit getestet werden, wird zusätzlich noch ein Testdatensatz benötigt. Nun soll die Funktionsweise des Early Stoppings beschrieben werden: Das Netzwerk wird mit den Trainingsdaten trainiert. Währenddessen wird mit den Stoppdaten laufend der Stoppfehler des Netzwerkes ermittelt. Hört der Stoppfehler auf zu sinken, oder beginnt anzusteigen, wird das Training beendet. [19, 42, 22] Ein konkretes Beispiel für den Verlauf von Trainings- und Stoppfehler während des Trainings eines Netzwerks, zeigt Abbildung 3.25.

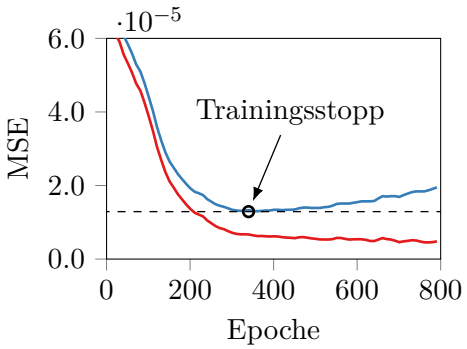


Abbildung 3.25: Verlauf von —— Trainingsfehler und —— Stoppfehler beim Early Stopping. Erstellt mit eigenen Daten, nach [19].

Ab einem gewissen Trainingsfortschritt beginnt der Stoppfehler (——) anzusteigen. Das Training wird zu diesem Zeitpunkt beendet, obwohl der Trainingsfehler (——) bei Fortsetzung des Trainings weiter sinken würde. Das Training wird also beendet, wenn die

Generalisierungsfähigkeit des Netzwerkes, ausgedrückt durch den Stoppfehler, schlechter wird.

Die Fehler eines Netzwerkes schwanken während des Trainings. Das Training würde somit, durch das zuvor beschriebene Early Stopping, beim ersten kleinen Anstieg des Stoppfehlers beendet werden. Lokale Minima im Fehlerverlauf könnten so nicht verlassen werden. Um das zu umgehen, wird Early Stopping mit Geduld verwendet. Die Geduld wird dabei in Epochen oder Iterationen angegeben. Der zuletzt erreichte Minimalwert des Stoppfehlers wird laufend im Training gespeichert und aktualisiert. Erst wenn der Minimalwert innerhalb der Geduld nicht mehr unterschritten wird, wird das Training beendet. Nach Beendigung des Trainings wird das Netzwerk üblicherweise auf die Netzparameter (Gewichte und Bias) zurückgesetzt, mit denen der geringste Stoppfehler erreicht werden konnte. [19, 42, 22]

Um ein Ende des Trainings zu garantieren, wird eine minimale Verbesserung und/oder eine maximale Epochenanzahl für das Early Stopping mit Geduld vorgegeben. Die minimale Verbesserung wird als Fehlerdifferenz des Stoppfehlers angegeben. Ist die Verbesserung des Stoppfehlers innerhalb der Geduld kleiner als die minimale Verbesserung, wird das Training beendet. Greift das Early Stopping nicht, wird das Training spätestens bei Erreichen der maximalen Epochenanzahl beendet. [46, 29]

Ensemblebildung

Ensemblemethoden werden zur Regularisierung im Maschinenlernen verwendet. Ein Ensemble kombiniert verschiedene Maschinenlernmodelle, um Vorhersagen zu treffen. Die Modelle werden dabei oft mit verschiedenen Datensätzen trainiert. Eine Ensemblemethode, zum Beispiel in der Anwendung für Bilderkennung, ist die Modellmittelung (engl. model averaging). Mehrere, separat trainierte Modelle treffen dabei jeweils eine Vorhersage für ein Bild, zum Beispiel „Hund“ oder „Katze“. Die häufigste Vorhersage wird gewählt und vom Modellensemble ausgegeben. [22]

Bei neuronalen Netzen wird der Trainingsverlauf durch Zufallszahlen beeinflusst. Beispielsweise werden die Gewichte zu Beginn des Trainings zufallsbasiert initialisiert. Bei Verwendung von Mini-Batches werden die Daten außerdem zu Beginn jeder Epoche, vor Bildung der Mini-Batches, zufallsbasiert gemischt. Eine Ensemblebildung kann deshalb auch sinnvoll sein, wenn die Modelle mit den gleichen Datensätzen bzw. den gleichen Hyperparametern trainiert werden. [22] Dieser Ansatz wurde in der vorliegenden Masterarbeit, in Kombination mit der Modellmittelung, verwendet. Die Ausgabe des Ensembles $\boldsymbol{y}_{Ens}$ wurde durch den Mittelwert der Ausgaben $\boldsymbol{y}_i$ der n Einzelmodelle berechnet:

$$\boldsymbol{y}_{Ens} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i \tag{3.46}$$

Es sei noch Folgendes angemerkt: Eine Zusammenfassung der n Modelle in einem einzigen Modell – durch Mittelwertbildung der Gewichte und Bias – ist als Alternativansatz, aufgrund der Nichtlinearitäten in neuronalen Netzen, nicht möglich.

4 Entwicklung der Prozeduren zur Modellerstellung

Bei Maschinenlernmodellen handelt es sich um selbstlernende Systeme. Um den großen Vorteil des selbstständigen Lernens nicht zunichte zu machen, werden zumeist weitgehend automatisierte Prozeduren zur Modellerstellung bzw. Hyperparameteroptimierung verwendet. Für diese Masterarbeit wurden drei Prozeduren, die auf den in Kapitel 3 erarbeiteten Grundlagen sowie eigenen Überlegungen basieren, entwickelt und programmiert. Bereits in Maschinenlernbibliotheken vorhandene Gesamtlösungen wurden aufgrund diverser Einschränkungen nicht verwendet.

Zu Beginn des Kapitels werden die verwendete Software und Hardware angeführt. Danach werden die drei Prozeduren bzw. Programme zur Modellerstellung beschrieben. Es werden die wesentlichen Programmabläufe und die zugrundeliegenden Überlegungen behandelt – auf die Darstellung umfangreicher Programmablaufpläne oder auf Auszüge aus dem Programmcode wird dabei jedoch verzichtet. Abschließend werden noch Benchmark-Ergebnisse, zum Training neuronaler Netze auf Prozessor und Grafikkarte, gezeigt.

4.1 Verwendete Software

Alle Programme wurden in der Programmiersprache Python3 geschrieben. Die neuronalen Netze wurden mit der Bibliothek TensorFlow über die integrierte High-Level API Keras erstellt. TensorFlow ist eine quelloffene Bibliothek für numerische Berechnungen und wird vorwiegend für die Ausführung von Maschinenlernalgorithmen verwendet. Keras ist eine Bibliothek für die Erstellung neuronaler Netze. Als High-Level API in TensorFlow erleichtert sie die Erstellung gängiger Netztypen enorm. Die verwendeten Versionen sind nachfolgend aufgelistet:

- Python 3.5.2
- TensorFlow 1.5.0 [57]
- Keras 2.1.2 (als TensorFlow High-Level API) [57, 29]

Zur Unterstützung der Datenverarbeitung und Auswertung wurden zusätzlich folgende Bibliotheken verwendet:

- Pandas

- NumPy
- Scikit-learn
- Matplotlib
- Seaborn

4.2 Verwendete Hardware

Zur Ausführung der Berechnungen wurden zwei Desktop-Rechner verwendet: „Desktop i7“ verfügte über einen schnellen[1] Vierkernprozessor mit Hyperthreading, „Desktop Xeon“ verfügte hingegen über einen langsameren Vierkernprozessor ohne Hyperthreading, dafür aber über eine schnelle[2] Grafikkarte. Die Spezifikationen beider Rechner sind nachfolgend aufgelistet:

Desktop i7

- Linux Ubuntu
- Core i7-6700K @ 4.3 GHz, 4 Kerne mit HT
- 16 GiB RAM

Desktop Xeon

- Linux Xubuntu
- Xeon X3360 @ 2.83 GHz, 4 Kerne ohne HT
- 8 GiB RAM
- GTX Titan X, 12 GiB GDDR5

4.3 Modellerstellung mit zufallsbasierter Datenteilung

Die Modellerstellung mit zufallsbasierter Datenteilung besteht im Wesentlichen aus zwei Teilen: Im ersten Teil werden die Hyperparameter optimiert und im zweiten wird das finale Modellensemble erstellt. Den grundlegenden Programmablauf zeigt Abbildung 4.2, wobei der Datenfluss vorerst ausgeklammert wird.

Die linke Seite des Ablaufplans beschreibt die Optimierung der Hyperparameter (HP). Nach Start des Programms beginnt die äußere Schleife mit der Auswahl einer HP-Kombination durch die Zufalls- oder die Rastersuche (vgl. Abschnitt 3.3.1). Mit den ausgewählten Parametern werden dann, in einer inneren Schleife, wiederholt Modelle trainiert und nach Stopp des Trainings getestet. Nach Ende der letzten Wiederholung wird der Mittelwert der ermittelten Testfehler gebildet. Sind weitere Parameterkombinationen

[1] Zum Zeitpunkt der Verfassung der Arbeit.
[2] Ebenso.

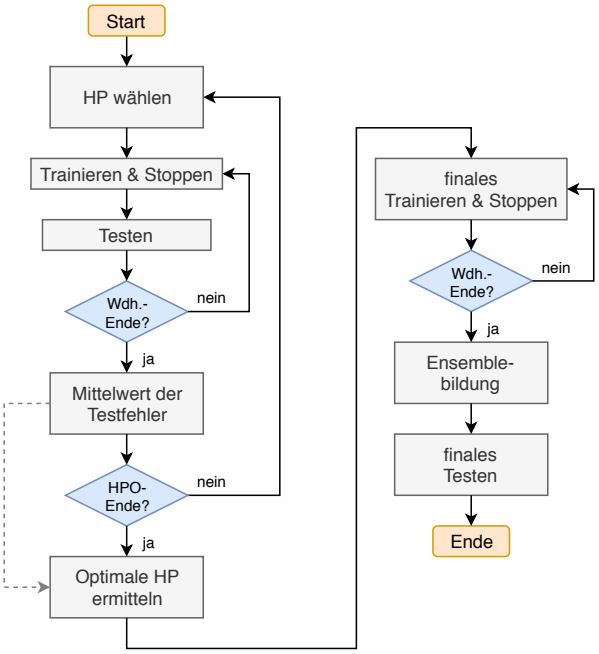


Abbildung 4.1: Vereinfachter Programmablaufplan der Modellerstellung mit zufallsbasierter Datenteilung.

zu testen, beginnt die äußere Schleife erneut. Am Ende der HP-Optimierung wird die Parameterkombination mit dem kleinsten mittleren Testfehler ermittelt und als Optimum festgelegt. In der rechten Seite des Ablaufplans wird das finale Modellensemble (vgl. Abschnitt 3.3.8) erstellt. Mit den optimalen HP werden, erneut in einer Schleife, wiederholt Modelle trainiert und nach Stopp des Trainings getestet. Aus diesen Modellen wird dann das finale Modellensemble gebildet, das seine Ausgabe als Mittelwert der Ausgaben der Einzelmodelle berechnet, vgl. Gleichung (3.46). Abschließend wird das Ensemble in einem finalen Test getestet.

Die Anzahl der durch die Zufalls- oder die Rastersuche ausgewählten HP-Kombinationen wird vom Entwickler vorgegeben, ebenso die Anzahl der Wiederholungen in der HP-Optimierung und für die Ensemblebildung. Der Stopp der Trainingsvorgänge erfolgt in allen Fällen durch Early Stopping (vgl. Abschnitt 3.3.8). Die Wiederholungen in der HP-Optimierung sind notwendig, da die Gewichte der neuronalen Netze zu Trainingsbeginn zufallsbasiert initialisiert werden. Diese Initialisierung hat Einfluss auf den Verlauf des Trainings und somit auch auf dessen Stopp. Durch die Wiederholungen soll vermieden werden, dass einzelne „glückliche“ oder „unglückliche“ Initialisierungen das Optimierungsergebnis beeinflussen. Die zur Ensemblebildung notwendigen Wiederholungsrechnungen haben denselben Zweck. Die Ensemblebildung ist außerdem eine Regularisierungsmethode, d.h. sie dient der Vermeidung von Überanpassung, vgl. Ab-

schnitt 3.3.8. Werden p Parameterkombinationen vorgegeben, w_{HPO} Wiederholungen in der HP-Optimierung und w_E Wiederholungen zur Ensemblebildung durchgeführt, werden während der Modellerstellung insgesamt

$$n_M = p \cdot w_{HPO} + w_E \tag{4.1}$$

Modelle trainiert und getestet. Dies erlaubt eine Abschätzung der benötigten Rechenzeit. Es sollen nun die Datenaufteilung und -verwendung näher beschrieben werden, die in Abbildung 4.2 dargestellt sind.

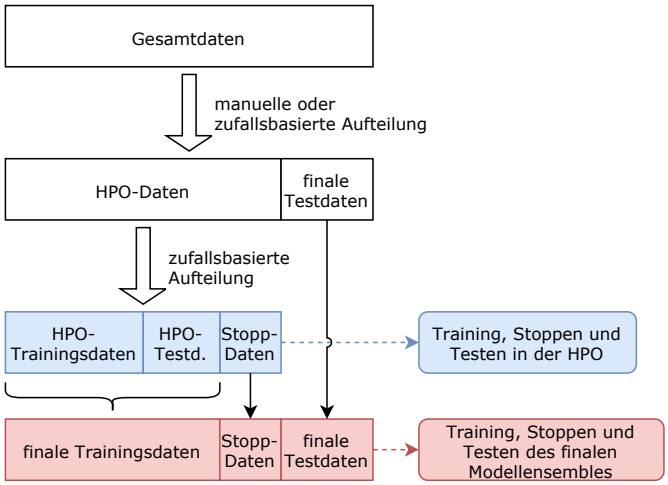


Abbildung 4.2: Datenaufteilung und -verwendung bei der Modellerstellung mit zufallsbasierter Datenteilung.

Die zur Verfügung stehenden Gesamtdaten, zum Beispiel Messdaten, werden in einem ersten Schritt manuell oder zufallsbasiert in HPO-Daten und finale Testdaten aufgeteilt. Letztere werden vorerst einbehalten, erstere zufallsbasiert in drei weitere Datensätze aufgeteilt: die HPO-Trainingsdaten, die HPO-Testdaten und die Stoppdaten. Diese drei Datensätze werden, wie ihre Bezeichnung bereits verrät, zum Training, Stopp des Trainings und Testen in der HP-Optimierung verwendet (in Abbildung 4.2 blau dargestellt). Die HPO-Testdaten werden vom Modell in der HP-Optimierung, die in gewisser Weise ein Trainingsvorgang ist, vom Modell „gesehen“: Der mit ihnen berechnete Testfehler wird zur Ermittlung der optimalen HP-Kombination verwendet. Die resultierende optimale Kombination ist deshalb auf diese Daten tendenziell überangepasst. Für den abschließenden Test, mit dem die Generalisierungsfähigkeit des finalen Modells abgeschätzt werden soll, sollen diese Daten deshalb nicht verwendet werden. Das ist der Grund, warum in der ersten Aufteilung die finalen Testdaten erstellt wurden. Um die HPO-Testdaten bei der Bildung des finalen Modellensembles jedoch nicht ungenutzt zu lassen, werden sie nach

der HP-Optimierung mit den HPO-Trainingsdaten in finale Trainingsdaten zusammengefasst. Diese, die Stoppdaten und die zu Beginn einbehaltenen finalen Testdaten werden abschließend zur Erstellung und zum Testen des finalen Modellensembles verwendet (in Abbildung 4.2 rot dargestellt). Genau genommen werden auch die Stoppdaten in der HP-Optimierung vom Modell „gesehen“. Der mit ihnen berechnete Fehler wird jedoch nie direkt, weder im Training noch in der HP-Optimierung, zur Auswahl von Modellparametern verwendet. Die Auswirkungen einer Überanpassung an die Stoppdaten wurden deshalb als gering eingeschätzt.

Anstatt das Training durch Early Stopping zu beenden, ist es auch möglich, die Anzahl der Trainingsepochen als Hyperparameter zu optimieren. Fasst man jedoch HPO-Trainings- und Testdaten zu den finalen Trainingsdaten zusammen, wie zuvor beschrieben, ist Folgendes zu beachten: Die finalen Trainingsdaten beinhalten dann mehr Trainingsbeispiele als die HPO-Trainingsdaten. In einer Epoche im finalen Training werden daher mehr Iterationen ausgeführt, als in einer Epoche in der HP-Optimierung, siehe auch Gleichung (3.10). Die Anzahl der Epochen eignet sich in diesem Fall nicht als zu optimierende Größe und es ist besser, die Gesamtanzahl der Iterationen zu optimieren. Trotzdem hat diese Methode einen Nachteil: Netzwerke mit ungünstig initialisierten Gewichten werden oft langsam trainiert, d.h. bei Vorgabe der Gesamtanzahl an Iterationen werden solche Netzwerke unvollständig trainiert. Early Stopping hingegen beendet das Training im Allgemeinen erst, wenn im Training keine Verbesserung des Stoppfehlers mehr erreicht werden kann. Aus diesen Gründen wurde bei den in dieser Masterarbeit verwendeten Prozeduren das Training immer durch Early Stopping beendet.

Die zufallsbasierte Aufteilung der Datensätze läuft folgendermaßen ab: Ein Datensatz A soll in zwei Datensätze B und C aufgeteilt werden. Dazu muss für B zuerst ein Prozentsatz p_B vorgegeben werden. Die in A enthaltenen Beispiele werden dann zufällig gemischt. Aus den ersten p_B Prozent der Beispiele wird der Datensatz B gebildet. Aus den restlichen Beispielen wird Datensatz C gebildet, der somit $100 - p_B$ Prozent der Beispiele aus A enthält. Abschließend sollen noch kurz die Vor- und Nachteile der umgesetzten Datenteilung beschrieben werden: Die Aufteilung benötigt wenig Rechenleistung und ist einfach umzusetzen, da sie einmalig zu Beginn des Programms erfolgt. Die Datensätze bleiben aber während der gesamten Modellerstellung gleich, was Nachteile mit sich bringt: Sind zum Beispiel durch die zufällige Aufteilung in den Trainingsdaten wichtige Datenbeispiele nicht enthalten, hat dies einen negativen Einfluss auf die gesamte Modellerstellung. Weiters ist eine Überanpassung der Hyperparameter (und somit des finalen Modells) an die HPO-Testdaten zu erwarten. Die Gründe dafür wurden in diesem Abschnitt bereits behandelt.

4.4 Modellerstellung mit Kreuzvalidierung

Um die mit zufallsbasierter Datenteilung zu erwartende Überanpassung in der HP-Optimierung zu vermeiden, wurde die Modellerstellung auch mit Kreuzvalidierung (KV, engl. cross-validation) umgesetzt. Der grundlegende Programmablauf ist dabei gleich wie der bei der Modellerstellung mit zufallsbasierter Datenteilung, vgl. Abbildung 4.1. Die Datenaufteilung und -verwendung, dargestellt in Abbildung 4.3 unterscheiden sich jedoch.

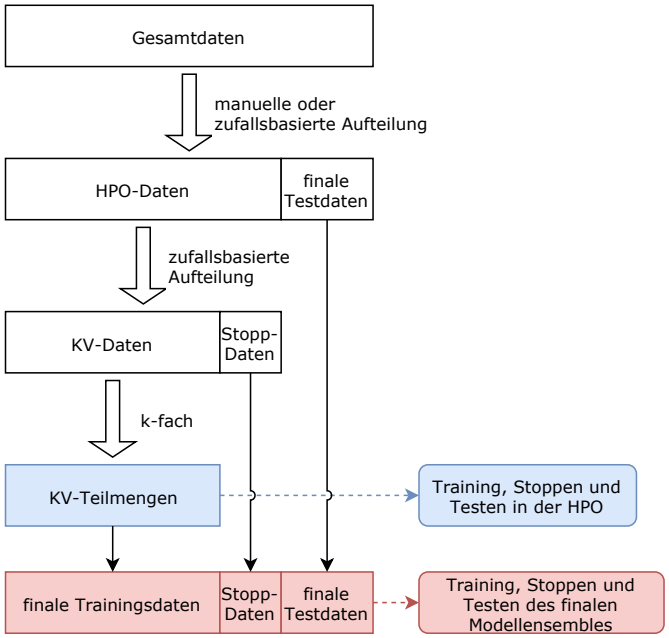


Abbildung 4.3: Datenaufteilung und -verwendung bei der Modellerstellung mit Kreuzvalidierung.

Die zur Verfügung stehenden Gesamtdaten, zum Beispiel Messdaten, werden in einem ersten Schritt manuell oder zufallsbasiert in HPO-Daten und finale Testdaten aufgeteilt. Letztere werden vorerst einbehalten, erstere in zwei weitere Datensätze aufgeteilt: die KV-Daten und die Stoppdaten. Die Stoppdaten werden auch einbehalten. Die KV-Daten werden dann k-fach in KV-Teilmengen unterteilt, die zum Training, Stopp des Trainings und Testen in der HP-Optimierung verwendet werden (in Abbildung 4.3 blau dargestellt). k entspricht der Anzahl an Wiederholungsrechnungen für jede HP-Kombination und wird ≥ 2 gewählt. Die KV-Daten (d.h. die zusammengefassten KV-Teilmengen), die Stoppdaten und die finalen Testdaten werden nach der HP-Optimierung zur Erstellung und zum Testen des finalen Modellensembles verwendet (in Abbildung 4.3 rot dargestellt). Wie die Aufteilung der KV-Daten durchgeführt wird und die Kreuzvalidierung funktioniert, soll nun im Detail betrachtet werden, siehe Abbildung 4.4.

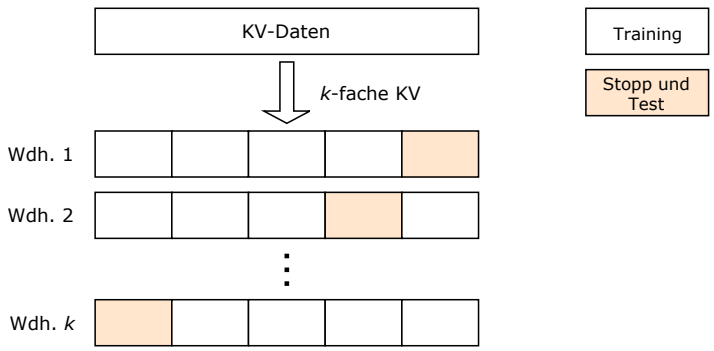


Abbildung 4.4: Aufteilung der KV-Daten in der HP-Optimierung mit Kreuzvalidierung, nach [47].

Die KV-Daten werden gemischt und in k gleich große Teilmengen aufgeteilt (k-fache Kreuzvalidierung). In jeder der k Wiederholungen wird eine andere der Teilmengen zum Stoppen und Testen verwendet, während alle übrigen Teilmengen zum Training verwendet werden. Die Teilmenge zum Stoppen und Testen wird durch eine zufallsbasierte Aufteilung weiter in zwei Datensätze, in Stopp- und Testdaten, unterteilt, was aber in Abbildung 4.4 nicht weiter dargestellt ist. Aus den Testfehlern der Wiederholungen wird der mittlere Testfehler der untersuchten HP-Kombination berechnet. Eine Überanpassung der HP auf die Testdaten ist somit unwahrscheinlich, da der mittlere Testfehler mit unterschiedlichen Datenauswahlszenarien ermittelt wird. Werden p Parameterkombinationen vorgegeben, k-fache Kreuzvalidierung in der HP-Optimierung und w_E Wiederholungen zur Ensemblebildung durchgeführt, werden während der Modellerstellung insgesamt

$$n_M = p \cdot k + w_E \tag{4.2}$$

Modelle trainiert und getestet. Die benötigte Rechenzeit der Modellerstellung mit Kreuzvalidierung beträgt, bei vergleichbaren Einstellungen, in etwa das Gleiche wie die Rechenzeit der Modellerstellung mit zufallsbasierter Datenteilung: Gleichungen (4.2) und (4.1) haben dasselbe Ergebnis, wenn $k = w_{HPO}$ ist. Die Kreuzvalidierung weist auch Nachteile auf. Die Größe der Datensätze ist abhängig von der Anzahl der Wiederholungen k. Ist k sehr groß oder klein, weisen die Datensätze Größenverhältnisse auf, die sich negativ auf das Ergebnis der Modellerstellung auswirken können.

Genau betrachtet kann es außerdem, trotz Kreuzvalidierung, zu einer Überanpassung der optimalen HP an die KV-Daten kommen: Die KV-Daten werden, es sei noch einmal auf Abbildung 4.3 hingewiesen, zufallsbasiert gewählt und bleiben dann während der gesamten Modellerstellung gleich. Die optimalen HP werden dadurch mit genau dieser Auswahl der Daten ermittelt, was zu Überanpassung führen kann. [13]

4.5 Modellerstellung mit verschachtelter Kreuzvalidierung

Bei einer HP-Optimierung durch Kreuzvalidierung kann, wie zuvor ausgeführt, Überanpassung auftreten. Diese Überanpassung tritt jedoch nicht mehr auf, wenn das Minimum des Fehlers für die Datenauswahl bei ähnlichen Hyperparametern auftritt wie das Minimum des Fehlers der – im Allgemeinen unbekannten – wahren Testdaten. Um zu erklären, wann das der Fall ist, soll noch einmal die Varianz der Näherung betrachtet werden, siehe Gleichung (3.17). Ist die Varianz niedrig, ändert sich die Näherung mit variiertem Datensatz nicht stark. Die HP, die den Fehler für eine bestimmte Datenauswahl minimieren, garantieren gute Generalisierung. Eine Methode um die Varianz in der HP-Optimierung zu untersuchen, ist die verschachtelte Kreuzvalidierung (VKV). [13] Die Datenaufteilung der VKV ist in Abbildung 4.5 dargestellt.

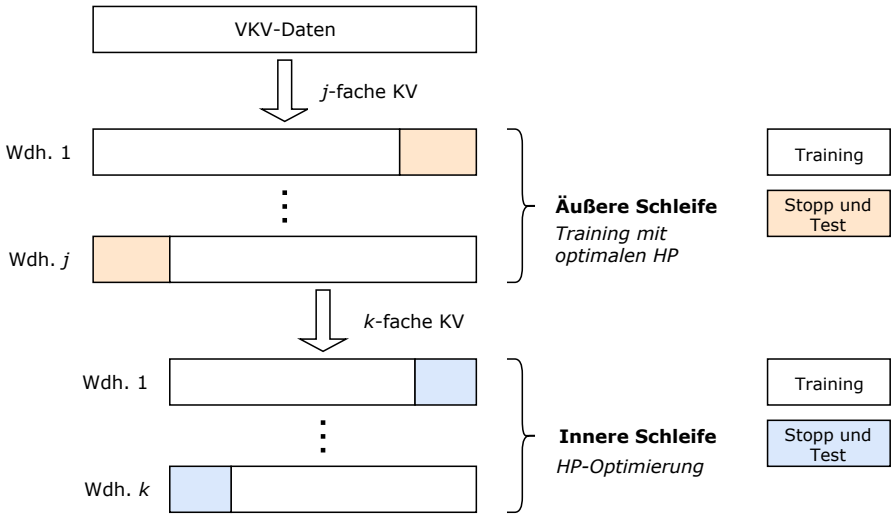


Abbildung 4.5: Aufteilung der Daten in der HP-Optimierung mit verschachtelter Kreuzvalidierung (VKV), nach [47].

Die VKV besteht aus zwei verschachtelten Schleifen: einer äußeren Schleife mit k Wiederholungen und einer inneren Schleife mit j Wiederholungen. Die innere Schleife dient der Modellerstellung, die äußere Schleife ist dazu da, um die Modellerstellung zu testen. In beiden Schleifen wird eine einfache Kreuzvalidierung durchgeführt. Eine Aufteilung der Daten ist nicht mehr notwendig, die Gesamtdaten werden als Daten für die VKV verwendet und werden als VKV-Daten bezeichnet. Die durch die Aufteilung in der äußeren Schleife erhaltenen Trainingsdaten werden in der inneren Schleife zur HP-Optimierung verwendet. Das Modell wird dann, mit den optimalen Hyperparametern aus der inneren Schleife, in der äußeren Schleife trainiert und getestet. Als Ergebnis erhält man j Testfehler sowie j optimale HP-Kombinationen. [47] Weichen die optimalen HP-Kombinationen

stark voneinander ab, ist dies ein Hinweis darauf, dass die Modellerstellung nicht wie vorgesehen funktioniert. Entweder ist die Wahl der Hyperparameter egal (was die Modellerstellung obsolet macht) oder die Hyperparameter führen zu einer eingeschränkten Generalisierungsfähigkeit des finalen Modells. Sind die Parameter ähnlich, d.h. die Varianz gering, ist die Modellerstellung geeignet, um das finale Modell zu ermitteln. Zur Ermittlung des finalen Modells werden die VKV-Daten (bzw. die Gesamtdaten) für eine HP-Optimierung durch einfache Kreuzvalidierung verwendet. Es wird also nur die innere Schleife noch einmal ausgeführt und die erhaltenen Hyperparameter sind das Optimum.

