Jonas Jakob Volgger, BSc

Modellierung der Reaktionskinetik einer reaktiven Strömung in OpenFOAM mittels Künstlichem Neuronalen Netzwerk

Masterarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

eingereicht an der

Technische Universität Graz

Betreuer

Ass.Prof. Dipl.-Ing. Dr.techn. René Josef Prieler

Institut für Wärmetechnik
Leiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Christoph Hochenauer

Graz, März 2024

Kurzfassung

Die vorliegende Arbeit beschreibt das Training und die Implementierung künstlicher neuronaler Netze für die CFD-Simulation der turbulenten Sandia Flamme D mit der Open-Source CFD-Software OpenFOAM. Durch den Einsatz von maschinellem Lernen in der Berechnung reaktiver Strömungen soll eine signifikante Reduktion der Rechenzeit erreicht werden, um Ressourcen bei der Entwicklung wärmetechnischer Anlagen einsparen zu können.

Das Training künstlicher neuronaler Netze beinhaltet auch die Erstellung und Vorbereitung entsprechender Datensätze, die einen maßgeblichen Einfluss auf die Vorhersagekraft der Netze haben. Es werden verschiedene Methoden der Datenauswertung und -aufbereitung diskutiert und verglichen. Schließlich werden die Ergebnisse der CFD-Simulationen mit verschiedenen künstlichen neuronalen Netzen untereinander und mit Referenzsimulationen verglichen, um sie sowohl hinsichtlich ihrer Rechenzeit als auch der Genauigkeit bewerten zu können.

Es wurden mehrere unterschiedliche neuronale Netze mithilfe der Keras-API auf Basis des TensorFlow Backend erstellt, die sich hinsichtlich der Datensätze unterscheiden, auf Basis derer sie trainiert worden sind, sowie auch in Hinblick auf das Preprocessing der Datensätze. Für alle Netze wurde dieselbe Netzstruktur gewählt, was eindrücklich zeigt, welchen Einfluss die richtige Wahl und Vorbereitung der Trainingsdaten auf die Netzperformance hat.

Es zeigte sich, dass es nötig war, den Datensatz auf mehrere künstliche neuronale Netze aufzuteilen, um die Reaktionskinetik in der Sandia Flamme D ausreichend genau wiedergeben zu können. Dieses Ergebnis konnte jedoch nur auf Kosten der Generalisierungsfähigkeit der Netze erreicht werden, indem sie mit Daten trainiert wurden, die bei der standardmäßigen Simulation der Flamme auftraten. Mit einem rein synthetischen Datensatz, der für einer größere Vielfalt an Flammen gültig wäre, konnte das Ergebnis nicht repliziert werden. Mithilfe dieser neuronalen Netze konnte eine Beschleunigung der Simulation von bis zu 76% im Vergleich zum `StandardChemistryModel` von OpenFOAM erreicht werden.

Abstract

This thesis describes the training and implementation of artificial neural networks for the CFD simulation of the turbulent Sandia flame D using the open source CFD-software OpenFOAM. By using machine learning in the calculation of reactive flows, a significant reduction in computing time can be achieved in order to save resources in the development of thermal engineering systems.

The training of artificial neural networks also includes the creation and preparation of corresponding data sets, which have a significant influence on the performance of the networks. Various methods of data evaluation and preparation are discussed and compared. Finally, the results of the CFD simulations with different artificial neural networks are compared with each other and with reference simulations and evaluated in terms of both computing time and accuracy.

Using the Keras API and the TensorFlow backend, multiple neural networks were created on the basis of different datasets and different preprocessing methods for those datasets were compared. All of those neural networks were created using the same net topology, which shows how important the correct choice and preprocessing of the training data has on the predictive performance of the neural network.

It turned out that it was necessary to split the dataset to be treated by multiple artificial neural networks in order to be able to approximate the chemical kinetics in the Sandia flame D to a sufficient degree. This result could only be achieved by compromising on the generalization capabilities of the neural networks by training them on data obtained from the standard CFD-simulation of the flame. Training the network on a purely synthetic dataset, which would be valid for a broader range of flames, did not yield the same level of accuracy. Using these neural networks, an acceleration of the CFD-simulation of up to 76% relative to OpenFOAM's `StandardChemistryModel` was achieved.

Inhaltsverzeichnis

Abkürzungsverzeichnis

CFD computational fluid dynamics

CSTR continuously stirred tank reactor

PSR perfectly stirred reactor

PFR plug flow reactor

DNS direct numerical simulation

EDM Eddy Dissipation Model

EDC Eddy Dissipation Concept

KNN Künstliche neuronale Netze

RNN recurrent neural network

SCM Standard Chemistry Model

MSE mean squared error

RMSE root mean squared error

RMS root mean square

PINN physics informed neural network

FVM Finite-Volumen-Methode

PaSR Partially Stirred Reactor

1 Einleitung und Motivation

Die Simulation reaktiver Strömungen mittels *computational fluid dynamics (CFD)* ist sowohl in der Forschung als auch in der Industrie von höchster Bedeutung. Anwendungsbeispiele beinhalten die Auslegung von Brennern für die Stahlindustrie [28], die Untersuchung und Optimierung der Verbrennungsvorgänge in Verbrennungskraftmaschinen [34] oder die Entwicklung von Gasturbinen auf Basis alternativer Brennstoffe [31]. Die numerische Simulation bietet eine bewährte Alternative und Ergänzung zu Modellversuchen oder Messungen in situ, da einerseits Kosten für mehrere Iterationen an Prototypen gespart und andererseits Einsichten in die Strömungsverhältnisse gewonnen werden können, die aufgrund mangelnder Messmethoden sonst nicht zugänglich wären.

Für diese Simulationen ist von zentraler Bedeutung, die Geschwindigkeiten der auftretenden chemischen Reaktionen in der Gasphasenverbrennung möglichst genau bestimmen zu können. Dafür muss im Allgemeinen in jeder Rechenzelle der Reaktionsmechanismus als System partieller Differentialgleichungen gelöst werden. Die Reaktionsraten der einzelnen Spezies sind dabei meist nicht nur nichtlinear gekoppelt, sondern unterscheiden sich oft um mehrere Größenordnung [24], was zu einem sehr steifen und in der Folge äußerst rechenaufwendigem Anfangswertproblem führt. Dieser erhöhte Rechenaufwand stellt meist den limitierenden Faktor dar, der der raschen Entwicklung neuer Anlagen oder beispielsweise auch der Umstellung bestehender Anlagen auf alternative Brennstoffe entgegenwirkt.

Künstliche neuronale Netze (KNN) werden schon seit den 1960er Jahren in der Modellierung physikalischer Systeme eingesetzt und haben für technische Anwendungen vor allem in der Regelungstechnik weite Verbreitung gefunden [47]. Der Vorteil von KNN gegenüber traditionellen Modellierungsmethoden liegt in der Kombination aus niedrigem Rechenaufwand bei gleichzeitig sehr guter Vorhersagegüte nichtlinearer Zusammenhänge in komplexen Systemen [21],[30].

In dieser Arbeit sollten KNNs verwendet werden, um ebenjenen rechenintensiven Schritt der numerischen Integration der Reaktionskinetik zu ersetzen, um die CFD-Simulation der turbulenten Sandia Flamme D zu beschleunigen. Für die CFD-Simulation

wurde die Open-Source C++ Software OpenFOAM [10] verwendet, die neuronalen Netze wurden mittels der Python Keras-API [5] auf Basis von TensorFlow [27] erstellt. Die Koppelung des Python-Codes mit der C++-Umgebung von OpenFOAM erfolgte mit der pybind11-Bibliothek [17].

Folgende Fragestellungen sollten im Laufe dieser Arbeit geklärt werden:

1. Kann mit KNN eine Beschleunigung der CFD-Simulation erreicht werden?
2. Wie gut ist die Vorhersagekraft, die dabei mittels KNN erreicht werden kann?
3. Ist eine allgemeine Formulierung erreicht worden, die für alle Flammen gültig ist?

Dafür mussten zunächst Referenzsimulationen der Sandia Flamme D mit dem *Eddy Dissipation Concept (EDC)*-Verbrennungsmodell und dem *Standard Chemistry Model (SCM)* von OpenFOAM durchgeführt und die Rechenzeit bestimmt werden. Weiters mussten Datensätze für das Netztraining erstellt und aufbereitet werden. Mittels Hyperparameter-Tuning sollte die passende Topologie für die künstlichen neuronalen Netze bestimmt werden, bevor sie auf Basis der unterschiedlichen Datensätze trainiert wurden. Als nächstes musste der OpenFOAM-Quellcode angepasst werden, um die KNN in die Simulation miteinbinden zu können. Verschiedene Ansätze für die Aufbereitung der Datensätze sollten getestet werden, um die vorliegenden Fragestellungen bestmöglich behandeln zu können.

2 Grundlagen

Das folgende Kapitel soll einen groben Überblick über die theoretischen Grundlagen liefern, die für das Verständnis der Thematik von zentraler Bedeutung sind. Das schließt sowohl den pyhsikalisch/thermodynamischen Inhalt als auch das Thema der Numerik der CFD-Simulationen und der künstlichen neuronalen Netze mit ein.

2.1 Reaktionskinetik

Als Reaktionskinetik bezeichnet man jenes Teilgebiet der Thermodynamik, das sich mit der Modellierung der Geschwindigkeit chemischer Reaktionen innerhalb eines System beschäftigt. Die Informationen dieses Kapitels sind allgemeiner Natur und orientieren sich am Inhalt des Buchs „Grundlagen der Kinetik“ von Hamann et al. [11].

2.1.1 Reaktionsgeschwindigkeit

In einem thermodynamischen System mit chemisch reaktiven Bestandteilen finden chemische Reaktionen mit einer endlichen Geschwindigkeit r statt. Die Reaktionsgeschwindigkeit r einer allgemeinen, reversiblen chemischen Reaktion

$$\left|\nu_{St\ A}\right|\mathrm{A} + \left|\nu_{St\ B}\right|\mathrm{B} + \ldots \underset{\mathrm{k}_r}{\overset{\mathrm{k}_h}{\rightleftharpoons}} \nu_{\mathrm{St\ E}}\mathrm{E} + \nu_{\mathrm{St\ F}}\mathrm{F} + \ldots$$

hängt dabei im Allgemeinen stark von der Temperatur des Systems und den Konzentrationen $[A], \ldots$ der Reaktanden in $^{\mathrm{mol}}/_{\mathrm{m}^3}$ ab:

$$r = -k_h \cdot [A]^a \cdot [B]^b \cdot \ldots + k_r \cdot [E]^e \cdot [F]^f \cdot \ldots \quad {}^{1}/_{\mathrm{s}} \tag{2.1}$$

Durch Multiplikation mit dem stöchiometrischen Koeffizienten einer Referenzspezies $\nu_{St\ A}$ kann damit die zeitliche Änderung der Konzentration einer Spezies $[A]$ beschrieben werden:

$$\frac{d[A]}{dt} = \nu_{St\ A} \cdot r \quad \text{mol/s} \tag{2.2}$$

Die Summe der Exponenten $a, b, ..., e, f, ...$ beschreibt die Ordnung der Reaktion. Die Faktoren k_h und k_r der Hin- und Rückreaktion werden als Geschwindigkeitskonstanten oder Arrhenius-Faktoren bezeichnet und jeweils näherungsweise von der Arrhenius-Gleichung beschrieben:

$$k = A \cdot e^{-\frac{E_a}{RT}} \tag{2.3}$$

A bezeichnet einen Prä-Exponentialfaktor, E_a ist die Aktivierungsenergie der Reaktion, R ist die universelle Gaskonstante und T die Temperatur des thermodynamischen Systems. Die Einheit des Präexponentialfaktor A und damit der Geschwindigkeitskonstante k ist abhängig von der Ordnung der Reaktion. Oft ist auch der Präexponentialfaktor A (geringfügig) abhängig von der Temperatur. In diesem Fall kann auch eine modifizierte Form der Arrhenius-Gleichung angesetzt werden:

$$k = B \cdot T^n \cdot e^{-\frac{E_a}{RT}} \tag{2.4}$$

2.1.2 Reaktionsmodelle

Im Allgemeinen findet nicht nur eine einzelne globale Reaktion in einem reaktiven System statt, sondern eine Vielzahl an Elementarreaktionen mit unterschiedlichen Geschwindigkeiten. Diese Reaktionen werden zu sogenannten Reaktionsmechanismen zusammengefasst. Abhängig von den vorhandenen Spezies und der geforderten Präzision können für ein ingenieurtechnisches Problem verschiedene Reaktionsmechanismen in Frage kommen. Tabelle 2.1 stellt einige Reaktionsmechanismen zur CH_4-Verbrennung dar.

Tabelle 2.1: Einige Reaktionsmechanismen der CH_4-Verbrennung

Name	Spezies	Reaktionen
GRI-Mech 3.0 [39]	53	325
San Diego [4]	57	268
Smooke46 [40]	17	46
Jones-Lindstedt (Frassoldati) [7]	10	6

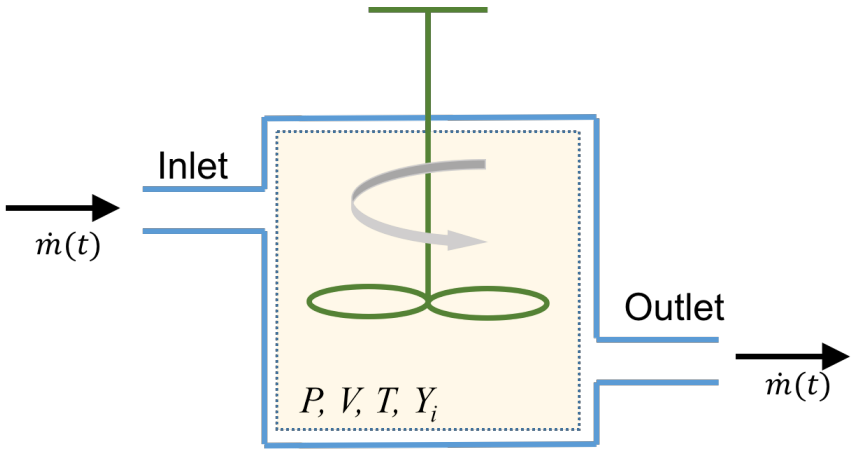


Abbildung 2.1: Schematische Darstellung eines Rührkesselreaktors. Quelle: [43]

2.1.3 Reaktormodelle

Die zwei - für diese Arbeit - bedeutendsten Modelle für die Betrachtung von reaktiven Systemen werden nachfolgend erläutert.

Rührkesselreaktor

Beim idealen kontinuierlichen Rührkessel (engl. *continuously stirred tank reactor (CSTR)* oder *perfectly stirred reactor (PSR)*) wird davon ausgegangen, dass ein homogenes Fluidgemisch mit einer gewissen Zusammensetzung Φ_0 und einem gewissen Massenstrom $\dot{m}$ in den Reaktor eintritt (siehe Abbildung 2.1). Bei Eintritt in den Reaktor werden die Edukte instantan mit dem vorhandenen Fluidgemisch gemischt und chemische Reaktionen können auftreten. Abhängig vom Massenstrom $\dot{m}$ des Fluids und dem Volumen V des Reaktors ergibt sich eine durchschnittliche Verweildauer τ^* eines Fluidpartikels im Rührkessel nach Gleichung 2.5.

$$\tau^* = \frac{V}{\dot{m}} \quad [s] \tag{2.5}$$

Der stationäre Zustand des Rührkessels ergibt sich nun nach Gleichung 2.6 als derjenige, in dem die chemischen Reaktionsraten $\dot{\omega}_i$ der einzelnen Spezies i im Gleichgewicht mit den ein- und austretenden Massenströmen sind [3].

$$\frac{dY_i}{dt} = \dot{\omega}_i + \frac{1}{\tau}(Y_{aus,i} - Y_{ein,i}) \stackrel{!}{=} 0 \quad [1/s] \tag{2.6}$$

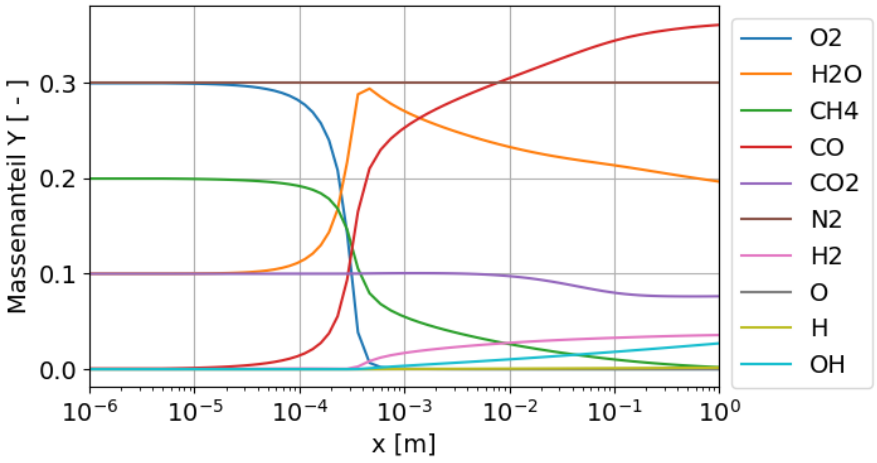


Abbildung 2.2: Verlauf der Gaszusammensetzung in einem Rohrreaktor bei Verbrennung von CH_4 mit dem modifizierten *Jones-Lindstedt* Reaktionsmechanismus entlang der Rohrlänge x. Berechnet mit Cantera [9]

Je nach Anwendungsfall kann der PSR beispielsweise adiabat ($\frac{dh}{dt} = 0$) oder isotherm ($\frac{dT}{dt} = 0$) sowie isobar ($\frac{dp}{dt} = 0$) oder isochor ($\frac{d\rho}{dt} = 0$) betrieben werden.

Rohrreaktor

Beim idealen Rohrreaktor (engl. *plug flow reactor (PFR)*) strömt das Fluidgemisch mit Zusammensetzung Φ_0 und Massenstrom $\dot{m}$ entlang eines Rohres mit konstantem Querschnitt A. Es wird angenommen, dass das Gemisch in radialer Richtung ideal gemischt wird:

$$\frac{dY_i}{dr} = 0 \tag{2.7}$$

Des Weiteren wird Diffusion in axialer Richtung nicht berücksichtigt. Damit kann, ausgehend von einer Eingangszusammensetzung Φ_0, jedem Ort x im Strömungsrohr eine Zusammensetzung $\Phi(x)$ zugeordnet werden. Abbildung 2.2 zeigt beispielhaft den Verlauf der Gaszusammensetzung in einem Rohrreaktor bei Verbrennung von CH_4 mit dem modifizierten *Jones-Lindstedt* Reaktionsmechanismus. Bei Lagrange'scher Betrachtung eines Fluidpartikels, das das Rohr bei $x = 0$ zum Zeitpunkt $t = 0$ betritt, kann dem Partikel unter Kenntnis der Geschwindigkeit v aus Gleichung 2.8 folglich auch eine Zusammensetzung $\Phi(t)$ zugeordnet werden. Die zeitliche Entwicklung der Zusammensetzung wird hierbei durch Gleichung 2.9 beschrieben.

$$v(x) = \frac{\dot{m}}{\rho(x)A} \quad [m/s] \tag{2.8}$$

$$\frac{dY_i}{dt} = \dot{\omega}_i \quad [1/s] \tag{2.9}$$

Man beachte, dass beim PFR im Gegensatz zum PSR der Mischungsterm mit der Umgebung vernachlässigt wird [3].