Werden p Parameterkombinationen vorgegeben, k-fache bzw. j-fache Kreuzvalidierung in der äußeren bzw. inneren Schleife und w_E Wiederholungen zur Ensemblebildung durchgeführt, werden während der Modellerstellung insgesamt

$$n_M = p \cdot j \cdot k + w_E \tag{4.3}$$

Modelle trainiert und getestet. Die meist sehr hohe Rechenzeit der Modellerstellung mit verschachtelter Kreuzvalidierung ist ein großer Nachteil dieser Methode.

4.6 Datenpipelines

Für viele Maschinenlernmodelle ist es erforderlich oder vorteilhaft, die Daten vor ihrer Verwendung zu transformieren. Im Laufe einer Modellerstellung kann es notwendig sein, viele verschiedene Datensätze denselben Transformationen zu unterziehen. Um diese Prozesse einfacher und fehlerarm umzusetzen, können sogenannte Datenpipelines verwendet werden. Für die Erstellung einer Pipeline bietet sich das sklearn.pipeline Modul von Scikit-learn an. In Kombination mit dem Modul sklearn.preprocessing, das bereits viele fertige Bausteine für Transformationen enthält, können vielseitige Pipelines erstellt und die meisten Standardanwendungen abgedeckt werden. Die beiden Module wurden auch in der vorliegenden Masterarbeit verwendet. [19, 51]

Als Beispiel soll nun eine Pipeline beschrieben werden, die Tabellendaten anhand des Namens auswählt, logarithmiert und dann linear skaliert, siehe Abbildung 4.6.

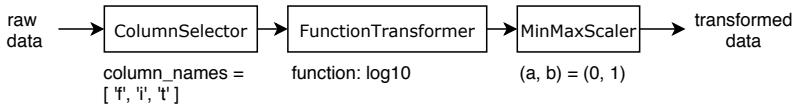


Abbildung 4.6: Beispiel für eine Datenpipeline.

Die Datenauswahl erfolgt durch die Klasse ColumnSelector[1]. Der Klasse werden vom Entwickler die Spaltennamen der zu verwendenden Eingaben vorgegeben, im Beispiel sind das f, i und t. Die Logarithmierung erfolgt durch die Klasse FunctionTransformer, der eine auf die Daten anzuwendende Funktion vorgegeben wird. Im Beispiel ist das der dekadische Logarithmus $\log_{10}$. Die Klasse MinMaxScaler skaliert die Daten schließlich linear. Vorgegeben werden die Werte a und b, die den Wertebereich festlegen, auf den die Daten transformiert werden. Im Beispiel sind das die Werte 0 und 1. Die Klasse MinMaxScaler unterscheidet sich dabei in einem wesentlichen Punkt von den beiden anderen Klassen: Die angewendete Transformation hängt von den Daten selbst ab und zwar von den auftretenden Minimal- und Maximalwerten jeder Eingabe. Die Skalierung wird für alle Werte jeder Eingabe wie folgt durchgeführt: [51]

$$x_{scaled} = \left(\frac{x - x_{min}}{x_{max} - x_{min}} \right) (b - a) + a \tag{4.4}$$

Darin ist x der zu skalierende Wert, x_{min} der Minimalwert und x_{max} der Maximalwert der Eingabe. x_{min} (bzw. x_{max}) kann für verschiedene Datensätze, aber auch für verschiedene Teile ein und desselben Datensatzes, variieren. Solche veränderlichen Parameter der Pipeline müssen auf die Datensätze angepasst werden. Pipelines haben deshalb zwei wichtige Methoden: [51]

- fit: Die Parameter der Pipeline werden auf die Daten angepasst.
- transform: Die Pipeline transformiert die Daten.

Die Methode fit wird dabei, in jedem Teil der Modellerstellung, nur auf die Trainingsdatensätze angewandt. Danach wird die Methode transform auf Trainings- und Testdaten angewandt (und auf die Stoppdaten falls vorhanden). Wird die Methode fit fälschlicherweise auf die Gesamtdaten angewendet und werden diese danach in Trainings- und Testdaten aufgeteilt, enthalten die Trainingsdaten im Allgemeinen bereits Informationen der Testdaten. Der berechnete Testfehler ist dann eine zu optimistische Schätzung des Generalisierungsfehlers. [19, 51]

4.7 Benchmarks: CPU vs. GPU für das Training

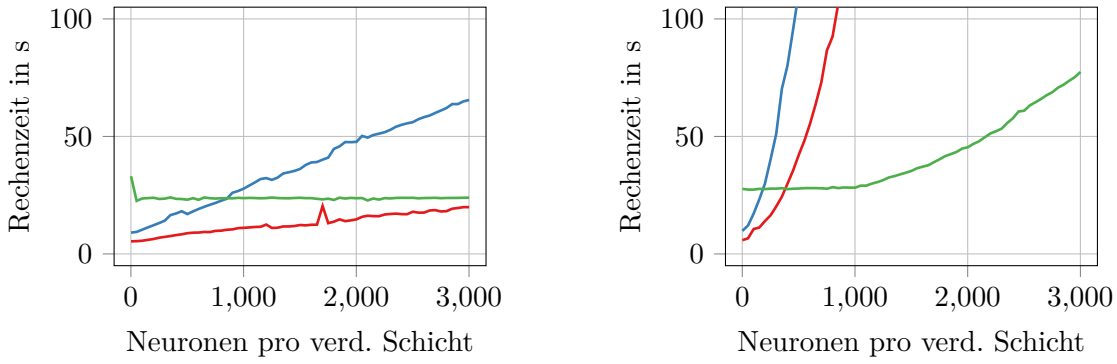
Die im Laufe des Kapitels beschriebenen Modellerstellungsprozeduren wurden mit einem einfachen Desktoprechner entwickelt und programmiert. Dies geschah in der Erwartung, die benötigten Rechenzeiten später durch Einsatz einer Grafikkarte (GeForce GTX Titan X) massiv verkürzen zu können. Nach der Fertigstellung der Programme – bei ersten

[1] Diese ist nicht Bestandteil von Scikit-learn und wurde für die Arbeit extra erstellt.

Berechnungen mit der Grafikkarte – stellte sich jedoch heraus, dass die Berechnungen sogar länger dauerten, als mit dem Prozessor des zuvor erwähnten Desktoprechners. Aufgrund dieser völlig unerwarteten Ergebnisse wurden einfache Benchmarks erstellt. Verglichen wurden zwei CPUs (i7-6700K @ 4.3 GHz, 4C/8T und Xeon X3360 @ 2.83 GHz, 4C/4T) und die bereits erwähnte GPU (GeForce GTX Titan X). Untersucht wurde der Einfluss der Schicht- und Neuronenanzahl, sowie der Batchgröße auf die benötigte Trainingszeit.

4.7.1 Einfluss der Schicht- und Neuronenanzahl

Untersucht wurde ein neuronales Netzwerk mit 4 Neuronen in der Eingabeschicht und 1 Neuron in der Ausgabeschicht. Die Anzahl der Ein- und Ausgaben war somit gleich, wie bei den später in Kapitel 5 vorgestellten Polarisationsmodellen. Der Datensatz, der für das Training in den Benchmarks verwendet wurde, umfasste 3430 Trainingsbeispiele. Die Batchgröße betrug 256. Variiert wurden die Anzahl der verdeckten Schichten und die Anzahl der Neuronen pro verdeckter Schicht. Mit jeder Konfiguration wurden 500 Epochen berechnet und die benötigte Rechenzeit aufgezeichnet. Abbildung 4.7 zeigt die Rechenzeitverläufe.



(a) 1 verdeckte Schicht.

(b) 2 verdeckte Schichten.

Abbildung 4.7: Benötigte Rechenzeiten für 500 Epochen bei Variation der Anzahl an verdeckten Schichten und der Neuronenanzahl pro verdeckter Schicht. Berechnung mit einem —— i7-6700K @ 4.3 GHz, 4C/8T, einem —— Xeon X3360 @ 2.83 GHz, 4C/4T und einer —— GeForce GTX Titan X.

Der Benchmark für eine verdeckte Schicht, Abbildung 4.7a, zeigt: Die Rechenzeit der beiden CPUs (——, ——) steigt mit der Neuronenanzahl näherungsweise linear. Das lässt sich zum Teil durch die Größe der beiden Gewichtsmatrizen des neuronalen Netzes

erklären: diese haben, bei n Neuronen in der verdeckten Schicht, die Größe $4 \times n$ bzw. $n \times 1$. Die Anzahl der Gewichte steigt somit linear mit n. Die i7-CPU (——) benötigt, wie zu erwarten, durchgehend nur die Hälfte bis ein Drittel der Rechenzeit der langsameren Xeon-CPU (——). Die Rechenzeit der GPU (——) bleibt für alle untersuchten Neuronenanzahlen in etwa gleich. Das deutet darauf hin, dass die hoch parallelisierten Strukturen[1] der GPU nicht ausgelastet werden können. Erstaunlicherweise benötigt die GPU dabei für alle untersuchten Neuronenanzahlen mehr Rechenzeit als die i7-CPU (——). Im Vergleich mit der Xeon-CPU (——) ist die GPU erst ab ca. 850 Neuronen in der verdeckten Schicht schneller. Eine mögliche Begründung für die hohen Rechenzeiten der GPU ist deren langsame Anbindung an die Rechnerarchitektur durch PCI Express. Außerdem werden nur die neuronalen Netze selbst auf der GPU berechnet, die restlichen Programmabläufe auf der CPU. Folglich müssen während der Berechnung häufig Daten zwischen dem RAM und dem Grafikspeicher übertragen werden.

Der Benchmark für zwei verdeckte Schichten, Abbildung 4.7b, zeigt einen näherungsweise quadratischen Anstieg der Rechenzeit der beiden CPUs (——, ——) mit der Neuronenanzahl pro verdeckter Schicht. Bei jeweils n Neuronen in den beiden verdeckten Schichten besitzt das neuronale Netz drei Gewichtsmatrizen mit den Größen $4 \times n$, $n \times n$ und $n \times 1$. Die Gesamtanzahl der Gewichte steigt näherungsweise quadratisch mit n. Die benötigte Rechenzeit der GPU (——) bleibt bis ca. 1000 Neuronen pro verdeckter Schicht konstant und steigt dann ebenso näherungsweise quadratisch an. Das deutet darauf hin, dass die GPU erst ab dieser Neuronenanzahl ausgelastet werden kann. Die GPU benötigt, bei zwei verdeckten Schichten, ab ca. 180 bzw. 380 Neuronen pro verdeckter Schicht weniger Rechenzeit als die Xeon-CPU bzw. i7-CPU.

Der Benchmark für drei verdeckte Schichten soll hier nicht mehr gezeigt werden. Die CPUs weisen darin einen näherungsweise quadratischen Anstieg der Rechenzeit auf, allerdings mit steilerem Anstieg als bei zwei verdeckten Schichten. Die GPU wird, bei drei verdeckten Schichten, ab ca. 800 Neuronen pro verdeckter Schicht ausgelastet. Sie benötigt ab ca. 140 bzw. 300 Neuronen pro verdeckter Schicht weniger Rechenzeit als die Xeon-CPU bzw. i7-CPU.

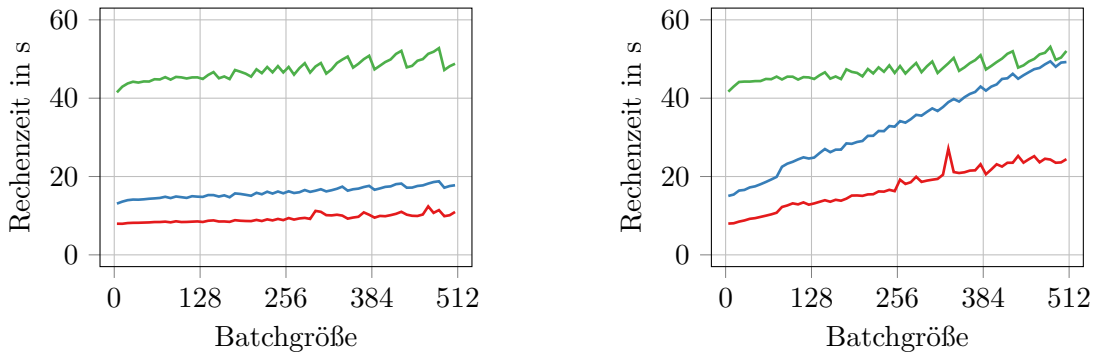
Die im Verlauf der Masterarbeit erstellten neuronalen Netze wiesen maximal drei verdeckte Schichten und in allen Fällen weniger als 100 Neuronen pro verdeckter Schicht auf. Zusammengefasst ist der Einsatz einer GPU für die Berechnung einfacher neuronaler Netze, wie sie für diese Arbeit erstellt wurden, sinnlos bzw. wirkt sich sogar nachteilig auf die benötigte Rechenzeit aus. Die Optimierung der Hyperparameter eines Modells, zum Beispiel durch Raster- oder Zufallssuche, kann aber i.A. parallelisiert werden. Dies würde zu einer besseren Auslastung der GPU führen. Die verwendete Keras API für TensorFlow enthält jedoch keine Methoden für die direkte Parallelisierung auf GPUs.

[1] Die GeForce GTX Titan X besitzt 3072 CUDA Cores.

Dies trifft auch auf andere gängige Maschinenlernbibliotheken zu, weshalb der Ansatz in der Masterarbeit nicht weiterverfolgt wurde.

4.7.2 Einfluss der Batchgröße

Um die theoretischen Betrachtungen zur Wahl der Batchgröße (vgl. Abschnitt 3.3.6) zu ergänzen, wurden weitere Benchmarks erstellt. Es sollte ermittelt werden, inwiefern sich die Batchgröße auf die Berechnungszeit einer Iteration im Training auswirkt. Das untersuchte Netzwerk bestand wieder aus 4 Neuronen in der Eingabeschicht und 1 Neuron in der Ausgabeschicht. Außerdem enthielt das Netzwerk 3 verdeckte Schichten. Variiert wurden die Neuronenanzahl in den verdeckten Schichten und die Batchgröße. Das Netzwerk wurde für jede Konfiguration 10 000 Iterationen lang trainiert. Die Rechenzeitverläufe über der Batchgröße zeigt Abbildung 4.8.



(a) 10 Neuronen pro verdeckter Schicht.

(b) 100 Neuronen pro verdeckter Schicht.

Abbildung 4.8: Benötigte Rechenzeiten für 10 000 Iterationen bei drei verdeckten Schichten und variierter Neuronenanzahl pro verdeckter Schicht. Berechnung mit einem —— i7-6700K @ 4.3 GHz, 4C/8T, einem —— Xeon X3360 @ 2.83 GHz, 4C/4T und einer —— GeForce GTX Titan X.

Bei je 10 Neuronen in 3 verdeckten Schichten, Abbildung 4.8a, steigen die Berechnungszeiten – für beide CPUs (——, ——) und die GPU (——) – mit der Batchgröße kaum. Beispielsweise benötigt die i7-CPU bei einer Batchgröße von 16 für 10 000 Iterationen eine Rechenzeit von 8.03 s. Bei einer Batchgröße von 512 beträgt die Rechenzeit 11.0 s

Bei je 100 Neuronen in 3 verdeckten Schichten, Abbildung 4.8b, steigen die Berechnungszeiten der CPUs (——, ——) mit der Batchgröße stärker an, als in Abbildung 4.8a. In diesem Fall benötigt die i7-CPU bei Batchgröße 16 8.34 s und bei Batchgröße 512 bereits 24.1 s. Der Verlauf der Berechnungszeit der GPU (——) bleibt durch die Erhöhung der Neuronenanzahl weitgehend unverändert, vgl. Abbildungen 4.8a und 4.8b.

Zusammengefasst kann die Batchgröße, beim einfachen Netzwerk mit 3 verdeckten Schichten mit je 10 Neuronen, ohne nennenswerte Geschwindigkeitseinbußen erhöht werden. Die Genauigkeit der Gradientennäherung jeder Iteration steigt dadurch. Für das kompliziertere Netzwerk, mit 3 verdeckten Schichten mit je 100 Neuronen, führt eine Erhöhung der Batchgröße zu einer längeren Iterationsdauer. Die Genauigkeit der Gradientennäherung steigt zwar, aber es können in derselben Zeit weniger Iterationen durchgeführt werden (was die Konvergenz i.A. verlangsamt).

5 Berechnung der Netzmodelle

Mit den im vorigen Kapitel beschriebenen Prozeduren bzw. Programmen wurden im Verlauf der Masterarbeit verschiedene SOFC-Modelle erstellt. Modelliert wurden Polarisationskurven und die Zellimpedanz, die zusammen eine Charakterisierung der wesentlichen Zelleigenschaften ermöglichen. In diesem Kapitel werden die Erstellung der Modelle, deren Auswertung und deren weitere Optimierung beschrieben, wobei zuerst die Polarisationsmodelle und anschließend die Impedanzmodelle behandelt werden.

Zu den verwendeten Fehlergrößen

Hier wird ein kurzer Überblick über die in diesem Kapitel vorkommenden Fehlergrößen gegeben. In den Gleichungen kommen folgende Größen vor: die Ausgabe des Modells y_i für einen Datenpunkt i, die zugehörige Soll-Ausgabe g_i und die Anzahl n_p der Datenpunkte des Datensatzes, für den der Fehler berechnet wird. Bei Modellen mit zwei Ausgabegrößen werden die Fehlergrößen für jede Ausgabegröße separat berechnet. Der bereits aus den Grundlagen bekannte mittlere quadratische Fehler (mean squared error, MSE), vgl. Gleichung (3.44), wird für eine einzelne Ausgabegröße mit

$$\mathrm{MSE} = \frac{1}{n_p} \sum_{i=1}^{n_p} (y_i - g_i)^2 \tag{5.1}$$

berechnet. Der mittlere relative Fehlerbetrag (mean absolute percentage error, MAPE) wird mit folgender Gleichung bestimmt:

$$\mathrm{MAPE} = \frac{100}{n_p} \sum_{i=1}^{n_p} \left| \frac{y_i - g_i}{g_i} \right| \quad \text{in } \% \tag{5.2}$$

Der MAPE wird, wenn die Soll-Ausgabe g_i eines Datenpunktes i null oder beinahe null ist, unendlich bzw. sehr groß. Das heißt, einzelne hohe Fehlerwerte können auch den mittleren Fehler stark erhöhen. Aus diesem Grund wird gleichzeitig zum MAPE auch immer der symmetrische mittlere relative Fehlerbetrag (symmetric mean absolute percentage error, SMAPE) angegeben:

$$\text{SMAPE} = \frac{2 \cdot 100}{n_p} \sum_{i=1}^{n_p} \frac{|y_i - g_i|}{|y_i| + |g_i|} \quad \text{in } \% \tag{5.3}$$

Der SMAPE eines einzelnen Datenpunktes kann nie größer werden als 200 %. Der SMAPE gewichtet demzufolge einzelne hohe Fehlerwerte geringer als der MAPE. Wenn die Abweichungen zwischen Vorhersagen y_i und Soll-Ausgaben g_i jedoch gering ist, sind SMAPE und MAPE näherungsweise gleich groß.

5.1 Modellierung von Polarisationskurven

In diesem Abschnitt werden die zur Modellierung von Polarisationskurven erstellten neuronalen Netze vorgestellt. Zuerst wird auf die zur Modellerstellung verwendeten Daten eingegangen. Danach werden generelle Einstellungen der Modellerstellungsprogramme und deren Wahl behandelt. Anschließend wird die Erstellung zweier Modelle beschrieben und die Qualität der Modellierung diskutiert: Für das erste Modell wurde der zu modellierende Spannungsbereich nach unten begrenzt. Der schwierig zu modellierende Bereich, in dem die Konzentrationsverluste auftreten, wurde so vermieden. Die Berechnung und Optimierung des Modells wird hier Schritt für Schritt beschrieben, um einen Überblick über die verwendete Vorgehensweise zu schaffen. Das zweite Modell modelliert den gesamten von den Daten umfassten Spannungsbereich. Der Fokus liegt hier auf den Ergebnissen.

5.1.1 Verwendete Daten

Um eine Polarisationskurve (vgl. Abschnitt 2.4) modellieren zu können, sind in jedem Fall die beiden Größen Zellspannung und Stromdichte erforderlich. Sollen die Betriebsbedingungen der Zelle in die Modellierung miteinbezogen werden, sind die Zelltemperatur, die Gaszusammensetzungen und die Volumenströme weitere relevante Größen. Die Zellspannung und die auftretenden Verluste werden durch diese Größen maßgeblich beeinflusst.

Die zur Erstellung der Polarisationsmodelle verwendeten Daten wurden mit einem physikalischen Modell einer planaren, anodengestützten SOFC generiert. Entwickelt wurde dieses Modell von Thomas Thaller im Laufe seiner Masterarbeit [58]. Das Modell ermöglicht an der Brennstoffseite die Vorgabe eines Gasgemisches aus Wasserstoff, Wasserdampf und Stickstoff. An der Sauerstoffseite kann hingegen ein Gemisch aus Sauerstoff und Stickstoff vorgegeben werden. Die Gaszusammensetzung im Strömungskanal wird im Modell als konstant angenommen und ist das logarithmische Mittel aus Eingangs- und Ausgangsgaszusammensetzung. Veränderungen quer zur Strömungsrichtung werden vernachlässigt. Es wird nur Stofftransport in den Elektroden in Zelldickenrichtung abgebildet,

das heißt es handelt sich um eine eindimensionale Betrachtung. Das Matlab-Modell berechnet die Zellspannung ausgehend von der Nernstspannung, von der die Aktivierungs-, Konzentrations- und ohm'sche Verluste, ausgedrückt als Spannungsdifferenzen, subtrahiert werden. Für eine genauere Beschreibung des Modells sei auf die bereits erwähnte Masterarbeit verwiesen. [58] Das Modell wurde in Matlab implementiert und soll im weiteren Verlauf als „Matlab-Modell“ bezeichnet werden, um es von anderen Modellen begrifflich zu trennen.

Bei allen im weiteren Verlauf des Kapitels angegebenen Gasanteilen x_i handelt es sich um in Prozent angegebene Stoffmengenanteile bzw. Volumenanteile (Betrachtung als ideales Gasgemisch). Um die in der Praxis meist eingeschränkte Verfügbarkeit an Messdaten abzubilden, wurden mit dem Matlab-Modell Pseudo-Messdaten simuliert. Diese wurden zur Modellerstellung durch neuronale Netze verwendet und umfassten für den realen Zellbetrieb relevante Betriebsparameter: Variiert wurden der Wasserstoffanteil x_{H_2} (30 %, 50 % und 70 %) und der Wasserdampfanteil x_{H_2O} (0 %, 10 % und 20 %) auf der Brennstoffseite, sowie die Zelltemperatur ϑ (725 °C, 775 °C und 825 °C). Der Stickstoffanteil auf der Brennstoffseite wurde so festgelegt, dass die Summe der Gasanteile 100 % beträgt: $x_{N_2} = (100 - x_{H_2} - x_{H_2O})\%$. Die Volumenströme $\dot{V}_F = 2\,\mathrm{l\,min^{-1}}$ (brennstoffseitig) und $\dot{V}_O = 2\,\mathrm{l\,min^{-1}}$ (sauerstoffseitig) sowie die Gaszusammensetzung (sauerstoffseitig) wurden konstant gehalten. Dies ergibt in Summe 27 verschiedene Betriebsbedingungen (9 Gaszusammensetzungen bei 3 Temperaturen). Einen Überblick über die variierten und konstanten Parameter bietet Tabelle 5.1.

Tabelle 5.1: Variierte und konstante Parameter des Matlab-Modells bei der Simulation der Pseudo-Messdaten.

Temperatur	Brennstoffseite				Sauerstoffseite		
ϑ	x_{H_2}	x_{H_2O}	x_{N_2}	$\dot{V}_F$	x_{O_2}	x_{N_2}	$\dot{V}_O$
°C	%	%	%	$\mathrm{l\,min^{-1}}$	%	%	$\mathrm{l\,min^{-1}}$
725 °C, 775 °C und 825 °C	30 %, 50 % und 70 %	0 %, 10 % und 20 %	$100 - x_{H_2} - x_{H_2O}$	2.0	21	79	2.0

Für die Simulation der Polarisationskurven wurde für jede Betriebsbedingung die Stromdichte von $0\,\mathrm{mA\,cm^{-2}}$ Schrittweise um $2\,\mathrm{mA\,cm^{-2}}$ erhöht, die Zellspannung berechnet und aufgezeichnet. Die Aufzeichnung wurde in zwei Fällen gestoppt: Erstens, wenn eine Zellspannung von 0 V erreicht wurde und die Kurve vollständig war. Zweitens, wenn eine weitere Erhöhung der Stromdichte im Matlab-Modell eine Verletzung der Massenbilanz oder negative Partialdrücke an den Elektrodenoberflächen verursacht hätte. Durch Einsatz des Matlab-Modells zur Datengenerierung war die den Daten zugrundeliegende wahre Funktion bekannt. Um diesen Vorteil zu nutzen, wurden innerhalb der Grenzen der zuvor beschriebenen Pseudo-Messdaten umfangreiche, wahre Testdaten erstellt: Variiert wurden wieder der Wasserstoffanteil x_{H_2} (30 % bis 70 % in Schritten von 5 %) und der Wasserdampfanteil x_{H_2O} (0 % bis 20 % in Schritten von 5 %) auf der Brennstoffseite sowie

die Zelltemperatur ϑ (725 °C bis 825 °C in Schritten von 25 °C). Die Pseudo-Messdaten waren in den wahren Testdaten nicht enthalten.

Die Nernstspannung einer Brennstoffzelle, vgl. Gleichung (2.16), geht bei trockenem Brenngas und gegen null gehender Stromdichte, d.h. im Leerlauf der Zelle, gegen unendlich. Die Zellspannung wird im Matlab-Modell ausgehend von der Nernstspannung ermittelt. Es muss dem Matlab-Modell daher immer ein minimaler Wasserdampfanteil vorgegeben werden. Betriebsbedingungen mit $x_{H_2O} = 0\,\%$ wurden genau genommen mit $x_{H_2O} = 0.001\,\%$ berechnet. Deshalb kann die vom Matlab-Modell berechnete Zellspannung nahe dem Leerlauf größer sein als die Standardzellspannung, die für Wasserstoffbrennstoffzellen in etwa 1.23 V beträgt.

5.1.2 Globale Einstellungen der Modellerstellung

Die Modellerstellung mit verschachtelter Kreuzvalidierung wurde, aufgrund der hohen benötigten Rechenzeiten, nur für erste Proberechnungen verwendet. Die Modellerstellung mit zufallsbasierter Datenteilung und die Modellerstellung mit Kreuzvalidierung wurden, mit den gleichen Einstellungen, immer beide ausgeführt. Keine der Methoden wies jedoch erkennbare Vorteile gegenüber der anderen auf, weder hinsichtlich der Rechenzeit noch der Ergebnisqualität. Um die direkte Vergleichbarkeit aufeinanderfolgender Ergebnisse zu wahren, werden nur Ergebnisse der Modellerstellung mit zufallsbasierter Datenteilung behandelt.

Bei der Modellerstellung mit zufallsbasierter Datenteilung wurden viele Hyperparameter – sowohl Hyperparameter der neuronalen Netze selbst, als auch Einstellungen der Modellerstellung – konstant gehalten. Welche Parameter das waren und wie sie gewählt wurden, wird in diesem Abschnitt beschrieben. Diese Vorgehensweise hat die Anzahl der möglichen Parameterkombinationen erheblich reduziert, was die Optimierung der Parameter und die Interpretation der erhaltenen Ergebnisse erleichterte. Es sei darauf hingewiesen, dass das jedoch zu Problemen führen kann, wenn einer der konstant gewählten Parameter ausschlaggebend für die erfolgreiche Modellerstellung ist.

Die gewählten Einstellungen der Modellerstellung sind in Tabelle 5.2 zusammengefasst. Für eine Beschreibung der darin vorkommenden Datensätze und wofür diese bei der Modellerstellung genau verwendet wurden, sei auf Abschnitt 4.3 verwiesen.

Die *Daten* wurden zunächst über Prozentsätze zufallsbasiert in HPO-Daten und finale Testdaten (80 % und 20 %) aufgeteilt. Die *HPO-Daten* wurden dann ebenso über vorgegebene Prozentsätze zufallsbasiert in HPO-Trainings-, Stopp- und HPO-Testdaten (80 %, 4 % und 16 %) aufgeteilt. Die angegebenen Prozentsätze wurden analog zu häufig verwendeten Werten in Tutorials und der Literatur gewählt. Die Anzahl der *Wiederholungen* in der *HP-Optimierung* wurde mit 5 festgelegt. Das war ein Kompromiss zwischen

Tabelle 5.2: Konstant gewählte Einstellungen der Modellerstellung mit zufallsbasierter Datenteilung für alle berechneten Polarisationsmodelle.