2.2 CFD-Simulation reaktiver Strömungen

Im folgenden Abschnitt werden die Besonderheiten der CFD-Simulation von chemisch reaktiven, einphasigen Mehrkomponenten-Strömungen beschrieben. Die Informationen in diesem Abschnitt stammen aus dem Buch „Numerische Verbrennungssimulation“ von Peter Gerlinger [8] sowie der Publikation „The Eddy Dissipation Concept - A bridge between science and technology“ von Bjørn F. Magnussen [26].

2.2.1 Spezies-Transportgleichung

Im Gegensatz zur herkömmlichen CFD-Simulation einer Ein-Komponenten-Strömung müssen für die Strömung eines einphasigen Gemischs von N Spezies $N-1$ zusätzliche Spezies-Transportgleichungen für die Massenanteile Y_i gelöst werden:

$$\frac{\partial}{\partial t}(\rho Y_i) + \nabla \cdot (\rho \vec{v} Y_i) = -\nabla \cdot \vec{J}_i + \overline{R}_i \tag{2.10}$$

Der Massenanteil der N-ten Spezies wird aus der Bedingung bestimmt, dass die Summe der Massenanteile gleich 1 sein muss:

$$Y_N = 1 - \sum_{i=1}^{N-1} Y_i \quad [-] \tag{2.11}$$

$\vec{J}_i$ in Gleichung 2.10 bezeichnet die Diffusion aufgrund von Konzentrationsgradienten, die beispielsweise nach dem Fick’schen Gesetz modelliert wird. Darauf wird in dieser Arbeit nicht näher eingegangen.

2.2.2 Turbulenz-Chemie-Interaktion

$\overline{R}_i$ in Gleichung 2.10 ist der Quellterm der i-ten Spezies aufgrund chemischer Reaktionen. Im einfachsten Fall entspricht dies der chemischen Reaktionsrate $\dot{\omega}_i$, so beispielsweise bei laminarer Strömung oder direct numerical simulation (DNS). Bei technisch relevanten Strömungen jedoch muss beinahe ausnahmslos ein Turbulenzmodell angewandt werden, um diejenigen Effekte der Turbulenz auf die Strömung berücksichtigen zu können, die sich in Größenordnungen abspielen, die kleiner sind als die Netzgittergrößen.

In diesen Fällen muss auch der Effekt modelliert werden, den die Turbulenz auf die Reaktionskinetik in der Flamme hat. Bei zu geringer turbulenter Vermischung können die Edukte nicht ausreichend aneinander geführt werden, was die effektive Reaktionsraten der chemischen Reaktionen reduziert oder sogar eine Zündung verhindern kann.

Eine wichtige Kenngröße zur Charakterisierung turbulenter reaktiver Strömungen ist die turbulente Damköhler-Zahl Da_t. Sie beschreibt das Verhältnis der typischen Zeitskala einer turbulenten Strömung τ_0 und der Zeitskala der chemischen Reaktionen τ_R (siehe Gleichung 2.12). Bei einer großen Damköhler-Zahl spricht man von einer mischungslimitierten Verbrennung: Die Edukte reagieren quasi sofort, sobald sie aufgrund der turbulenten Mischung miteinander in Berührung kommen. Ein relevantes Modell für diese Art der Verbrennung ist das *Eddy Dissipation Model (EDM)* von Magnussen und Hjertager (1977) [25].

$$Da_t = \frac{\tau_0}{\tau_R} \quad [-] \tag{2.12}$$

Bei ungenügend großer Damköhler Zahl muss sowohl die Geschwindigkeit der chemischen Reaktionen als auch die turbulente Mischung berücksichtigt werden (*detailed chemistry*). Für diese Art der Verbrennung eignet sich beispielsweise das *EDC* [26], das der Fokus dieser Arbeit ist.

Beim EDC wird davon ausgegangen, dass die chemischen Reaktionen nur innerhalb der kleinsten turbulenten Wirbel in der Strömung stattfinden, den *fine structures*. Die Größe dieser Strukturen und deren Massenaustausch mit der Umgebung ist eng mit der Turbulenz der Strömung verbunden. In radialer Richtung ist deren Ausdehnung in der Größenordnung der *Kalmogorov-Länge* η (siehe Abbildung 2.3), während die Ausdehnung in axialer Richtung viel größer ist.

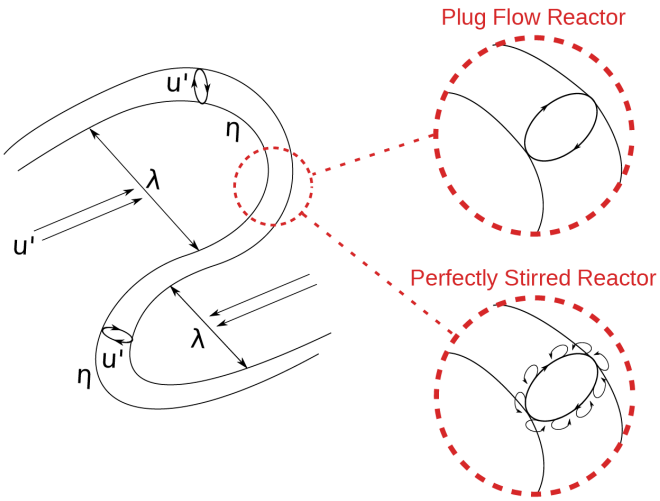


Abbildung 2.3: Schematische Darstellung der *fine structures* und deren Behandlung als PFR oder PSR. Quelle: [3]

Innerhalb dieser *fine structures* werden die Edukte ausreichend schnell vermischt: Sie können als ideale Rührkessel (*PSR*) betrachtet werden. Die durchschnittliche Verweildauer eines Fluidpartikels in den reaktiven Zonen wird unter der Annahme von annähernd isotroper Turbulenz modelliert mit:

$$\tau^* = 0.41 \left(\frac{\nu}{\varepsilon}\right)^{\frac{1}{2}} \quad [s] \tag{2.13}$$

Hierbei bezeichnet ν die turbulente Viskosität und ε die turbulente Dissipation.

Die mittlere Reaktionsrate $\overline{R}_i$ einer Spezies i folgt laut [26] weiters nach Gleichung 2.14.

$$\overline{R}_i = \frac{\overline{\rho}}{\tau^*} \frac{\gamma_L^2 \chi}{1 - \gamma_L^2} (\overline{Y}_i - Y_i^*) \quad [kg/m^3/s] \tag{2.14}$$

$\overline{\rho}$ bezeichnet die mittlere Dichte des Kontrollvolumens und γ_L den Volumenanteil der *fine structures* in dem Kontrollvolumen, der im Weiteren in Abhängigkeit von der Turbulenz modelliert werden muss. χ ist der chemisch reaktive Anteil der *fine structures*, der bei klassischer Verbrennung meist mit 100% angenommen wird.

$\overline{Y}_i$ ist der mittlere Massenanteil der Spezies i im gesamten Kontrollvolumen, während Y_i^* dem Massenanteil derselben Spezies in der reaktiven Zone entspricht. In der Analogie des idealen Reaktors entspricht $\overline{Y}_i$ der Zusammensetzung Φ_0 des eintretenden

Fluidstromes in den *PSR* und Y_i^* der Zusammensetzung Φ_{τ^*} der austretenden Fluide, die den Reaktor mit einer durchschnittlichen Verweildauer τ^* durchströmt haben.

Um diese Zusammensetzung Φ_{τ^*} bestimmen zu können, muss das Differentialgleichungssystem 2.6 ausgehend von $\Phi(t = 0) = \Phi_0$ numerisch integriert werden, bis ein stationärer Zustand erreicht ist. Die chemischen Reaktionsraten $\dot{\omega}_i$ der Spezies sind dabei im Allgemeinen ausgeprägt nichtlinear gekoppelt, was zu einem sehr steifen Anfangswertproblem führt. Dies führt dazu, dass eine sehr kleine Integrations-Schrittweite gewählt werden muss, um auch die kürzesten Zeitskalen auflösen zu können, aber das System trotzdem über einen langen Zeitraum betrachtet werden muss, um den Gleichgewichtszustand zu erreichen. Der Mischungsterm kann auch zu numerischer Instabilität führen, womit beispielsweise periodisch oszillierende Zustandsänderungen auftreten können, die eine konvergente Lösung unmöglich machen.

Als Alternative bietet sich an, die *fine structures* als Rohrreaktor (*PFR*) gemäß Differentialgleichungssystem 2.9 zu behandeln: der Zustand Φ_{τ^*} ergibt sich dann als derjenige Zustand, den ein Fluidpartikel nach einer Zeit von $t = \tau^*$ erreicht. Damit ist im Allgemeinen sowohl die Anzahl der benötigten Integrationsschritte reduziert als auch die numerische Stabilität erhöht. Markus Bösenhofer et al. [3] kommen zu dem Schluss, dass die Behandlung als PFR für die klassische Verbrennung in den meisten Fällen zulässig ist, aber in bestimmten Fällen eine detailliertere Betrachung benötigt wird.

2.3 Künstliche neuronale Netze

Der folgende Abschnitt soll einen groben Überblick über die Grundlagen von künstlichen neurononalen Netzen verschaffen, vor allem über deren Aufbau und Training. Die Informationen hierzu sind hauptsächlich dem Buch "Neuronale Netze kompakt"von Daniel Sonnet [42] entnommen.

2.3.1 Allgemeines und Aufbau

Das KNN ist ein Teilgebiet des Maschinellen Lernens (*machine learning*) und somit der künstlichen Intelligenz und dient der Datenverarbeitung. Anwendungsbeispiele beinhalten die Bild- oder Spracherkennung, die Zeitfolgenabschätzung oder die Regelung technischer Systeme.

Die grundsätzliche Idee solcher *machine learning*-Systeme ist, dass einem Lernalgorithmus ein Datensatz präsentiert wird, aus dem von dem Programm selbstständig

Zusammenhänge erkannt werden sollen, die in weiterer Folge auf ähnliche Datensätze angewandt werden können. So könnten einem maschinellen Lernalgorithmus beispielsweise Bilder von verschiedenen Kleidungsstücken gezeigt werden, mit dem Ziel, dass das System nach Abschluss des Trainings diese Kleidungsstücke auch auf neuen Bildern unterscheiden kann. Diese Art der qualitativen Unterscheidung wird als *Kategorisierung* bezeichnet. Ein neuronales Netz könnte auch darauf trainiert werden, quantitative Zusammenhänge zu erkennen, was als *Regression* bezeichnet wird.

Domingo [6] postulierte, dass künstliche neuronale Netze reine Interpolatoren sind: Sie sind eine näherungsweise Darstellung des Datensatzes, auf dessen Basis sie trainiert wurde. Daraus folgt, dass sie überhaupt nicht - oder bestenfalls sehr beschränkt - für Extrapolation außerhalb des Parameterraums des zugrundeliegenden Datensatzes angewandt werden können.

Meist werden maschinelle Lernalgorithmen in zwei Gruppen unterteilt: überwachtes Lernen (*supervised learning*) und unüberwachtes Lernen (*unsupervised learning*). Beim unüberwachten Lernen kann dem System bei jeder Abschätzung (*prediction*) P_i aufgrund von Eingabedaten (*input*) I_i auch die "richtige" Lösung (*label*) L_i mitgeteilt werden. Damit hat das System eine direkte Rückmeldung, wie korrekt die Abschätzung war, und kann diese Information in das Training übernehmen. Beim unüberwachten Lernen wird dem Algorithmus diese Information nicht übergeben, und es muss selbstständig versuchen, Zusammenhänge zu erkennen.

Künstliche neuronale Netze können sowohl mittels überwachtem als auch mittels unüberwachtem Lernen trainiert werden. Sie sind dabei in Schichten (*layers*) aufgebaut, die wiederum aus Neuronen (*neurons* oder *units*) bestehen. Die Neuronen der unterschiedlichen Schichten sind über sogenannte Kanten miteinander verbunden.

Im einfachsten Fall eines sogenannten *Feedfordward*-Netzes sind alle Neuronen einer Eingabe-Schicht mit allen Neuronen der nächsten Schicht verbunden, die wiederum mit allen Neuronen der darauffolgenden Schicht verbunden sind. Solche Schichten bezeichnet man als *deeply connected*. Die letzte dieser Schichten stellt die Ausgabe des KNN dar, die das gewünschte Ergebnis erzeugen soll. Alle dazwischenliegenden Schichten werden als versteckte Schichten (*hidden layers*) bezeichnet, da bei diesen Schichten kein direkter Informationsaustausch nach außerhalb des Netzes stattfindet. Abbildung 2.4 zeigt schematisch den Aufbau eines *Feedfordward*-Netzes mit Eingabeschicht, versteckten Schichten und Ausgabeschicht.

Grundsätzlich gibt es eine Vielzahl an Möglichkeiten der Netztopologie. So ist es beispielsweise möglich, dass Neuronen einer Schicht auch mit Neuronen einer nicht direkt

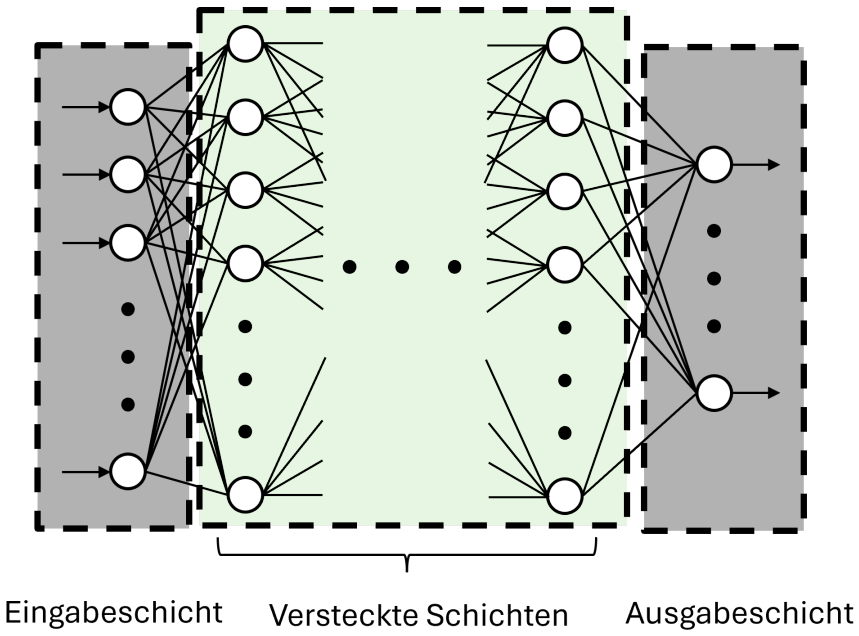


Abbildung 2.4: Schematische Darstellung eines *Feedfordward*-Netzes. Angelehnt an: [42]

nachfolgenden Schicht verbunden sind; in diesem Fall spricht man von einer *Abkürzung* (*shortcut*). Weiters können Neuronen auch Daten an Neuronen weitergeben, die in vorherigen Schichten liegen, hier spricht man von *recurrent neural network (RNN)*.

Bei *Feedforward*-Netzen mit mehr als einer versteckten Schicht spricht man im Allgemeinen von *Deep Neural Networks*, zu *dt.* etwa tiefe neuronale Netze. Die *Tiefe* eines Netzes bezeichnet die Anzahl seiner Schichten, die *Breite* des Netzes bezeichnet die Anzahl der Neuronen in diesen Schichten.

Abbildung 2.5 zeigt den typischen Aufbau eines solchen Neurons in einem Feedforward-Netz. Das Neuron j erhält I Eingabewerte $o_1, o_2, ..., o_I$ von I Neuronen der vorherigen Schicht. Diese Werte werden mit den Kantengewichten $w_{1j}, w_{2j}, ..., w_{Ij}$ (*weights*) gewichtet und anschließend summiert, um die skalare Netzeingabe net_j zu erhalten. Zusätzlich wird ein sogenannter *Bias* b_j zur Netzeingabe addiert:

$$net_j = b_j + \sum_{i=1}^{I} w_{ij} \cdot o_i \tag{2.15}$$

Auf diese Netzeingabe wird nun eine sogenannte Aktivierungsfunktion angewandt. Diese ist nötig, um auch nichtlineare Zusammenhänge darstellen zu können, da mittels gewichteter Summierung (was einer Linearkombination der vorherigen Netzausgaben

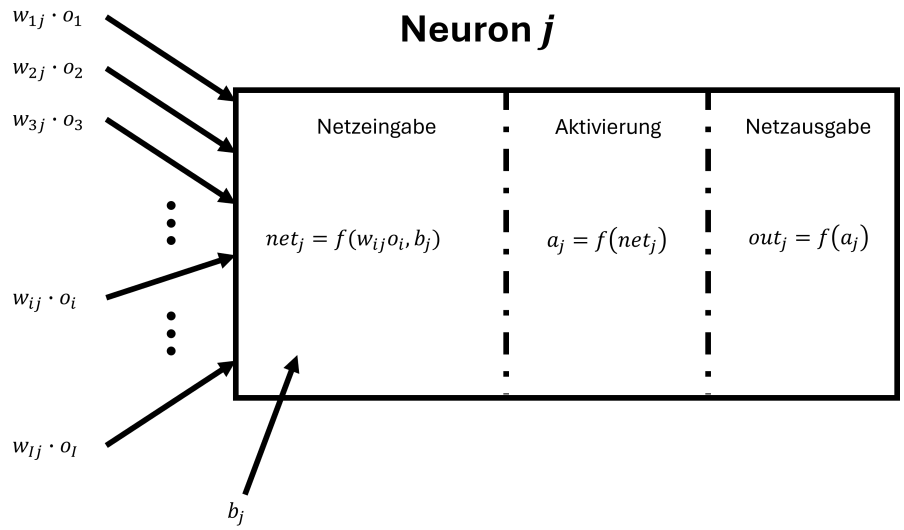


Abbildung 2.5: Typischer Aufbau eines einzelnen Neurons in einem Künstlichen neuronalen Netzwerk. Angelehnt an: [42]

entspricht) nur *linear trennbare* Daten repräsentiert werden können. Es werden eine Vielzahl unterschiedlicher Aktivierungsfunktionen angewandt, die sich hinsichtlich ihrer Eigenschaften wie Stetigkeit und Differenzierbarkeit und ihres Wertebereichs unterscheiden. Abbildung 2.6 zeigt eine repräsentative Auswahl der gängigsten Aktivierungsfunktionen.

Um die Netzausgabe zu erzeugen, könnte auf die Ausgabe der gewählten Aktivierungsfunktion eine weitere Funktion $f(a_j)$ angewandt werden, meist wird jedoch die Aktivierung selbst als Netzausgabe verwendet:

$$out_j = a_j \tag{2.16}$$

2.3.2 Netztraining

Die große Herausforderung beim Einsatz künstlicher neuronaler Netze besteht im Netztraining, also darin, einen Satz von Gewichten w_{ij} zu finden, mittels derer ein gegebener Datensatz „möglichst gut“ approximiert wird. Um die Güte einer gegebenen Abschätzung quantitativ beschreiben zu können, muss zuerst eine sogenannte Fehlerfunktion L (*loss function*) definiert werden. Diese soll auf Basis einer Anzahl von Netzausgaben (einem *Batch*) und der dazugehörigen korrekten Ergebnisse einen Skalar berechnen, der beschreibt, wie nahe die Vorhersagen an den gewünschten Ergebnissen liegen. Je

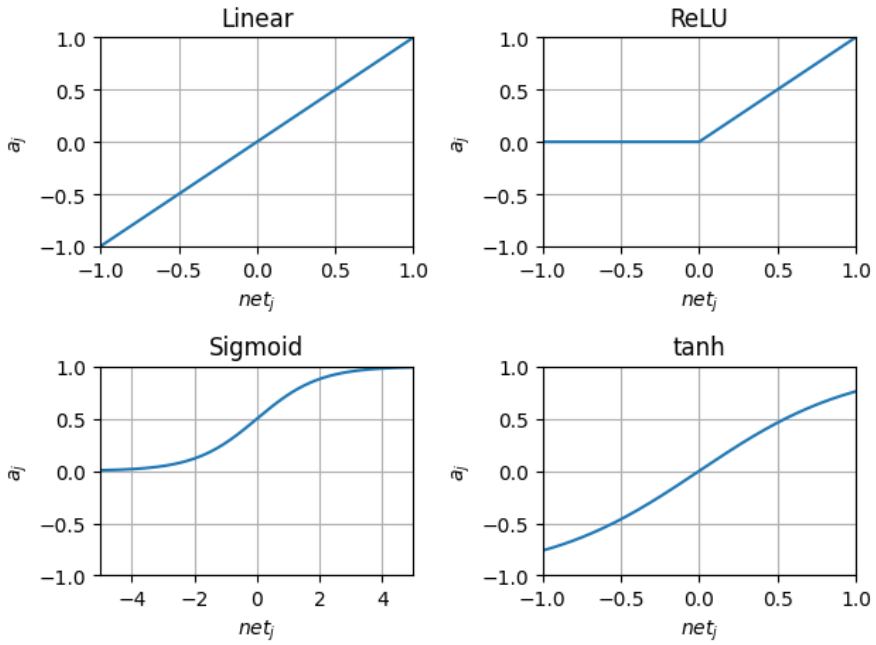


Abbildung 2.6: Auswahl gängiger Aktivierungsfunktionen

Tabelle 2.2: Beispiele für Fehlerfunktionen

Name	Funktion	Anwendung
Mean Absolute Error	$\frac{1}{N}\sum_{i=1}^{N}\lvert y_i^{soll} - y_i^{ist}\rvert$	Regression
Mean Squared Error	$\frac{1}{N}\sum_{i=1}^{N}(y_i^{soll} - y_i^{ist})^2$	Regression
Poisson	$\frac{1}{N}\sum_{i=1}^{N}(y_i^{ist} - y_i^{soll} \cdot y_i^{ist})$	Klassifizierung

nachdem, ob es sich beispielsweise um ein Klassifizierungsproblem oder um Regression handelt, kommen andere Fehlerfunktionen in Frage. Tabelle 2.2 zeigt beispielhaft einige geläufige Fehlerfunktionen für Regressions- und Klassifizierungsprobleme.

Beim Netztraining werden die Gewichte zu Beginn zufällig initialisiert. Mit diesem neuronalen Netz aus zufälligen Gewichten wird eine Vorhersage auf einem oder mehreren Datenpunkten (ein *Batch*) aus dem Trainingsdatensatz angestellt und der Wert der Fehlerfunktion berechnet. Das Ziel des Netzes ist es nun, die skalare Fehlerfunktion zu minimieren. Somit wird das Problem auf eine klassische Optimierungsaufgabe zurückgeführt, die zum Ziel hat, das globale Minimum der Fehlerfunktion $L(w_{ij}, b_j)$ im Raum der möglichen Gewichtungen w_{ij}, b_j zu finden. Ein *Batch* kann dabei aus einem einzelnen Datenpunkt bestehen, aus mehreren Punkten oder aus dem gesamten Daten-

satz. Das Durchschreiten des gesamten Datensatzes wird als eine *Epoche* bezeichnet, wobei ein Training für mehrere Hundert oder Tausend Epochen andauern kann.

Die meisten der verwendeten Optimierungsalgorithmen bauen auf der sogenannten *Error-Backpropagation* auf, um die Fehlerfunktion zu minimieren. Hierbei werden zunächst die partiellen Ableitungen der Fehlerfunktion in Bezug auf alle Gewichtungen berechnet. Anschließend werden die Gewichte in die Richtung des fallenden Gradienten angepasst. Im *machine-learning*-Paket *TensorFlow* werden die partiellen Ableitungen durch die sogenannte automatische Differenzierung (*automatic differentiation* [44]) berechnet. Hierbei werden beim *Forward-Pass* (also dem Auswerten des Ist-Zustands des Netzes) Schritt für Schritt die mathematischen Operationen und deren Ableitungen gespeichert, die durchgeführt werden, wenn das Netz ausgehend von der Eingabeschicht die Werte der Ausgabeschicht und schließlich die Fehlerfunktion ermittelt. Nun können - ausgehend von der Ausgabe-Schicht - die partiellen Ableitungen gemäß der Kettenregel so multipliziert werden, dass alle partiellen Ableitungen der Fehlerfunktion in Bezug auf die Gewichte ermittelt werden können (*Backpropagation*). Abbildung 2.7 zeigt schematisch, wie durch die automatische Differenzierung alle partiellen Ableitungen zwischen den Neuronen ermittelt werden. Um nun beispielsweise das Gewicht b aktualisieren zu können, muss die partielle Ableitung $\frac{\partial d}{\partial b}$ mittels Kettenregel berechnet werden:

$$\frac{\partial d}{\partial b} = \frac{\partial d}{\partial c} \cdot \frac{\partial c}{\partial b} = a \tag{2.17}$$

Die Gewichte werden nun entlang des Gradienten und um eine *Lernrate* gewichtet angepasst. Diese Schritte werden mit neuen Batches so lange wiederholt, bis der Fehler nicht weiter sinkt und das Netztraining somit konvergiert ist.

2.3.3 Überanpassung

Hornik et al. [14] konnten zeigen, dass grundsätzlich jeder beliebig komplexe Datensatz von einem mehrschichtigen Feedforward-Netz approximiert werden kann, wenn es nur genügend Neuronen besitzt, und bezeichneten diese Gruppe von Netzen daher als *universelle Approximatoren*.

Zu tiefe und zu breite Netze bringen jedoch auch Pobleme mit sich: Viele Parameter müssen optimiert werden, daher erfordert der Lernprozess einen hohen Rechenaufwand und Speicherbedarf. Zusätzlich werden hohe Anforderungen an den Lernalgorithmus

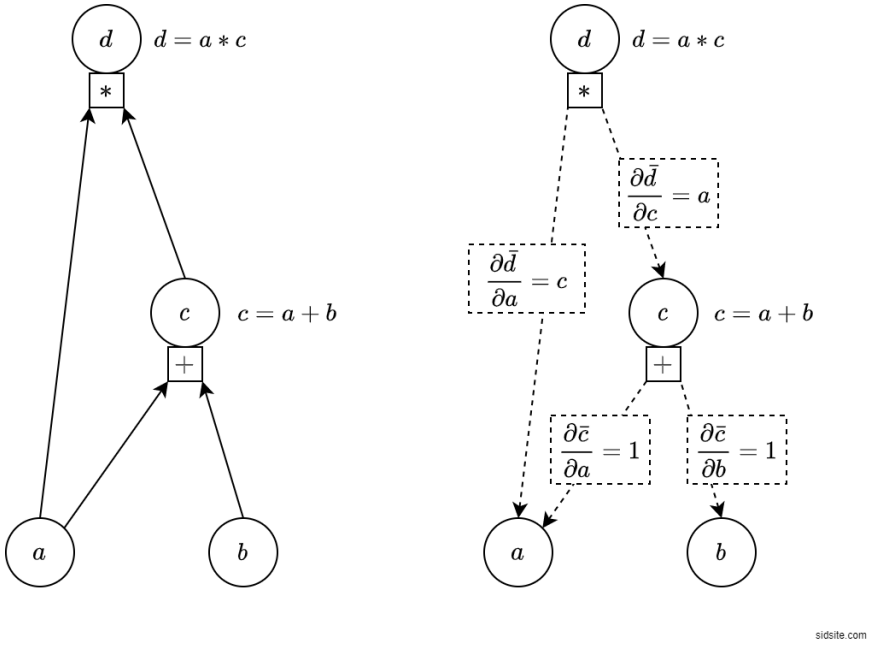


Abbildung 2.7: Schematische Darstellung der automatischen Differenzierung. Quelle: [36]

gestellt, der teilweise mehrere Millionen Parameter variieren muss, um die Kostenfunktion zu minimieren. Weiters muss darauf geachtet werden, dass die Netze die Datensätze nicht zu präzise lernen: Wenn das Netz zu viele Neuronen besitzt, können auch zufällige Schwankungen im Trainingsdatensatz vom Netz erfasst werden und es kann potentiell unerwünscht auf neue Daten reagieren. Dies wird als Überanpassung (engl. *overfitting*) bezeichnet und ist deshalb problematisch, weil das Netz die Fähigkeit verliert, abstrakte Zusammenhänge aus den Daten zu erkennen, die auf weitere Datensätze angewandt werden können. Abbildung 2.8 zeigt schematisch, welche Ergebnisse ein Netz produzieren kann, das zur Überanpassung angeregt wird.

Während des Trainings kann kontrolliert werden, ob das neuronale Netz zur Überanpassung angeregt wurde, indem nach jeder Epoche Vorhersagen auf einem Datensatz angestellt werden, der nicht zum Training verwendet wurde. Wenn die Vorhersagegüte auf diesem Validations-Datensatz stark von derjenigen auf dem Trainingsdatensatz abweicht, kann davon ausgegangen werden, dass Überanpassung auftritt, und das Training kann abgebrochen werden.

Maßnahmen zur Vermeidung von Überanpassung sind beispielsweise, weniger Schichten/Neuronen vorzusehen, oder die Gewichtungen selbst in die Kostenfunktion mitaufzunehmen, sodass vermieden wird, dass sie zu groß - und überflüssige Verbindungen gar eliminiert - werden [29].

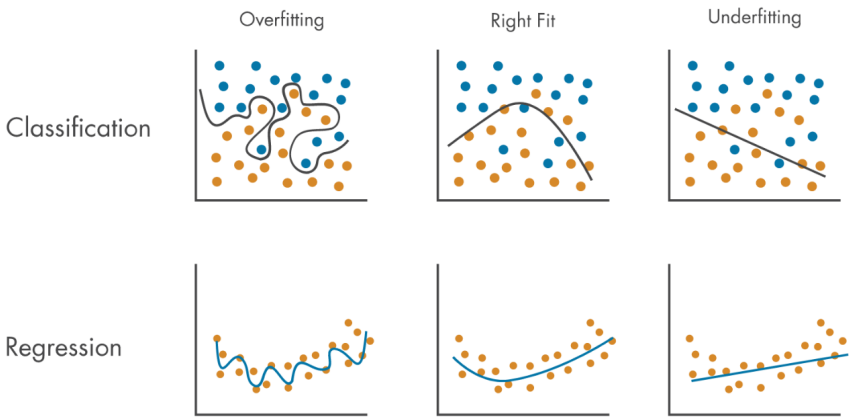


Abbildung 2.8: Überanpassung für Klassifikations- und Regressionsprobleme. Quelle: [45]

2.3.4 Physik-informierte neuronale Netze

Wenn KNN zur Behandlung physikalischer Problemstellungen herangezogen werden, kann es zielführend sein, physikalische Gesetzte direkt in den Lernprozess miteinzubinden. So könnte beispielsweise das Residuum eines Erhaltungssatzes in die Kostenfunktion mitaufgenommen werden (*„soft constraint“*), oder es könnten zusätzliche Schichten eingeführt werden, die die Erhaltung einer Gleichung erzwingen (*„hard constraint“*). Diese Art von neuronalen Netzen werden als Physik-informierte neuronale Netze bezeichnet (*engl.: physics informed neural network (PINN)*) [37].

3 CFD Simulation

Das folgende Kapitel beschreibt die CFD-Simulation der Sandia Flamme D sowie die Software, die dafür verwendet wurde.

3.1 OpenFOAM

OpenFOAM [10] ist eine Open-Source C/C++ Software für die numerische Strömungssimulation mit der *Finite-Volumen-Methode (FVM)*. Das Softwarepaket enthält einen Löser (*solver*) und weitere Applikationen (*utilities*), die auf über 150 Bibliotheken (*libraries*) zugreifen, um strömungstechnische Probleme numerisch zu lösen. Der Löser kann auf verschiedene Lösermodule zugreifen, die jeweils für unterschiedliche Arten von Strömungen verantwortlich sind (inkompressibel/kompressibel, einphasig/mehrphasig, stationär/transient, ...).

Das Lösermodul, das für die Berechnung reaktiver Strömungen verwendet wird, heißt `multicomponentFluid`. Dieses Modul ist für einphasige, reaktive, kompressible oder inkompressible, stationäre oder transiente Strömung geeignet. Es basiert auf dem `PIMPLE`-Algorithmus, einer Kombination aus den weit verbreiteten `SIMPLE` und `PISO` Algorithmen (siehe dazu das Buch „Mathematics, Numerics, Derivations and OpenFOAM“ von Holzmann [13]).

Für die Verbrennungsmodellierung stehen standardmäßig verschiedene Modelle zur Verfügung, wie das EDC und das *Partially Stirred Reactor (PaSR)*-Modell. In dieser Arbeit wurde das EDC in der Formulierung von 2005 verwendet [26], aber die künstlichen neuronalen Netze könnten grundsätzlich auch mit dem PaSR-Modell verwendet werden.

Entscheidend für die benötigte Rechenzeit und die Genauigkeit der Simulation ist die Wahl des Lösers für die äußerst steifen Differentialgleichungen, sowie die gewählten Lösertoleranzen. Imren und Haworth [15] kommen zu dem Schluss, dass der `SEULEX`-Löser einen besseren Kompromiss zwischen Rechenaufwand und Genauigkeiten darstellt als Alternativen wie der `CVODE`-Löser. Dafür müssen in OpenFOAM vom Nutzer Werte für

Tabelle 3.1: Abmessungen der Sandia Flamme D. Quelle: [2]

Außendurchmesser des Hauptstrahls	7.2 mm
Innendurchmesser des Pilotstrahls	7.7 mm
Außendurchmesser des Pilotstrahls	18.2 mm
Windtunnel Abmaße	30 cm x 30 cm

die gewünschte relative (`relTol`) und absolute Toleranz (`absTol`) angegeben werden, wobei der `SEULEX`-Löser auch bei vergleichsweise groben relativen Toleranzen noch brauchbare Ergebnisse liefern kann. Der Werte für die absolute Toleranz sollte in jedem Fall sehr klein gewählt werden.

3.2 Sandia Flamme D

1997 wurden an den *Sandia National Laboratories* [38] verschiedene teilweise vorgemischte turbulente Flammen präzise vermessen. Die Daten für die Flammen mit den Kennungen C, D, E und F wurden im Verlauf des *International Workshop on Measurement and Computation of Turbulent Nonpremixed Flames* [46] veröffentlicht.

Für diese Arbeit wurde die Flamme D als Vergleichssimulation herangezogen, da sie bereits in OpenFOAM als Tutorial-Simulation implementiert war und auch sonst in zahlreichen Publikationen numerisch untersucht wurde. Abbildung 3.1 zeigt den Versuchsaufbau der Flamme D. Ein Gemisch aus CH_4 und Luft im Verhältnis $1:3$ nach Volumen wurde im Zentrum des Brenners mit einem Durchmesser von 7.2 mm zugeführt. Rund um den Auslass dieses Hauptstrahls wurde eine Pilotflamme zur Stabilisierung und Zündung des Hauptstrahls gezündet, der eine Mischung aus C_2H_2, H_2, Luft, CO_2 und N_2 zugeführt wurde. Tabelle 3.1 listet die Abmessungen des Brenners und des darunterliegenden Windtunnels.

Der stabilisierende Pilotstrahl sollte eine Rauchgasrezirkulation simulieren und wurde in der Zusammensetzung so eingestellt, dass er die Verhältnisse nach der Flamme bei $T = 1880\,\mathrm{K}$ repräsentierte. Tabelle 3.2 listet die resultierende Zusammensetzung des Pilotstrahls. Flamme D sollte eine Reynolds-Zahl am Austritt des Hauptstrahls von $Re = 22400$ aufweisen, was einer durchschnittlichen Austrittsgeschwindigkeit von $v = 49.4\,\mathrm{m/s}$ entsprach. Mittels *Jones-Lindstedt*-Reaktionsmechanismus konnte in OpenFOAM für diese Randbedingungen keine Zündung der Flamme vorhergesagt werden, daher wurde in der Simulation die Eintrittsgeschwindigkeit auf $v = 10.0\,\mathrm{m/s}$ reduziert.