Einstellung	gewählt
Aufteilung Daten	
HPO-Daten	80 %
finale Testdaten	20 %
Aufteilung HPO-Daten	
HPO-Trainingsdaten	80 %
Stoppdaten	4 %
HPO-Testdaten	16 %
Wiederholungen	
in HP-Optimierung	5
in Ensemblebildung	10
Datenpipeline	
Transformation EG	lin. Skalierung

der Genauigkeit des mittleren Testfehlers in der HP-Optimierung und der Rechenzeit der Modellerstellung. Beispielsweise müsste, bei Verdopplung der Wiederholungen, die Anzahl der untersuchten HP-Kombinationen halbiert werden, damit die Gesamtrechenzeit gleich bleibt, siehe Gleichung 4.1. Ob die Verdopplung der Wiederholungen dann zu besseren Ergebnissen führt, ist kaum abschätzbar. Die Anzahl der *Wiederholungen* in der *Ensemblebildung* betrug 10. Werden in der Modellerstellung viele HP-Kombinationen untersucht, hat dieser Parameter wenig Einfluss auf die Gesamtrechenzeit, siehe erneut Gleichung (4.1). Die *Datenpipeline* wurde so aufgebaut, dass die Werte jeder Eingabegröße linear skaliert werden – das heißt, sie werden auf einen Wertebereich zwischen 0.2 und 0.8 transformiert. Weisen Eingabe- und Ausgabegrößen eines neuronalen Netzwerkes dieselbe Größenordnung auf, beschleunigt das die Konvergenz [39].

Nun sollen die konstant gewählten Hyperparameter der neuronalen Netze behandelt werden. Diese wurden bei der Erstellung aller in diesem Kapitel vorgestellten Polarisationsmodelle beibehalten. Eine Übersicht bietet Tabelle 5.3.

Die *Bias* wurden alle mit 0 initialisiert, vgl. Abschnitt 3.3.4. Als *Optimierer* wurde der Adam-Algorithmus gewählt, vgl. Abschnitt 3.3.5. Die meisten Optimierer haben unterschiedliche Parameter und diese wiederum unterschiedliche, von der Problemstellung abhängige, optimale Parameterbereiche. Durch die feste Vorgabe des Optimierers entfällt daher ein erheblicher Optimier- und Programmieraufwand. Zusätzlich ist es so für den Entwickler einfacher, Ergebnisse zu interpretieren und Erfahrungswerte zu sammeln. Die *Batchgröße*, vgl. Abschnitt 3.3.6, wurde konstant gesetzt, da das Lernverhalten über die Lernrate angepasst wurde. Mit den Benchmarks in Abschnitt 4.7.2 wurde bereits gezeigt,

Tabelle 5.3: Konstant gewählte Hyperparameter für alle berechneten Polarisationsmodelle.

Hyperparameter	gewählt
Biasinitialisierung	mit 0
Optimierer	Adam
Batchgröße	512
Kostenfunktion	MSE
Min. Verbesserung	$1 \cdot 10^{-7}$
Max. Epochenanzahl	20 000

dass eine Batchgröße von 512 einen guten Kompromiss zwischen Genauigkeit der Gradientennäherung und Konvergenzgeschwindigkeit darstellt (für ein ähnlich kompliziertes Modell wie die hier behandelten). Die *Kostenfunktion* wurde vor allem aus Gründen der Vergleichbarkeit fest als die mittlere quadratische Abweichung (mean squared error, MSE) vorgegeben, vgl. Abschnitt 3.3.7. Eine Optimierung der Hyperparameter inklusive Variation der Kostenfunktion ist wenig sinnvoll, da die berechneten Kosten verschiedener Funktionen nicht direkt vergleichbar sind[1]. Die *minimale Verbesserung* der Kosten im Early Stopping, vgl. Abschnitt 3.3.8, wurde um etwa drei Größenordnungen kleiner gewählt als die zu erwartenden Kosten am Ende des Trainings. Somit werden kleine Änderungen vom Early Stopping als solche erkannt und gleichzeitig sind keine Probleme mit der Maschinengenauigkeit zu erwarten. Die *maximale Epochenanzahl* wurde so hoch gewählt, dass die Berechnungen zum Großteil durch das Early Stopping beendet werden, bevor sie diese Epochenanzahl erreichen.

5.1.3 Modellierung von Leerlauf bis 0.6 V

Zuerst wurde ein Modell für den Spannungsbereich von Leerlauf bis 0.6 V erstellt. Die Begrenzung der Modellierung wurde durch eine selektive Datenauswahl erreicht: Datenpunkte mit Spannungswerten kleiner als 0.6 V, wurden aus den zur Modellerstellung verwendeten Pseudo-Messdaten entfernt. Die Ausgabegröße des Modells war die Zellspannung E, die Eingabegrößen waren der Wasserstoffanteil $x_{\mathrm{H_2}}$ (brennstoffseitig), der Wasserdampfanteil $x_{\mathrm{H_2O}}$ (brennstoffseitig), die Zelltemperatur ϑ und die konstante Stromdichte i. Abbildung 5.1 zeigt eine Darstellung des Modells als Blackbox, mit den Eingabe- und Ausgabegrößen. Tabelle 5.4 zeigt einen Ausschnitt der zur Modellerstellung verwendeten Daten, bevor sie der Datenpipeline zugeführt wurden, d.h. in untransformiertem Zustand. Im Gesamten umfassten die Daten 11 901 Datenpunkte. Die umfangreicheren, wahren Testdaten wurden ebenso auf Spannungen bis 0.6 V begrenzt und umfassten 88 270 Datenpunkte.

[1] Das Kriterium zur Ermittlung der optimalen Hyperparameter sind ja gerade die Kosten.

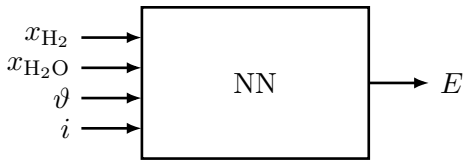


Abbildung 5.1: Vereinfachte Darstellung des Polarisationsmodells mit Eingabe- und Ausgabegrößen.

Tabelle 5.4: Ausschnitt der zur Erstellung des Polarisationsmodells verwendeten Daten.

Eingaben				Ausgabe
x_{H_2}	x_{H_2O}	i	ϑ	E
%	%	mA cm^{-2}	°C	V
30	0	0	725	1.407
30	0	2	725	1.224
30	0	4	725	1.197
⋮	⋮	⋮	⋮	⋮

Die Hyperparameter, die in der Modellerstellung optimiert wurden, waren: die Art der Gewichtsinitialisierung, die Wahl der Aktivierungsfunktion, die Anzahl der verdeckten Schichten, die Anzahl der Neuronen pro verdeckter Schicht, die Lernrate des Optimierers und die Geduld des Early Stoppings. Die durchsuchten Wertebereiche dieser Parameter sind in Tabelle 5.5 angegeben.

Tabelle 5.5: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das ermittelte Optimum. Berechnungszeit ~ 3h.

Hyperparameter	Durchsuchter Bereich	Optimum
Gewichtsinitialisierung	nach Glorot u. Bengio, nach He et al.	nach Glorot u. Bengio
Aktivierungsfunktion	Sigmoid, ReLU, ELU	Sigmoid
verdeckte Schichten	1 bis 3	3
Neuronen pro verd. Sch.	3 bis 15	11
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$1 \cdot 10^{-2}$
Geduld in Epochen	5 bis 20	20

Aus welchen Überlegungen heraus diese gewählt wurden, soll nun beschrieben werden: Für die Gewichtsinitialisierung und Aktivierungsfunktionen wurden, nach eigenem Ermessen, gängige Methoden bzw. Funktionen ausgewählt. Die Wertebereiche für die Anzahl an verdeckten Schichten und die Neuronen pro verdeckter Schicht orientierten sich an [39]: Darin wurde ein einfaches Polarisationsmodell (mit 3 Eingabegrößen) durch ein neuronales Netzwerk, mit einer verdeckten Schicht mit 3 Neuronen, umgesetzt. Ein komplizierteres

Modell (mit 9 Eingabegrößen) in derselben Veröffentlichung benötigte ebenso nur eine verdeckte Schicht, diese enthielt aber 7 Neuronen. Beide Modelle in [39] wurden so einfach wie möglich gehalten, um die Generalisierung zu verbessern (Anwendung von Ockhams Rasiermesser, vgl. Abschnitt 3.2.4). Die generelle Gültigkeit von Ockhams Rasiermesser für das Maschinenlernen wurde jedoch widerlegt [16], weshalb das Prinzip für die hier erstellten Polarisationsmodelle nicht zur Anwendung kommen sollte. Es wurde deshalb auch über deutlich kompliziertere Modelle optimiert (1 bis 3 verdeckte Schichten und 3 bis 15 Neuronen pro verdeckter Schicht). Der Bereich für die Lernrate umfasste zwei Größenordnungen ($1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$). Bei Lernraten kleiner als $1 \cdot 10^{-4}$ dauerte das Training einzelner Versuchsrechnungen zu lange, bei Lernraten größer als $5 \cdot 10^{-2}$ begannen die Fehlerverläufe zu oszillieren und konvergierten gegen hohe Fehler (oder gar nicht). Die Geduld wurde für einen Bereich von 5 bis 20 Epochen optimiert.

Aus dem durch die Hyperparameter aufgespannten Raum, wurden durch die Zufallssuche 144 Parameterkombinationen ausgewählt und die beste Kombination ermittelt. Als berechnetes Optimum ergaben sich die Initialisierung nach Glorot und Bengio, die Sigmoid-Aktivierungsfunktion, 3 verdeckte Schichten mit je 11 Neuronen, eine Lernrate von $1 \cdot 10^{-2}$ und eine Geduld für das Early Stopping von 20 Epochen. Diese Werte sind ebenso in Tabelle 5.5 angegeben. Die Berechnungszeit mit der CPU des Desktop Xeon (Xeon X3360 @ 2.83 GHz, 4C/4T) betrug etwas mehr als 3 Stunden.

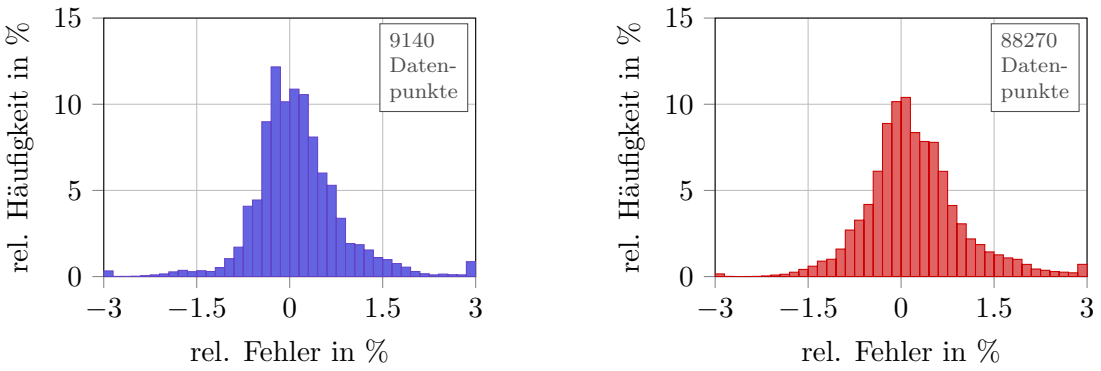
Das mit dem ermittelten HP-Optimum berechnete finale Modell, ein Ensemble aus 10 trainierten Modellen, erreichte einen MSE von $8.74 \cdot 10^{-5}/1.28 \cdot 10^{-4}/7.89 \cdot 10^{-5}$ und einen MAPE von 0.53 %/0.55 %/0.60 % auf den finalen Trainingsdaten/finalen Testdaten/wahren Testdaten. Zusammengefasst sind diese Ergebnisse in Tabelle 5.6.

Tabelle 5.6: Fehlergrößen des finalen Modells.

	MSE	MAPE	SMAPE
Datensatz	-	%	%
finale Trainingsdaten	$8.74 \cdot 10^{-5}$	0.53	0.53
finale Testdaten	$1.28 \cdot 10^{-4}$	0.55	0.55
wahre Testdaten	$7.89 \cdot 10^{-5}$	0.60	0.60

Ungewöhnlich ist, dass der MSE der wahren Testdaten geringer ist als der MSE der finalen Trainingsdaten. Zusätzlich sind noch die Histogramme des Fehlers, auf den finalen Trainingsdaten und wahren Testdaten, in Abbildung 5.2 dargestellt. Als Fehlergröße wurde der relative Fehler gewählt, da dieser einfach interpretiert werden kann.

Der überwiegende Teil (90 %) der relativen Fehler tritt bei den finalen Trainingsdaten im Bereich von −1.12 % bis 1.12 % auf, siehe Abbildung 5.2a. Bei den wahren Testdaten umfasst der Bereich −1.32 % bis 1.32 %, siehe Abbildung 5.2b. Die Unter- und Überlaufcontainer beider Histogramme, in Abbildungen 5.2a und 5.2b, zeigen, dass auch



(a) Finale Trainingsdaten.

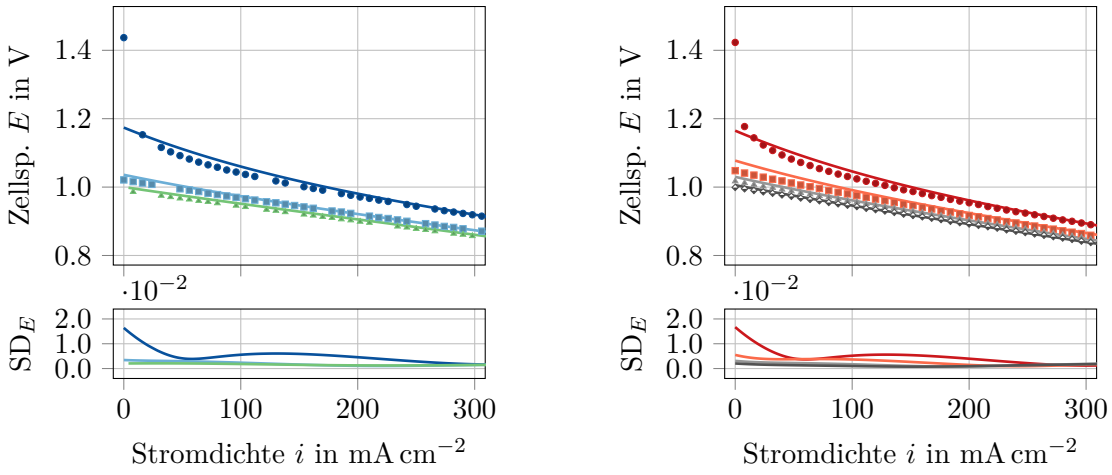
(b) Wahre Testdaten.

Abbildung 5.2: Histogramme des relativen Fehlers für finale Trainings- und wahre Testdaten. Darstellung mit Unter- und Überlaufcontainer.

Fehlerbeträge größer als 3 % auftreten. Bei beiden Datensätzen treten diese Fehler hauptsächlich in zwei Bereichen an den Modellgrenzen auf: Bei 0 % H_2O brennstoffseitig und niedrigen Stromdichten von 0 mA cm^{-2} bis 4 mA cm^{-2} erreichen die relativen Fehlerbeträge bis zu 20.4 %/20.1 % (finale Trainingsdaten/wahre Testdaten). Bei Zellspannungen nahe 0.6 V und Wasserstoffanteilen von 30 % bis 35 % erreichen die relativen Fehlerbeträge bis zu 8.20 %/9.92 % (finale Trainingsdaten/wahre Testdaten).

Es werden nun ausgewählte Modellvorhersagen für die Bereiche mit hohen Fehlern gezeigt. Vorhersagen für den Verlauf der Zellspannung E für den Bereich geringer Stromdichten i, zeigt Abbildung 5.3. Darin zeigt Abbildung 5.3a einen Ausschnitt der finalen Trainingsdaten bei einer Zelltemperatur von 775 °C und 50 % H_2 bei variiertem Wasserdampfanteil (in N_2). Abbildung 5.3b zeigt einen Ausschnitt der wahren Testdaten bei einer Zelltemperatur von 750 °C und 40 % H_2 bei variiertem Wasserdampfanteil (in N_2).

In beiden Datensätzen, siehe Abbildungen 5.3a bzw. 5.3b, tritt ein abrupter Spannungsabfall bei Stromdichten i von circa 0 mA cm^{-2} bis 20 mA cm^{-2} und einem H_2O-Anteil von 0 % (• bzw. •) auf. Dieser Spannungsabfall wird, zu Beginn, in erster Linie durch die Berechnung der Nernstspannung verursacht, vgl. Abschnitt 5.1.1 und ab Stromdichten von 6 mA cm^{-2} durch die Aktivierungsverluste an den Zellelektroden. Der maximale, relative Fehlerbetrag zwischen Modell (— bzw. —) und den Daten des Matlab-Modells (• bzw. •) beträgt, zu Beginn des abrupten Abfalls, bei beiden Datensätzen etwas mehr als 20 %. Zusätzlich weichen dort die Vorhersagen der einzelnen Modelle im Ensemble voneinander ab, weshalb die Standardabweichung erhöht ist. Demzufolge hängt in diesem Bereich das Trainingsergebnis eines einzelnen Modells stark von der zufälligen Initialisierung der Gewichte ab, was im Allgemeinen nicht erwünscht ist. Bei einem H_2O-Anteil von 10 % (▪) bzw. 5 % (▪), in Abbildung 5.3a bzw. 5.3b, tritt der Spannungsabfall durch die



(a) Finale Trainingsdaten. Ausschnitt bei einer Zelltemperatur von 775 °C, 50 % H_2 und • 0 %, ▪ 10 % und ▲ 20 % H_2O.

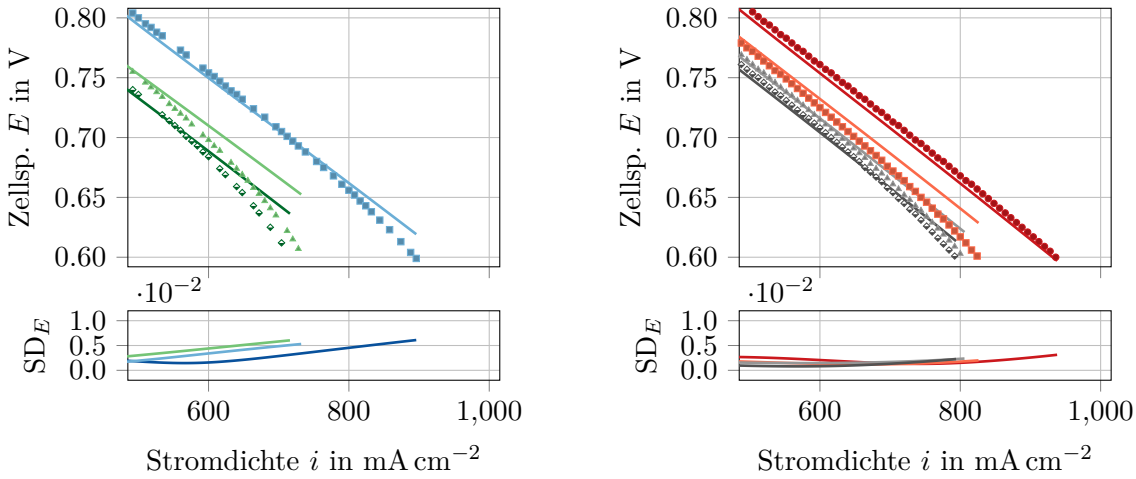
(b) Wahre Testdaten. Ausschnitt bei einer Zelltemperatur von 750 °C, 40 % H_2 und • 0 %, ▪ 5 %, ▲ 10 % und ◆ 15 % H_2O.

Abbildung 5.3: Vergleich der —— Vorhersage des finalen Modellensembles mit den Daten des Matlab-Modells, für geringe Stromdichten i. SD_E ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

Nernstspannung nicht mehr auf, aber der relative Fehlerbetrag zwischen Modellvorhersage und den Daten bleibt hoch: Es treten Werte über 2 % auf. Das ist auf den ersten Blick nicht viel, ist aber in etwa gleich groß wie die Änderung der Zellspannung durch Variation des Wasserdampfanteils, zum Beispiel von 5 % H_2O (▪) auf 10 % H_2O (▲) bei den wahren Testdaten. Erst im linearen Bereich, in dem vorwiegend ohm'sche Verluste auftreten, werden die Abweichungen gering. Das ist der Fall ab Stromdichten i von etwa 200 mA cm^{-2}.

Vorhersagen für den Verlauf der Zellspannung E, nahe der Modellgrenze von 0.6 V, d.h. im Bereich hoher Stromdichten i, zeigt Abbildung 5.4. Darin zeigt Abbildung 5.4a einen Ausschnitt der finalen Trainingsdaten bei einer Zelltemperatur von 825 °C und 30 % H_2 bei variiertem Wasserdampfanteil (in N_2). Abbildung 5.4b zeigt einen Ausschnitt der wahren Testdaten bei einer Zelltemperatur von 800 °C und 35 % H_2 bei variiertem Wasserdampfanteil (in N_2).

In beiden Datensätzen, siehe Abbildungen 5.4a bzw. 5.4b, tritt mit steigender Stromdichte ein vom ohm'schen Bereich abweichender, nichtlinearer Spannungsabfall auf. Dieser wird durch zunehmende Konzentrationsverluste, d.h. sinkende Eduktkonzentrationen an den Elektroden, verursacht und tritt bei allen dargestellten Wasserdampfanteilen auf. In beiden Abbildungen, 5.4a und 5.4b, treten für alle dargestellten Wasserdampfanteile



(a) Finale Trainingsdaten. Ausschnitt bei einer Zelltemperatur von 825 °C, 30 % H_2 und ▪ 0 %, ▲ 10 % und ◆ 20 % H_2O.

(b) Wahre Testdaten. Ausschnitt bei einer Zelltemperatur von 800 °C, 35 % H_2 und • 0 %, ▪ 5 %, ▲ 10 % und ◆ 15 % H_2O.

Abbildung 5.4: Vergleich der —— Vorhersage des finalen Modellensembles mit den Daten des Matlab-Modells, für hohe Stromdichten i. SD_E ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

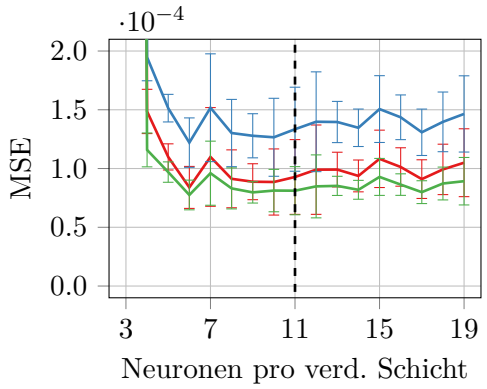
im Bereich der Konzentrationsverluste hohe Abweichungen zwischen Modellvorhersage und Daten des Matlab-Modells auf. Nahe der Modellgrenze von 0.6 V kommen relative Fehlerbeträge mit bis zu 10 % vor. Die einzige Ausnahme ist die Vorhersage für 0 % H_2O (•) in Abbildung 5.4b. Die Standardabweichung der Modelle im Ensemble steigt generell, mit Annäherung an die Modellgrenze von 0.6 V, in beiden Datensätzen an.

Die eben beschriebenen Probleme des Modells, die Eigenheiten der Daten wiederzugeben, treten dabei sowohl bei den finalen Trainingsdaten als auch bei den wahren Testdaten auf. Das deutet darauf hin, dass das Modell unterangepasst ist. Der MAPE des Modells, der auf den finalen Trainingsdaten/wahren Testdaten nur 0.53 %/0.60 % beträgt, vgl. Tabelle 5.6, würde hingegen – auf den ersten Blick – ein gut angepasstes Modell vermuten lassen. Ein Grund für den niedrigen MAPE ist die hohe Anzahl an Datenpunkten im linearen Bereich, die vom Modell genau abgebildet werden. In den Bereichen, in denen die Spannung stark abfällt, sind vergleichsweise wenig Datenpunkte enthalten. Deren Anteil am MAPE, der ein mittlerer Fehler ist, ist deshalb gering. In den Daten ist außerdem nur der Beginn des konzentrationsbedingten Spannungsabfalls enthalten, da die Zellspannung auf Werte größer gleich 0.6 V beschränkt wurde. Bei Wasserstoffanteilen von 40 % und mehr liegt die Spannung bis 0.6 V zudem noch im linearen, ohm'schen Bereich. Der konzentrationsbedingte Spannungsabfall tritt daher nur bei etwa einem Drittel aller

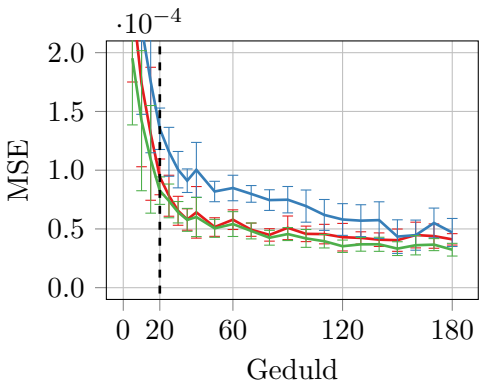
simulierten Polarisationskurven auf. Die dort entstehenden Fehler haben demzufolge ebenso einen reduzierten Anteil am MAPE.

Um das Modell auf Unteranpassung zu untersuchen, wurden Lernkurven erstellt, die im Folgenden kurz beschrieben werden: Um eine Lernkurve zu ermitteln, wird ein Hyperparameter des Modells variiert und das Modell für jeden Wert neu trainiert und getestet. Alle anderen Hyperparameter bleiben währenddessen unverändert. Als Ergebnis erhält man den Verlauf des Testfehlers in Abhängigkeit des variierten Parameters. Lernkurven werden, aufgrund ihrer Bezeichnung, häufig mit dem Fehlerverlauf während des Trainings verwechselt (d.h. dem Verlauf des Fehlers in Abhängigkeit der Epoche). Obwohl die Erstellung von Lernkurven vielfache Trainingsvorgänge beinhaltet, sind die dabei entstehenden Fehlerverläufe nicht von Interesse, sondern nur die Endergebnisse der Tests. Ein Vorteil von Lernkurven ist die einfache Anwendung. Nachteil der Methode ist, dass für jede Lernkurve nur eine Dimension des Parameterraums variiert wird: Ist die gleichzeitige Änderung eines zweiten Parameters notwendig, um einen geringen Testfehler zu erreichen, kann das aus der Lernkurve nicht abgelesen werden. Anders formuliert, wird nur ein kleiner Teil des Parameterraumes untersucht. Das bedeutet, dass viele vielleicht gut funktionierende Parameterkombinationen ausgelassen werden.

Sowohl zu einfache Modelle als auch zu kurzes Training sind häufige Gründe für Unteranpassung. Es wurden daher Lernkurven für die Hyperparameter *Neuronen pro verdeckter Schicht* und *Geduld* erstellt, die in Abbildung 5.5 dargestellt sind.



(a) Lernkurve für den Parameter *Neuronen pro verdeckter Schicht*.



(b) Lernkurve für den Parameter *Geduld*.

Abbildung 5.5: Lernkurven (Fehlerverlauf bei Variation eines Parameters). Der MSE ist jeweils ein Mittelwert aus 10 Wiederholungen. Die zugehörigen Standardabweichungen sind als Fehlerbalken dargestellt. MSE auf den — finalen Trainingsdaten, den — finalen Testdaten und den — wahren Testdaten.

Für die ursprünglich in der Modellerstellung durchsuchten Parameterbereiche und das

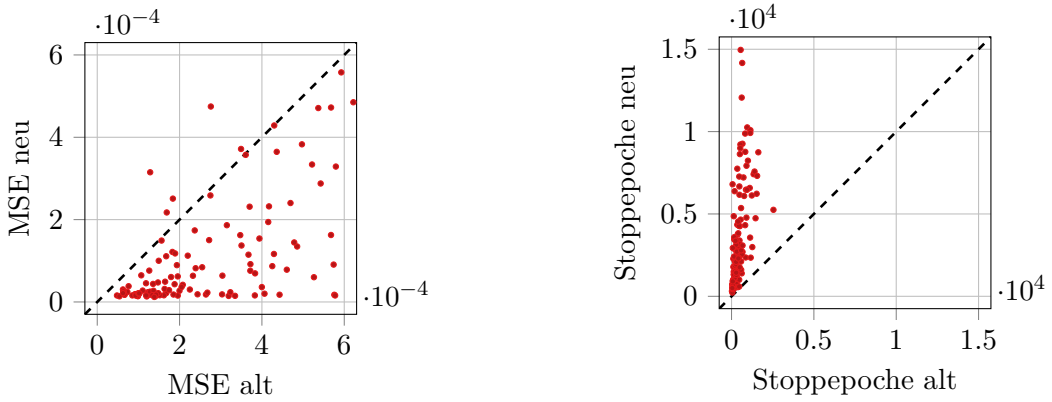
gefundene Optimum, sei noch einmal auf Tabelle 5.5 verwiesen: Für den Parameter Neuronen pro verdeckter Schicht wurde der Bereich 3 bis 15 durchsucht, das gefundene Optimum betrug 11. Das deutete darauf hin, dass eine Erhöhung der Neuronenanzahl nicht der wesentliche Faktor für die Reduktion des MSE war. Bestätigt wurde dies durch die Lernkurve des Parameters in Abbildung 5.5a. Diese zeigt den Verlauf des MSE bei Variation der Neuronenanzahl pro verdeckter Schicht von 3 bis 19. Die gestrichelte Linie kennzeichnet die Ausgangskonfiguration, 11 Neuronen pro verdeckter Schicht, des finalen Modells. Der MSE, sowohl für die finalen Trainingsdaten (——), die finalen Testdaten (——) als auch die wahren Testdaten (——), sinkt mit Veränderung des Parameters, ausgehend vom Wert 11, nicht nennenswert. Bei sehr geringer Neuronenanzahl steigt der MSE, wenig überraschend, sogar an.