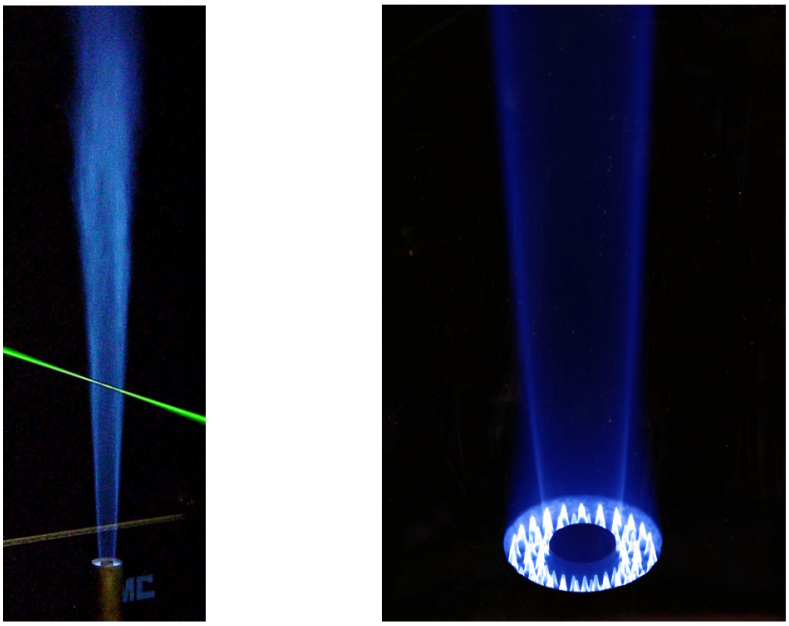
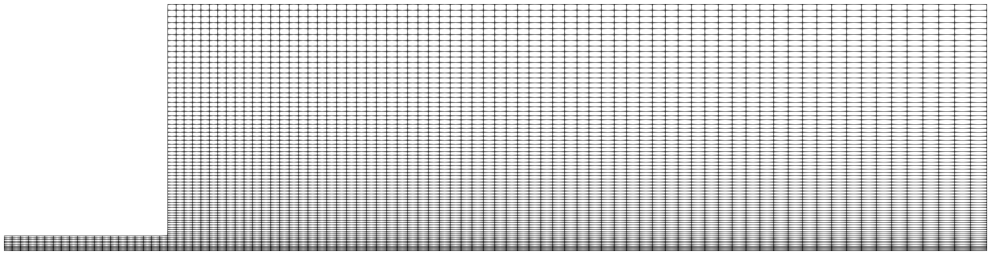


Abbildung 3.1: Die Sandia Flamme D. Quelle: [2]



(a) Das blockstrukturierte Netzgitter

(b) Resultierendes Temperaturfeld

Abbildung 3.2: 2D-Simulation der Sandia Flamme D mit dem Jones-Lindstedt Reaktionsmechanismus, dargestellt in ParaView [1]

Tabelle 3.2: Zusammensetzung des Pilotstrahls der Sandia Flamme D. Quelle: [2]

T	1880 K	Y_H	$2.48 \cdot 10^{-5}$
ρ	0.180 kg/m³	Y_{H_2O}	0.0942
Y_{N_2}	0.7342	Y_{CO}	$4.07 \cdot 10^{-3}$
Y_{O_2}	0.0540	Y_{CO_2}	0.1098
Y_O	$7.47 \cdot 10^{-4}$	Y_{OH}	0.0028
Y_{H_2}	$1.29 \cdot 10^{-4}$	Y_{NO}	$4.8 \cdot 10^{-6}$

Für die Berechnung wurde die Rotationssymmetrie ausgenützt und die Strömung lediglich zweidimensional betrachtet. Die Strömungsdomäne wurde mittels eines blockstrukturierten Netzgitters in 5170 Rechenzellen unterteilt (siehe Abbildung 3.2a). Abbildung 3.2b zeigt das resultierende Temperaturfeld der CFD-Simulation der Flamme.

3.3 Einbindung der Künstlichen neuronalen Netze

Die KNN für diese Arbeit wurden in Python mithilfe der Keras API [5] auf Basis des TensorFlow [27] Backend erstellt. Um ein schnelles Prototyping zu ermöglichen, wurde ein Weg gesucht, um die trainierten KNN direkt in Python ausführen zu können, ohne bei jeder Änderung den OpenFOAM Quellcode anpassen zu müssen.

Für solche Problemstellungen wurde von Wenzel et al. [17] eine C++ Bibliothek namens *pybind11* geschaffen, die es ermöglicht, Daten zwischen C++ Code und Python-Code zu transferieren. Diese Bibliothek wurde in den OpenFOAM Quellcode integriert, um während der Programmlaufzeit (*runtime*) ein Python-Skript ausführen lassen zu können, das mittels KNN die benötigten Reaktionsraten R_i für die Speziestransportgleichung liefert (siehe Kapitel 2.2.1).

Abbildung 3.3 zeigt schematisch den Algorithmus, der in OpenFOAM bei der Berechnung einer reaktiven Strömung für jede Rechenzelle zu jedem Zeitschritt ausgeführt wird, um die Quellterme $\overline{R}_i$ für die Speziestransportgleichung zu erhalten. Standardmäßig werden dabei die folgenden Schritte ausgeführt:

1. Das Verbrennungsmodell (hier `EddyDissipationConcept`) wird vom Löser (`multicomponentFluid`) aufgerufen, um die Quellterme $\overline{R}_i$ vorab zu berechnen
2. Das EDC berechnet einen Vorfaktor $\kappa = \frac{\gamma_L^2}{1-\gamma_L^2}$ und eine durchschnittliche Verweildauer $\tau^* = C_\tau \cdot \sqrt{\frac{\nu}{\varepsilon}}$

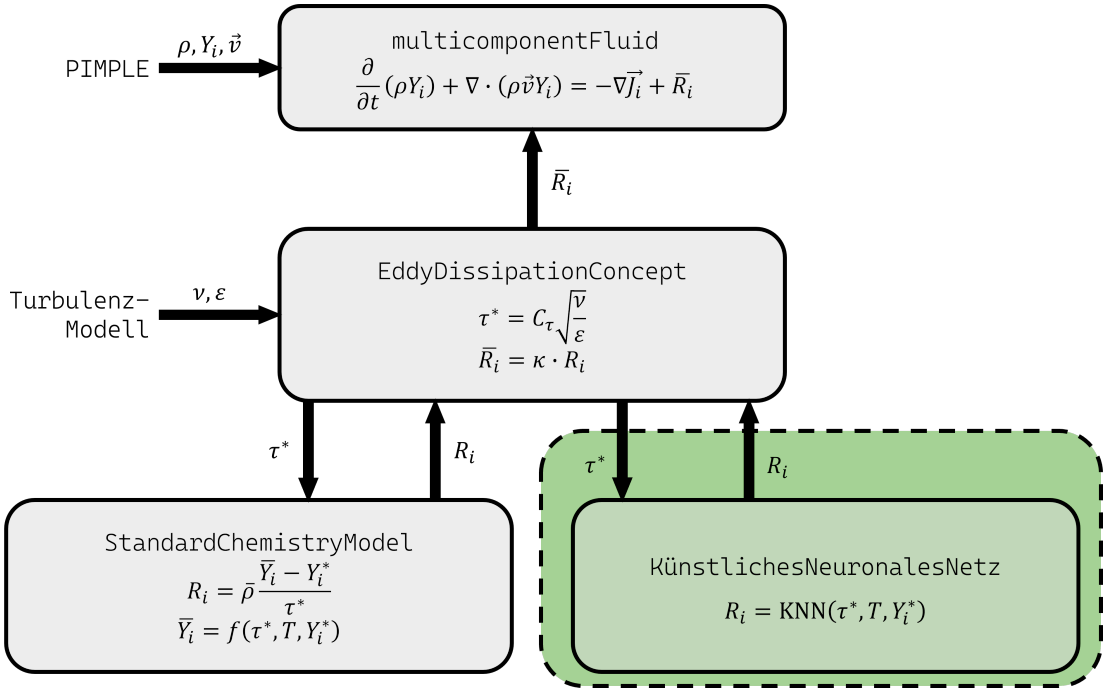


Abbildung 3.3: Schematische Darstellung der Berechnung des Quellterms für die Speziestransportgleichung in OpenFOAM

3. Die Verweildauer τ^* wird an das Standard Chemistry Model (SCM) übergeben, um die Reaktionskinetik zu berechnen
4. Das SCM berechnet die Reaktionsraten $R_i = \frac{\bar{\rho}(\overline{Y}_i - Y_i^*)}{\tau^*}$ mithilfe des gewählten Reaktionsmechanismus
5. Das EDC übergibt die Quellterme $\overline{R}_i = \kappa \cdot R_i$ an den Löser
6. Der Löser löst die Spezies-Transportgleichung 2.10, um die Felder der Massenanteile Y_i zu aktualisieren

Das KNN sollte bei Punkt 4 eingreifen und das SCM in dieser Funktion ersetzen, indem die Reaktionsraten R_i für das gesamte Strömungsfeld direkt abgeschätzt werden (grüne Box in Abbildung 3.3).

4 Trainingsdaten

Der Trainingsdatensatz, auf dessen Basis die Kantengewichte des künstlichen neuronalen Netzes mithilfe eines Lernalgorithmus angepasst werden, ist von zentraler Bedeutung für die spätere Vorhersagekraft des Netzes. Die KNN, die im Rahmen dieser Arbeit erschaffen wurden, sind mit einem *überwachten* Lernalgorithmus trainiert worden. Dafür wird eine große Anzahl an Datenpunkten benötigt, die aus jeweils zusammengehörenden Eingabe- und Ausgabedaten bestehen. Das neuronale Netz erhält bei jedem Trainingsschritt eine Menge von Eingabedaten (einen *batch*) und erzeugt eine Vorhersage, die mit den richtigen Ausgabedaten verglichen wird (siehe Kapitel 2.3.2). Dieses Kapitel beschreibt das Vorgehen bei der Erstellung, Analyse und Aufbereitung der Trainingsdaten, die für diese Arbeit verwendet wurden.

4.1 Jones-Lindstedt Reaktionsmechanismus

Um den Rechenaufwand und den Speicherbedarf einzuschränken, wurde für die Datenerzeugung der modifizierte Jones-Lindstedt Reaktionsmechanismus in der Formulierung von Frassoldati et al. [7] für Methanverbrennung angewandt. Es handelt sich dabei um einen skeletalen Mechanismus, der die Reaktion von 9 Spezies (exklusive des als chemisch inert betrachteten Stickstoffs N_2) mithilfe von lediglich 6 Teilreaktionen beschreibt. Tabelle 4.1 zeigt den Reaktionsmechanismus inklusive aller relevanten Parameter. Die betrachteten Spezies sind: $O_2, CH_4, H_2O, CO_2, H_2, N_2, CO, H, O$ und OH.

4.2 Cantera

Cantera ist eine Open-Source-Software-Bibliothek für Problemstellungen im Bereich der Thermodynamik und speziell der Reaktionskinetik [9]. Der Funktionsumfang von Cantera steht dem Benutzer sowohl über ein Python-Interface als auch über MATLAB, Fortran oder C/C++ zur Verfügung. Die Software bietet dem Nutzer die Möglichkeit, verschiedenste Typen von Reaktoren zu erstellen und miteinander zu

Tabelle 4.1: Jones-Lindstedt Reaktionsmechanismus. Darstellung angelehnt an [7]

Reaktion	Prä Exponential-faktor	Temperatur-exponent	Aktivierungs-energie $[\mathrm{cal/mol}]$
$CH_4 + 0.5\,O_2 \longrightarrow CO + 2\,H_2$	$3.06 \cdot 10^{10}$	0	$3.00 \cdot 10^4$
$CH_4 + H_2O \longrightarrow CO + 3\,H_2$	$3.84 \cdot 10^9$	0	$3.00 \cdot 10^4$
$CO + H_2O \longleftrightarrow CO_2 + H_2$	$2.01 \cdot 10^9$	0	$2.00 \cdot 10^4$
$H_2 + 0.5\,O_2 \longleftrightarrow H_2O$	$8.06 \cdot 10^{16}$	-1	$4.00 \cdot 10^4$
$O_2 \longleftrightarrow 2\,O$	$1.50 \cdot 10^9$	0	$1.13 \cdot 10^5$
$H_2O \longrightarrow H + OH$	$2.30 \cdot 10^{22}$	-3	$1.20 \cdot 10^5$

koppeln. Nach Auswahl des gewünschten Reaktionsmechanismus können die Anfangs- und Randbedingungen für die Reaktoren oder das Reaktornetzwerk festgelegt und der weitere Verlauf der Zustände berechnet werden.

4.3 Gewähltes Reaktormodell

Wie bereits in Kapitel 2.1.3 beschrieben, könnte die Reaktionskinetik sowohl in einem Rohrreaktor als auch in einem Rührkessel-Reaktor betrachtet werden.

Die Rührkessel-Betrachtung wäre die physikalisch korrektere, aber sie hat für die vorliegende Aufgabe einen entscheidenden Nachteil: Zur Variierung der durchschnittlichen Verweildauer τ^* müsste für jeden Wert τ_i^* ein neuer Rührkessel numerisch bis zum chemischen Gleichgewicht integriert werden. Im Vergleich dazu bietet jede Auswertung eines Rohrreaktors zu einem Zeitpunkt τ einen neuen Datenpunkt mit $\tau_i^* = \tau$. Bösenhofer et al. [3] kamen weiters zu dem Schluss, dass die Behandlung der *fine structures* als Rohrreaktoren bei der klassischen Verbrennung in den meisten Fällen zulässig ist. Aus diesen Gründen wurde für diese Arbeit der Rohrreaktor-Ansatz gewählt, wodurch die Rechenzeit drastisch reduziert werden konnte. Der Reaktor wurde dabei als adiabat und isobar betrachtet.

4.4 Parameterraum

Um das künstliche neuronale Netz trainieren zu können, musste ein Datensatz aufgebaut werden, der eine möglichst repräsentative Auswahl an möglichen Zuständen

in den Rechenzellen der CFD-Simulation beinhaltet. Hierfür wurde zuerst untersucht, welche Zustände bei der Simulation der Sandia Flamme-D (siehe Kapitel 3) mittels OpenFOAM auftreten.

Die relevanten Größen für die Berechnung der Reaktionsraten sind dabei jeweils die Temperatur T, die Massenanteile der 10 Spezies $Y_1, Y_2, ..., Y_{10}$, die durchschnittliche Verweildauer τ^*, der Druck p sowie die Dichte ρ. Die Untersuchung sollte vorerst nur auf inkompressible, isobare Strömung beschränkt werden, daher konnte der Druck $p = 1\,\text{atm}$ als konstant betrachtet werden. Die Dichte ρ konnte mithilfe der Idealgasgleichung 4.1 aus dem Druck p, der Temperatur T und der spezifischen Gaskonstante R für das jeweilige Gemisch bestimmt werden.

$$\rho = \frac{p}{RT} \quad [\text{kg/m}^3] \tag{4.1}$$

Mit der Bedingung, dass die Summe der Massenanteile Y aller Spezies gleich 1 sein muss (Gleichung 4.2), konnte das thermodynamische System weiter auf 11 Freiheitsgrade beschränkt werden: $\{\tau^*, T, Y_1, Y_2, ..., Y_9\}$

$$\sum_{i=1}^{10} Y_i \stackrel{!}{=} 1 \tag{4.2}$$

Die Zustände $\{\tau^*, T, Y_1, Y_2, ..., Y_9\}$, die bei der Simulation der Sandia Flamme D bei Berechnung mit SCM auftraten, wurden zu **Datensatz A** zusammengefasst. Tabelle 4.2 zeigt die auftretenden Minima und Maxima der genannten 11 Zustandsgrößen im Datensatz A. Um die Reaktionsraten R_i in dieser Flamme bestmöglich mittels eines KNN abschätzen zu können, sollte der Datensatz, auf dessen Basis das KNN trainiert wird, in etwa dieselbe statistische Verteilung der Zustandsgrößen aufweisen wie die behandelte Simulation. Diese Verteilung hängt aber maßgeblich von der Zusammensetzung und den Strömungsverhältnissen der betrachteten Flamme ab. Soll das KNN für eine möglichst große Bandbreite an verschiedenen reaktiven Strömungen gute Ergebnisse liefern, so muss die Verteilung der Trainingsdaten auch so gewählt werden, dass sie auf verschiedene Flammen zutrifft.

Um eine solche Verteilung realisieren zu können, sollte nun ein Datensatz generiert werden, indem eine große Anzahl von Rohrreaktoren mit unterschiedlichen, zufällig generierten Anfangsbedingungen $\{T, Y_1, Y_2, ..., Y_{10}\}$ simuliert werden. Die Intervalle, aus denen die Werte für die Anfangsbedinungen gezogen wurden, wurden dabei in Anlehnung an die Grenzen aus Datensatz A bestimmt, jedoch mit etwas größeren Bandbreiten, um die Reaktionsraten auch in ähnlichen Flammen abschätzen zu können.

Tabelle 4.2: Bandbreiten der auftretenden Zustandsgrößen in den Trainingsdatensätzen

	Datensatz A		Datensatz B		
	Minimum	Maximum	Minimum	Maximum	
τ^*	$3.6 \cdot 10^{-5}$	0.057	$1.0 \cdot 10^{-5}$	0.10	[s]
T	291.01	$1.9 \cdot 10^3$	250.00	$2.3 \cdot 10^3$	[K]
Y_{O_2}	0.00	0.23	0.00	0.65	$[-]$
Y_{H_2O}	0.00	0.15	0.00	0.60	$[-]$
Y_{CH_4}	0.00	0.16	0.00	0.69	$[-]$
Y_{CO}	0.00	0.11	0.00	0.65	$[-]$
Y_{CO_2}	0.00	0.11	0.00	0.54	$[-]$
Y_{H_2}	0.00	0.0013	0.00	0.58	$[-]$
Y_O	0.00	$7.5 \cdot 10^{-4}$	0.00	0.0086	$[-]$
Y_H	0.00	$2.8 \cdot 10^{-5}$	0.00	0.004	$[-]$
Y_{OH}	0.00	0.0029	0.00	0.018	$[-]$

Tabelle 4.2 zeigt auch die Minima und Maxima der Eingabegrößen in dem resultierenden **Datensatz B**. Die Anfangsbedingungen für den Rohrreaktor wurden dabei mit folgendem Algorithmus nach der Monte-Carlo-Methode bestimmt:

1. Die Temperatur T wird aus dem Intervall $[250, 2300\,]K$ gemäß einer Gleichverteilung zufällig bestimmt.
2. Die vorläufigen Massenanteile $Y_1^*, Y_2^*, ..., Y_{10}^*$ werden gemäß einer Gleichverteilung aus den jeweilig gewählten Intervallen gezogen. Dabei kann es vorkommen, dass die Summe der Massenanteile größer oder kleiner 1.0 ist.
3. Die Massenanteile $Y_1, Y_2, ..., Y_{10}$ werden durch Divison des jeweiligen vorläufigen Massenanteils durch die Summe aller vorläufigen Massenanteile $\sum Y_i^*$ bestimmt:

$$Y_i = \frac{Y_i^*}{\sum Y_i^*} \tag{4.3}$$

   Durch diese Normierung kann sichergestellt werden, dass Gleichung 4.2 stets erfüllt ist.

4.5 Ausgangsgrößen

Um die zu den jeweiligen Eingangsgrößen gehörenden Output-Daten zu erhalten, wurde der Zustand des Rohrreaktors für jeden Anfangszustand an 30 Zeitpunkten τ^*

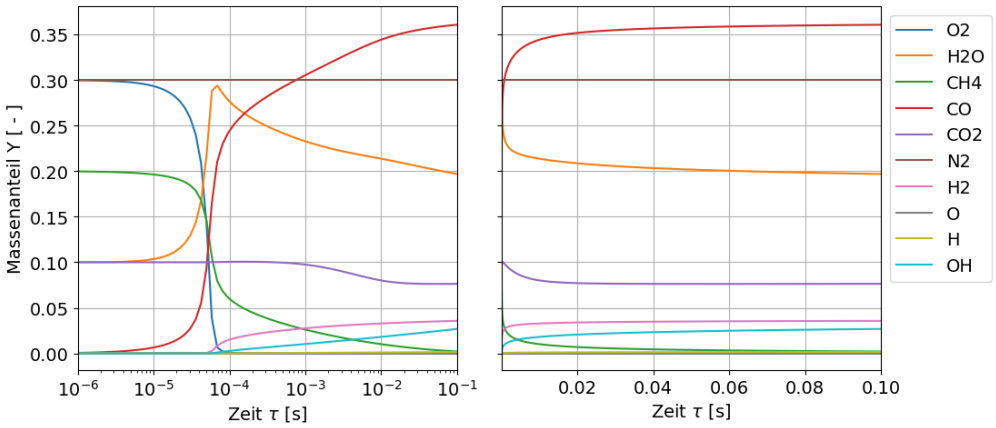


Abbildung 4.1: Vergleich der zeitlichen Skalierung bei Betrachtung des Rohrreaktors

ausgewertet, die logarithmisch verteilt zwischen 10^{-5} s und 10^{-1} s gewählt wurden. Abbildung 4.1 zeigt die Rechtfertigung für die logarithmische Auflösung der Datenpunkte: die größten Gradienten in den Daten traten bei Verweildauern $\tau < 10^{-3}$ s auf, was eine hohe Auflösung in diesem Bereich erforderte. Gleichzeitig musste der Reaktor aber bis zu einer Dauer im Bereich von $\tau = 0.1$ s ausgewertet werden, um den Parameterraum des Datensatzes A (siehe Tabelle 4.2) vollständig abdecken zu können. Dieser Bereich konnte - aufgrund der geringeren auftretenden Gradienten - mit gröberer Auflösung dargestellt werden, um Ressourcen zu schonen.

Damit bestand ein einzelner Datenpunkt für das Netztraining jeweils aus einer Verweildauer τ^*, einem Eintrittszustand $\Phi_0 = \{T, Y_1, Y_2, ..., Y_9\}$ und dem Zustand $\Phi(\tau^*) = \{Y_1, Y_2, ..., Y_9\}$. Die Gesamtheit der Zustände Φ und der dazugehörigen Verweildauern τ^* in Datensatz A umfasste in etwa $2.3 \cdot 10^7$ Datenpunkte mit einem Speicherbedarf von 2.8 GB. Weiters wurden für Datensatz B gemäß oben genanntem Algorithmus $2.6 \cdot 10^6$ Anfangsbedingungen zufällig generiert und für jeweils 30 Zeitschritte ausgewertet, um insgesamt $7.8 \cdot 10^7$ Datenpunkte mit einem Speicherbedarf von ungefähr 5.4 GB zu erhalten.

4.6 Preprocessing des Datensatzes

Die Datenaufbereitung ist ein weiteres wichtiges Werkzeug, um die Konvergenz des Trainings von künstlichen neuronalen Netzen zu verbessern. So sollten die Daten beispielsweise durch Normierung in eine Größenordnung von $O(1)$ gebracht werden, um zu verhindern, dass zu große Gradienten im Netztraining auftreten, was

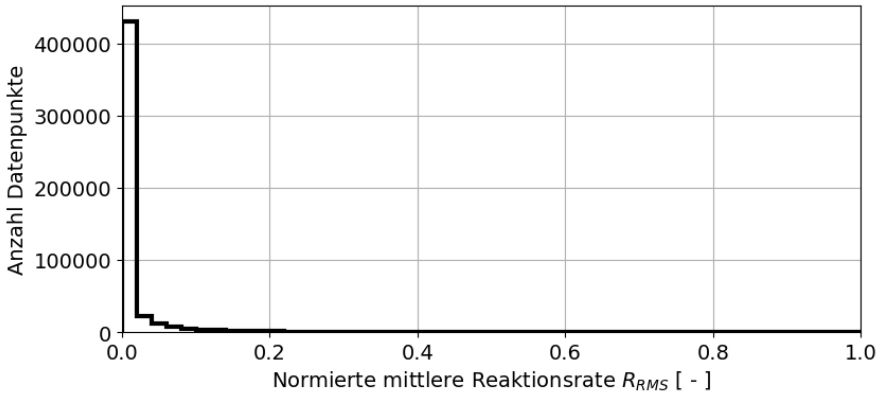
Tabelle 4.3: Normierungsfunktionen

Min-Max-Normierung	$\tilde{y} = \frac{y - y_{min}}{y_{max} - y_{min}}$
z-Standardisierung	$\tilde{y} = \frac{y - \mu}{\sigma}$
Logarithmisch	$\tilde{y} = log(y)$
Wurzelfunktion	$\tilde{y} = y^{1/n}$

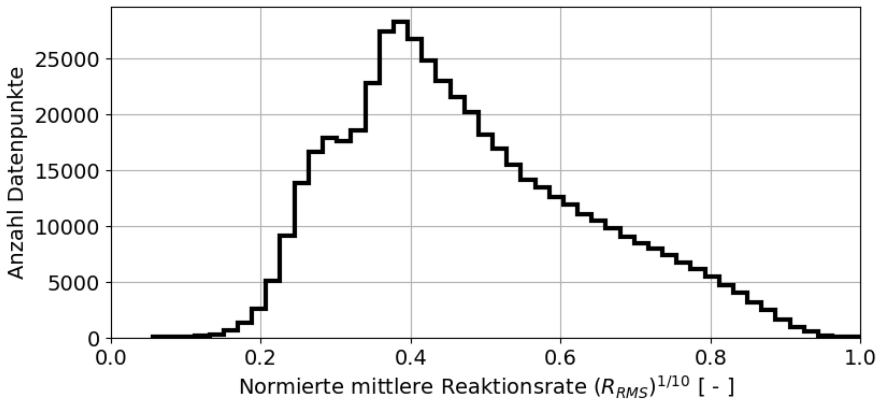
folglich aus Gründen der Stabilität eine geringere Lernrate erfordern und damit den Lernprozess verlangsamen würde [16]. Yann A. LeCun et al. [22] haben festgestellt, dass das Training normalerweise am schnellsten konvergiert, wenn alle Input-Daten standard-normalverteilt sind, also eine Normalverteilung mit Mittelwert von 0 und Standardabweichung 1 aufweisen. Wenn die zugrundeliegenden Daten nicht normalverteilt sind, sollten die Daten dennoch in eine Größenordnung von $O(1)$ gebracht werden. In Tabelle 4.3 sind einige gängige Funktionen für die Datennormierung gelistet.

Die z-Standardisierung ist nur angebracht, wenn die zugrundeliegenden Daten annähernd normalverteilt sind, und transformiert diese Daten dann auf eine Standardnormalverteilung. Die Min-Max-Normierung transformiert die Daten auf das Intervall $[0, 1]$. Es kann auch hilfreich sein, einen Logarithmus oder eine Wurzelfunktion auf die Daten anzuwenden, um die Verteilung näher an eine Normalverteilung heranzuführen, wie beispielsweise in Abbildung 4.2 gezeigt. Abbildung 4.2a zeigt die mittleren Reaktionsraten R_{RMS}, nachdem sie mittels Min-Max-Normierung auf den Wertebereich $[0, 1]$ abgebildet wurden. Abbildung 4.2a zeigt dieselben Daten, aber transformiert mittels $f(R) = (R)^{1/10}$, was die Verteilung näher an eine Normalverteilung heranführt.

In vielen Fällen (siehe beispielsweise Prieler et al. [33] oder He et al. [12]) kann es auch nützlich sein, den Datensatz auf mehrere KNN aufzuteilen, um so eine bessere Performance zu erreichen. Radaideh et al. [35] konnten zeigen, dass die Verweildauer τ^* ein sehr guter Prädiktor für die Reaktionsraten R_i des Systems ist. Daher war sie für diese Arbeit gut geeignet, um als Aufteilungskriterium für die Datensätze zu dienen. Abbildung 4.3 zeigt, wie stark die mittleren Reaktionsraten R_{RMS} der Datenpunkte vor allem im Datensatz B von der Verweildauer abhängen.

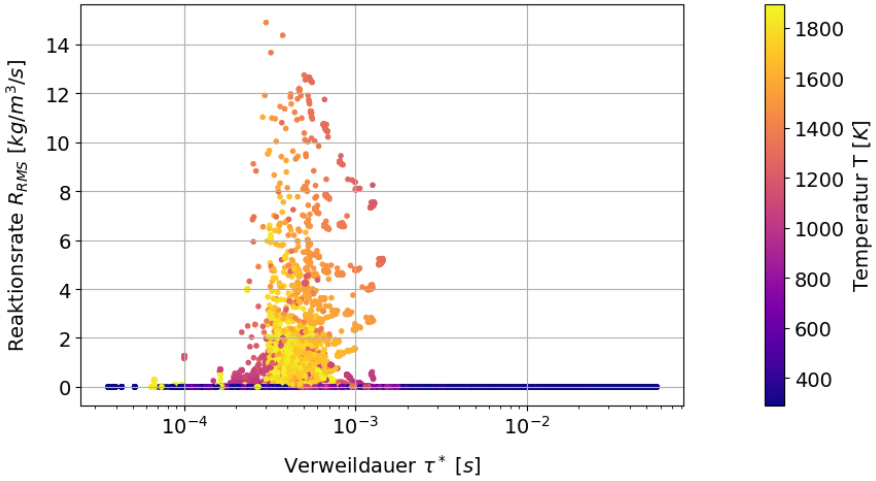


(a) Verteilung der normierten mittleren Reaktionsraten R_{RMS}

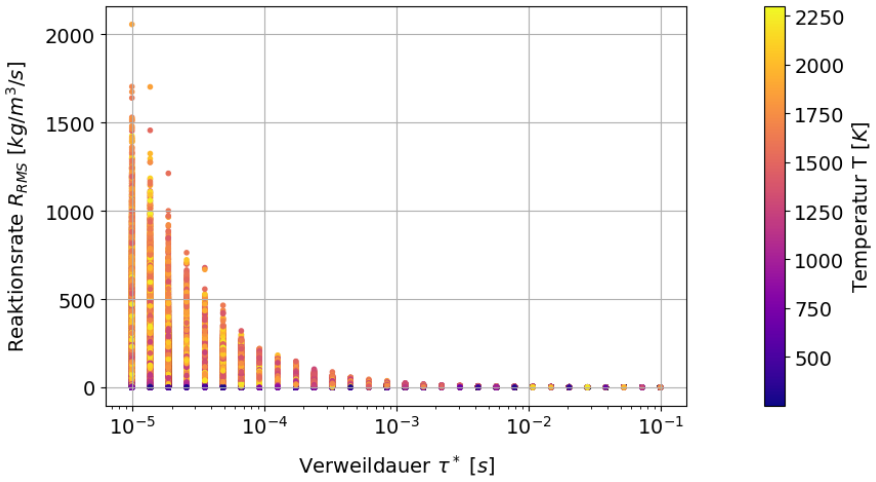


(b) Verteilung der mittleren Reaktionsraten mit Wurzelfunktion $(R_{RMS})^{1/10}$

Abbildung 4.2: Datensatz B: Einfluss des Preprocessing auf die statistische Verteilung der Datenpunkte



(a) Datensatz A



(b) Datensatz B

Abbildung 4.3: Vergleich der auftretenden durchschnittlichen Reaktionsraten R_{RMS} in den Datensätzen

5 Netztopologie und Training

Entscheidend für die Vorhersagekraft eines KNN ist der Aufbau des Netzes. Das beinhaltet die Anzahl der Schichten und Neuronen, deren Verbindungen untereinander, die genutzten Aktivierungsfunktionen und die Fehlerfunktion. Weiters ist der verwendete Trainingsalgorithmus von besonderer Bedeutung, mit gewählter Lernrate und Lernraten-Abfall sowie der Batch-Größe.

5.1 Lernalgorithmus

Als Lernalgorithmus für alle in dieser Arbeit trainierten KNN wurde der *ADAM*-Algorithmus verwendet. Dabei handelt es sich laut Kingma et al. [19] um einen rechnerisch effizienten und robusten stochastischen Optimierungs-Algorithmus, der sich besonders für Probleme mit großen Datensätzen und vielen Parametern eignet.

Die Lernrate wird für unterschiedliche Parameter einzeln angepasst, wobei auch die Veränderung der Lernraten mittels eines gewichteten gleitenden Durchschnitts kontrolliert wird.

Die Batch-Größe des Lernprozesses hat einen maßgeblichen Einfluss auf die Generalisierungsfähigkeit des neuronalen Netzes. Im Allgemeinen gilt, je größer die Batches, desto schlechter generalisiert das Netz; das bedeutet, die Vorhersagegüte des Netzes auf einem Validierungs-Datensatz nimmt ab (siehe beispielsweise [18]). Andererseits haben größere Batches den Vorteil, dass die Berechnung pro Iterationsschritt besser parallelisiert werden kann, wodurch die Trainingszeit stark reduziert wird. Die Batch-Größe in der vorliegenden Arbeit wurde aus diesem Grund so groß wie möglich gewählt, ohne den verfügbaren Arbeitsspeicher auf der Grafikkarte (Nvidia GeForce GTX 1060 6 GB) zu überschreiten.

5.2 Hyperparameter-Tuning

Die Parameter, die zwar über den Aufbau des Netzes direkten Einfluss auf die Perfomance des Netzes haben, aber nicht vom Lernalgorithmus trainiert werden, bezeichnet man als Hyperparameter. Dazu gehören beispielsweise die Anzahl der versteckten Schichten und die Anzahl an Neuronen je Schicht.

Für die Auswahl der geeignete Hyperparameter wurde der Keras-Tuner [32] verwendet. Dabei wird mittels eines Optimierungs-Algorithmus (z.B. Bayes'sche Optimierung oder *Hyperband* [23]) ein Satz an Hyperparametern gesucht, mit Hilfe derer der Datensatz am besten approximiert werden kann.

Dabei musste auch beachtet werden, dass das Netztraining selbst nicht zu viele Ressourcen beansprucht, um im Rahmen dieser Arbeit eine größere Anzahl an Varianten der Datenaufbereitung testen zu können.

Im Verlauf dieser Arbeit hat sich für die getesteten Netze eine Netzstruktur von 4 **versteckten Schichten** mit **jeweils** 256 **Neuronen** als ein guter Kompromiss zwischen Trainingsaufwand und Netzperformance herausgestellt. Abbildung 5.1 zeigt den Aufbau eines solchen neuronalen Netzes aus den verschiedenen Schichten. Bei jeder Schicht ist angegeben, mit wie vielen Neuronen der vorhergehenden Schicht sie verbunden ist (*input*) und wie viele Neuronen sie selbst besitzt (*input*).

Als Aktivierungsfunktion für die Neuronen der Eingabeschicht und der versteckten Schichten wurde die *Rectified Linear Unit (ReLU)* verwendet (siehe Abbildung 2.6). Diese Funktion ist besonders gut geeignet für das Training von sehr tiefen neuronalen Netzen mit einer Vielzahl von versteckten Schichten [20]. Auf die Ausgabeschicht wurde eine lineare Aktivierung angewandt.

Als Kostenfunktion wurde die mittlere quadratische Abweichung (engl. mean squared error (MSE)) gewählt.

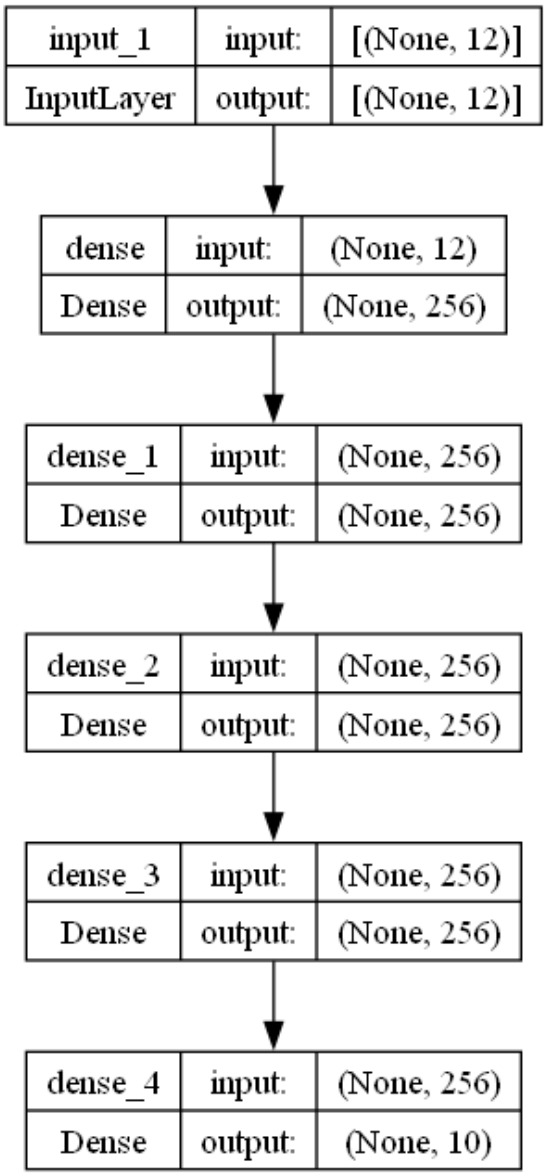


Abbildung 5.1: Aufbau der getesteten neuronalen Netze

6 Ergebnisse

Im folgenden Kapitel werden die künstlichen neuronalen Netze präsentiert, die im Verlauf dieser Arbeit erstellt worden sind. Die Netze werden bezüglich ihrer Vorhersagekraft auf den zugrundeliegenden Datensätzen bewertet und ihre Vorhersagegüte bei der CFD-Simulation der Sandia Flamme D wird untersucht. Das Kapitel soll einen logischen und chronologischen Überblick über die Herangehensweise bei der Entwicklung und Verbesserung der verschiedenen KNN verschaffen. Tabelle 6.1 zeigt eine Übersicht aller erstellten Netze und der jeweils zugrundeliegenden Datensätze.

Tabelle 6.1: Übersicht über die erstellten neuronalen Netze

KNN	Datensatz	Anzahl Netze	Normierung	Zusatz
1	A	1	Min-Max	
2	AB	1	Min-Max	
3	AB	1	Wurzelfunktion	
4	AB	4	Wurzelfunktion	
5	B	4	Wurzelfunktion	
6	B	4	Wurzelfunktion	Soft-Constraint
7	B	4	Wurzelfunktion	Hard-Constraint

6.1 KNN-1 auf Basis der Daten der Sandia Flamme D

Das erste neuronale Netz (KNN-1) wurde auf Basis von Datensatz A (siehe Kapitel 4.4) trainiert, was den Zuständen entsprach, die bei der CFD-Simulation der Sandia Flamme D mittels SCM in OpenFOAM auftraten. Dadurch sollte das Netz in der Lage sein, für diese eine Flamme passende Ergebnisse zu liefern.