Anders verhielt es sich mit dem Parameter Geduld (des Early Stopping). Für den ursprünglich durchsuchten Bereich sei erneut auf Tabelle 5.5 verwiesen: Der Bereich betrug 5 bis 20 Epochen und das gefundene Optimum betrug 20 Epochen – es lag also genau auf der oberen Grenze des Bereichs. Die Lernkurve des Parameters, in Abbildung 5.5b, zeigt den Verlauf des MSE bei Variation der Geduld von 0 bis 180 Epochen. Die Ausgangskonfiguration des finalen Modells, eine Geduld von 20 Epochen, ist wieder mit einer gestrichelten Linie gekennzeichnet. Bei Erhöhung der Geduld, ausgehend vom Wert 20, sinkt der MSE für die finalen Trainingsdaten (——), die finalen Testdaten (——) und die wahren Testdaten (——) stark. Beispielsweise beträgt der MSE auf den wahren Testdaten, bei einer Geduld von 20 Epochen, $8.26 \cdot 10^{-5}$. Bei einer Geduld von 180 Epochen beträgt er nur noch $3.23 \cdot 10^{-5}$. Der durchsuchte Bereich in der Hyperparameteroptimierung, der eine Geduld von 5 bis 20 Epochen umfasste, war demzufolge ungünstig bzw. zu klein gewählt. Mit dieser Erkenntnis war wiederum das Ergebnis der Optimierung als Ganzes in Frage zu stellen. Die Optimierungsrechnung wurde aus diesem Grund erneut durchgeführt. Das Minimum des MSE der Testdaten lag, in der bereits gezeigten Lernkurve in Abbildung 5.5b, bei einer Geduld von 150. Der Bereich der Geduld für die neue Rechnung wurde mit 143 bis 158 gewählt. Die durchsuchten Bereiche für die Geduld und die anderen Parameter der neuen Rechnung sind in Tabelle 5.7 zusammengefasst. Es wurde außerdem die gleiche Seed für die Zufallssuche verwendet, wie in der ersten Rechnung: Die Parameterkombinationen, über die optimiert wurde, waren somit – ausgenommen der erhöhten Geduld – identisch. Die Berechnungszeit mit der CPU des Desktop Xeon (Xeon X3360 @ 2.83 GHz, 4C/4T) betrug diesmal nicht ganz 23 Stunden. Als Optimum ergab sich die Initialisierung nach He et al., die Sigmoidfunktion als Aktivierungsfunktion, 2 verdeckte Schichten mit je 12 Neuronen, eine Lernrate von $1 \cdot 10^{-1}$ und eine Geduld von 147.

Um den Einfluss der erhöhten Geduld auf die HP-Optimierung sichtbar zu machen, wurden die während der Optimierung erstellten Modelle untersucht. Zur Erinnerung: Der MSE und die Stoppepoche in der HP-Optimierung sind Mittelwerte aus 5 Wiederholungen, wobei der MSE auf den HPO-Testdaten ermittelt wird.

Tabelle 5.7: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das ermittelte Optimum. Neue Rechnung mit erhöhtem Geduldbereich. Berechnungszeit ~ 23h.

Hyperparameter	Durchsuchter Bereich	Optimum
Gewichtsinitialisierung	nach Glorot u. Bengio, nach He et al.	nach He et al.
Aktivierungsfunktion	Sigmoid, ReLU, ELU	Sigmoid
verdeckte Schichten	1 bis 4	2
Neuronen pro verd. Sch.	3 bis 16	12
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$1 \cdot 10^{-1}$
Geduld	143 bis 158	147



(a) Vergleich des MSE.

(b) Vergleich der mittleren Stoppepoche.

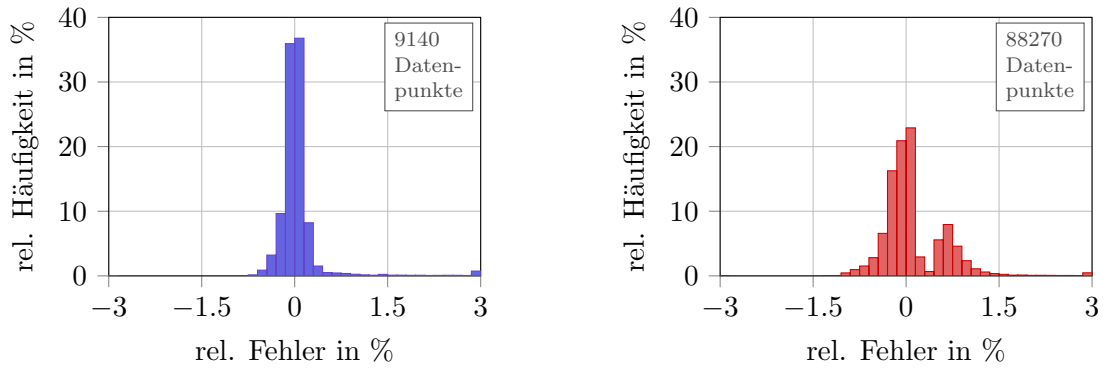
Abbildung 5.6: Streudiagramme zum Vergleich des MSE bzw. der Stoppepoche der in der HP-Optimierung berechneten Modelle.

In Abbildung 5.6a sind die MSE der neuen Modelle (mit erhöhter Geduld) über den MSE der alten Modelle in einem Streudiagramm dargestellt. Jeder Punkt entspricht derselben Parameterkombination (ausgenommen der veränderten Geduld). Es ist außerdem die Identität als gestrichelte Trennlinie eingezeichnet: Jeder Punkt in Abbildung 5.6a, der sich unter der gestrichelten Trennlinie befindet, steht für ein Modell, dessen MSE durch die erhöhte Geduld verringert wurde. Das trifft auf den überwiegenden Teil der Modelle zu. In Abbildung 5.6b sind hingegen die Stoppepochen der neuen Modelle (mit erhöhter Geduld) über denen der alten Modelle dargestellt. Wieder ist die Identität als Trennlinie eingezeichnet. Die Abbildung zeigt, dass alle Modelle durch die erhöhte Geduld länger trainiert wurden, das heißt eine höhere Stoppepoche aufwiesen. Das ist auch der Grund für die wesentlich höhere Berechnungszeit der neu durchgeführten HP-Optimierung (23 Stunden) gegenüber der alten (3 Stunden).

Das mit dem neuen HP-Optimum berechnete finale Modell, wieder ein Ensemble aus 10 trainierten Modellen, erreichte einen MSE von $3.08 \cdot 10^{-5}/4.59 \cdot 10^{-5}/2.93 \cdot 10^{-5}$ und einen MAPE von 0.17 %/0.18 %/0.34 % auf den finalen Trainingsdaten/finalen Testdaten/wahren Testdaten. Zusammengefasst sind diese Ergebnisse in Tabelle 5.8. Im Vergleich mit den MSE des alten Modells, Tabelle 5.6, betragen die MSE des neuen Modells durchweg weniger als die Hälfte. Das spiegelt sich auch in den Fehlerhistogrammen wieder, dargestellt in Abbildung 5.7. Erneut wird der relative Fehler als Fehlergröße betrachtet.

Tabelle 5.8: Fehler des neu berechneten finalen Modells.

Datensatz	MSE -	MAPE %	SMAPE %
finale Trainingsdaten	$3.08 \cdot 10^{-5}$	0.17	0.17
finale Testdaten	$4.59 \cdot 10^{-5}$	0.18	0.18
wahre Testdaten	$2.93 \cdot 10^{-5}$	0.34	0.34



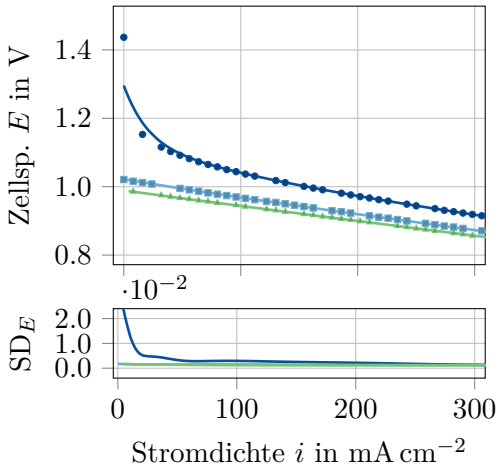
(a) Finale Trainingsdaten.

(b) Wahre Testdaten.

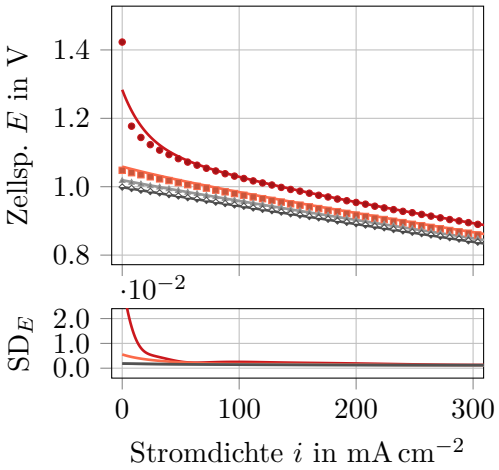
Abbildung 5.7: Histogramme des relativen Fehlers für finale Trainings- und wahre Testdaten des neu berechneten finalen Modells. Darstellung mit Unter- und Überlaufcontainer.

Der überwiegende Anteil der Fehler (90 %) liegt bei den finalen Trainingsdaten im Bereich von −0.29 % bis 0.29 %, siehe Abbildung 5.7a (beim alten Modell waren es noch −1.12 % bis 1.12 %). Das Histogramm der wahren Testdaten ist etwas breiter, 90 % der Fehler liegen hier in dem Bereich von −0.79 % bis 0.79 %, siehe Abbildung 5.7b (gegenüber −1.32 % bis 1.32 % beim alten Modell). Die Überlaufcontainer beider Histogramme, in Abbildungen 5.7a und 5.7b, zeigen an, dass wieder Fehler größer als 3 % auftreten. Diese treten bei beiden Datensätzen in denselben Bereichen auf wie schon beim alten Modell: Bei 0 % H_2O brennstoffseitig und niedrigen Stromdichten von $0\,\mathrm{mA\,cm^{-2}}$ bis $4\,\mathrm{mA\,cm^{-2}}$ treten relative Fehlerbeträge bis zu 9.82 %/10.1 % (finale Trainingsdaten/wahre Testdaten) auf.

Bei Zellspannungen nahe 0.6 V und Wasserstoffanteilen von 30 % bis 35 % treten relative Fehlerbeträge bis zu 3.84 %/6.63 % (finale Trainingsdaten/wahre Testdaten) auf. Auch für das neue Modell sollen diese Bereiche anhand von ausgewählten Modellvorhersagen gezeigt werden. Dazu werden dieselben Bereiche, die schon beim alten Modell betrachtet wurden, dargestellt. Die Vorhersagen für den Verlauf der Zellspannung E für den Bereich geringer Stromdichten i, zeigt Abbildung 5.8 (vgl. Abbildung 5.3 für das alte Modell). Darin zeigt Abbildung 5.8a einen Ausschnitt der finalen Trainingsdaten bei einer Zelltemperatur von 775 °C und 50 % H_2 bei variiertem Wasserdampfanteil (in N_2). Abbildung 5.8b zeigt einen Ausschnitt der wahren Testdaten bei einer Zelltemperatur von 750 °C und 40 % H_2 bei variiertem Wasserdampfanteil (in N_2).



(a) Finale Trainingsdaten. Ausschnitt bei einer Zelltemperatur von 775 °C, 50 % H_2 und • 0 %, ▪ 10 % und ▲ 20 % H_2O.



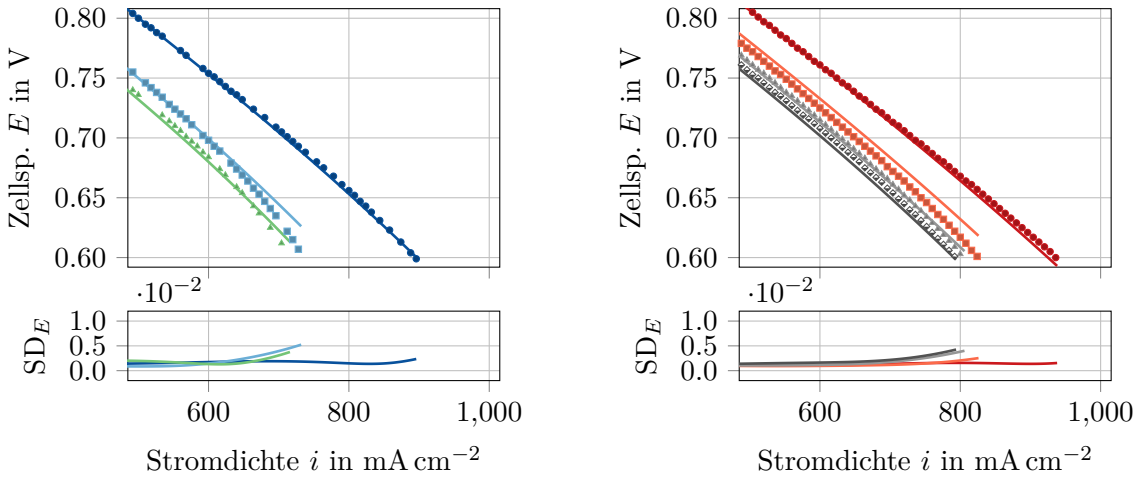
(b) Wahre Testdaten. Ausschnitt bei einer Zelltemperatur von 750 °C, 40 % H_2 und • 0 %, ▪ 5 %, ▲ 10 % und ◆ 15 % H_2O.

Abbildung 5.8: Vergleich der Vorhersage des —— finalen Modellensembles mit den Daten des Matlab-Modells, für geringe Stromdichten i. SD_E ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

Das neue Modell, in Abbildungen 5.8a und 5.8b, gibt die Spannungsverläufe der Messungen, in beiden Datensätzen, generell besser wieder, als das alte Modell (vgl. Abbildungen 5.3a und 5.3b). Die größten Abweichungen der Modellvorhersage zu den Daten des Matlab-Modells treten, auch beim neuen Modell, zu Beginn des abrupten Spannungsabfalls auf, siehe Abbildungen 5.8a und 5.8b: Der maximale, relative Fehlerbetrag zwischen Modell (—— bzw. ——) und den Daten des Matlab-Modells (• bzw. •) beträgt, für beide Datensätze, in etwa 10 %. Beim alten Modell betrug er noch etwas mehr als 20 %. Die Standardabweichung, für 0 % H_2O und Stromdichten i bis 50 mA cm^{-2} (—— bzw. ——), ist beim neuen Modell höher, als beim alten Modell. Ab Stromdichten von 50 mA cm^{-2}

und mehr ist sie jedoch geringer.

Vorhersagen für den Verlauf der Zellspannung E, nahe der Modellgrenze von 0.6 V, d.h. im Bereich hoher Stromdichten i, zeigt Abbildung 5.9 (vgl. Abbildung 5.4 für das alte Modell). Darin zeigt Abbildung 5.9a einen Ausschnitt der finalen Trainingsdaten bei einer Zelltemperatur von 825 °C und 30 % H_2 bei variiertem Wasserdampfanteil (in N_2). Abbildung 5.9b zeigt einen Ausschnitt der wahren Testdaten bei einer Zelltemperatur von 800 °C und 35 % H_2 bei variiertem Wasserdampfanteil (in N_2).



(a) Finale Trainingsdaten. Ausschnitt bei einer Zelltemperatur von 825 °C, 30 % H_2 und • 0 %, ▪ 10 % und ▲ 20 % H_2O.

(b) Wahre Testdaten. Ausschnitt bei einer Zelltemperatur von 800 °C, 35 % H_2 und • 0 %, ▪ 5 %, ▲ 10 % und ◆ 15 % H_2O.

Abbildung 5.9: Vergleich der Vorhersage des —— finalen Modellensembles mit den Daten des Matlab-Modells, für hohe Stromdichten i. SD_E ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

Die Abweichung zwischen Modellvorhersage und Daten ist beim neuen Modell, in Abbildungen 5.9a und 5.9b, generell geringer, als beim alten Modell (vgl. Abbildungen 5.4a und 5.4b). Die höchsten Abweichungen treten bei 10 % H_2O (▪) in Abbildung 5.9a und bei 5 % H_2O (▪) in Abbildung 5.9b auf.

Um weiteres Optimierungspotential auszuloten, wurden auch für das neue Modell Lernkurven erstellt. Untersucht wurden die Anzahl der verdeckten Schichten, die Anzahl der Neuronen pro verdeckter Schicht, die Lernrate und die Geduld. Es war jedoch keine nennenswerte Verbesserung mehr möglich.

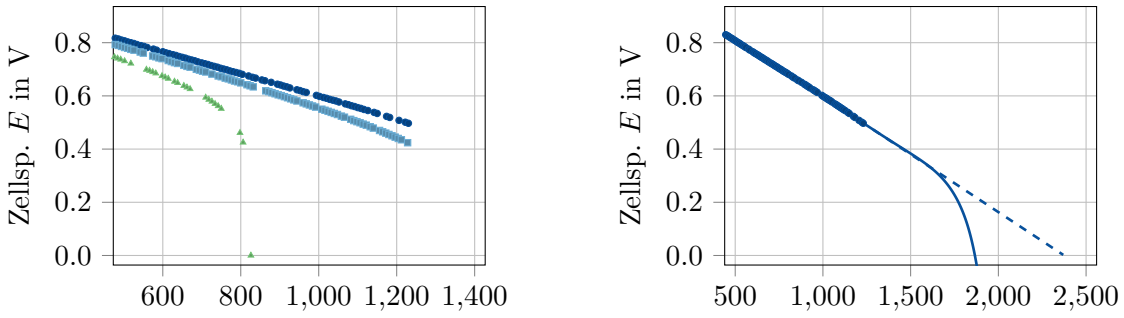
5.1.4 Modellierung des gesamten Spannungsbereiches

Bei den im vorigen Abschnitt behandelten Modellen waren die verwendeten Daten auf den Spannungsbereich von Leerlauf bis 0.6 V beschränkt. Es wurde daher untersucht, ob auch die Bereiche der Polarisationskurven, in denen die Konzentrationsverluste verstärkt auftreten, modelliert werden können. Dazu wurde ein Modell mit den gesamten Pseudo-Messdaten erstellt, d.h. für den Spannungsbereich von Leerlauf bis 0 V. Insgesamt umfassten die zur Modellerstellung verwendeten Daten somit 15066 Datenpunkte. Die wahren Testdaten für den gesamten Spannungsbereich umfassten 115583 Datenpunkte. Ein- und Ausgabegrößen des Modells und die Vorgehensweise bei der Modellerstellung waren gleich, wie bei der Modellierung des beschränkten Spannungsbereichs. Die durchsuchten Parameterbereiche der HP-Optimierung waren geringfügig anders, worauf aber nicht mehr näher eingegangen wird. Nachfolgend werden kurz die Herausforderungen bei der Modellierung zusammengefasst.

Das physikalische Matlab-Modell, mit dem die Pseudo-Messdaten generiert wurden, gab für viele Betriebsbedingungen Polarisationskurven aus, die vor dem Auftreten der Konzentrationsverluste endeten. Bedingt war das durch physikalische Grenzen der Modellierungsansätze, die Details dazu wurden bereits in Abschnitt 5.1.1 behandelt. Abbildung 5.10a zeigt dies anhand eines Ausschnitts der finalen Trainingsdaten bei einer Zelltemperatur von 775 °C, 10 % H_2O und variiertem Wasserstoffanteil. Die generierten Daten enthalten bei 30 % H_2 (▲) den durch die Konzentrationsverluste verursachten Spannungsabfall auf 0 V, bei 50 % H_2 (■) und 70 % H_2 (●) jedoch nicht. Somit ergeben sich, für verschiedene Betriebsbedingungen, verschiedene untere Spannungswerte als Modellgrenze. Für die finalen Trainingsdaten sind diese Spannungswerte bekannt, für neue Daten, zum Beispiel im praktischen Einsatz des Modells, jedoch nicht. Das heißt, es ist für neue Daten nicht genau bekannt, bis zu welchem Spannungswert die Modellvorhersagen innerhalb der vorgesehenen Grenzen liegen. Macht das Modell außerhalb seiner Grenzen dann noch sehr ungenaue oder unphysikalische Vorhersagen, kann das zu erheblichen Problemen führen. Abbildung 5.10b zeigt je ein Beispiel für eine physikalische (——) und eine unphysikalische Vorhersage (- - -) für die Zellspannung E. Unphysikalische Vorhersagen ähnlicher Art traten mehrfach bei während der Arbeit erstellten Modellen auf, konnten aber durch geeignete Parameterwahl vermieden werden.

Abschließend werden noch Vorhersagen des fertig optimierten Modells gezeigt, siehe Abbildung 5.11. Darin zeigt Abbildung 5.11a einen Ausschnitt der finalen Trainingsdaten bei einer Zelltemperatur von 775 °C und 10 % H_2O bei variiertem Wasserstoffanteil (in N_2). Abbildung 5.11b zeigt einen Ausschnitt der wahren Testdaten bei einer Zelltemperatur von 800 °C und 15 % H_2O bei variiertem Wasserstoffanteil (in N_2).

Für die finalen Trainingsdaten, in Abbildung 5.11a, konnten für alle dargestellten Wasserstoffanteile geringe Abweichungen zwischen den Modellvorhersagen und Daten des



(a) Ausschnitt der finalen Trainingsdaten bei ▲ 30 %, ■ 50 % und ● 70 % H_2. Es ergeben sich verschiedene Spannungswerte als untere Modellgrenze.

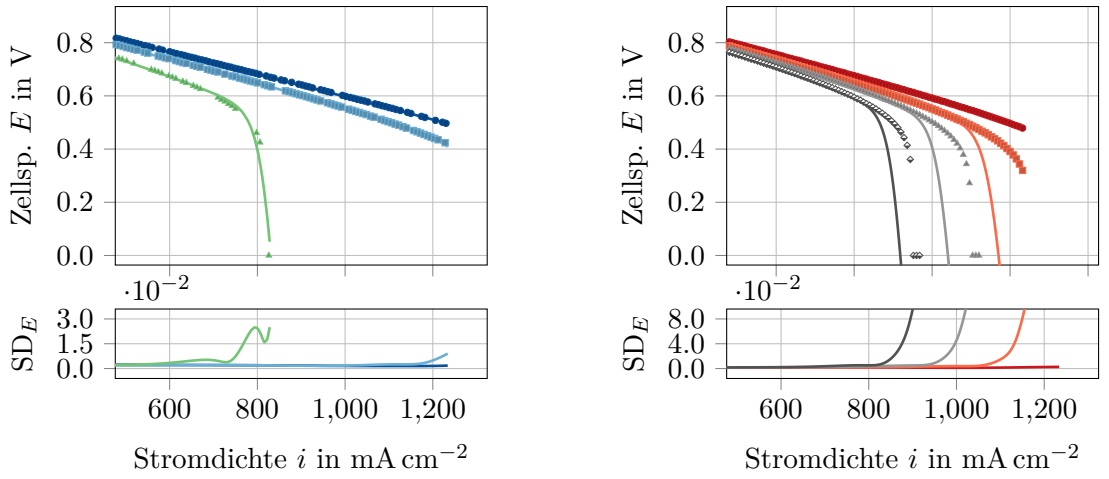
(b) Eine —— physikalisch plausible und eine - - - physikalisch unplausible Vorhersage außerhalb der Modellgrenzen, für die ● finalen Trainingsdaten bei 70 % H_2.

Abbildung 5.10: Herausforderungen bei der Modellierung des gesamten Spannungsbereichs. Dargestellt anhand eines Auszugs der durch das Matlab-Modell generierten finalen Trainingsdaten bei einer Zelltemperatur von 775 °C, 10 % H_2O und variiertem Wasserstoffanteil.

Matlab-Modells erreicht werden. Der Spannungsabfall durch die Konzentrationsverluste in den Daten bei 30 % H_2 (▲) wird von den Modellvorhersagen (——) gut wiedergegeben. Die Standardabweichung SD_E (——) der Modelle im Ensemble steigt im Bereich der Konzentrationsverluste, was auf ungleichmäßiges Training der Modelle hinweist.

Bei den finalen Testdaten, in 5.11b, wird der Spannungsabfall durch die Konzentrationsverluste qualitativ richtig wiedergegeben. Zu sehen ist das an den Modellvorhersagen für 35 %, 40 % und 45 % H_2 (◆/▲/■) im Vergleich mit den Daten des Matlab-Modells (——/——/——). Einzig für 55 % H_2 (●) liegt der, durch die Konzentrationsverluste verursachte, Spannungsabfall der Modellvorhersage (——) außerhalb des Bereichs der wahren Testdaten (d.h. bei Stromdichten i deutlich über 1200 mA cm^{-2}). Die vorhergesagte Spannung fällt für die drei erstgenannten Wasserstoffanteile jedoch früher ab, als in den finalen Testdaten. Dadurch entstehen große relative Fehler. Die Standardabweichung SD_E ist auch bei den Vorhersagen für die finalen Testdaten, im Bereich der Konzentrationsverluste, groß.

Die gezeigten Abweichungen zwischen den Vorhersagen des Modells und den finalen Testdaten, siehe Abbildung 5.11b, konnten durch eine weitere Modelloptimierung nicht weiter verringert werden. Zum Erfolg führte jedoch eine Erhöhung der Menge an Pseudo-Messdaten, durch eine feinere Auflösung der Betriebsparameter des Matlab-Modells: Der Wasserstoffanteil x_{H_2}, der Wasserdampfanteil x_{H_2O} und die Zelltemperatur ϑ wurden mit je 5 Werten variiert, anstatt mit je 3 Werten wie zuvor. Die wahren Testdaten mussten deshalb angepasst werden, wurden aber wieder innerhalb der Grenzen der Pseudo-Messdaten generiert. Das resultierende Modell konnte die konzentrationsbedingten



(a) Finale Trainingsdaten. Ausschnitt bei einer Zelltemperatur von 775 °C, 10 % H_2O und ▲ 30 %, ■ 50 % und ● 70 % H_2.

(b) Finale Testdaten. Ausschnitt bei einer Zelltemperatur von 800 °C, 15 % H_2O und ◆ 35 %, ▲ 40 %, ■ 45 % und ● 55 % H_2.

Abbildung 5.11: Vergleich der Vorhersage des —— finalen Modellensembles mit den Daten des Matlab-Modells, für hohe Stromdichten i. SD_E ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

Spannungsabfälle für die wahren Testdaten mit guter Genauigkeit vorhersagen. Für Daten aus Messungen an einer echten Zelle würde sich der Ansatz, die Datenmenge zu erhöhen, jedoch nur bedingt eignen. Immerhin würde das Messungen von Polarisationskurven bei $5^3 = 125$ Betriebsbedingungen (im Gegensatz zu $3^3 = 27$) erfordern und somit einen enormen Mehraufwand bedeuten. Deshalb wird an dieser Stelle auf eine Darstellung des Modells verzichtet.

5.2 Modellierung der Zellimpedanz

In diesem Abschnitt werden die zur Modellierung der Zellimpedanz erstellten neuronalen Netze vorgestellt. In der ersten Hälfte des Abschnitts wird auf die zur Modellerstellung verwendeten Messdaten und auf den Messaufbau eingegangen. Es werden Untersuchungen gezeigt, die vor der Berechnung der Modelle durchgeführt wurden, um einige Einstellungen der Modellerstellung zu optimieren. Danach werden die Einstellungen der Modellerstellungsprogramme im Gesamten behandelt. In der zweiten Hälfte des Abschnitts werden die berechneten Impedanzmodelle vorgestellt, wobei die Modelle, ausgehend von einem einfachen Realteilmodell, sukzessive aufeinander aufbauen.

5.2.1 Verwendete Daten

Die zur Erstellung der Impedanzmodelle verwendeten Daten stammen aus einer Versuchsreihe an einem SOFC-Einzelzellenprüfstand. Dieser ist schematisch in Abbildung 5.12 dargestellt.

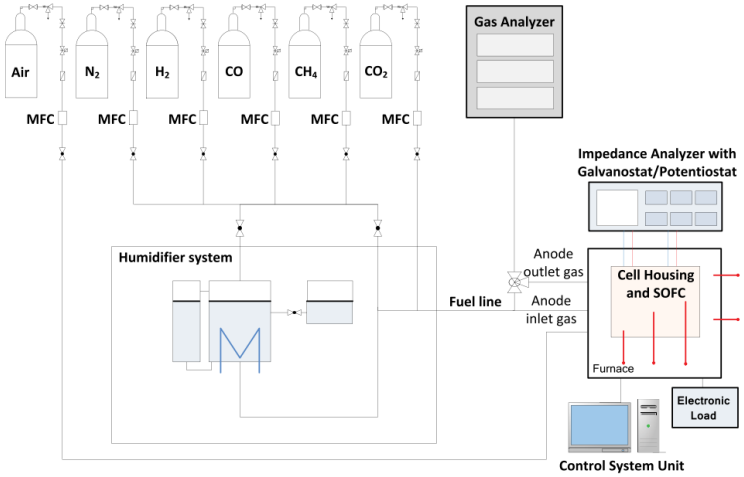


Abbildung 5.12: Schema des Einzelzellenprüfstandes, aus [55].