Alle Eingangs- und Ausgangsgrößen im Datensatz A wurden zu Beginn mittels Min-Max-Normierung auf den Wertebereich $[0, 1]$ abgebildet (siehe dazu Kapitel 4.6). Ein Hyperparameter-Tuning mit dem *Hyperband*-Optimierungsverfahren aus dem Keras-Tuner (siehe Kapitel 5.2) ergab eine optimale Netzstruktur für das vorliegende Problem

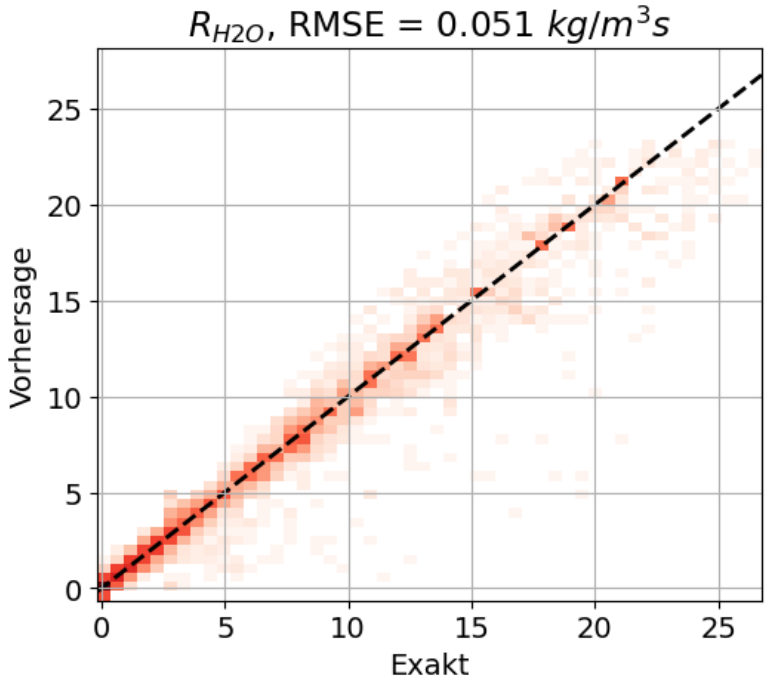


Abbildung 6.1: Vorhersagen von KNN-1 auf Basis von Datensatz A

von 4 versteckten Schichten mit jeweils 256 Neuronen. Für die Aktivierung der Neuronen aller Schichten außer der Außgabeschicht wurde die *Rectified Linear Unit* (ReLU) verwendet, siehe dazu Abbildung 2.6. Auf die Neuronen der Ausgabeschicht wurde die lineare Aktivierung angewandt.

Abbildung 6.1 zeigt beispielhaft die Performance von KNN-1 für das Abschätzen der Reaktionsrate des Wasserdampfs R_{H_2O}. Auf der Ordinate ist die Netzvorhersage für die Reaktionsrate über der „tatsächlichen“, exakten Reaktionsrate auf der Abszisse aufgetragen. Es handelt sich hierbei um ein 2D-Histogramm mit einer Auflösung von $50x50$ Feldern, wobei die Intensität der Farbe die Anzahl der Datenpunkte widerspiegelt, die sich im jeweiligen Feld befinden. Bestenfalls sollten sich alle Punkte auf der Winkelhalbierenden wiederfinden, die als schwarz gestrichelte Linie eingezeichnet ist. Die Farbskala ist logarithmisch skaliert, die hellsten Felder beinhalten dementsprechend um mehrere Größenordnungen weniger Punkte als die dunkelsten.

Abbildung 6.2 zeigt den Vergleich der CFD-Simulation mit KNN-1 mit der Referenzsimulation. Die Temperatur erreichte in der Flamme mit KNN-1 Werte von bis zu 3500 K, höhere Werte lässt OpenFOAM standardmäßig nicht zu. Die aufgetretenen Reaktionsraten waren im Durchschnitt um den Faktor 30 höher als bei der Referenzsimulation.

(a) Temperatur T der Flamme mit KNN-1 (oben) und SCM (unten)

(b) RMS der Reaktionsraten R in der Flamme mit KNN-1 (oben) und SCM (unten)

Abbildung 6.2: Vergleich der Ergebnisse des KNN-1 mit dem SCM für die Sandia Flamme D

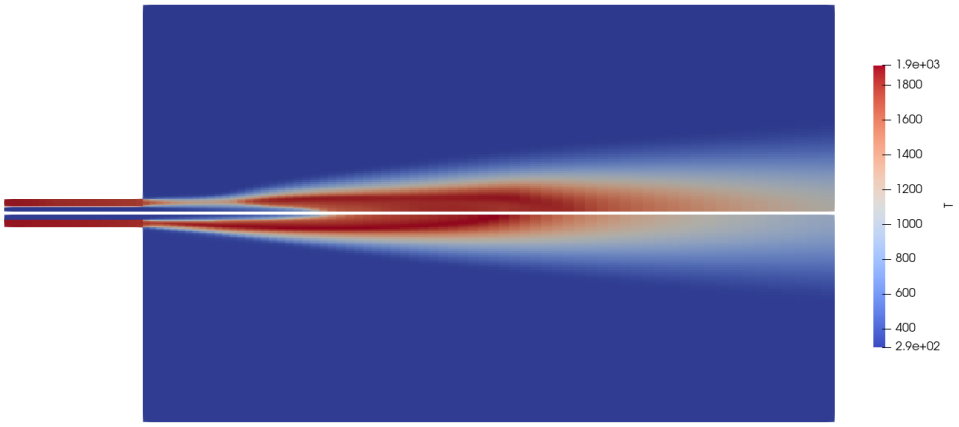


Abbildung 6.3: Vergleich der Temperaturverteilung der Sandia Flamme D berechnet mit KNN-1 (oben) und mit dem *Standard Chemistry Model* (unten)

Die KNN-Simulation wurde hier vorzeitig abgebrochen, da keine Besserung ersichtlich war.

Das Problem konnte im Auftreten kleiner Fehler zu Beginn der Simulation identifiziert werden, die dazu führten, dass die auftretenden Zustände in den Rechenzellen nicht mehr durch den Trainingsdatensatz abgedeckt waren. So ist beispielsweise die von Temperatur $T = 3500\,\mathrm{K}$ weit jenseits der maximal auftretenden Temperatur $T_{max} = 1880\,\mathrm{K}$ aus Datensatz A. Wie in Kapitel 2.3.1 erwähnt, sind KNN nicht in der Lage, bei Extrapolation zufriedenstellende Ergebnisse zu liefern, was den hohen Fehler in der Simulation erklären konnte: Waren die Zustände einmal außerhalb des Parameterraums des Trainingsdatensatzes, führten die größer werdenden Fehler dazu, dass sich die Zustände noch weiter vom Trainingsdatensatz entfernten.

Eine naheliegende Lösung war es, den OpenFOAM Quellcode so anzupassen, dass für jede Zelle untersucht wird, ob sich der jeweilige Zustand innerhalb der Grenzen des Trainingsdatensatzes befindet. Falls ja, sollte das KNN-1 zur Berechnung der Reaktionsraten herangezogen werden, andernfalls sollte das SCM für diese Zelle eingreifen.

Abbildung 6.3 zeigt als Ergebnis dieser Simulation die Temperatur-Contourplots. Die Temperaturen und Kompositionen konnten sehr gut an die Referenzsimulation angenähert werden, aber es wurden nur ca. 20% der Zellen mittels KNN behandelt, wodurch die erwünschte Beschleunigung der Simulation ausblieb. Alle anderen Zellen wurden nur mittels SCM behandelt, daher war es nicht überraschend, dass sich die Ergebnisse stark ähnelten.

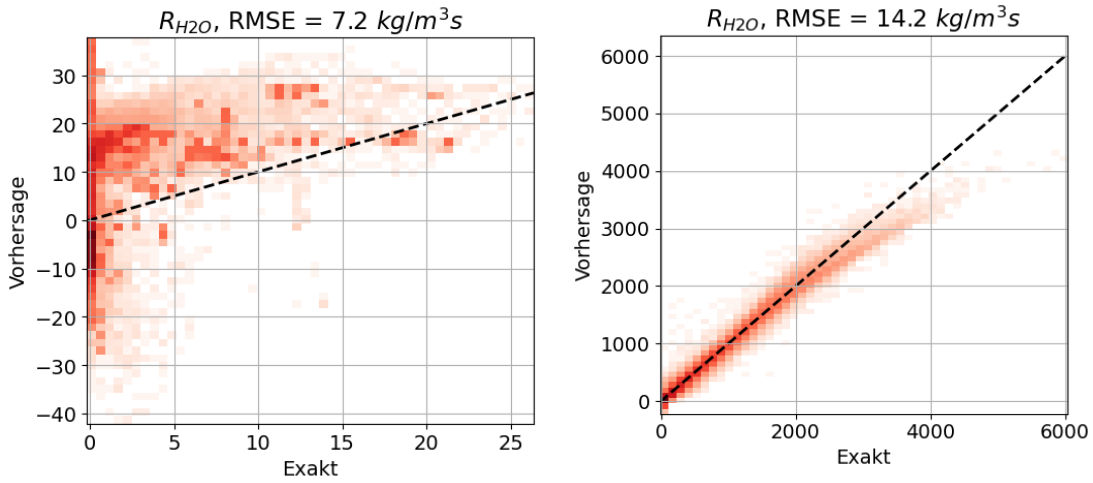
6.2 KNN-2 und KNN-3 auf Basis des erweiterten Sandia-D Datensatzes

Der nächste Schritt war nun, die Grenzen des abgedeckten Parameter-Raums zu erweitern, sodass alle Zellen in der Simulations-Domäne mittels KNN berechnet werden konnten. Dafür wurden dem Datensatz A zusätzlich Punkte aus dem Datensatz B (siehe Kapitel 4.5) beigemischt, um auch Zustände ins Training miteinzubinden, die nicht in Datensatz A vorkamen. Der resultierende, erweiterte Datensatz AB bestand zu 50% aus Punkten des Datensatzes A und zu 50% aus Punkten des Datensatzes B. Somit konnten die Grenzen des Trainingsdatensatzes auf die Grenzen von Datensatz B erweitert werden (siehe hierfür Tabelle 4.2)

Für das KNN-2 wurde wiederum eine Struktur von 4 versteckten Schichten mit jeweils 256 Neuronen gewählt. Als Aktivierungsfunktion wurde für alle Neuronen - außer jenen in der Ausgabeschicht - die ReLU-Funktion beibehalten. Der Datensatz AB wurde mittels Min-Max-Normierung auf den Wertebereich $[0, 1]$ abgebildet. Abbildung 6.4 zeigt schließlich beispielhaft die Vorhersagekraft des trainierten KNN-2 für die Reaktionsrate R_{H_2O}. Abbildung 6.4b zeigt eine zufriedenstellende Vorhersage auf Basis des gesamten Trainingssatzes AB bei Reaktionsraten $R_{RMS} < 2000\,\mathrm{kg/m^3s}$. Abbildung 6.4a zeigt jedoch, dass KNN-2 quasi keine Vorhersagekraft für den Datensatz A hat, obwohl dieser eine Teilmenge des Trainingsdatensatzes AB darstellt. Die Vorhersagekraft des KNN für den Datensatz A ist deshalb entscheidend, da dieser die Zustände beinhaltet, die bei der CFD-Simulation der Sandia Flamme D mit dem SCM auftreten, und daher auch vom KNN richtig abgeschätzt werden müssen. Beim Vergleich mit Abbildung 6.1 kann festgestellt werden, dass der root mean squared error (RMSE) bei KNN-2 um den Faktor 141 größer ist als bei KNN-1, wenn beide Netze Vorhersagen für denselben Datensatz A anstellen.

Dieser Effekt kann dadurch erklärt werden, dass die auftretenden Werte R_i in Datensatz B um bis zu zwei Größenordnungen größer waren als die entsprechenden Werte im Datensatz A, siehe dazu auch Abbildung 4.3. Abbildung 6.5 zeigt schließlich das Simulationsergebnis, das mittels KNN-2 erzielt werden konnte. Aufgrund des erweiterten Parameterraums gegenüber KNN-1 konnten über 50% der Zellen mit dem KNN berechnet werden, aber es konnte sich keine Flamme ausbilden, da der Fehler bei der Vorhersage der Reaktionsraten zu groß war.

Um das Problem der unterschiedlichen Größenordnungen der Ausgabegrößen in den Datensätzen A und B zu beheben, musste das Preprocessing dementsprechend angepasst werden. Eine lineare Transformation wie die Min-Max-Normierung oder die



(a) Vorhersagen Datensatz A

(b) Vorhersagen Datensatz AB

Abbildung 6.4: Vorhersagen des KNN-2 auf Basis unterschiedlicher Datensätze

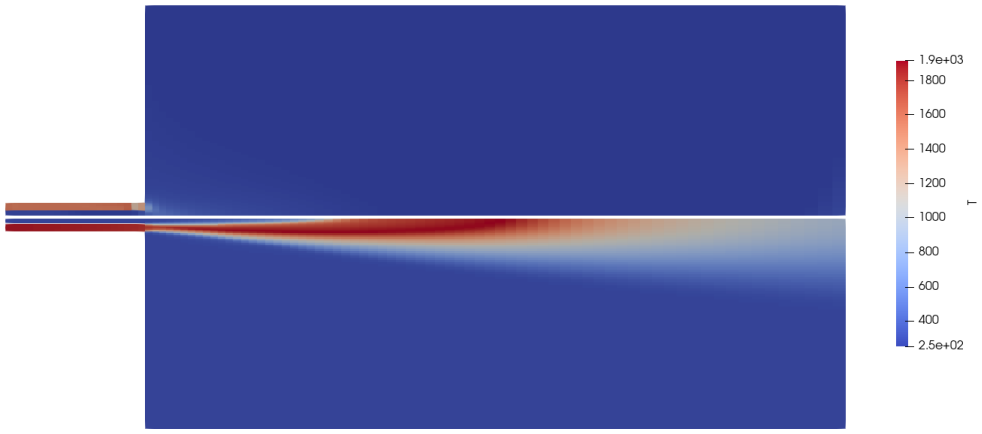
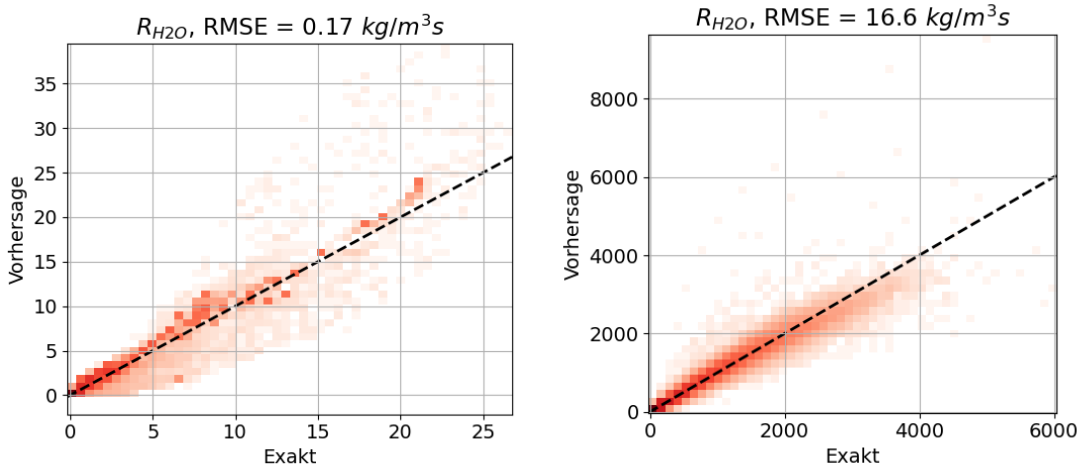


Abbildung 6.5: Vergleich der Temperaturverteilung der Sandia Flamme D berechnet mit KNN-2 (oben) und mit dem SCM (unten)



(a) Vorhersagen Datensatz A

(b) Vorhersagen Datensatz AB

Abbildung 6.6: Vorhersagen des KNN-3 auf Basis unterschiedlicher Datensätze

z-Standardisierung würde die Verhältnisse der Reaktionsraten beibehalten und war somit nicht zielführend. Eine logarithmische Transformation war nicht anwendbar, da auch Zustände mit $R_i = 0.0$ auftreten konnten, wo der Logarithmus eine Singularität aufweist. Aus diesen Gründen wurde eine Wurzelfunktion in Verbindung mit einer linearen Skalierung für jede Ausgabegröße angewandt, vgl. Gleichung 6.1. $\tilde{R}_i$ beschreibt dabei die normierte Reaktionsrate der Spezies i.

$$\tilde{R}_i = \begin{cases} \left(\frac{R_i}{max|R_i|}\right)^{\frac{1}{n}} & \text{wenn } R_i \geq 0.0 \\ -\left(\frac{|R_i|}{max|R_i|}\right)^{\frac{1}{n}} & \text{sonst} \end{cases} \tag{6.1}$$

Das KNN-3 wurde nun auf Basis von Datensatz AB mit Preprocessing gemäß Gleichung 6.1 trainiert. Für n wurde der Wert 3 gewählt. Die Topologie wurde wiederum nicht verändert: 4 versteckte Schichten mit jeweils 256 Neuronen. Abbildung 6.6 zeigt das Ergebnis des Netztrainings beispielhaft anhand der Vorhersagen für R_{H_2O}. Durch das Preprocessing konnte der Fehler für betragsmäßig kleine Reaktionsraten $|R_i|$ dramatisch gesenkt werden, beim gezeigten Beispiel von R_{H_2O} im Datensatz A um den Faktor 42. Der Fehler im gesamten Datensatz AB stieg hingegen um etwa 17%.

Abbildung 6.7 zeigt das resultierende Ergebnis der CFD-Simulation mit KNN-3 im Vergleich zur Referenzsimulation mit SCM. Obwohl hier 100% der Rechenzellen mit dem neuronalen Netz behandelt wurden, konnte die Form der Flamme qualitativ äußerst genau angenähert werden. Bei genauerer Untersuchung des Temperatur- und

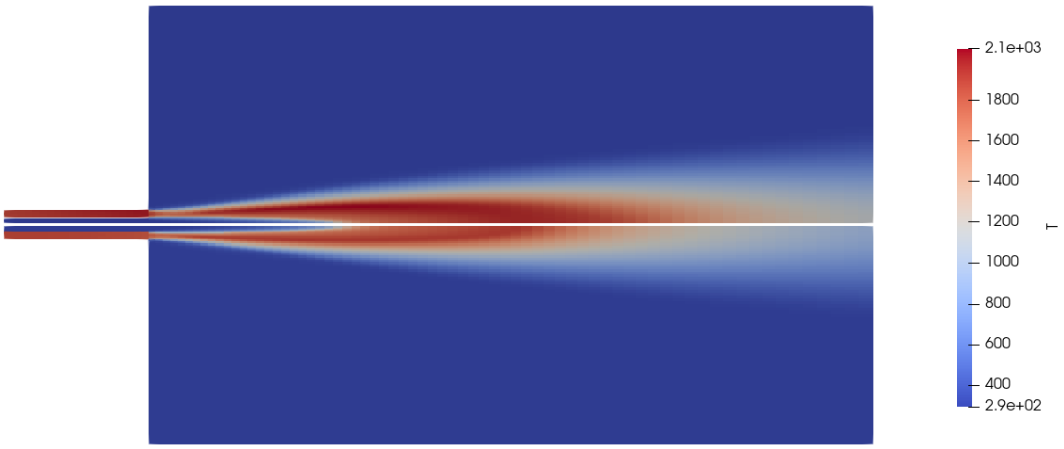
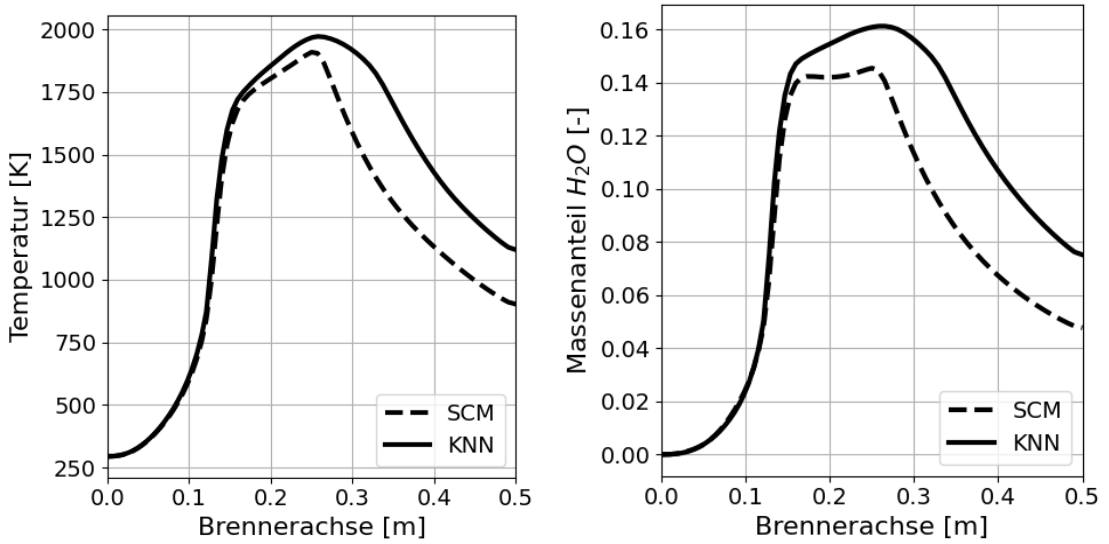


Abbildung 6.7: Vergleich der Temperaturverteilung der Sandia Flamme D berechnet mit KNN-3 (oben) und mit dem SCM (unten)

CO-Verlaufs im Zentrum der Flamme konnten dennoch einige Abweichungen festgestellt werden, siehe dazu Abbildung 6.8. Die Temperatur am Austritt wurde mit KNN um 24% zu hoch vorhergesagt, der Massenanteil von H_2O um 57% zu hoch.

6.3 KNN-4 auf Basis eines aufgeteilten Datensatzes

Ein Hauptproblem für die Vorhersagekraft für den Datensatz A eines KNN, das auf dem Datensatz AB trainiert worden ist, schien die große Diskrepanz in der Größenordnung der maximal auftretenden Reaktionsraten R_i zu sein. Beim Vergleich der Abbildungung 6.6a und 6.1 kann festgestellt werden, dass der Fehler für die Vorhersage von R_{H_2O} bei KNN-1, das nur auf dem Datensatz A trainiert wurde, um einen Faktor von 3.3 geringer war als bei KNN-3, das auf dem erweiterten Datensatz AB trainiert wurde. Der nächste Schritt war nun, den Trainingsdatensatz aufzuteilen, und jeweils ein KNN nur auf einen Teil des Datensatzes zu trainieren. Die Aufteilung sollte möglichst so erfolgen, dass die Größenordnung der Reaktionsraten R_i gut a priori abgeschätzt werden konnte, sodass folglich jedes Netz mit Reaktionsraten unterschiedlicher Größenordnungen konfrontiert wird. Abbildung 4.3b zeigt, dass die Verweildauer τ^* ein sehr guter Prädiktor für die durchschnittliche Reaktionsrate R_{RMS} im Datensatz AB war, daher wurde der Datensatz AB für das Training von KNN-4 in 4 Intervalle von τ^* unterteilt. Das Modell KNN-4 bestand somit aus 4 einzelnen KNN, wobei die abzuschätzenden Eingangsdaten für die Vorhersage gemäß ihrer Verweildauer τ^* auf die 4 Netze aufgeteilt wurden.



(a) Temperaturverlauf im Zentrum der Flamme

(b) Verlauf der H_2O-Massenanteile im Zentrum der Flamme

Abbildung 6.8: Vergleich der Ergebnisse von KNN-3 mit dem SCM für die Sandia Flamme D

Abbildung 6.9 zeigt auf, wie die Daten eingeteilt wurden. Das Netz 1 war zuständig für das geschlossene Intervall der Verweildauern $\tau_1^* \in [1.0 \cdot 10^{-5}; 4.9 \cdot 10^{-5}]s$, Netz 2 für $\tau_2^* \in [4.9 \cdot 10^{-5}; 2.4 \cdot 10^{-4}]s$, Netz 3 für $\tau_3^* \in [2.4 \cdot 10^{-4}; 1.2 \cdot 10^{-3}]s$ und Netz 4 schließlich für $\tau_4^* \in [1.2 \cdot 10^{-3}; 1.0 \cdot 10^{-1}]s$. Man beachte, dass die Grenzwerte der Intervalle selbst jeweils in beide angrenzenden Intervalle aufgenommen wurden.

Für alle 4 neuronalen Netze wurde wiederum dieselbe Netzstruktur von 4 versteckten Schichten mit jeweils 256 Neuronen gewählt. Abbildung 6.10 zeigt, welcher Zuwachs an Vorhersagekraft durch diese Anpassung möglich war. Im Vergleich mit Abbildung 6.6 zeigt sich, dass der durchschnittliche Fehler bei der Vorhersage auf Basis von Datensatz A von KNN-4 um einen Faktor von 10.6 gegenüber KNN-3 gesenkt werden konnte. Der durchschnittliche Fehler bezüglich des gesamten Datensatzes AB konnte um den Faktor 19.8 gesenkt werden.

Auch in der CFD-Simulation der Flamme zeigt sich der Zuwachs an Genauigkeit: Abbildung 6.12 zeigt den Vergleich der Verläufe von Temperatur und H_2O-Massenanteilen entlang des Zentrums der Sandia Flamme D, wie sie jeweils mithilfe von KNN-4 und SCM berechnet wurden. Die Spitzentemperatur der Flamme konnte im Vergleich zur Berechnung mit dem SCM bis auf 9.5 K genau abgeschätzt werden, was einem relativen Fehler von lediglich 0.5% entsprach.

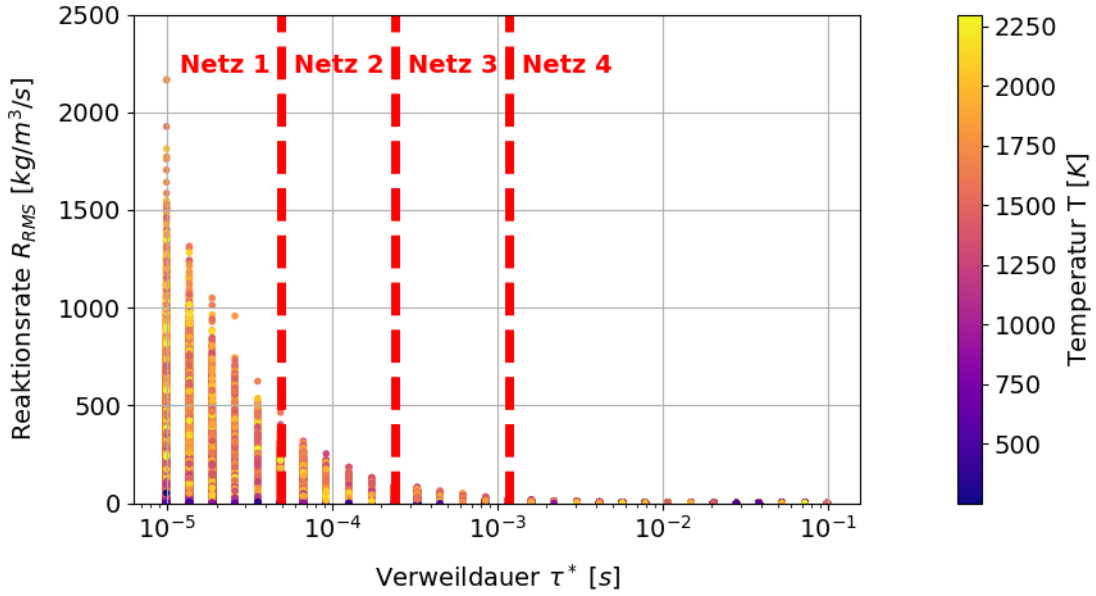
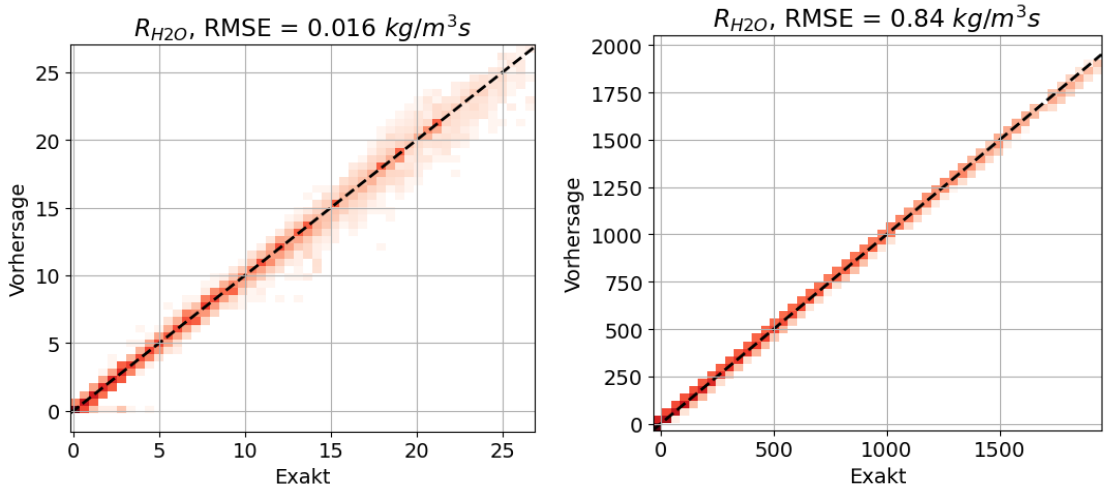


Abbildung 6.9: Aufteilung der Trainingsdaten auf 4 KNN gemäß der Verweildauer τ^*



(a) Vorhersagen Datensatz A

(b) Vorhersagen Datensatz AB

Abbildung 6.10: Vorhersagen des KNN-4 auf Basis unterschiedlicher Datensätze

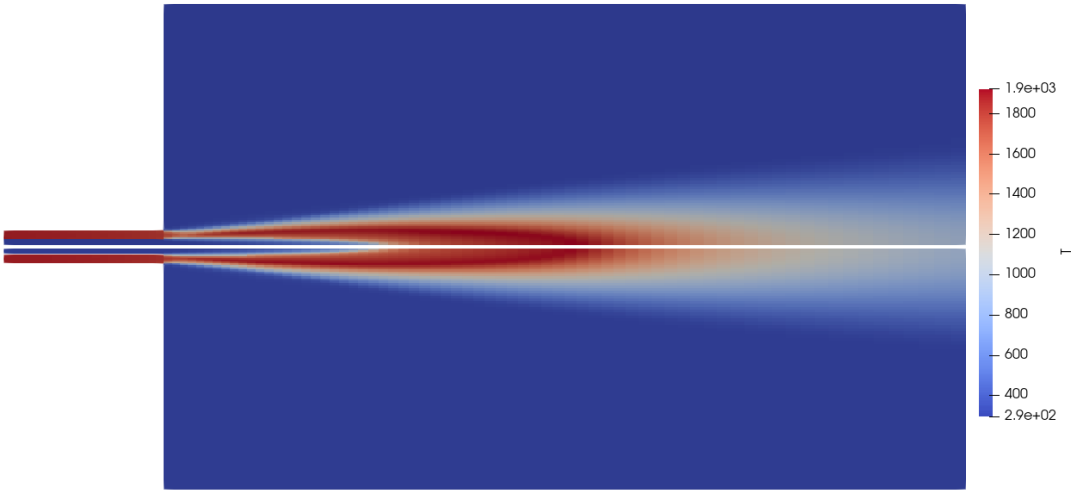
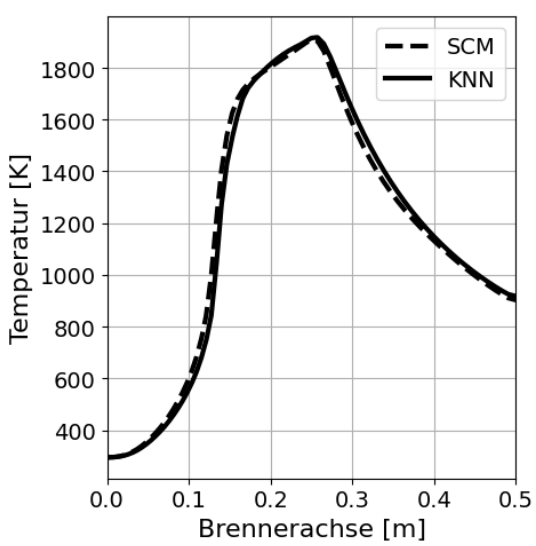
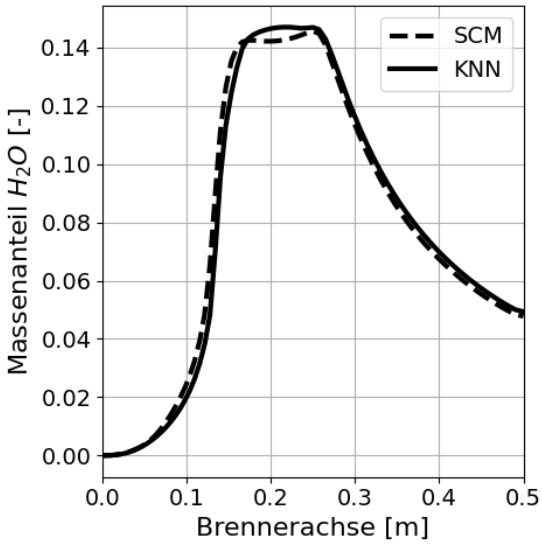


Abbildung 6.11: Vergleich der Temperaturverteilung der Sandia Flamme D berechnet mit KNN-4 (oben) und mit dem SCM (unten)



(a) Temperaturverlauf im Zentrum der Flamme



(b) Verlauf der H_2O-Massenanteile im Zentrum der Flamme

Abbildung 6.12: Vergleich der Ergebnisse von KNN-4 mit dem SCM für die Sandia Flamme D

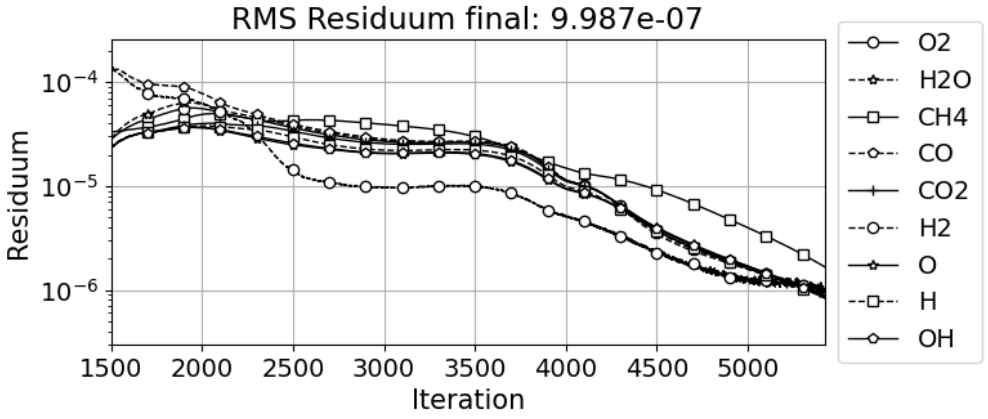
Tabelle 6.2: Vergleich der Rechenzeit der Sandia Flamme D in Sekunden mit SCM und KNN-4

	1 CPU	4 CPU	1 CPU + GPU
SCM	474	224	474
KNN-4	334	207	269
rel. Beschleunigung	+42%	+9%	+76%

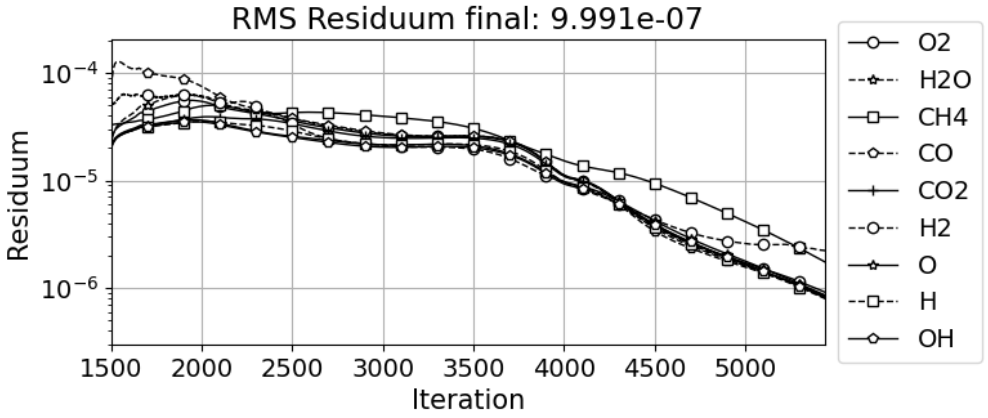
6.3.1 Rechenzeit

Um die Rechenzeiten vergleichen zu können, wurden die CFD-Simulationen, bei denen das KNN-4 eingesetzt wurde, sowie jene, die mit dem SCM berechnet wurden, auf demselben PC mit dem Prozessor Intel i7-6700k mit 4 physischen Kernen und einer NVIDIA Geforce GTX 1060 Grafikkarte mit 6GB GDDR5 RAM durchgeführt. Die Simulationen wurde dabei jeweils einmal mit 1 und einmal mit 4 CPU-Kernen durchgeführt, um auch zu untersuchen, wie gut sich die Berechnung mittels KNN parallelisieren lässt. Für den Vergleich galt eine Simulation als abgeschlossen, wenn der root mean square (RMS) der Residuen aller Massenanteile Y_i unter 10^{-6} fiel. Abbildung 6.13 zeigt vergleichsweise die Konvergenz der Simulationen mit KNN und mit SCM. Aufgrund der herausragenden Vorhersagen des KNN-4 zeigten beide Simulationen beinahe dasselbe Konvergenzverhalten, lediglich das Residuum des Massenanteils von Wasserstoff Y_{H2} war etwas niedriger mit dem SCM.