Die Gasregelstrecke des Prüfstandes enthält einzelne Gaskomponenten, aus denen das Brenngas gemischt wird. In den Zuleitungen für jede Gaskomponente ist ein Massendurchflussregler (mass flow controller, MFC) verbaut, sodass die Zusammensetzung des Brenngases angepasst werden kann. Die Befeuchtungseinheit (humidifier system), genauer gesagt ein Bubbler, dient bei Bedarf zur Befeuchtung des trockenen Brenngases. Ein Gasanalysator ermöglicht die Bestimmung der Brenngaszusammensetzung in der Zu- und Ableitung der Zelle. Der keramische Zellhalter, der die SOFC enthält, befindet sich in einem geregelt beheizten Ein-Zonen-Rohrofen. Mit einem Impedanzmessgerät und einer elektrischen Last wird die SOFC, in verschiedenen Betriebspunkten, vermessen. [55]

Verbaut war eine planare, anodengestützte Brennstoffzelle mit einer elektrochemisch aktiven Fläche von $80\,\mathrm{cm}^2$. Die Brennstoffelektrode (Anode) der Zelle bestand aus Ni/YSZ, der Elektrolyt aus YSZ, die Trennschicht aus gedoptem $CerO_2$ und die Sauerstoffelektrode (Kathode) aus LSCF. Die Anode wurde mit einem Nickelnetz kontaktiert und die Kathode mit einem Platinnetz. Die elektrische Verbindung der Netze zum Prüfstand erfolgte durch Platindrähte. Zur Spannungsmessung wurde je ein dünner Platindraht mit dem Nickel- und dem Platinnetz verbunden. Sowohl Brenngas- als auch Luftzuleitungen wurden vor dem Rohrofen in Edelstahl ausgeführt und auf $300\,^\circ\mathrm{C}$ beheizt. Innerhalb des Ofens

wurden Zu- und Ableitungen von Brenngas und Luft als Keramikrohre ausgeführt. Die Gastemperaturen innerhalb der Keramikrohre, d.h. innerhalb der Zu- und Ableitungen, wurde mittels Typ N Thermoelementen gemessen, ebenso die Temperatur im Rohrofen und die Temperatur des Zellhalters. Als elektrische Last und Impedanzmessgerät wurde das Kombi-Gerät BioLogic SP-150 verwendet. Durch Verwendung des BioLogic VMP3B-80 Boosters wurde der maximale Laststrom auf 80 A erhöht, aber die maximale Spannung auf 3 V eingeschränkt. Der Booster schränkt die maximale Messfrequenz der EIS auf 10 kHz ein. [58]

Die Versuchsreihe umfasste umfangreiche Impedanzmessungen, hier werden aber nur die in dieser Arbeit verwendeten Messungen beschrieben. Die Gaszusammensetzung auf der Brennstoffseite wurde mit 60 % H_2 und 40 % N_2 konstant gehalten. Das Brenngas wurde nicht befeuchtet. Auf der Sauerstoffseite wurde synthetische Luft, mit 21 % O_2 und 79 % N_2, zugeführt. Die Gasvolumenströme waren ebenso konstant, mit $1.2\,\mathrm{l\,min^{-1}}$ und $1.6\,\mathrm{l\,min^{-1}}$ auf der Brennstoff- bzw. Sauerstoffseite. Die Ofentemperatur wurde mit 700 °C, 750 °C und 800 °C variiert. Die verschiedenen Betriebsbedingungen bei den Messungen sind in Tabelle 5.9 zusammengefasst. [58]

Tabelle 5.9: Betriebsbedingungen der SOFC bei den Impedanzmessungen. [58]

Ofen	Brennstoffseite				Sauerstoffseite		
ϑ	x_{H_2}	x_{H_2O}	x_{N_2}	$\dot{V}_F$	x_{O_2}	x_{N_2}	$\dot{V}_O$
°C	%	%	%	$\mathrm{l\,min^{-1}}$	%	%	$\mathrm{l\,min^{-1}}$
700	60	-	40	1.2	21	79	1.6
750	60	-	40	1.2	21	79	1.6
800	60	-	40	1.2	21	79	1.6

Die Anregung bei den Impedanzmessungen erfolgte mit Wechselstrom. Die Amplitude der Anregung betrug 4 % des im jeweiligen statischen Betriebspunkt fließenden Gleichstroms. Bei Messungen im Leerlauf betrug die Amplitude $0.4\,\mathrm{mA\,cm^{-2}}$. Bei einer Ofentemperatur von 700 °C wurde in Betriebspunkten mit Stromdichten von $0.0\,\mathrm{mA\,cm^{-2}}$ bis $250.0\,\mathrm{mA\,cm^{-2}}$, bei 750 °C bis $350.0\,\mathrm{mA\,cm^{-2}}$ und bei 800 °C bis $550.0\,\mathrm{mA\,cm^{-2}}$ gemessen. In allen drei Fällen gab es bis $150.0\,\mathrm{mA\,cm^{-2}}$ alle $10\,\mathrm{mA\,cm^{-2}}$ einen Messpunkt, ab $150.0\,\mathrm{mA\,cm^{-2}}$ alle $50\,\mathrm{mA\,cm^{-2}}$. Die Frequenz umfasste den Bereich von 0.1 Hz bis 10 000 Hz. Jede Frequenzdekade wurde in 10 Messpunkte unterteilt. [58] Zur Veranschaulichung sind die Impedanzmessdaten für eine Ofentemperatur von $\vartheta = 700\,°\mathrm{C}$ und 60 % H_2 in N_2 in Abbildung 5.13 in einem Nyquist-Diagramm dargestellt. Die Zellimpedanz Z sinkt mit steigender Stromdichte i. Von den zwei charakteristischen Kreisbögen, die sich für jede Stromdichte i ausbilden, kann auf Verlustprozesse an den Elektroden geschlossen werden.

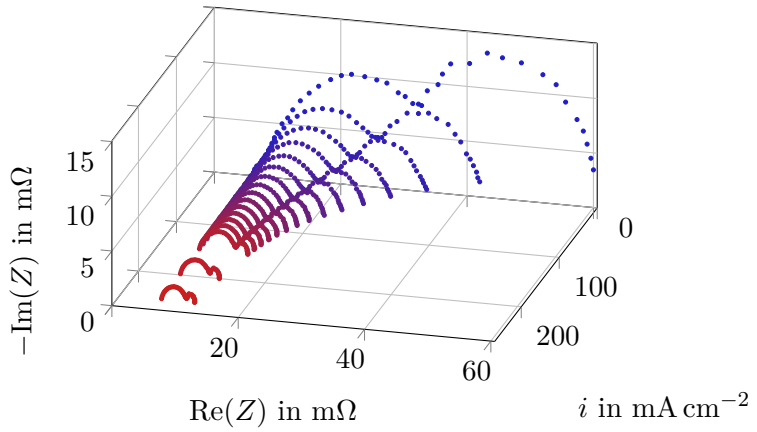


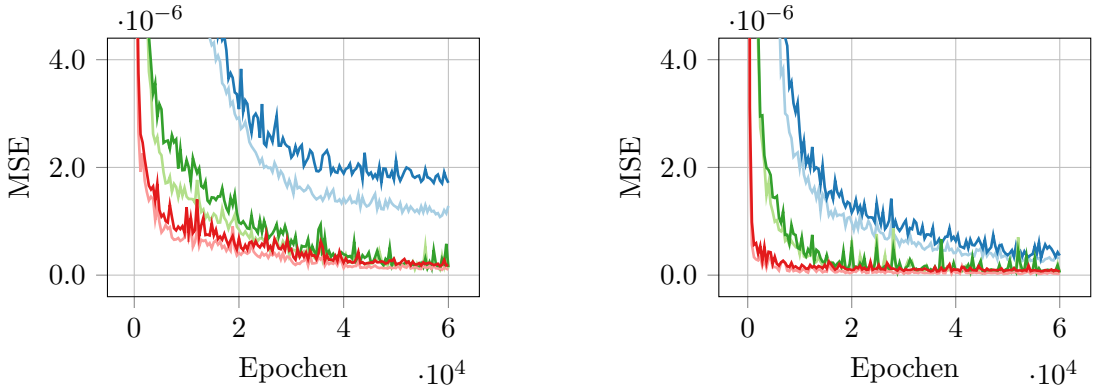
Abbildung 5.13: Impedanzmessdaten bei einer Ofentemperatur von 700 °C und 60 % H_2 in N_2 auf der Brennstoffseite, dargestellt als Nyquist-Diagramm. Variierte Stromdichte i von • 10.0 mA cm^{-2} bis • 250.0 mA cm^{-2}.

5.2.2 Vorab durchgeführte Untersuchungen

Mehrere durchgeführte Rastersuchen für verschiedene Impedanzmodelle – mit ähnlichen Einstellungen, wie sie für die Polarisationsmodelle verwendet wurden – führten sämtlich zu nicht zufriedenstellenden Ergebnissen. Die resultierenden Modelle waren stark unterangepasst. Aus diesem Grund wurden einfache Untersuchungen angestellt, um die wesentlichen Einflüsse der Hyperparameter auf die Modellberechnung zu erkennen. Die in diesen Untersuchungen berechneten Modelle wiesen 3 verdeckte Schichten mit je 10 Neuronen auf. Verwendet wurden die Daten der Realteilmodelle, die später in Abschnitt 5.2.4 genauer beschrieben werden. Die Testdaten, 15 % der Gesamtdaten, wurden zufallsbasiert ausgewählt. Die übrigen 85 % wurden als Trainingsdaten verwendet.

Zuerst wird die Transformation der Eingangsdaten betrachtet. Bei den bisherigen Berechnungen wurden die Eingangsdaten linear zwischen einem Minimal- und einem Maximalwert skaliert, genauer zwischen 0.2 und 0.8. Die verwendete Initialisierung nach Glorot und Bengio, vgl. Abschnitt 3.3.3, wurde jedoch mit der Annahme hergeleitet, dass die Varianz bzw. die Standardabweichung der Eingabegrößen gleich ist. Deshalb soll hier die lineare Skalierung mit einer Standardisierung verglichen werden. Die Standardisierung transformiert die Werte jeder Eingabegröße auf Mittelwert 0 und Varianz bzw. Standardabweichung 1. Berechnet wurden für die lineare Skalierung und die Standardisierung der Fehlerverlauf im Training je dreier Modelle mit verschiedenen Aktivierungsfunktionen (Sigmoid, Tanh, ReLU). Ein Fehlerverlauf ist dabei immer ein Mittelwert aus den Verläufen 15 zufällig initialisierter Netzwerke. Optimiert wurden die inneren Parameter (Gewichte und Bias) der Modelle mit dem Adam-Algorithmus mit einer Lernrate von

$\eta = 0.001$ und den Parametern $\beta_1 = 0.9$ und $\beta_2 = 0.999$. Abbildung 5.14 zeigt die Fehlerverläufe im Vergleich. Die niedrigsten, im Training erreichten Testfehler, sind in Tabelle 5.10 zusammengefasst. Die aufgeführten Trainingsfehler stammen dabei aus derselben Epoche wie die jeweils ermittelten niedrigsten Testfehler.



(a) Lineare Skalierung der Eingabegrößen.

(b) Standardisierung der Eingabegrößen.

Abbildung 5.14: Fehlerverläufe (Mittelwert aus dem Training je 15 zufällig initialisierter Netzwerke) für die lineare Skalierung und die Standardisierung. MSE des —/— Sigmoid-Netzwerkes, des —/— Tanh-Netzwerkes und des —/— ReLU Netzwerkes für die Trainings-/Testdaten.

Tabelle 5.10: Niedrigste erreichte Testfehler (MSE) während des Trainings, mit den zugehörigen Trainingsfehlern. Vergleich zwischen linearer Skalierung und Standardisierung für die Aktivierungsfunktionen Sigmoid, Tanh und ReLU. Die zugehörigen Fehlerverläufe sind in Abbildung 5.14 dargestellt.

	lin. Skalierung		Standardisierung	
Akt.Fkt.	Training	Test	Training	Test
Sigmoid	$1.10 \cdot 10^{-6}$	$1.54 \cdot 10^{-6}$	$1.98 \cdot 10^{-7}$	$2.53 \cdot 10^{-7}$
Tanh	$9.48 \cdot 10^{-8}$	$6.66 \cdot 10^{-8}$	$3.37 \cdot 10^{-8}$	$2.80 \cdot 10^{-8}$
ReLU	$9.78 \cdot 10^{-8}$	$1.48 \cdot 10^{-7}$	$2.04 \cdot 10^{-8}$	$5.97 \cdot 10^{-8}$

Die Fehlerverläufe in Abbildungen 5.14a und 5.14b – sowohl die der linearen Skalierung als auch der Standardisierung – zeigen, dass das ReLU-Netzwerk (—/—) am schnellsten konvergiert. Das Tanh-Netzwerk (—/—) konvergiert etwas langsamer, das Sigmoid-Netzwerk (—/—) hingegen deutlich langsamer. Das ReLU-Netzwerk und das Tanh-Netzwerk erreichen auch einen geringeren Testfehler als das Sigmoid-Netzwerk ($5.97 \cdot 10^{-8}$ und $2.80 \cdot 10^{-8}$ gegenüber $2.53 \cdot 10^{-7}$), vgl. Tabelle 5.10. Ob das Sigmoid-Netzwerk bei längerem Training einen vergleichbar geringen Fehler erreichen könnte, kann aus den Daten nicht abgeleitet werden. Der Vergleich zwischen Abbildung 5.14a und 5.14b zeigt, dass die Standardisierung die Konvergenz im Training für alle 3 untersuchten

Aktivierungsfunktionen beschleunigt. Zudem können mit der Standardisierung deutlich geringere Testfehler erreicht werden, als mit der linearen Skalierung, siehe Tabelle 5.10.

Als Nächstes soll der Einfluss unterschiedlicher Optimierer dargestellt werden. Untersucht wurde der Gradientenabstieg mit Moment, der RMSprop-Algorithmus und der Adam-Algorithmus, vgl. Abschnitt 3.3.5. Die Parameter aller 3 Optimierer wurden nach den Standardeinstellungen der Keras API gewählt. Die Standardeinstellungen sind zusammengefasst in Tabelle 5.11.

Tabelle 5.11: Standardparameter für die Optimierer in der Keras API.

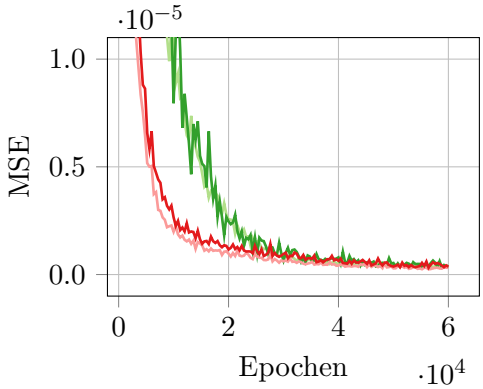
Gradientenabstieg		RMSprop		Adam		
η	α	η	β	η	β_1	β_2
0.01	0.9	0.001	0.9	0.001	0.9	0.999

Diese Vorgehensweise erlaubt nur einen sehr groben Überblick, da die Parameter (insbesondere die Lernrate) eigentlich für jeden Algorithmus optimiert werden sollten. Solche weiterführenden Untersuchungen hätten jedoch den Zeitrahmen der vorliegenden Arbeit gesprengt. Als Transformation der Eingabedaten wurde, aufgrund der vorher gezeigten Ergebnisse, die Standardisierung gewählt. Erneut wurden die Fehlerverläufe je eines Sigmoid-, Tanh- und ReLU-Netzwerkes berechnet. Die Vorgehensweise und restliche Konfiguration waren ansonsten gleich wie zuvor. Dargestellt sind die Fehlerverläufe in Abbildung 5.15. Die niedrigsten, im Training erreichten Testfehler, sind in Tabelle 5.12 zusammengefasst.

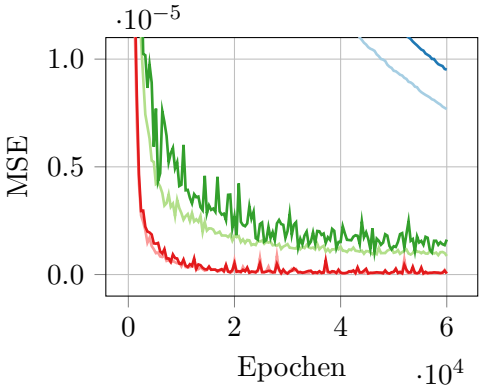
Tabelle 5.12: Niedrigste erreichte Testfehler (MSE) während des Trainings mit den zugehörigen Trainingsfehlern. Vergleich der Optimierer für die Aktivierungsfunktionen Sigmoid, Tanh und ReLU. Die zugehörigen Fehlerverläufe sind in Abbildung 5.15 dargestellt.

	Gradientenabstieg		RMSprop		Adam	
Akt.Fkt.	Training	Test	Training	Test	Training	Test
Sigmoid	$8.33 \cdot 10^{-5}$	$1.12 \cdot 10^{-4}$	$5.33 \cdot 10^{-7}$	$2.10 \cdot 10^{-7}$	$1.98 \cdot 10^{-7}$	$2.53 \cdot 10^{-7}$
Tanh	$7.67 \cdot 10^{-6}$	$9.46 \cdot 10^{-6}$	$1.13 \cdot 10^{-6}$	$5.36 \cdot 10^{-7}$	$3.37 \cdot 10^{-8}$	$2.80 \cdot 10^{-8}$
ReLU	$9.01 \cdot 10^{-7}$	$1.55 \cdot 10^{-6}$	$6.92 \cdot 10^{-7}$	$2.65 \cdot 10^{-7}$	$2.04 \cdot 10^{-8}$	$5.97 \cdot 10^{-8}$

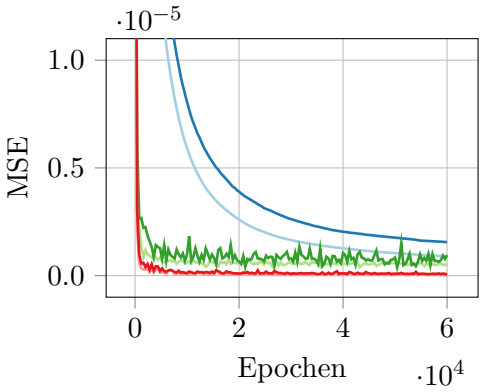
Abbildungen 5.15a, 5.15b und 5.15c zeigen, dass der Adam-Algorithmus (——/——) für alle drei Aktivierungsfunktionen am schnellsten konvergiert. Der RMSprop-Algorithmus (——/——) konvergiert langsamer, der Gradientenabstieg mit Moment (——/——) im Vergleich dazu deutlich langsamer. In Abbildungen 5.15a und 5.15c liegt der Fehlerverlauf des Gradientenabstiegs mit Moment sogar außerhalb des dargestellten Bereichs. Der Adam-Algorithmus erreicht bei Verwendung der Tanh- und ReLU-Aktivierungsfunktion die geringsten Testfehler ($2.80 \cdot 10^{-8}$ und $5.97 \cdot 10^{-8}$), vgl. Tabelle 5.12. Bei Verwendung der Sigmoid-Aktivierungsfunktion erreicht der RMSprop-Algorithmus den geringsten



(a) Sigmoid-Netzwerk.



(b) Tanh-Netzwerk.

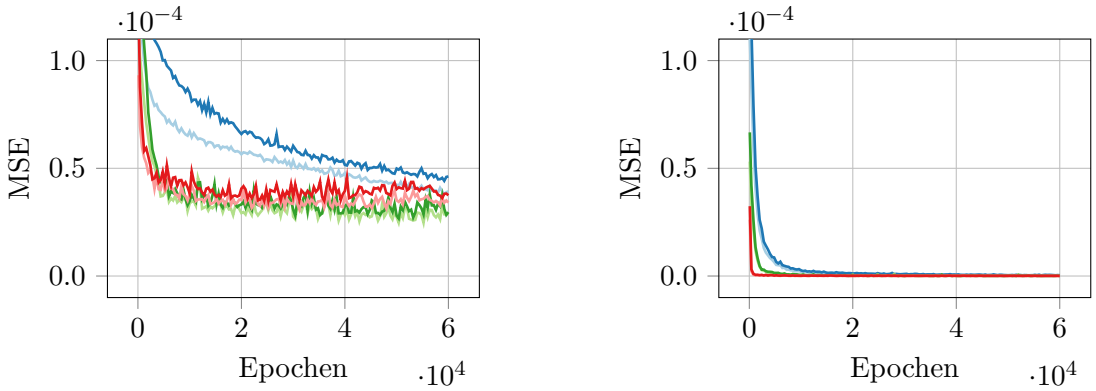


(c) ReLU-Netzwerk.

Abbildung 5.15: Fehlerverläufe (Mittelwert aus dem Training je 15 zufällig initialisierter Netzwerke) für die Optimierer. MSE des —/— Gradientenabstiegs mit Moment, des —/— RMSprop-Algorithmus und des —/— Adam-Algorithmus für die Trainings-/Testdaten.

Testfehler ($2.10 \cdot 10^{-7}$), der sich aber in derselben Größenordnung wie der Testfehler des Adam-Algorithmus ($2.53 \cdot 10^{-7}$) befindet. Erneut sei auf die eingeschränkte Aussagekraft des Vergleichs hingewiesen, da nur die Standardeinstellungen der Optimierer verwendet wurden.

Für die bisher durchgeführten Untersuchungen wurde die Anregefrequenz f der Daten, vor weiteren Transformationen (z.B. der Standardisierung), dekadisch logarithmiert. Die Idee hinter dieser Vorgehensweise soll an dieser Stelle kurz dargelegt werden: Die Messdaten weisen 10 Messpunkte pro Frequenzdekade auf, die gleichmäßig auf den Logarithmus der Frequenz (den Wertebereich des Exponenten) verteilt sind. Dies hat zur Folge, dass ein Frequenzschritt bei den niedrigsten gemessenen Frequenzen in etwa 0.027 Hz beträgt, bei den höchsten Frequenzen aber in etwa 2100 Hz. Wird die Frequenz logarithmiert, sind alle Schritte zwischen den erhaltenen Exponenten gleich groß, was das Training des neuronalen Netzes erleichtert. Diese Annahme wurde bereits im Voraus durch eine Vergleichsrechnung bestätigt, soll hier aber noch einmal im Rahmen der anderen Untersuchungen gezeigt werden. Es wurden Fehlerverläufe je eines Sigmoid-, Tanh-, und ReLU-Netzwerkes berechnet. Verwendet wurden die Standardisierung und der Adam-Optimierer (mit den Standardparametern). Die Fehlerverläufe sind in Abbildung 5.16 dargestellt. Die niedrigsten, im Training erreichten Testfehler, sind in Tabelle 5.13 zusammengefasst.



(a) Ohne Logarithmierung der Frequenz.

(b) Mit Logarithmierung der Frequenz.

Abbildung 5.16: Fehlerverläufe (Mittelwert aus dem Training je 15 zufällig initialisierter Netzwerke), mit und ohne Logarithmierung der Anregefrequenz f. MSE des —/— Sigmoid-Netzwerkes, des —/— Tanh-Netzwerkes und des —/— ReLU-Netzwerkes für die Trainings-/Testdaten.

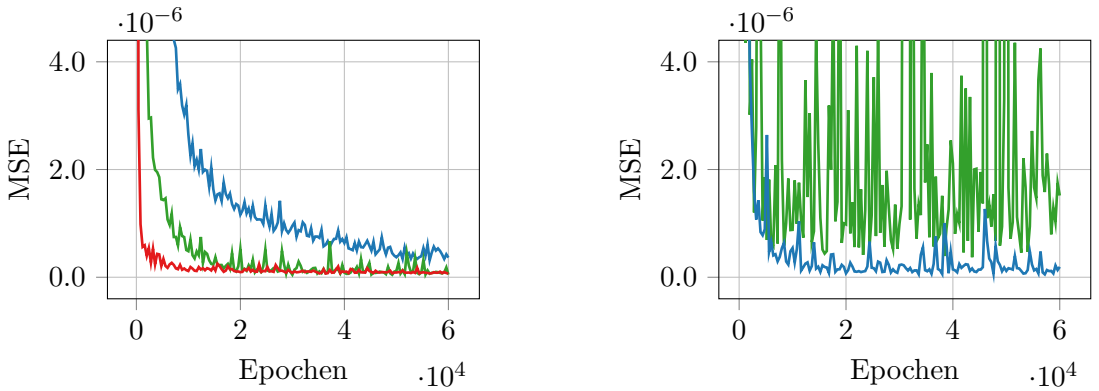
Ein Vergleich von Abbildungen 5.16a und 5.16b zeigt, dass die Logarithmierung der Frequenz die Konvergenz im Training enorm beschleunigt. Mit Logarithmierung werden, für alle drei Aktivierungsfunktionen, durchwegs um Größenordnungen kleinere Testfehler

Tabelle 5.13: Niedrigste erreichte Testfehler (MSE) während des Trainings, mit den zugehörigen Trainingsfehlern. Vergleich mit und ohne Logarithmierung der Anregefrequenz f, für die Aktivierungsfunktionen Sigmoid, Tanh und ReLU. Die zugehörigen Fehlerverläufe sind in Abbildung 5.16 dargestellt.

	ohne $\log_{10}(f)$		mit $\log_{10}(f)$	
Akt.Fkt.	Training	Test	Training	Test
Sigmoid	$3.92 \cdot 10^{-5}$	$4.32 \cdot 10^{-5}$	$1.98 \cdot 10^{-7}$	$2.53 \cdot 10^{-7}$
Tanh	$2.41 \cdot 10^{-5}$	$2.54 \cdot 10^{-5}$	$3.37 \cdot 10^{-8}$	$2.80 \cdot 10^{-8}$
ReLU	$3.15 \cdot 10^{-5}$	$3.22 \cdot 10^{-5}$	$2.04 \cdot 10^{-8}$	$5.97 \cdot 10^{-8}$

erreicht ($2.53 \cdot 10^{-7}$, $2.80 \cdot 10^{-8}$ und $5.97 \cdot 10^{-8}$), als ohne Logarithmierung ($4.32 \cdot 10^{-5}$, $2.54 \cdot 10^{-5}$ und $3.22 \cdot 10^{-5}$), vgl. Tabelle 5.13.

Abschließend soll noch der Einfluss der Lernrate untersucht werden. Aufgrund der bereits gezeigten Ergebnisse, wurde die Anregefrequenz zuerst logarithmiert und danach wurden alle Eingabegrößen standardisiert. Als Optimierer wurde der Adam-Algorithmus verwendet. Es wurden, mit einer erhöhten Lernrate von $\eta = 0.01$ gegenüber $\eta = 0.001$, erneut die Fehlerverläufe der Sigmoid-, Tanh- und ReLU-Netzwerke berechnet. Die Vorgehensweise und restliche Konfiguration waren ansonsten wieder gleich wie zuvor. Abbildung 5.17 zeigt die berechneten Fehlerverläufe, wobei diesmal die Trainingsfehler aus Gründen der Übersichtlichkeit von der Darstellung ausgenommen wurden. Die niedrigsten, im Training erreichten Testfehler, sind in Tabelle 5.10 zusammengefasst.



(a) Lernrate $\eta = 0.001$.

(b) Lernrate $\eta = 0.01$.

Abbildung 5.17: Fehlerverläufe (Mittelwert aus dem Training je 15 zufällig initialisierter Netzwerke) für die Lernraten $\eta = 0.001$ und $\eta = 0.01$. MSE des — Sigmoid-Netzwerkes, des — Tanh-Netzwerkes und des — ReLU Netzwerkes für die Testdaten. Der Fehlerverlauf des ReLU-Netzwerkes liegt in (b) außerhalb des dargestellten Bereiches.

Der Vergleich zwischen Abbildungen 5.17a und 5.17b zeigt, dass das Sigmoid-Netzwerk

Tabelle 5.14: Niedrigste erreichte Testfehler (MSE) während des Trainings mit den zugehörigen Trainingsfehlern. Vergleich zwischen den Lernraten $\eta = 0.001$ und $\eta = 0.01$ für die Aktivierungsfunktionen Sigmoid, Tanh und ReLU. Die zugehörigen Fehlerverläufe sind in Abbildung 5.17 dargestellt.

	$\eta = 0.001$		$\eta = 0.01$	
Akt.Fkt.	Training	Test	Training	Test
Sigmoid	$1.98 \cdot 10^{-7}$	$2.53 \cdot 10^{-7}$	$2.21 \cdot 10^{-8}$	$2.29 \cdot 10^{-8}$
Tanh	$3.37 \cdot 10^{-8}$	$2.80 \cdot 10^{-8}$	$2.99 \cdot 10^{-7}$	$2.18 \cdot 10^{-7}$
ReLU	$2.04 \cdot 10^{-8}$	$5.97 \cdot 10^{-8}$	$1.41 \cdot 10^{-5}$	$1.70 \cdot 10^{-5}$

(——) von der erhöhten Lernrate $\eta = 0.01$ profitiert. Es konvergiert schneller und erreicht geringere Fehler. Auf die Tanh- und ReLU-Netzwerke wirkt sich die erhöhte Lernrate nachteilig aus: Der Fehler des Tanh-Netzwerks (——) oszilliert stark. Der Fehler des ReLU-Netzwerkes (——) konvergiert zwar schnell, aber gegen einen hohen Wert, sodass der Verlauf sogar außerhalb des dargestellten Bereichs liegt. Möglicherweise handelt es sich dabei um das „dying ReLU“ Problem, vgl. Abschnitt 3.3.2. Dieses tritt unter anderem auf, wenn ReLUs durch zu hohe Lernraten in ihren inaktiven Bereich, d.h. in Sättigung, gebracht werden. Das Training des Netzwerkes wird dadurch behindert oder sogar ganz unterbunden. Für die Gegenüberstellung der erreichten Fehler sei noch einmal auf Tabelle 5.10 verwiesen. Unerwartet ist, dass der niedrigste erreichte Testfehler des Sigmoid-Netzwerkes bei erhöhter Lernrate ($2.29 \cdot 10^{-8}$) sogar geringer ist als der der Tanh- und ReLU-Netzwerke bei der ursprünglichen Lernrate von $\eta = 0.001$ ($2.80 \cdot 10^{-8}$ und $5.97 \cdot 10^{-8}$), vgl. Tabelle 5.14. Die wichtigsten Ergebnisse dieses Abschnitts sollen nun kurz zusammengefasst werden:

- Die Standardisierung bietet Vorteile gegenüber der linearen Skalierung: Netzwerke mit den Aktivierungsfunktionen Sigmoid, Tanh und ReLU, wurden mit der Standardisierung schneller trainiert und erreichten geringere Fehler.
- Der Adam-Optimierer führte, bei Netzwerken mit den Aktivierungsfunktionen Sigmoid, Tanh und ReLU, zu schnellerer Konvergenz als der Gradientenabstieg mit Impuls und RMSprop. Bei den Netzwerken mit Tanh und ReLU, führte er auch zu geringeren Testfehlern.
- Die (dekadische) Logarithmierung der Anregefrequenz f beschleunigt das Training enorm und führt zu um Größenordnungen kleineren Testfehlern.
- Sigmoid-Netzwerke benötigen eine höhere Lernrate als Tanh- und ReLU-Netzwerke: Das Training des Sigmoid-Netzwerks wurde durch die Erhöhung der Lernrate von $\eta = 0.001$ auf $\eta = 0.01$ beschleunigt und die niedrigsten erreichten Fehler verringert. Im Gegensatz dazu stiegen die Fehler bei den Tanh- und ReLU-Netzwerken durch die Erhöhung der Lernrate stark.

Es sei darauf hingewiesen, dass diese Erkenntnisse nicht allgemein gültig sind. Durch die

sequentiell aufeinander aufbauenden Rechnungen, konnte nur ein kleiner Teil der Wechselwirkungen der Hyperparameter ermittelt werden. Außerdem wurden die Rechnungen nur für einen bestimmten Anwendungsfall, die Modellierung des Realteils (wie später in Abschnitt 5.2.4 beschrieben), angestellt. Deshalb wurde die Annahme getroffen, dass die anderen in dieser Masterarbeit berechneten Impedanzmodelle soweit Ähnlichkeiten mit dem Realteilmodell aufweisen, dass sich die Ergebnisse übertragen lassen.

5.2.3 Globale Einstellungen der Modellerstellung

Bei Erstellung der Polarisationsmodelle, bereits behandelt in Abschnitt 5.1, wurden immer sowohl die Modellerstellung mit zufallsbasierter Datenteilung als auch die Modellerstellung mit Kreuzvalidierung ausgeführt. Zur Erstellung der Impedanzmodelle wurde nur mehr die Modellerstellung mit zufallsbasierter Datenteilung verwendet. Der Hauptgrund für diese Vorgehensweise war, dass sehr viele verschiedene Rechnungen notwendig waren, bis die Erstellung brauchbarer Impedanzmodelle gelang. Durch die Beschränkung auf nur eine Modellerstellungsprozedur konnten die hohen benötigten Rechenzeiten halbiert werden. Außerdem hatte sich bei Erstellung der Polarisationsmodelle gezeigt, dass sich die beiden genannten Modellerstellungsprozeduren hinsichtlich der Ergebnisqualität nicht wesentlich unterschieden, vgl. Abschnitt 5.1.2.

Bei der Erstellung der verschiedenen Impedanzmodelle mit zufallsbasierter Datenteilung, wurden – wie bei den Polarisationsmodellen – viele Hyperparameter konstant gehalten. Welche Parameter das waren und wie sie gewählt wurden, wird im Folgenden behandelt. Die gewählten Einstellungen der Modellerstellung sind in Tabelle 5.15 zusammengefasst. Für eine Beschreibung der darin vorkommenden Datensätze und wofür diese bei der Modellerstellung verwendet werden, sei auf Abschnitt 4.3 verwiesen.

Die *Daten* wurden manuell in HPO-Daten und finale Testdaten aufgeteilt. Genau genommen wurden die finalen Testdaten manuell ausgewählt und der Rest der Daten als HPO-Daten verwendet. Durch diese Vorgehensweise kann beeinflusst werden, wie das Generalisierungsverhalten des finalen Modells getestet wird. Die manuelle Wahl der Testdaten wird im späteren Verlauf dieses Kapitels für die verschiedenen Modellierungsprobleme gesondert beschrieben, da teilweise unterschiedliche Datensätze verwendet wurden. Die *HPO-Daten* wurden danach, über vorgegebene Prozentsätze, zufallsbasiert in HPO-Trainings-, Stopp- und Testdaten (70 %, 15 % und 15 %) aufgeteilt. Der Anteil der Stoppdaten an den HPO-Daten (15 %) wurde gegenüber den Polarisationsmodellen (4 %, vgl. Abschnitt 5.1.2) erhöht. Dadurch wurde ein zu frühes Stoppen des Trainings, aufgrund eines zu kleinen und nicht repräsentativen Stoppdatensatzes, vermieden. Die Anzahl der *Wiederholungen* in der *HP-Optimierung* wurde, im Vergleich zu den Polarisationsmodellen, verdreifacht (von 5 auf 15). Bei mehrfacher Ausführung derselben Modellerstellung (für die Realteilmodelle), mit nur 5 Wiederholungen, schwankten die gefundenen, optimalen

Tabelle 5.15: Konstant gewählte Einstellungen der Modellerstellung mit zufallsbasierter Datenteilung für alle berechneten EIS-Modelle.

Einstellung	gewählt
Aufteilung Daten	
HPO-Daten	manuell
finale Testdaten	manuell
Aufteilung HPO-Daten	
HPO-Trainingsdaten	70 %
Stoppdaten	15 %
HPO-Testdaten	15 %
Wiederholungen	
in HP-Optimierung	15
in Ensemblebildung	30
Datenpipeline	
Transformation f	$\log_{10}$
Transformation EG	Standardisierung

Hyperparameter sehr. Um die Gesamtrechenzeit der Modellerstellung, bei Verdreifachung der Wiederholungen in der HP-Optimierung, in etwa gleich zu halten, muss jedoch die Anzahl der untersuchten HP-Kombinationen gedrittelt werden. Dazu sei wieder auf Gleichung (4.1) verwiesen. Ob dies dann in Summe zu besseren Ergebnissen führt ist, wie bei den Polarisationsmodellen, kaum abschätzbar. Die *Wiederholungen* in der *Ensemblebildung* wurden, im Vergleich zu den Polarisationsmodellen, ebenfalls verdreifacht (von 10 auf 30). Nach Gleichung (4.1) erhöht sich damit die Gesamtanzahl an in der Modellerstellung berechneter Modelle von 760 auf 780. Die Gesamtrechenzeit der Modellerstellung steigt dadurch nicht wesentlich. Die *Datenpipeline* wurde so aufgebaut, dass die Frequenzwerte der Impedanzdaten zuerst logarithmiert wurden. Die Logarithmierung beschleunigte, wie in den Untersuchungen in Abschnitt 5.2.2 gezeigt wurde, die Konvergenz im Training und führte zu geringeren Testfehlern. Nach der Logarithmierung der Frequenz wurden die Werte jeder Eingabegröße standardisiert – das heißt, sie wurden auf Mittelwert 0 und Varianz 1 transformiert. Bei den Polarisationsmodellen wurden die Werte der Eingabegrößen noch linear mit einem Minimal- und einem Maximalwert skaliert. Dass die Standardisierung Vorteile gegenüber der linearen Skalierung bietet, wurde auch in den Untersuchungen im bereits erwähnten Abschnitt 5.2.2 gezeigt.

Nun sollen die konstant gewählten Hyperparameter der neuronalen Netze behandelt werden. Diese wurden bei der Erstellung aller in diesem Kapitel vorgestellten Impedanzmodelle beibehalten. Eine Übersicht bietet Tabelle 5.16.

Die *Gewichtsinitialisierung*, vgl. Abschnitt 3.3.3, wurde immer nach Glorot und Bengio

Tabelle 5.16: Konstant gewählte Hyperparameter für alle berechneten Impedanzmodelle.

Hyperparameter	gewählt
Gewichtsinitialisierung	nach Glorot u. Bengio
Biasinitialisierung	mit 0
Optimierer	Adam
Batchgröße	64
Kostenfunktion	MSE
Min. Verbesserung	$1 \cdot 10^{-11}$
Max. Epochenanzahl	60 000

durchgeführt. Obwohl diese für ReLU-Netzwerke hergeleitet wurde, eignet sie sich auch für Sigmoid-Netzwerke, um die Sättigung zu verringern. Die *Bias*, vgl. Abschnitt 3.3.4, wurden alle mit 0 initialisiert. Die Wahl des Adam-Algorithmus, vgl. Abschnitt 3.3.5, als *Optimierer*, hatte bei den Polarisationsmodellen noch eher praktische Gründe. Die in Abschnitt 5.2.2 gewonnenen Erkenntnisse bestätigten diese Wahl für die Impedanzmodelle: der Adam-Optimierer führte, verglichen mit zwei anderen Optimierern, zu schnellerer Konvergenz und geringeren Testfehlern. Die *Batchgröße*, vgl. Abschnitt 3.3.6, wurde – wie bei den Polarisationsmodellen – konstant gesetzt, da das Lernverhalten über die Lernrate angepasst wurde. Die gewählte Größe von 64 erscheint nicht groß, die HPO-Trainingsdaten der einfachsten Impedanzmodelle umfassten jedoch nur 490 Datenpunkte. Die *Kostenfunktion*, die *minimale Verbesserung* und die *maximale Epochenanzahl* wurden aus den gleichen Überlegungen gewählt, wie schon bei den Polarisationsmodellen, siehe Abschnitt 5.2.3.

5.2.4 Modellierung des Realteils

Als Einstieg in die Modellierung der Zellimpedanz wurde zunächst nur der Realteil der Impedanz $\mathrm{Re}(Z)$ modelliert, der somit die Ausgabegröße des Modells war. Verwendet wurden die Messdaten bei einer Temperatur von 700 °C, sowie konstanten Volumenströmen und Gaszusammensetzungen, vgl. Tabelle 5.9. Damit ergaben sich als Eingabegrößen der Modelle die Gleichstromdichte i, die den statischen Betriebspunkt definiert, um den die Zelle zur Impedanzmessung angeregt wird und die Anregefrequenz f. Abbildung 5.18 zeigt das Modell als Blackbox, mit den Eingabe- und Ausgabegrößen. Tabelle 5.17 zeigt einen Ausschnitt der zur Modellerstellung verwendeten Daten, bevor sie der Datenpipeline zugeführt wurden, d.h. in untransformiertem Zustand. Im Gesamten umfassten die Daten 900 Datenpunkte.

Die Testdaten wurden manuell ausgewählt. Gewählt wurden alle Datenpunkte bei einer Gleichstromdichte von 10, 70, 130 und 200 $\mathrm{mA\,cm^{-2}}$. Es wurde versucht, mit dieser Wahl

Abbildung 5.18: Blackboxansicht des Realteilmodells mit Eingabe- und Ausgabegrößen.

Tabelle 5.17: Ausschnitt der zur Erstellung der Realteilmodelle verwendeten Daten.

Eingaben		Ausgabe
i	f	Re(Z)
mA cm^{-2}	Hz	mΩ
20.0	0.103	43.2
20.0	0.130	42.9
20.0	0.164	42.7
⋮	⋮	⋮

den gesamten Stromdichtebereich gut abzudecken, um die Generalisierung des finalen Modells umfassend zu testen. Alle übrigen Datenpunkte bildeten somit die HPO-Daten, die zur Optimierung der Hyperparameter verwendet wurden. Die weiteren, globalen Einstellungen bei der Modellerstellung wurden bereits in Abschnitt 5.2.3 beschrieben.

Die Hyperparameter, die in der Modellerstellung optimiert wurden, waren: die Wahl der Aktivierungsfunktion, die Anzahl der verdeckten Schichten, die Anzahl der Neuronen pro verdeckter Schicht, die Lernrate des Optimierers und die Geduld des Early Stoppings. Die durchsuchten Wertebereiche dieser Parameter sind in Tabelle 5.18 angegeben.

Tabelle 5.18: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das ermittelte Optimum. Berechnungszeit ~ 46h.

Hyperparameter	Durchsuchter Bereich	Optimum
Aktivierungsfunktion	Sigmoid, Tanh und ReLU	Sigmoid
verdeckte Schichten	1 bis 3	3
Neuronen pro verd. Sch.	3 bis 10	9
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$1 \cdot 10^{-2}$
Geduld in Epochen	500 bis 1000	662

Die vorgegebenen Parameterbereiche, in Tabelle 5.18, orientierten sich generell an denen der Polarisationsmodelle (vgl. Tabelle 5.5): Für die Realteilmodelle wurden 3 Aktivierungsfunktionen untersucht, diesmal wurde aber die Tanh-Funktion (anstatt der ELU-Funktion bei den Polarisationsmodellen) verwendet. Da die Realteilmodelle nur 2 (anstatt 4) Eingabegrößen aufwiesen wurde angenommen, dass diese weniger komplizierte Netzwerke benötigen: Die Bereichsgrenzen der verdeckten Schichtanzahl blieben gleich, die Anzahl

der Neuronen pro verdeckter Schicht wurde jedoch auf 10 (anstatt 15) begrenzt. Der Bereich der Lernrate blieb gleich wie bei den Polarisationsmodellen, die Geduld wurde jedoch mit 500 bis 1000 Epochen (anstatt 5 bis 20 Epochen) um ein Vielfaches höher angesetzt. Die Notwendigkeit dieser hohen Geduldwerte wurde durch mehrfache Versuchsrechnungen und dem Einsatz von Lernkurven ermittelt, deren Ergebnisse hier aber nicht gezeigt werden.

Aus dem durch die Hyperparameter aufgespannten Raum, wurden durch die Zufallssuche 50 Parameterkombinationen ausgewählt und die beste Kombination ermittelt. Als berechnetes Optimum ergaben sich die Sigmoid-Aktivierungsfunktion, 3 verdeckte Schichten mit je 9 Neuronen, eine Lernrate von $1 \cdot 10^{-2}$ und eine Geduld für das Early Stopping von 662 Epochen. Diese Werte sind ebenso in Tabelle 5.18 angegeben. Die Berechnungszeit mit der CPU des Desktop Xeon (Xeon X3360 @ 2.83 GHz, 4C/4T) betrug etwas mehr als 46 Stunden.

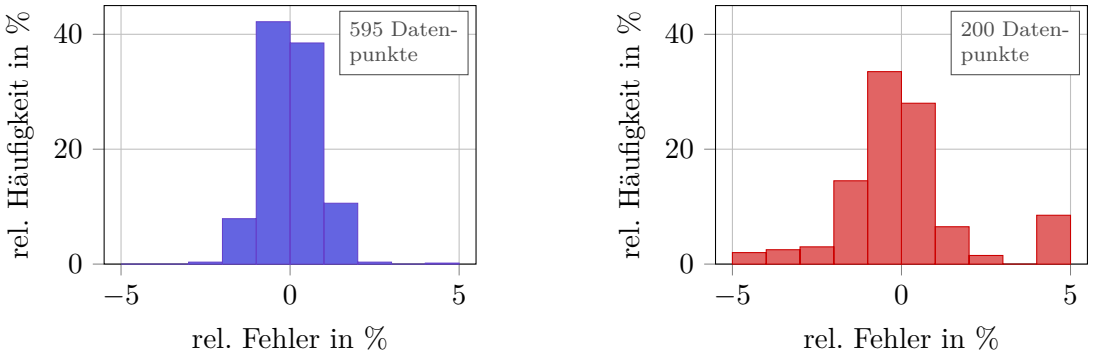
Das mit dem ermittelten HP-Optimum berechnete finale Modell, ein Ensemble aus 30 trainierten Modellen, erreichte einen MSE von $1.53 \cdot 10^{-8}$/ $1.37 \cdot 10^{-5}$ und einen MAPE von 0.57 %/2.31 % im Training/Test. Zusammengefasst sind diese Ergebnisse in Tabelle 5.19. Der Hauptgrund für die auffälligen, sehr großen Unterschiede zwischen dem finalen Trainings- und Testfehler ist, dass das Modell für einen Teil der finalen Testdaten ungenaue Vorhersagen macht. Das hängt mit der Wahl der finalen Testdaten zusammen, worauf im späteren Verlauf des Abschnitts aber noch näher eingegangen wird.

Tabelle 5.19: Modellierung des Realteils. Fehlergrößen des finalen Modells.

	MSE	MAPE	SMAPE
Datensatz	-	%	%
fin. Trainingsdaten	$1.53 \cdot 10^{-8}$	0.57	0.57
fin. Testdaten	$1.37 \cdot 10^{-5}$	2.31	2.15

Die Histogramme des relativen Fehlers, auf den finalen Trainings- und Testdaten, sind in Abbildung 5.19 dargestellt. Der überwiegende Teil (90 %) der relativen Fehler tritt bei den finalen Trainingsdaten im Bereich von -1.30 % bis 1.30 % auf, siehe Abbildung 5.19a. Bei den wahren Testdaten umfasst der Bereich -4.38 % bis 4.38 %, siehe Abbildung 5.19b. Die Unter- und Überlaufcontainer des Histogramms der finalen Testdaten in Abbildung 5.19b zeigen, dass auch Fehler mit Beträgen größer als 5 % auftreten. Diese Fehler treten nur in einem bestimmten Bereich der finalen Testdaten auf und zwar bei Gleichstromdichten von $i = 10\,\mathrm{mA\,cm^{-2}}$.

Dies wird, in den Vorhersagen des finalen Modellensembles für die Trainings- und Testdaten, in Abbildung 5.20, ersichtlich. Dargestellt sind Vorhersagen für den Realteil der Impedanz $\mathrm{Re}(Z)$ im Vergleich mit den Messdaten. Abbildungen 5.20a und 5.20b



(a) Finale Trainingsdaten.

(b) Finale Testdaten.

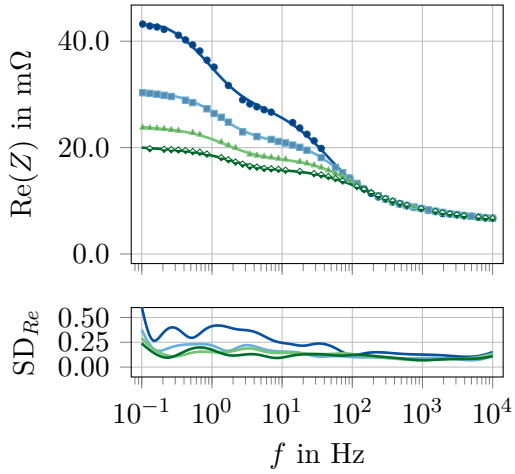
Abbildung 5.19: Modellierung des Realteils. Histogramme des relativen Fehlers für finale Trainings- und Testdaten. Darstellung mit Unter- und Überlaufcontainer.

zeigen darin Vorhersagen für einen Ausschnitt der finalen Trainingsdaten. Abbildungen 5.20c und 5.20d zeigen Vorhersagen für die gesamten finalen Testdaten.

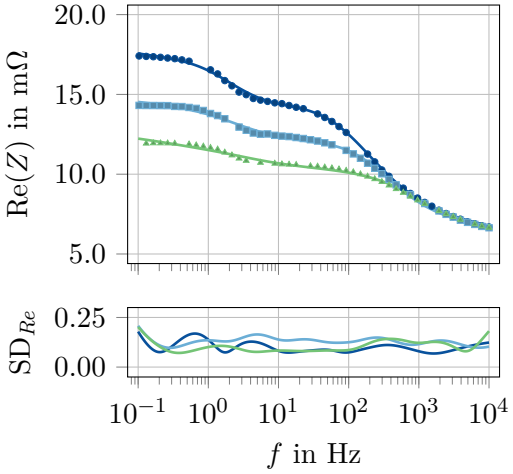
Die Modellvorhersagen weichen, für die finalen Trainingsdaten in Abbildungen 5.20a und 5.20b, für alle Stromdichten kaum nennenswert von den Messdaten ab. Bei den finalen Testdaten zeigt die Modellvorhersage (•) nur bei einer Gleichstromdichte i von $10\,\mathrm{mA\,cm^{-2}}$ große Abweichungen zu den Messdaten (—), für die drei anderen Stromdichten sind die Abweichungen gering. Die hohen Abweichungen, bis zu einem relativen Fehler von 43 %, treten dabei im Frequenzbereich von $1 \cdot 10^{-1}\,\mathrm{Hz}$ bis $1 \cdot 10^{1}\,\mathrm{Hz}$ auf. In diesem Bereich ist auch die Standardabweichung SD_{Re} der Vorhersagen der Modelle im Ensemble sehr hoch, sie erreicht bis zu $10\,\mathrm{mA\,cm^{-2}}$.

Wie bereits angesprochen wurde, hängt das Auftreten dieser großen Abweichungen mit der Wahl der finalen Testdaten zusammen. Dazu soll der Bereich von Stromdichten i von $0\,\mathrm{mA\,cm^{-2}}$ bis $20\,\mathrm{mA\,cm^{-2}}$ betrachtet werden. Die Messdaten enthalten dort bei $0\,\mathrm{mA\,cm^{-2}}$, $10\,\mathrm{mA\,cm^{-2}}$ und $20\,\mathrm{mA\,cm^{-2}}$ jeweils eine EIS-Messung. Da aber die Messdaten bei einer Gleichstromdichte von $10\,\mathrm{mA\,cm^{-2}}$ als finale Testdaten gewählt wurden, umfassen die finalen Trainingsdaten nur Datenpunkte bei $0\,\mathrm{mA\,cm^{-2}}$ und $20\,\mathrm{mA\,cm^{-2}}$. Die finalen Trainings- und Testdaten, mit den zugehörigen Modellvorhersagen, zeigt Abbildung 5.21.

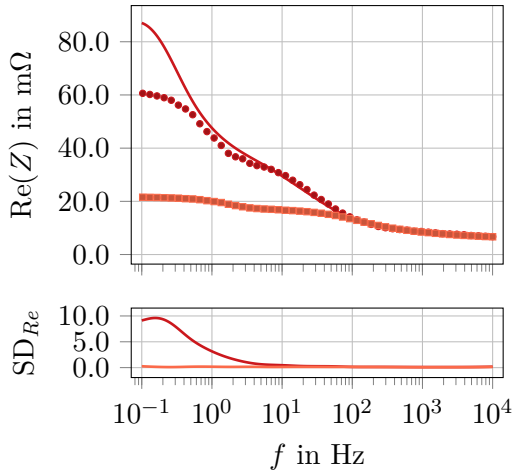
Die Vorhersagen des finalen Modells für die finalen Trainingsdaten (—/—) stimmen noch gut mit den Messdaten (•/▪) überein. Dies war zu erwarten, da das Modell mit diesen Daten trainiert wurde. Um nun eine Vorhersage für die Testdaten (•) zu treffen, interpoliert das Modell, in diesem Fall näherungsweise linear, zwischen den Trainingsdaten (•/▪). Es ist laut den Messdaten aber kein linearer Zusammenhang gegeben: Der Realteil der Impedanz $\mathrm{Re}(Z)$, nimmt in den Messdaten von $0\,\mathrm{mA\,cm^{-2}}$ (•) auf $10\,\mathrm{mA\,cm^{-2}}$ (•) um



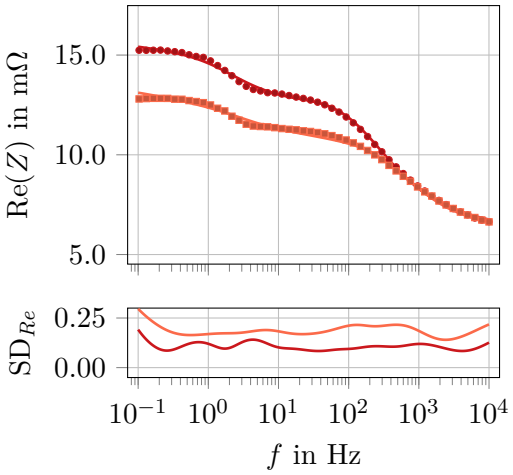
(a) Finale Trainingsdaten. Stromdichte i von • 20 mA cm^{-2}, ▪ 40 mA cm^{-2}, ▲ 60 mA cm^{-2} und ◆ 80 mA cm^{-2}.



(b) Finale Trainingsdaten. Stromdichte i von • 100 mA cm^{-2}, ▪ 150 mA cm^{-2} und ▲ 250 mA cm^{-2}.



(c) Finale Testdaten. Stromdichte i von • 10 mA cm^{-2} und ▪ 70 mA cm^{-2}.



(d) Finale Testdaten. Stromdichte i von • 130 mA cm^{-2} und ▪ 200 mA cm^{-2}.

Abbildung 5.20: Modellierung des Realteils. Vergleich der —— Vorhersage des finalen Modellensembles mit den Messdaten. SD$_{Re}$ ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

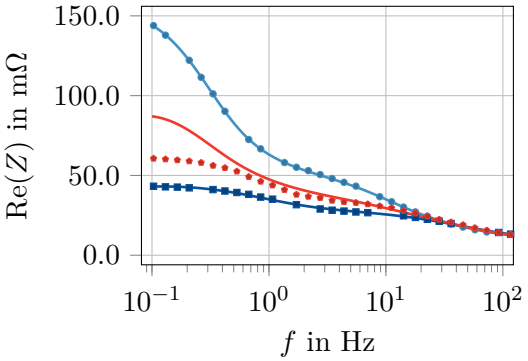


Abbildung 5.21: Modellierung des Realteils. Finale Trainingsdaten bei den Stromdichten i von • $0\,\mathrm{mA\,cm^{-2}}$ und ▪ $20\,\mathrm{mA\,cm^{-2}}$ mit den —/— Modellvorhersagen. Dazwischen sind die finalen Testdaten bei • $i = 10\,\mathrm{mA\,cm^{-2}}$ mit der — Modellvorhersage dargestellt.

ca. 60 % ab (bei $f = 1 \cdot 10^{-1}\,\mathrm{Hz}$), aber von $10\,\mathrm{mA\,cm^{-2}}$ (•) auf $20\,\mathrm{mA\,cm^{-2}}$ (▪) nur um ca. 30 %. Dies führt zu den großen Abweichungen zwischen der Modellvorhersage (—) und den finalen Testdaten (•) bei $i = 10\,\mathrm{mA\,cm^{-2}}$. Ein möglicher Grund dafür, dass dieses Problem bei den finalen Testdaten für Stromdichten i von $70\,\mathrm{mA\,cm^{-2}}$, $130\,\mathrm{mA\,cm^{-2}}$ und $200\,\mathrm{mA\,cm^{-2}}$ nicht auftritt ist, dass sich der Realteil $\mathrm{Re}(Z)$ bei höheren Stromdichten i weniger stark ändert.

Zusammengefasst standen dem Modell, durch die Wahl der finalen Testdaten, wichtige Informationen für das Training nicht zur Verfügung. Bei einer anderen Wahl der finalen Testdaten, könnte der Bereich von $0\,\mathrm{mA\,cm^{-2}}$ bis $20\,\mathrm{mA\,cm^{-2}}$ jedoch nicht auf seine Generalisierungsfähigkeit getestet werden. Dies ist ein Nachteil bzw. ein Trade-off der manuellen Datenwahl, der bei einer zufallsbasierten Datenteilung nicht auftritt.

5.2.5 Modellierung des Imaginärteils

Nach erfolgreicher Modellierung des Realteils, wurde der Imaginärteil der Zellimpedanz $\mathrm{Im}(Z)$ modelliert. Der Imaginärteil $\mathrm{Im}(Z)$ war demnach die Ausgabegröße des Modells. Ansonsten wurden dieselben Messdaten wie bei den Realteilmodellen verwendet, d.h. als Eingabegrößen ergaben sich die Gleichstromdichte i und die Anregefrequenz f. Das Modell ist in Abbildung 5.22 als Blackbox dargestellt und Tabelle 5.20 zeigt einen Ausschnitt der zur Modellerstellung verwendeten Daten. Im Gesamten umfassten die Daten, gleich wie bei den Realteilmodellen, 900 Datenpunkte.

Die finalen Testdaten wurden ebenso gleich gewählt wie bei den Realteilmodellen. Sie umfassten alle Datenpunkte bei einer Gleichstromdichte von $10\,\mathrm{mA\,cm^{-2}}$, $70\,\mathrm{mA\,cm^{-2}}$,

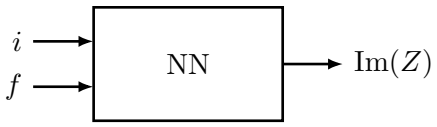


Abbildung 5.22: Blackboxansicht des Imaginärteilmodells mit Eingabe- und Ausgabegrößen.

Tabelle 5.20: Ausschnitt der zur Erstellung der Imaginärteilmodelle verwendeten Daten.

Eingaben		Ausgabe
i	f	$\mathrm{Im}(Z)$
$\mathrm{mA\,cm^{-2}}$	Hz	mΩ
20.0	0.103	−2.35
20.0	0.130	−2.70
20.0	0.164	−3.21
⋮	⋮	⋮

$130\,\mathrm{mA\,cm^{-2}}$ und $200\,\mathrm{mA\,cm^{-2}}$. Alle übrigen Datenpunkte der Messdaten bildeten wieder die HPO-Daten. Auch die Hyperparameter und deren durchsuchte Wertebereiche, angegeben in Tabelle 5.21 wurden, gleich wie bei den Realteilmodellen, beibehalten. Das Optimum, aus 50 durch die Zufallssuche ausgewählte Parameterkombinationen, ist ebenso in Tabelle 5.21 angegeben. Dieses stimmt mit dem Optimum der Realteilmodelle, vgl. Tabelle 5.18, überein. Das ist möglich, da die Seed der Zufallssuche für die Erstellung der Real- und Imaginärteilmodelle gleich gewählt wurde, d.h. die 50 Parameterkombinationen waren dieselben. Die gefundenen Parameter eignen sich demzufolge gut für beide Modellierungsprobleme.

Tabelle 5.21: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das berechnete Optimum. Berechnungszeit ~ 54h.

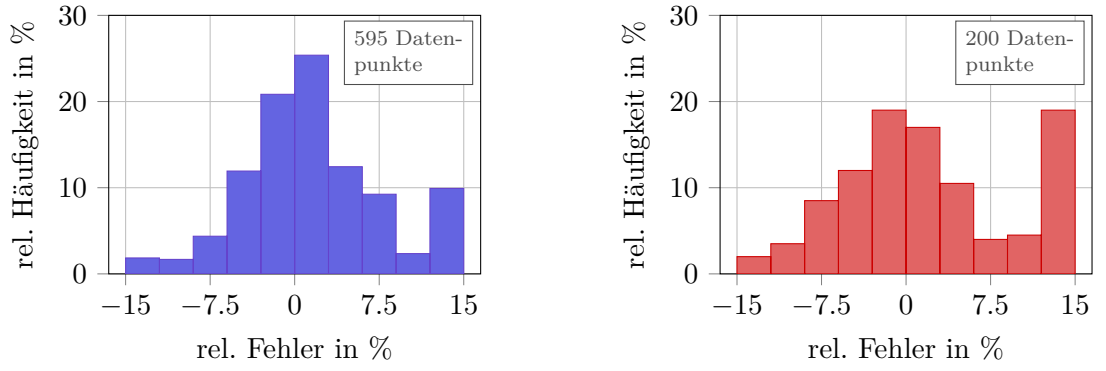
Hyperparameter	Durchsuchter Bereich	Optimum
Aktivierungsfunktion	Sigmoid, Tanh und ReLU	Sigmoid
verdeckte Schichten	1 bis 3	3
Neuronen pro verd. Sch.	3 bis 10	9
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$1 \cdot 10^{-2}$
Geduld	500 bis 1000	662

Das mit dem ermittelten HP-Optimum berechnete finale Modell, ein Ensemble aus 30 trainierten Modellen, erreichte einen MSE von $1.37 \cdot 10^{-8}$/ $4.17 \cdot 10^{-6}$ und einen MAPE von 8.78 %/11.5 % im Training/Test. Zusammengefasst sind diese Ergebnisse in Tabelle 5.22. Auffällig ist der sehr hohe MAPE auf den finalen Trainingsdaten (8.78 %), der vom MAPE auf den finalen Testdaten (16.6 %) sogar noch deutlich übertroffen wird.

Tabelle 5.22: Modellierung des Imaginärteils. Fehlergrößen des finalen Modells.

Datensatz	MSE -	MAPE %	SMAPE %
fin. Trainingsdaten	$1.37 \cdot 10^{-8}$	8.78	7.05
fin. Testdaten	$4.17 \cdot 10^{-6}$	16.6	11.5

Der Grund für das generell hohe Fehlerniveau sind, gehäuft an den Modellgrenzen auftretende, hohe relative Fehler. Ein Grund für den hohen Unterschied zwischen finalem Trainings- und Testfehler ist wieder, wie beim Realteilmodell, die Interpolation des Modells bzw. die Wahl der Testdaten. Diese Einflüsse werden aber etwas später in diesem Abschnitt noch genauer behandelt. Zuvor sollen aber die Histogramme des relativen Fehlers gezeigt werden. Diese sind in Abbildung 5.23 dargestellt. 90 % der relativen Fehler treten bei den finalen Trainingsdaten, siehe Abbildung 5.23a, im Bereich von −15 % bis 15 % auf. Bei den finalen Testdaten, siehe Abbildung 5.23b treten 90 % der relativen Fehler im Bereich von −29.5 % bis 29.5 % auf. Auf den ersten Blick ist das Modell somit völlig unbrauchbar.



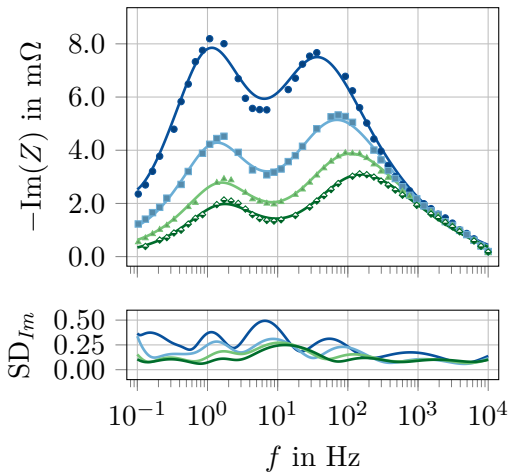
(a) Finale Trainingsdaten.

(b) Finale Testdaten.

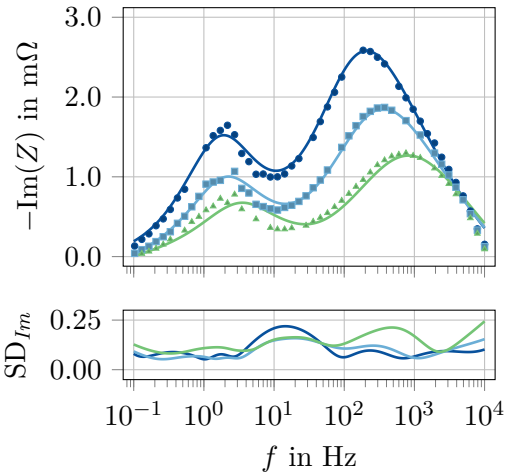
Abbildung 5.23: Modellierung des Imaginärteils. Histogramme des relativen Fehlers für finale Trainings- und Testdaten. Darstellung mit Unter- und Überlaufcontainer.

Dass das Modell, entgegen den bisherigen Ergebnissen, gute Vorhersagen macht, soll nun gezeigt werden. Abbildung 5.24 zeigt Vorhersagen des finalen Modellensembles für die finalen Trainings- und Testdaten. Darin werden die Vorhersagen für den (negativen) Imaginärteil der Zellimpedanz $-\mathrm{Im}(Z)$ mit den Messdaten verglichen. Abbildungen 5.24a und 5.24b zeigen Vorhersagen für einen Ausschnitt der finalen Trainingsdaten. Abbildungen 5.24c und 5.24d zeigen Vorhersagen für die gesamten finalen Testdaten.

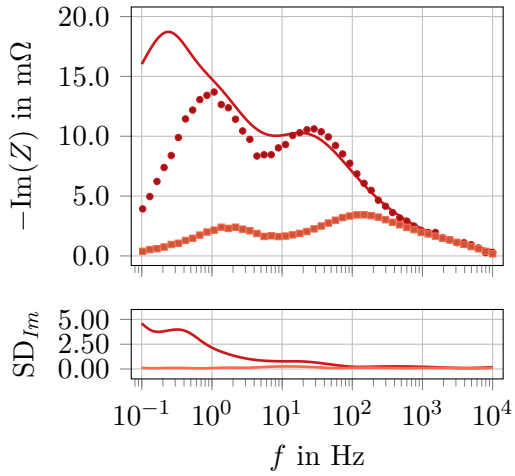
Zuerst sollen die Vorhersagen für die finalen Trainingsdaten in Abbildungen 5.24a und



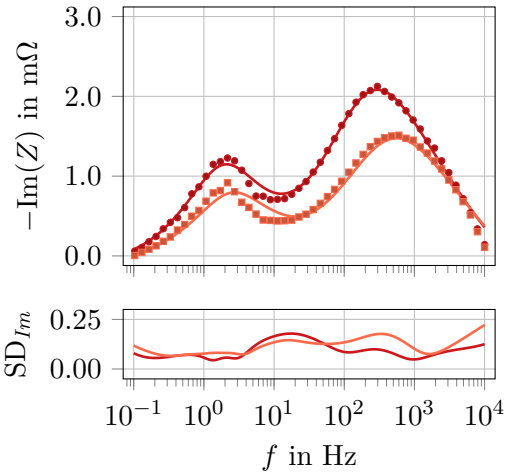
(a) Finale Trainingsdaten. Stromdichte i von • $20\,\mathrm{mA\,cm^{-2}}$, ▪ $40\,\mathrm{mA\,cm^{-2}}$, ▲ $60\,\mathrm{mA\,cm^{-2}}$ und ◆ $80\,\mathrm{mA\,cm^{-2}}$.



(b) Finale Trainingsdaten. Stromdichte i von • $100\,\mathrm{mA\,cm^{-2}}$, ▪ $150\,\mathrm{mA\,cm^{-2}}$ und ▲ $250\,\mathrm{mA\,cm^{-2}}$.



(c) Finale Testdaten. Stromdichte i von • $10\,\mathrm{mA\,cm^{-2}}$ und ▪ $70\,\mathrm{mA\,cm^{-2}}$.



(d) Finale Testdaten. Stromdichte i von • $130\,\mathrm{mA\,cm^{-2}}$ und ▪ $200\,\mathrm{mA\,cm^{-2}}$.

Abbildung 5.24: Modellierung des Imaginärteils. Vergleich der —— Vorhersagen des finalen Modellensembles mit den Messdaten. SD_{Im} ist die Standardabweichung der Vorhersagen der Modelle im Ensemble.

5.24b betrachtet werden. Die Messdaten weisen, für jede dargestellte Gleichstromdichte i, ein lokales Minimum des Imaginärteils $-\mathrm{Im}(Z)$ auf. Dieses befindet sich immer bei einer Anregefrequenz f von circa $1 \cdot 10^1$ Hz. Mit zunehmender Stromdichte i treten im erweiterten Bereich des lokalen Minimums (von 1 Hz bis $1 \cdot 10^2$ Hz), zunehmend relative Abweichungen zwischen Modell und Messdaten auf. Bei einer Stromdichte i von $250\,\mathrm{mA\,cm^{-2}}$ beträgt der größte relative Fehler zwischen Messdaten (▲) und Modellvorhersage (——) beispielsweise 51 %. Im Bereich von 1 Hz bis $1 \cdot 10^2$ Hz weisen auch die Standardabweichungen SD_{Im} der Modellvorhersagen, für alle Stromdichten, Maxima oder lokale Maxima auf (da $-\mathrm{Im}(Z)$ zugleich lokale Minima aufweist, sind die Standardabweichungen, relativ gesehen, noch höher). Die weitaus größten relativen Abweichungen treten jedoch an den Modellgrenzen, in der Nähe der Frequenzen $f = 1 \cdot 10^{-1}$ Hz bzw. $f = 1 \cdot 10^4$ Hz, auf. Gut zu erkennen ist das bei den vergleichsweise hohen Gleichstromdichten i in Abbildung 5.24b: Der Imaginärteil $-\mathrm{Im}(Z)$ geht an den Modellgrenzen gegen null, bzw. Werte nahe null. Kleinste absolute Abweichungen zwischen Modellvorhersage und Messdaten führen dann zu sehr hohen relativen Fehlern. Zum Beispiel beträgt, bei $i = 250\,\mathrm{mA\,cm^{-2}}$ und $f = 1 \cdot 10^4$ Hz, die relative Abweichung zwischen Vorhersage (——) und Messdaten (▲) in etwa 350 %.

Als nächstes werden die Vorhersagen für die finalen Testdaten in Abbildungen 5.24c und 5.24d betrachtet. Grundsätzlich treten dieselben Fehlerquellen auf, die zuvor für die finalen Trainingsdaten ausgeführt wurden. Zusätzlich treten jedoch hohe Abweichungen zwischen der Modellvorhersage (——) und den Messdaten (●) bei einer Wechselstromdichte i von $10\,\mathrm{mA\,cm^{-2}}$ auf. Der maximale relative Fehler, im Bereich von $1 \cdot 10^{-1}$ Hz bis $1 \cdot 10^1$ Hz, beträgt circa 310 %. Dieses Phänomen trat bereits beim Realteilmodell in Abschnitt 5.2.4 auf. Ursachen dafür sind die manuelle Wahl der finalen Testdaten und eine lineare Interpolation des neuronalen Netzes zwischen den finalen Trainingsdaten (bei einem tatsächlich nichtlinearen Zusammenhang).

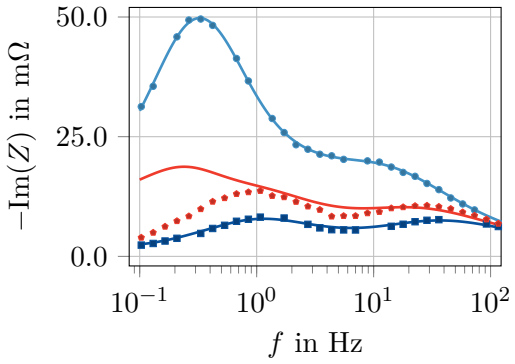


Abbildung 5.25: Modellierung des Imaginärteils. Finale Trainingsdaten bei den Stromdichten i von ● $0\,\mathrm{mA\,cm^{-2}}$ und ■ $20\,\mathrm{mA\,cm^{-2}}$ mit den ——/—— Modellvorhersagen. Dazwischen sind die finalen Testdaten bei ● $i = 10\,\mathrm{mA\,cm^{-2}}$ mit der —— Modellvorhersage dargestellt.

Abbildung 5.25 zeigt die Interpolation des Imaginärteilmodells für die finalen Testdaten bei $i = 10\,\mathrm{mA\,cm^{-2}}$ im Detail. Das Modell interpoliert zwischen den finalen Trainingsdaten bei $i = 0\,\mathrm{mA\,cm^{-2}}$ (•) und $i = 20\,\mathrm{mA\,cm^{-2}}$ (▪). Das Ergebnis, die Vorhersage für die finalen Testdaten bei $i = 10\,\mathrm{mA\,cm^{-2}}$ (——), weicht erheblich von den zugehörigen Messdaten (•) ab.

5.2.6 Modellierung der gesamten Impedanz

Nach erfolgreicher, separater Modellierung des Realteils und Imaginärteils der Impedanz, wurden beide Größen zusammen durch ein einziges Modell abgebildet. Der Imaginärteil $\mathrm{Im}(Z)$ und der Realteil $\mathrm{Re}(Z)$ waren somit die Ausgabegrößen, die Gleichstromdichte i und die Anregefrequenz f die Eingabegrößen des Modells. Das Modell ist in Abbildung 5.26 als Blackbox dargestellt. Die Daten umfassten insgesamt, wie bei den Real- und Imaginärteilmodellen, 900 Datenpunkte.

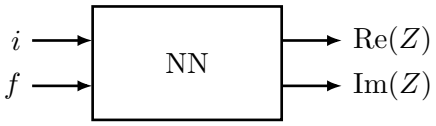


Abbildung 5.26: Blackboxansicht des Impedanzmodells mit Eingabe- und Ausgabegrößen.

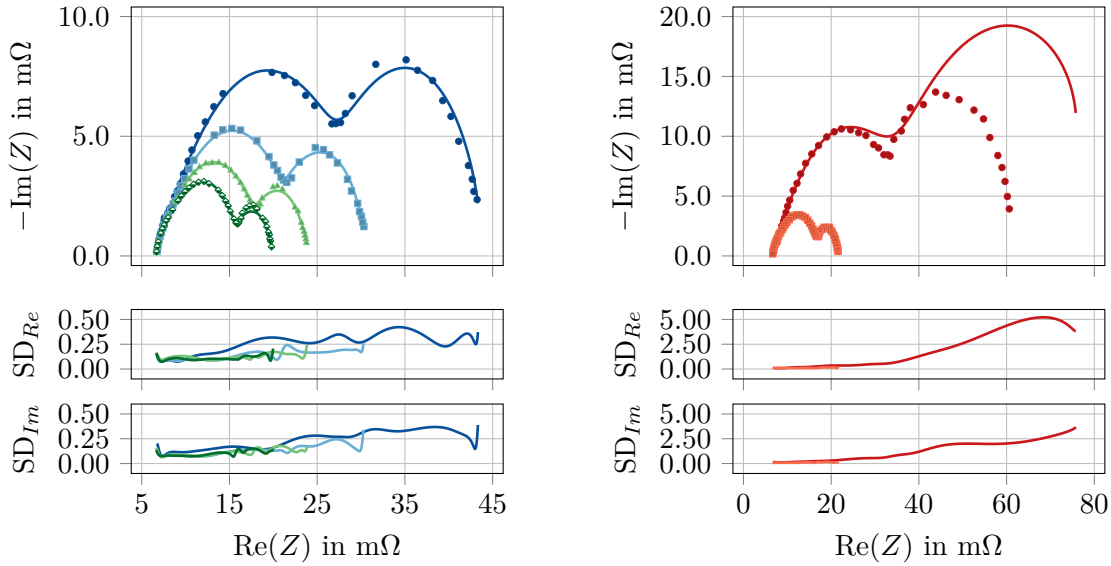
Bis auf eine Ausnahme war die gesamte Vorgehensweise bei den Berechnungen gleich, wie bei den Real- und Imaginärteilmodellen. Einzig der in der HP-Optimierung durchsuchte Bereich der Anzahl der Neuronen pro verdeckter Schicht, wurde nach oben von 10 auf 15 erweitert. Damit wurde der höheren Kompliziertheit des Modells Rechnung getragen. Die Parameterbereiche, die durchsucht wurden, sind mit ihren Optima in Tabelle 5.23 zusammengefasst.

Tabelle 5.23: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das berechnete Optimum. Berechnungszeit ~ 51h.

Hyperparameter	Durchsuchter Bereich	Optimum
Aktivierungsfunktion	Sigmoid, Tanh und ReLU	Sigmoid
verdeckte Schichten	1 bis 3	3
Neuronen pro verd. Sch.	3 bis 15	10
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$5 \cdot 10^{-3}$
Geduld	500 bis 1000	741

Mit dem berechneten, finalen Modell konnte eine vergleichbare Vorhersagequalität für Real- und Imaginärteil der Impedanz erreicht werden, wie mit der separaten Modellierung der Größen. Aus diesem Grund wird hier auf eine genaue Fehlerbetrachtung verzichtet. In

Abbildung 5.27 werden jedoch auszugsweise Vorhersagen des finalen Modellensembles für die Trainings- und Testdaten gezeigt. Die Darstellung erfolgt dabei als Nyquist-Diagramm, d.h. der negative Realteil $-\mathrm{Im}(Z)$ ist über dem Realteil $\mathrm{Re}(Z)$ aufgetragen. Abbildung 5.27a zeigt darin einen Ausschnitt der finalen Trainingsdaten, Abbildung 5.27b einen Ausschnitt der finalen Testdaten.



(a) Finale Trainingsdaten. Stromdichte i von • $20\,\mathrm{mA\,cm^{-2}}$, ▪ $40\,\mathrm{mA\,cm^{-2}}$, ▲ $60\,\mathrm{mA\,cm^{-2}}$ und ◆ $80\,\mathrm{mA\,cm^{-2}}$.

(b) Finale Testdaten. Stromdichte i von • $10\,\mathrm{mA\,cm^{-2}}$ und ▪ $70\,\mathrm{mA\,cm^{-2}}$.

Abbildung 5.27: Modellierung der gesamten Impedanz. Vergleich der —— Vorhersagen des finalen Modellensembles mit den Messdaten. SD_{Re} ist die Standardabweichung der Vorhersagen der Modelle im Ensemble für den Realteil, SD_{Im} für den Imaginärteil.

Die Vorhersagen für finale Trainings- und Testdaten, in Abbildungen 5.27a und 5.27b, stimmen für alle gezeigten Stromdichten i weitgehend mit den Messdaten überein. Einzig bei einer Stromdichte von $i = 10\,\mathrm{mA\,cm^{-2}}$ treten, bei den finalen Testdaten in Abbildung 5.27b, große Abweichungen zwischen Modellvorhersage (——) und Messung (•) auf. Auch bei diesem Modell ist das durch die manuelle Wahl der finalen Testdaten bedingt, was schon bei den separaten Modellen für Real- und Imaginärteil gezeigt wurde, vgl. Abschnitte 5.2.4 und 5.2.5.

5.2.7 Modellierung der gesamten Impedanz mit Temperaturabhängigkeit

Zum Abschluss wurde das Modell für die gesamte Impedanz noch um die Abhängigkeit von der Ofentemperatur erweitert. Die Eingabegrößen des Modells waren somit die Gleichstromdichte i, die Anregefrequenz f und die Ofentemperatur ϑ. Ausgabegrößen waren der Realteil $\mathrm{Re}(Z)$ und der Imaginärteil $\mathrm{Im}(Z)$ der Impedanz. Das Modell ist mit seinen Eingabe- und Ausgabegrößen als Blackbox in Abbildung 5.28 dargestellt. Es wurden somit alle Messdaten, der in Tabelle 5.9 angegebenen Betriebszustände der SOFC, für dieses Modell verwendet. Tabelle 5.24 zeigt einen Ausschnitt der insgesamt 3100 Datenpunkte umfassenden Daten.

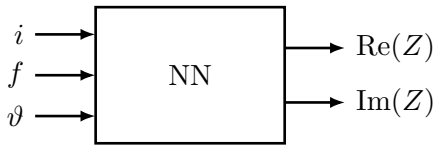


Abbildung 5.28: Blackboxansicht des Impedanzmodells mit Temperaturabhängigkeit, mit Eingabe- und Ausgabegrößen.

Tabelle 5.24: Ausschnitt der, zur Modellierung der gesamten Impedanz mit Temperaturabhängigkeit, verwendeten Daten.

Eingaben			Ausgaben	
i	f	ϑ	$\mathrm{Re}(Z)$	$\mathrm{Im}(Z)$
$\mathrm{mA\,cm^{-2}}$	Hz	°C	mΩ	mΩ
20	0.103	700	43.2	−2.35
20	0.130	700	42.9	−2.70
20	0.164	700	42.7	−3.21
⋮	⋮	⋮	⋮	⋮

Die finalen Testdaten wurden bei diesem Modell, für die drei gemessenen Temperaturen, bei jeweils verschiedenen Stromdichten gewählt: Bei 700 °C bei Stromdichten i von 10/70/130/200 $\mathrm{mA\,cm^{-2}}$, bei 750 °C bei 20/80/140/250 $\mathrm{mA\,cm^{-2}}$ und bei 800 °C bei 30/90/150/300/450 $\mathrm{mA\,cm^{-2}}$. Alle übrigen Messdaten bildeten die HPO-Daten. Die für die HP-Optimierung durchsuchten Parameterbereiche wurden, im Vergleich zur Modellierung ohne Temperaturabhängigkeit, angepasst: Die untere Grenze der Anzahl der verdeckten Schichten wurde von 1 auf 2 erhöht. Der Bereich der Anzahl der Neuronen pro verdeckter Schicht wurde von 3 bis 15 auf 10 bis 30 geändert. Die Parameterbereiche mit ihren zugehörigen, gefundenen Optima sind in Tabelle 5.25 angegeben.

Das finale Modellensemble erreichte einen MSE von $7.12 \cdot 10^{-7}/3.68 \cdot 10^{-7}$ für die Vorhersage des Realteils/Imaginärteils auf den finalen Testdaten, bzw. einen MAPE von

Tabelle 5.25: Hyperparameteroptimierung durch Zufallssuche: Durchsuchter Bereich und das berechnete Optimum. Berechnungszeit ~ 83h.

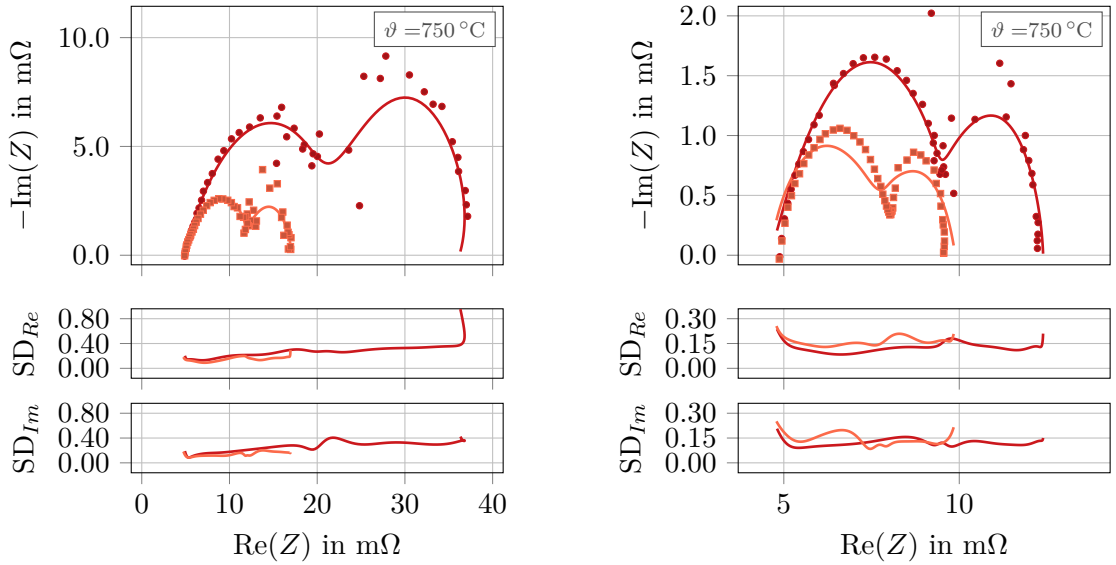
Hyperparameter	Durchsuchter Bereich	Optimum
Aktivierungsfunktion	Sigmoid, Tanh und ReLU	Sigmoid
verdeckte Schichten	2 bis 3	2
Neuronen pro verd. Sch.	10 bis 30	28
Lernrate	$1 \cdot 10^{-4}$ bis $5 \cdot 10^{-2}$	$5 \cdot 10^{-3}$
Geduld	500 bis 1000	703

2.04 %/36.9 %. Tabelle 5.26 bietet einen Überblick über die berechneten Fehlergrößen. Wie bereits bei dem Imaginärteilmodell gezeigt wurde, vgl. Abschnitt 5.2.5, entstehen durch kleinste absolute Abweichungen große relative Fehler an den Modellgrenzen. Der MAPE ist deshalb als Maß für die Vorhersagequalität des Imaginärteils nur bedingt geeignet. Durch Hinzunahme der Temperaturabhängigkeit ergaben sich jedoch weitere Eigenheiten des Modells, die anhand von Modellvorhersagen für die finalen Testdaten gezeigt werden sollen. Auf eine Darstellung der finalen Trainingsdaten wird im weiteren Verlauf verzichtet, die beschriebenen Phänomene traten bei diesen aber in vergleichbarer Weise auf.

Tabelle 5.26: Modellierung der gesamten Impedanz mit Temperaturabhängigkeit. Fehlergrößen des finalen Modells.

	MSE -		MAPE %		SMAPE %	
Datensatz	$\mathrm{Re}(Z)$	$\mathrm{Im}(Z)$	$\mathrm{Re}(Z)$	$\mathrm{Im}(Z)$	$\mathrm{Re}(Z)$	$\mathrm{Im}(Z)$
fin. Testdaten	$7.12 \cdot 10^{-7}$	$3.68 \cdot 10^{-7}$	2.04	36.9	2.00	23.7

Abbildung 5.29 zeigt einen Vergleich der Modellvorhersagen mit den finalen Testdaten, bei einer Temperatur von 750 °C, in einem Nyquist-Diagramm. Die Messdaten weisen bei dieser Temperatur ein starkes Rauschen auf, was in Abbildung 5.29a bei den Stromdichten i von 20 mA cm^{-2} und 80 mA cm^{-2} (•/▪) gut zu sehen ist. Die Modellvorhersagen geben, trotz des starken Rauschens, den typischen Verlauf eines SOFC Impedanzspektrums wieder. Eine Überanpassung des Modells ist nicht zu erkennen. Der berechnete finale Testfehler wird durch die verrauschten Messpunkte erhöht. Dies wäre sogar dann der Fall, wenn das Modell die den Messdaten zugrundeliegende, unbekannte wahre Funktion genau vorhersagen würde. Dies trifft bei Regression jedoch generell zu und ist keine besondere Eigenheit künstlicher neuronaler Netze. Bei einer Stromdichte i von 250 mA cm^{-2}, in Abbildung 5.29b, weichen die Modellvorhersagen (——) auch weit innerhalb der Modellgrenzen von den Messdaten (▪) stark ab. Betroffen ist der Bereich mit einem Realteil der Impedanz $\mathrm{Re}(Z)$ von 6 mΩ bis 9 mΩ. Möglicherweise handelt es sich dabei um eine Unteranpassung des Modells.



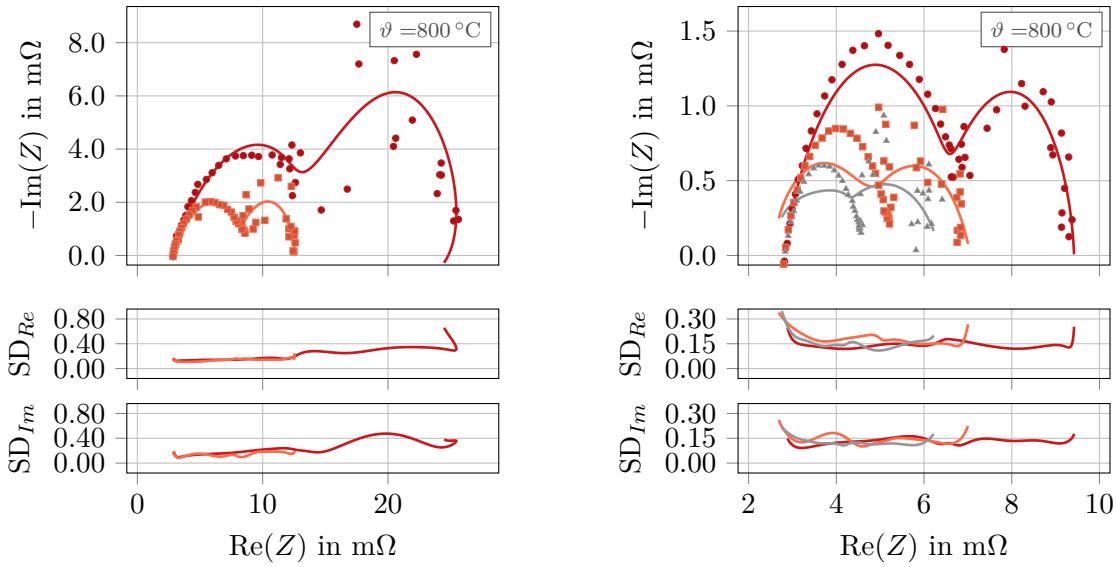
(a) Finale Testdaten. Stromdichte i von • $20\,\mathrm{mA\,cm^{-2}}$ und ▪ $80\,\mathrm{mA\,cm^{-2}}$.

(b) Finale Testdaten. Stromdichte i von • $140\,\mathrm{mA\,cm^{-2}}$ und ▪ $250\,\mathrm{mA\,cm^{-2}}$.

Abbildung 5.29: Modellierung der gesamten Impedanz, mit Temperaturabhängigkeit. Vergleich der —— Vorhersagen des finalen Modellensembles mit den Messdaten bei einer Temperatur von 750 °C. SD_{Re} ist die Standardabweichung der Vorhersagen der Modelle im Ensemble für den Realteil, SD_{Im} für den Imaginärteil.

Abbildung 5.30 zeigt einen Vergleich der Modellvorhersagen mit den finalen Testdaten, bei einer Temperatur von 800 °C. Auch hier tritt möglicherweise Unteranpassung bei hohen Stromdichten auf: Abbildung 5.30b zeigt hohe Abweichungen zwischen den Vorhersagen (——/——) und den Messdaten (▪/▲) bei Stromdichten von $300\,\mathrm{mA\,cm^{-2}}$ und $400\,\mathrm{mA\,cm^{-2}}$. Hier betrifft das den Bereich mit einem Realteil der Impedanz $\mathrm{Re}(Z)$ von $3\,\mathrm{m\Omega}$ bis $6\,\mathrm{m\Omega}$.

Eine mögliche Ursache für die gerade beschriebenen Abweichungen (bzw. die Unteranpassung), in Bereichen mit hohen Stromdichten, soll nun näher ausgeführt werden. Der Trainingsfehler der neuronalen Netze wird durch die mittlere quadratische Abweichung, zwischen der Netzvorhersage und den Trainingsdaten, ermittelt. Die Gewichte und Bias der neuronalen Netze werden im Training so angepasst, dass der Trainingsfehler minimiert wird. Bei hohen Stromdichten ist der Betrag der Impedanz jedoch i.A. um ein Vielfaches kleiner, als der Betrag der Impedanz bei kleinen Stromdichten. Hohe relative Abweichungen im Bereich kleiner Impedanzen führen daher nur zu geringen absoluten Abweichungen (und somit nur zu geringen Fehlerquadraten). Das bedeutet, dass sich der Trainingsfehler – durch Anpassen der Gewichte und Bias des neuronalen Netzes,



(a) Finale Testdaten. Stromdichte i von • 30 mA cm^{-2} und ▪ 90 mA cm^{-2}.

(b) Finale Testdaten. Stromdichte i von • 150 mA cm^{-2}, ▪ 300 mA cm^{-2} und ▲ 450 mA cm^{-2}.

Abbildung 5.30: Modellierung der gesamten Impedanz, mit Temperaturabhängigkeit. Vergleich der —— Vorhersagen des finalen Modellensembles mit den Messdaten bei einer Temperatur von 800 °C. SD_{Re} ist die Standardabweichung der Vorhersagen der Modelle im Ensemble für den Realteil, SD_{Im} für den Imaginärteil.

die die Vorhersagen für hohe Stromdichten beeinflussen – nur wenig ändert. Heuristisch betrachtet werden sich die neuronalen Netze im Training daher zuerst in Bereichen mit niedrigen Stromdichten an die Messdaten anpassen.

6 Zusammenfassung und Schlussfolgerungen

Im Rahmen dieser Masterarbeit wurden mathematische Modelle erstellt, um elektrochemische Eigenschaften von SOFCs zu modellieren. Modelliert wurden Polarisationskurven und die Zellimpedanz. Die Modellierung erfolgte durch vorwärtsverknüpfte, künstliche neuronale Netze. Es wurden geeignete Prozeduren entwickelt und implementiert, um eine weitgehend automatisierte Modellerstellung zu ermöglichen. Basis für die Polarisationsmodelle waren Daten aus einem physikalischen Zellmodell. Basis für die Impedanzmodelle waren Messdaten von einem SOFC-Einzelzellenprüfstand. Im Anschluss wurden die generierten Modelle auf ihre Vorhersagegenauigkeit und Generalisierungsfähigkeit untersucht. Dazu wurden verschiedene Vergleiche der Modellvorhersagen mit den Daten des physikalischen Modells bzw. den Messdaten angestellt.

Die Ergebnisse der Modellierung der Polarisationskurven können wie folgt zusammengefasst werden:

- Die Zellspannung konnte, abhängig von vier Eingabegrößen, erfolgreich modelliert werden. Die Eingabegrößen waren die brennstoffseitige Wasserstoff- und Wasserdampfkonzentration, die Zelltemperatur und die Stromdichte.
- Bei Beschränkung der modellierten Spannung auf den Bereich von der Leerlaufspannung bis 0.6 Volt, konnte eine quantitativ und qualitativ gute Übereinstimmung der Modellvorhersagen mit den Daten des physikalischen Zellmodells erreicht werden.
- Bei Modellierung des gesamten Spannungsbereichs, von der Leerlaufspannung bis 0 Volt, konnte der durch die Konzentrationsverluste verursachte Spannungsabfall bei hohen Stromdichten nur unzureichend modelliert werden. Einige der erstellten Modelle neigten außerdem zur Vorhersage unphysikalischer bzw. unplausibler Werte.

Bei der Modellierung der Zellimpedanz zeigte sich Folgendes:

- Real- und Imaginärteil der Impedanz konnten, abhängig von drei Eingabegrößen, erfolgreich modelliert werden. Die Eingabegrößen waren die Gleichstromdichte des statischen Betriebspunkts, um den die Zelle zur Impedanzmessung angeregt wird, die Anregefrequenz und die Zelltemperatur.
- Trotz bei gewissen Temperaturen in den Messdaten vorhandenem starkem Rauschen, stimmten die Modellvorhersagen qualitativ gut mit den Messdaten überein. Der Realteil konnte auch quantitativ genau modelliert werden, der Imaginärteil nur

mit großen relativen Fehlern an den Modellgrenzen. Diese waren aber durch in den Messdaten auftretende Werte des Imaginärteils bedingt, die nur geringfügig größer als Null waren (beinahe Division durch Null in der Fehlerberechnung), die absoluten Abweichungen blieben gering. Bei hohen Gleichstromdichten traten sowohl für Real- als auch Imaginärteil der Impedanz hohe relative Abweichungen auf, die möglicherweise durch Unteranpassung des Modells verursacht wurden.

Die gewonnenen Erkenntnisse bezüglich der benötigten Rechenzeiten, sowie der Modelle als auch der Modellerstellung, sind:

- Die zur Modellierung der Polarisationskurven und der Zellimpedanz erstellten neuronalen Netze, wiesen maximal 3 verdeckte Schichten und maximal 30 Neuronen pro verdeckter Schicht auf. Die erstellten Modelle waren tendenziell einfach, z.B. verglichen mit Modellen zur Bilderkennung, die mehrere hundert bis tausend Neuronen pro Schicht und mehr als 10 Schichten aufweisen können [23].
- Das Training dieser einfachen Netzwerke (und somit die Modellerstellung) konnte durch Einsatz einer leistungsfähigen Grafikkarte nicht beschleunigt werden. Vielmehr war das Training auf der Grafikkarte sogar etwas langsamer, als auf zwei verschiedenen Vierkern-Prozessoren. Mit Benchmarks konnte gezeigt werden, dass die Grafikkarte durch das Training der Modelle nicht ausgelastet wurde. Ein Grund dafür, warum die Grafikkarte sogar langsamer rechnete als die Prozessoren, könnte die häufig notwendige Datenübertragung zwischen Arbeitsspeicher des Rechners und der Grafikkarte sein.
- Eine direkte Parallelisierung der Hyperparameteroptimierung neuronaler Netze auf der Grafikkarte könnte die Berechnungen beschleunigen. Die verwendete Software enthielt jedoch keine dafür geeigneten Methoden. Versuche dies anderweitig umzusetzen wurden nicht unternommen.
- Aus den genannten Gründen wurden die Berechnungen größtenteils auf einem Vierkern-Prozessor durchgeführt. Die benötigten Rechenzeiten zur Modellerstellung betrugen bei den Polarisationsmodellen in etwa 20 Stunden, bei den Impedanzmodellen 50 bis 80 Stunden. Dies sind Werte, der in der Arbeit beschriebenen und implementierten „Modellerstellung mit zufallsbasierter Datenteilung“ und der „Modellerstellung mit Kreuzvalidierung“. Die aufwendigere „Modellerstellung mit verschachtelter Kreuzvalidierung“ hätte ein Vielfaches an Rechenzeit benötigt und wurde deshalb nicht eingesetzt.
- Berechnungen von Vorhersagen mit den fertigen Modellen waren sehr schnell, d.h. die nötigen Rechenzeiten lagen in der Größenordnung von wenigen Millisekunden pro 1000 Datenpunkte.

Abschließend seien noch Anmerkungen zur Modellerstellung und zur Auswertung der Modelle angeführt:

- Wurden die in der Hyperparameteroptimierung durchsuchten Parameterbereiche ungünstig gewählt, funktionierte die automatische Modellerstellung nicht oder nicht zufriedenstellend. Die richtigen Parameterbereiche zu finden, erforderte aufwendige Untersuchungen, oder gegen Ende der Arbeit auch Erfahrung.
- Einzelne Fehlergrößen waren für die erstellten Modelle nicht aussagekräftig genug, um die Qualität der Modellvorhersagen zu beurteilen. Um herauszufinden, ob wichtige, in den Daten auftretende Phänomene von den Modellen wiedergegeben wurden, waren genauere Untersuchungen der Modellvorhersagen erforderlich. Anders ist das zum Beispiel bei Anwendungen zur Bildererkennung, für die in der Literatur meist nur eine Genauigkeit oder Erfolgsrate des Modells angegeben wird (in Prozent der richtig erkannten Bilder).

Die in dieser Masterarbeit vorgestellten Modelle sind ein erster Schritt zur Modellierung von SOFCs durch künstliche neuronale Netze am Institut für Wärmetechnik. Für die Modelle würden sich vielfältige Einsatzmöglichkeiten anbieten, zum Beispiel zur Regelung, Echtzeitüberwachung oder Diagnose von SOFCs. Ebenso wäre ein Einsatz in Simulationen, die ein mathematisches Zellmodell erfordern, möglich. Dabei ließen sich die Modelle um weitere Ein- und Ausgabegrößen erweitern. Schlussendlich ist der gewählte Modellierungsansatz nicht an physikalische Gesetzmäßigkeiten gebunden, d.h. neben SOFCs ließen sich verschiedenste ingenieurstechnische Systeme durch künstliche neuronale Netze modellieren. Dazu sind meist nur geringfügige Änderungen an den Abläufen zur Modellerstellung nötig, weshalb neuronale Netze sehr flexibel eingesetzt werden können.

Appendix

Literatur

[1] Martin Abadi et al. „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems“. In: *arXiv* (2016). arXiv: 1603.04467.

[2] Ryan P. Adams. *A Tutorial on Bayesian Optimization for Machine Learning Machine Learning*. 2014. URL: https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf (besucht am 13. 01. 2019).

[3] S.B. Adler. „Sources of cell and electrode polarisation losses in SOFCs“. In: *High-Temperature Solid Oxide Fuel Cells for the 21st Century*. Elsevier Ltd., 2016. Kap. 11, S. 357–381. ISBN: 978-0-12-410453-2. DOI: 10.1016/B978-0-12-410453-2.00011-7.

[4] Angelica D. Amara und Jürgen Schmidhuber. *How bio-inspired deep learning keeps winning competitions: An interview with Dr. Juergen Schmidhuber on the future of neural networks*. 2012. URL: http://www.kurzweilai.net/how-bio-inspired-deep-learning-keeps-winning-competitions (besucht am 17. 09. 2018).

[5] Carlos Humberto Andrade-Moraes et al. „Cell number changes in Alzheimer's disease relate to dementia, not to plaques and tangles.“ In: *Brain : A Journal of Neurology* 136. Nr. 12 (2013), S. 3738–3752. DOI: 10.1093/brain/awt273.

[6] Frederico A.C. Azevedo et al. „Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled-Up Primate Brain“. In: *Journal of Comparative Neurology* 513. Nr. 5 (2009), S. 532–541. DOI: 10.1002/cne.21974.

[7] Atilim Gunes Baydin et al. „Automatic Differentiation in Machine Learning: a Survey“. In: *Journal of Machine Learning Research* 18 (2018), S. 1–43. URL: http://jmlr.org/papers/v18/17-468.html.

[8] Yoshua Bengio. „Practical Recommendations for Gradient-Based Training of Deep Architectures“. In: *arXiv* (2012). arXiv: 1206.5533.

[9] James Bergstra und Yoshua Bengio. „Random Search for Hyper-Parameter Optimization“. In: *Journal of Machine Learning Research* 13 (2012), S. 281–305. URL: http://www.jmlr.org/papers/v13/bergstra12a.html.

[10] James Bergstra et al. *Quadratic Polynomials Learn Better Image Features*. Techn. Ber. Université de Montréal, Department of Computer Science und Operations Research, 2009. URL: http://www.iro.umontreal.ca/~lisa/publications2/index.php/attachments/single/205.

[11] Yannick Bessekon et al. „Simulation of a SOFC/Battery powered vehicle“. In: *International Journal of Hydrogen Energy* 44. Nr. 3 (2019), S. 1905–1918. DOI: 10.1016/j.ijhydene.2018.11.126.

[12] BP. *BP Statistical Review of World Energy, 67th Edition*. 2018. URL: https://www.bp.com/content/dam/bp/business-sites/en/global/corporate/pdfs/energy-economics/statistical-review/bp-stats-review-2018-full-report.pdf.

[13] Gavin C. Cawley und Nicola L. C. Talbot. „On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation“. In: *Journal of Machine Learning Research* 11 (2010), 2079-2107. URL: http://www.jmlr.org/papers/v11/cawley10a.html.

[14] Djork-Arné Clevert, Thomas Unterthiner und Sepp Hochreiter. „Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)“. In: *arXiv* (2016). arXiv: 1511.07289.

[15] G. Cybenko. „Approximation by Superpositions of a Sigmoidal Function“. In: *Mathematics of Control, Signals, and Systems* 2. Nr. 4 (1989), S. 303–314. DOI: 10.1007/BF02551274.

[16] Pedro Domingos. „The Role of Occam's Razor in Knowledge Discovery“. In: *Data Mining and Knowledge Discovery* 3 (1999), S. 409–425. DOI: 10.1023/A:1009868929893.

[17] Richard O. Duda, Peter E. Hart und David G. Stork. *Pattern Classification*. Second Edi. Wiley-Interscience, 2000. ISBN: 978-0-471-05669-0.

[18] Harikishan R. Ellamla et al. „Current status of fuel cell based combined heat and power systems for residential sector“. In: *Journal of Power Sources* 293 (2015), S. 312–328. DOI: 10.1016/j.jpowsour.2015.05.050.

[19] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. First Edit. O'Reilly Media, 2017. ISBN: 978-1-491-96229-9.

[20] Xavier Glorot und Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. Bd. 9. 2010. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[21] Xavier Glorot, Antoine Bordes und Yoshua Bengio. „Deep Sparse Rectifier Neural Networks“. In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*. Bd. 15. 2011. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[22] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 978-0262035613. URL: http://www.deeplearningbook.org.

[23] Kaiming He et al. „Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification“. In: *arXiv* (2015). arXiv: 1502.01852.

[24] Geoffry Hinton. *Neural Networks for Machine Learning: Lecture 6*. 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (besucht am 25. 07. 2018).

[25] Sepp Hochreiter. „The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions“. In: *Int. Journal of Uncertainity, Fuzziness and Knowledge-Based Systems* 6. Nr. 2 (1998), S. 107–116. DOI: 10.1142/S0218488598000094.

[26] Qiu-An Huang et al. „A review of AC impedance modeling and validation in SOFC diagnosis“. In: *Electrochimica Acta* 52 (2007), S. 8144–8164. DOI: 10.1016/j.electacta.2007.05.071.

[27] T. Kawada und T. Horita. „Cathodes“. In: *High-Temperature Solid Oxide Fuel Cells for the 21st Century*. Elsevier Ltd., 2015. Kap. 6, S. 161–193. ISBN: 978-0-12-410453-2. DOI: 10.1016/B978-0-12-410453-2.00006-3.

[28] K. Kendall. „Introduction to SOFCs“. In: *High-Temperature Solid Oxide Fuel Cells for the 21st Century*. Elsevier Ltd., 2015. Kap. 1, S. 1–23. ISBN: 978-0-12-410453-2. DOI: 10.1016/B978-0-12-410453-2.00001-4.

[29] *Keras*. 2018. URL: https://keras.io/.

[30] Nitish Shirish Keskar et al. „On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima“. In: *arXiv* (2016). arXiv: 1609.04836.

[31] P. Kingma Kingma und Jimmy Lei Ba. „Adam: a Method for Stochastic Optimization“. In: *arXiv* (2014). arXiv: 1412.6980v8.

[32] Manfred Klell. „Höhere Thermodynamik“. In: *Vorlesung an der Technischen Universität Graz*. 2016.

[33] David Kriesel. *Ein kleiner Überblick über neuronale Netze*. 2007. URL: http://www.dkriesel.com/science/neural_networks.

[34] Rudolf Kruse et al. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 2. Auflage. Springer Vieweg, 2015. ISBN: 978-3-658-10903-5. DOI: 10.1007/978-3-658-10904-2.

[35] Yann LeCun et al. „Efficient BackProp“. In: *Neural Networks: Tricks of the Trade*. Springer, 1998. DOI: 10.1007/3-540-49430-8_2. URL: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf.

[36] Andrew L. Maas, Awni Y. Hannun und Andrew Y. Ng. „Rectifier Nonlinearities Improve Neural Network Acoustic Models“. In: *Proceedings of the 30 th International Conference on Machine Learning*. Bd. 28. 2013. URL: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[37] Dario Marra et al. „A neural network estimator of Solid Oxide Fuel Cell performance for on-field diagnostics and prognostics applications“. In: *Journal of Power Sources* 241 (2013), S. 320–329. DOI: 10.1016/j.jpowsour.2013.04.114.

[38] A. F. Massardo und F. Lubelli. „Internal Reforming Solid Oxide Fuel Cell-Gas Turbine Combined Cycles (IRSOFC-GT): Part A—Cell Model and Cycle Thermodynamic Analysis“. In: *Journal of Engineering for Gas Turbines and Power* 122. Nr. 1 (2000), S. 27–35. DOI: 10.1115/1.483187.

[39] Jarosław Milewski und Konrad Świrski. „Modelling the SOFC behaviours by artificial neural network“. In: *International Journal of Hydrogen Energy* 34. Nr. 13 (2009), S. 5546–5553. DOI: 10.1016/j.ijhydene.2009.04.068.

[40] N. Q. Minh. „Cell and Stack Design, Fabrication and Performance“. In: *High-Temperature Solid Oxide Fuel Cells for the 21st Century*. Elsevier Ltd., 2015. Kap. 8, S. 255–282. ISBN: 978-0-12-410453-2. DOI: 10.1016/B978-0-12-410453-2.00008-7.

[41] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN: 9780071154673.

[42] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com.

[43] Ryan O'Hayre et al. *Fuel Cell Fundamentals*. Third Edit. Wiley, 2016. ISBN: 978-1119113805. DOI: 10.1002/9781119191766.

[44] Bente Pakkenberg et al. „Aging and the human neocortex“. In: *Experimental Gerontology* 38. Nr. 1-2 (2003), S. 95–99. DOI: 10.1016/S0531-5565(02)00151-1.

[45] Allan Pinkus. „Approximation theory of the MLP model in neural networks“. In: *Acta Numerica* 8 (1999), S. 143–195. DOI: 10.1017/S0962492900002919.

[46] Lutz Prechelt. „Automatic early stopping using cross validation: Quantifying the criteria“. In: *Neural Networks* 11. Nr. 4 (Juni 1998), S. 761–767. DOI: 10.1016/S0893-6080(98)00010-0.

[47] Sebastian Raschka. *Python Machine Learning*. Second Edi. Packt Publishing, 2017. ISBN: 978-1-78712-593-3.

[48] F. Rosenblatt. „The perceptron: A probabilistic model for information storage and organization in the brain“. In: *Psychological Review* 65. Nr. 6 (1958), S. 386–408. DOI: 10.1037/h0042519.

[49] Arthur L. Samuel. „Some Studies in Machine Learning Using the Game of Checkers“. In: *IBM Journal* 3. Nr. 3 (1959).

[50] Jürgen Schmidhuber. „Deep Learning in neural networks: An overview“. In: *Neural Networks* 61 (2015), S. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

[51] *Scikit learn documentation*. 2018. URL: https://scikit-learn.org/stable/documentation.html (besucht am 13. 11. 2018).

[52] Salha Faleh Sghaier, Tahar Khir und Ammar Ben Brahim. „Energetic and exergetic parametric study of a SOFC-GT hybrid power plant“. In: *International Journal of Hydrogen Energy* 43. Nr. 6 (2018), S. 3542–3554. DOI: 10.1016/j.ijhydene.2017.08.216.

[53] S. Silbernagl und A. Despopoulos. *Taschenatlas Physiologie*. 8. Auflage. Georg Thieme Verlag KG, 2012. ISBN: 978-3-13-567708-8.

[54] Samuel L. Smith et al. „Don't Decay the Learning Rate, Increase the Batch Size“. In: *arXiv* (2017). arXiv: 1711.00489.

[55] Vanja Subotić, Christoph Schluckner und Christoph Hochenauer. „An experimental and numerical study of performance of large planar ESC-SOFCs and experimental investigation of carbon depositions“. In: *Journal of the Energy Institute* 89. Nr. 1 (2016), S. 121–137. DOI: 10.1016/j.joei.2015.01.004.

[56] Vanja Subotić et al. „In-situ electrochemical characterization methods for industrial-sized planar solid oxide fuel cells Part I: Methodology, qualification and detection of carbon deposition“. In: *Electrochimica Acta* 207 (2016), S. 224–236. DOI: 10.1016/j.electacta.2016.05.025.

[57] *TensorFlow*. 2018. URL: https://www.tensorflow.org/.

[58] Thomas Thaller. „Experimentelle Untersuchung und Modellierung reversibel betriebener Festoxidbrennstoffzellen“. In: *Masterarbeit an der Technischen Universität Graz* (2018).

[59] K. Wang et al. „A Review on solid oxide fuel cell models“. In: *International Journal of Hydrogen Energy* 36. Nr. 12 (2011), S. 7212–7228. DOI: 10.1016/j.ijhydene.2011.03.051.

[60] D. Randall Wilson und Tony R. Martinez. „The general inefficiency of batch training for gradient descent learning“. In: *Neural Networks* 16. Nr. 10 (2003), S. 1429–1451. DOI: 10.1016/S0893-6080(03)00138-2.

[61] W. Winkler. „Thermodynamics“. In: *High-Temperature Solid Oxide Fuel Cells for the 21st Century*. 2015. Kap. 3, S. 51–83. ISBN: 978-0-12-410453-2. DOI: 10.1016/B978-0-12-410453-2.00003-8.

[62] David H. Wolpert und William G. Macready. „No Free Lunch Theorems for Optimization“. In: *IEEE Transactions on Evolutionary Computation* 1. Nr. 1 (1997), S. 67–82. DOI: 10.1109/4235.585893.

[63] Pouya Zahadat und Jaroslaw Milewski. „Modeling electrical behavior of solid oxide electrolyzer cells by using artificial neural network“. In: *International Journal of Hydrogen Energy* 40. Nr. 23 (2015), S. 7246–7251. DOI: 10.1016/j.ijhydene.2015.04.042.

Abkürzungen und Symbole

Abkürzungen: SOFC

ASC	anode supported cell (anodengestützte Zelle)
CSC	cathode supported cell (kathodengestützte Zelle)
EIS	elektrochemische Impedanzspektroskopie
ESC	electrolyte supported cell (elektrolytgestützte Zelle)
LSCF	Lanthan Strontium Cobalt Ferrit
SOFC	solid oxide fuel cell (Festoxidbrennstoffzelle)
SSC	substrate supported cell (substratgestützte Zelle)
YSZ	yttriumstabilisiertes Zirconiumoxid

Abkürzungen: Neuronale Netze

Adam	adaptive moment estimation
ELU	exponential linear unit
HP	Hyperparameter
HPO	Hyperparameteroptimierung
MAE	mean absolute error
MAPE	mean absolute percentage error
MSE	mean squared error
ReLU	rectified linear unit
RMSprop	root mean square propagation
SMAPE	symmetric mean absolute percentage error

Symbole: SOFC

α	-	Transferkoeffizient
$\Delta_B H_m$	$\mathrm{J\,mol^{-1}}$	molare Bildungsenthalpie
$\Delta_B S_m$	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare Bildungsentropie
$\Delta_R G_m$	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare freie Reaktionsenthalpie
$\Delta_R H_m$	$\mathrm{J\,mol^{-1}}$	molare Reaktionsenthalpie

$\Delta_R S_m$	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare Reaktionsentropie
ν_{st}	-	stöchiometrischer Koeffizient
ω	$\mathrm{s^{-1}}$	Kreisfrequenz
ϕ	°	Phasenverschiebung
ϑ	°C	Temperatur
C_{mp}	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare Wärmekapazität
E	V	Zellspannung
E^0	V	Standardzellspannung
E_{rev}	V	reversible Zellspannung
E_N	V	Nernstspannung
E_T	V	temperaturabhängige reversible Zellspannung
F	$\mathrm{A\,s\,mol^{-1}}$	Faraday-Konstante
f	Hz	Frequenz
G_m	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare freie Enthalpie
H_m	$\mathrm{J\,mol^{-1}}$	molare Enthalpie
I	A	Strom
i	$\mathrm{mA\,cm^{-2}}$	Stromdichte
P	W	Leistung
p	Pa	Druck
p	$\mathrm{W\,cm^{-2}}$	Leistungsdichte
R	Ω	ohm'scher Widerstand
R_m	$\mathrm{J\,mol^{-1}\,K}$	allgemeine (molare) Gaskonstante
S_m	$\mathrm{J\,mol^{-1}\,K^{-1}}$	molare Entropie
T	K	Temperatur
V	V	Überspannung
x	%	Volumenanteil
Z	Ω	Impedanz
z_e	-	Elektronenzahl

Symbole: Neuronale Netze

η	-	Lernrate
$\sigma, \boldsymbol{\sigma}$	-	Aktivierungsfunktion
$\theta, \boldsymbol{\theta}$	-	Schwellenwertfunktion
$a, \boldsymbol{a}$	-	Aktivierung, Aktivierungsvektor
$b, \boldsymbol{b}$	-	Bias, Biasvektor

C	-	Kostenfunktion (Fehlerfunktion)
$g, \boldsymbol{g}$	-	Soll-Ausgabe
n_{batch}	-	Batchgröße
n_{train}	-	Anzahl der Trainingsbeispiele
$w, \boldsymbol{W}$	-	Gewicht, Gewichtsmatrix
$x, \boldsymbol{x}$	-	Eingabe eines Netzwerks
$y, \boldsymbol{y}$	-	Ausgabe eines Netzwerks
$z, \boldsymbol{z}$	-	Gewichtete Eingabe

Abbildungsverzeichnis

Tabellenverzeichnis