Der direkte Vergleich der Rechenzeiten gestaltete sich etwas schwieriger, da die Simulationsdauer mit dem SCM sehr stark von den gewählten Toleranzen für den `SELUEX` Differentialgleichungs-Löser abhing (siehe Abschnitt 3.1). Für den Vergleich wurde die Simulation mit dem SCM mit sehr groben Löser-Toleranzen von `relTol`$= 0.1$ und `absTol`$= 10^{-8}$ durchgeführt, um als „best case"-Szenario hinsichtlich der Rechenzeit für den SCM zu dienen. Tabelle 6.2 listet die resultierenden Rechenzeiten in Sekunden. Es zeigte sich, dass bei Berechnung auf einem Rechenkern trotz der groben Toleranzen bei dem SCM eine Beschleunigung der Simulation mittels KNN um 42% verzeichnet werden konnte. Bei Auslagerung der Vorhersage des neuronalen Netzes auf die GPU konnte die Rechenzeit sogar um 43% gesenkt werden, was einer Beschleunigung von 76% entsprach. Um das volle Potenzial des KNN auch bei CPU-Parallelisierung ausschöpfen zu können, müsste der Quellcode in OpenFOAM weiter daraufhin optimiert werden, was nicht der Fokus dieser Arbeit war.

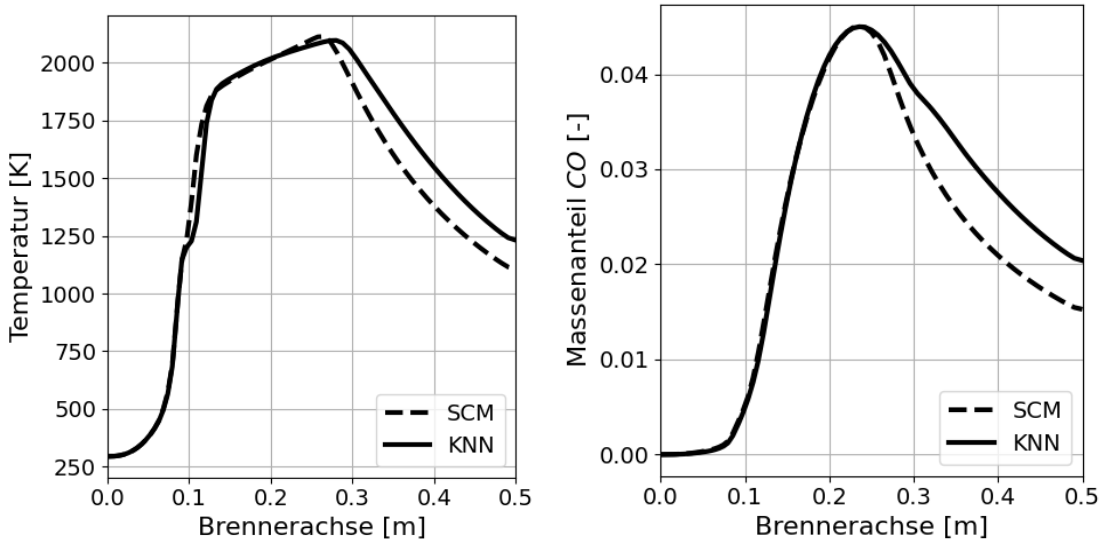


(a) Residuen der Simulation mit SCM



(b) Residuen der Simulation mit KNN-4

Abbildung 6.13: Vergleich der Konvergenz der Simulationen mit KNN-4 und mit dem SCM für die Sandia Flamme D



(a) Temperaturverlauf im Zentrum der Flamme

(b) Verlauf der *CO*-Massenanteile im Zentrum der Flamme

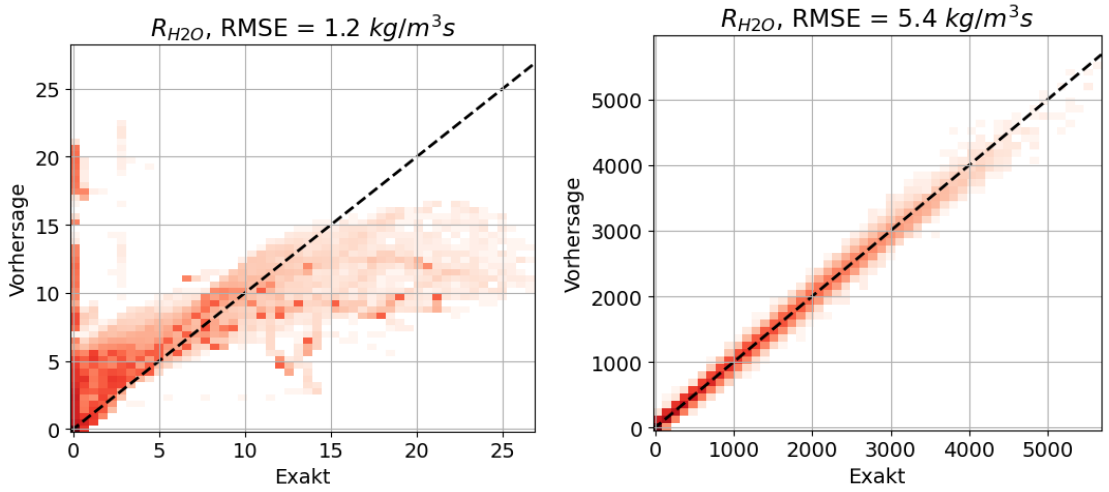
Abbildung 6.14: Vergleich der Ergebnisse von KNN-4 mit dem SCM für die modifizierte Flamme mit einem Hauptstrom-Gemisch aus 5% H_2, 10% CH_4 und Luft sowie einer Einströmgeschwindigkeit von 20 m/s

6.3.2 Generalisierung

Mit dem beschriebenen KNN-4 konnten gute Ergebnisse für die Sandia Flamme D erzielt werden, wie in Abbildung 6.12 deutlich erkennbar ist. Dadurch, dass das Netz zu 50% auf Basis der Daten dieser Flamme trainiert wurde, lässt sich das Ergebnis aber nicht auf andere Gemische und Strömungsbedingungen übertragen, das Netz kann also nicht gut *generalisieren*. Abbildung 6.14 zeigt zum Vergleich die Netzperformance, wenn der Flamme im Hauptstrahl anstelle eines 15% CH_4-Luft-Gemischs ein Gemisch mit 10% CH_4 und 5% H_2 als Brennstoff zugeführt wird. Weiters wurde die Einströmgeschwindigkeit des Brennstoff-Luft-Gemischs von 10 m/s auf 20 m/s erhöht, um auch die Reynoldszahl der Strömung zu verändern. Beim Vergleich mit Abbildung 6.12 kann festgestellt werden, dass die Vorhersagekraft des Netzes unter den veränderten Randbedingungen nicht mehr in gleichem Maße gegeben war.

6.4 KNN-5 auf Basis der Daten aus Cantera

Um dieses Problem der Generalisierung umgehen zu können, sollte nun ein KNN auf Basis eines Datensatzes trainiert werden, das möglichst gut auf eine Vielzahl



(a) Vorhersagen Datensatz A

(b) Vorhersagen Datensatz B

Abbildung 6.15: Vorhersagen des KNN-5 auf Basis unterschiedlicher Datensätze

verschiedener Flammen zutreffen würde. Hierfür war der Datensatz B mittels Monte-Carlo-Sampling geschaffen worden, siehe dazu Kapitel 4.4. Für die Topologie des KNN-5 wurden wiederum 4 versteckte Schichten mit jeweils 256 Neuronen gewählt. Das Preprocessing erfolgte auch hier mit der Wurzelfunktion gemäß Gleichung 6.1 und der Trainingsdatensatz wurde nach der Verweildauer τ^* in 4 Teile unterteilt. Abbildung 6.15 zeigt die Netzperformance bei der Vorhersage auf Basis der Datensätze A und B am Beispiel von R_{H_2O}. Aufgrund der geringen Anzahl an Datenpunkten in Datensatz B, deren Reaktionsraten dieselbe Größenordnung wie jene in Datensatz A aufwiesen, war die Vorhersagekraft entsprechend mangelhaft. Der RMSE bei der Vorhersage auf Datensatz A war mit KNN-5 um den Faktor 75 höher als mit KNN-4 (vgl. Abbildung 6.10a).

Dieses Ergebnis spiegelte sich auch im resultierenden Temperaturfeld wider, das in Abbildung 6.16a dargestellt ist. Die Strömung war bei der Simulation mit KNN bereits im Austritt des Pilotstrahls um über 200 K zu heiß. Der Grund hierfür kann in 6.16b erkannt werden: im Pilot-Rohr wurden vom KNN unphysikalisch hohe Reaktionsraten vorhergesagt. Die Hauptmischungs- und -reaktionszone direkt nach dem Auslass des Pilot- und Hauptstrahls konnte durch das KNN gut vorhergesagt werden, aber im weiteren Verlauf der Flamme führten betragsmäßig zu hoch geschätzte Reaktionsraten R_{O_2} von Sauerstoff und R_{H_2O} von Wasserdampf zu einer unphysikalisch langen und heißen Verbrennung.

Ein weiteres Problem betraf das Thema der Massenerhaltung. Bei einer chemischen

Reaktion kann sich zwar die Stoffmenge in mol ändern, die Masse muss aber zwischen Edukten und Produkten immer erhalten bleiben. Das führt zur Bedingung, dass die Summe der Reaktionsraten R_i in einer Rechenzelle gleich 0 sein muss, siehe Gleichung 6.2. Diese Bedingung ist bei der Abschätzung der Reaktionsraten durch das neuronale Netz nicht automatisch erfüllt. Abbildung 6.17 zeigt die Summe der Reaktionsraten $\sum R_i$ in den resultierenden Strömungsfeldern der Simulationen mit KNN-5 und mit SCM. Ein positiver Wert entspricht einer Quelle, ein negativer Wert entspricht einer Senke.

$$\sum_{i=1}^{10} R_i \stackrel{!}{=} 0\,\mathrm{kg/m^3s} \tag{6.2}$$

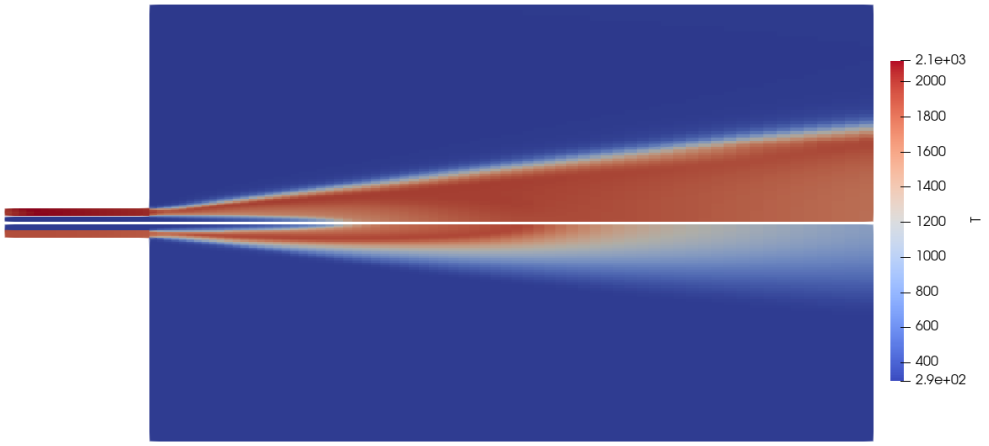
6.5 KNN-6 und KNN-7 als Physik-informierte neuronale Netze

Das KNN sollte nun so angepasst werden, dass die Masse erhalten bleibt. Dies war durch die Nutzung von PINN möglich, wie bereits in Abschnitt 2.3.4 angedeutet. Dabei konnten an dieser Stelle zwei grundlegende Herangehensweisen unterschieden werden:

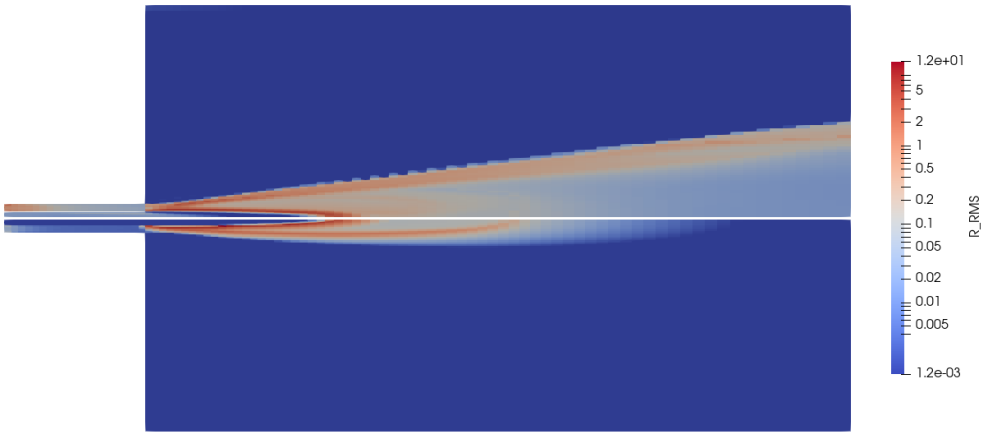
1. „Soft Constraint“: Die Kostenfunktion L wird so angepasst, dass die Summe der Reaktionsraten als Residuum der Massenerhaltung gewichtet um einen Faktor w mit in die Kosten aufgenommen wird. Der Faktor w musste in weiterer Folge empirisch ermittelt werden. (KNN-6)
2. „Hard Constraint“: Die Netzausgabe wird mit einer zusätzlichen Schicht oder Aktivierung so angepasst, dass die Massenerhaltung gezwungenermaßen erfüllt ist. (KNN-7)

6.5.1 KNN-6: Soft Constraint

Um die Erhaltung der Masse zu erzwingen, wurde die Kostenfunktion L um einen Term erweitert, der das Residuum der Gleichung 6.2 darstellt, gewichtet um einen Faktor w. Das Optimum für den Faktor w musste empirisch mittels Parametervariation bestimmt werden. Gleichung 6.3 zeigt die verwendete Kostenfunktion, um den Fehler für eine Vorhersage k zu berechnen. R_i bezeichnet hier die tatsächliche Reaktionsrate der Spezies i, $\tilde{R}_i$ die entsprechende Netzvorhersage. Der erste Term der Funktion stellt den gewöhnlichen *mean squared error (mse)* dar. N bezeichnet dabei die Anzahl der



(a) Contour-Plot der Temperatur T für die Simulation mit KNN-5 (oben) und mit dem SCM (unten)



(b) Contour-Plot der durchschnittlichen Reaktionsraten R_{RMS} für die Simulation mit KNN-5 (oben) und mit dem SCM (unten)

Abbildung 6.16: Contour-Plots als Vergleich der Simulationsergebnisse mit KNN-5 und mit dem SCM für die Sandia Flamme D

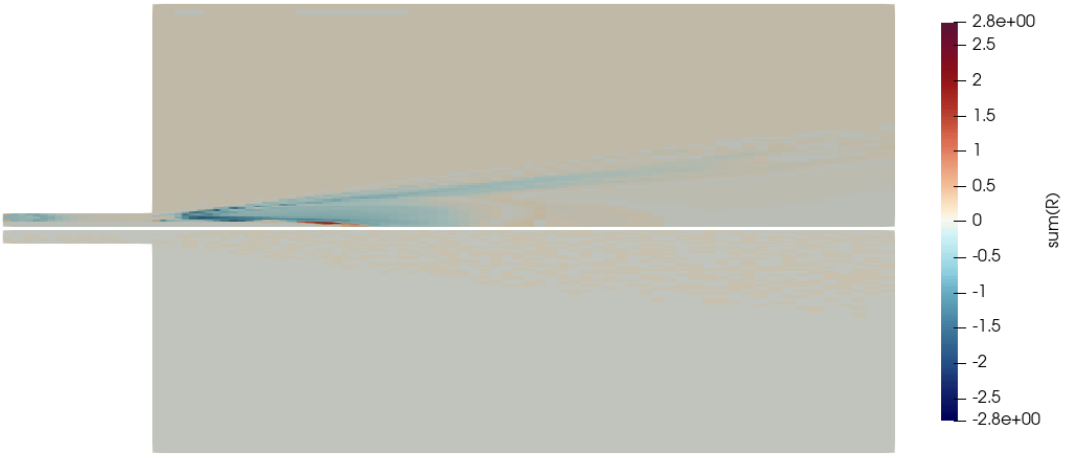


Abbildung 6.17: Vergleich der Einhaltung der Massenerhaltung bei chemischen Reaktionen in der Sandia Flamme D berechnet mit KNN-5 (oben) und mit dem SCM (unten)

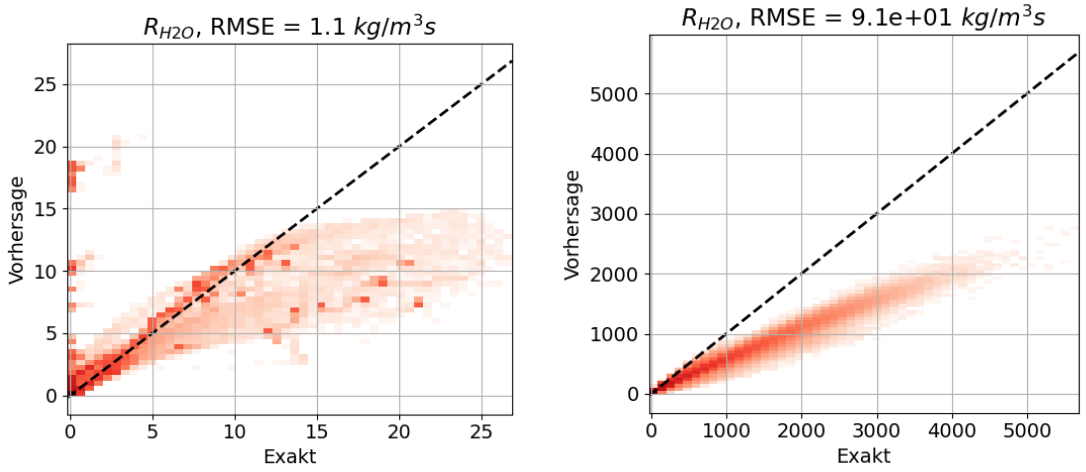
Spezies. Der zweite Term in der Funktion ist der Betrag des mittleren Residuums der Massenerhaltung, gewichtet um den Faktor w.

$$L_k(R, \tilde{R}) = \sum_{i=1}^{N} \frac{(R_i - \tilde{R}_i)^2}{N} + w \cdot \left| \sum_{i=1}^{N} \tilde{R}_i \right| \tag{6.3}$$

Als Ausgangspunkt für das hiermit erstellte KNN-6 diente das bereits trainierte KNN-5. Dieses wurde für 200 weitere Epochen mit der Fehlerfunktion 6.3 trainiert. Abbildung 6.18 zeigt beispielhaft die Vorhersagekraft des neuen Netzes anhand der Reaktionsraten für Wasserdampf R_{H_2O} mit einer Gewichtung $w = 0.01$. Beim Vergleich mit Abbildung 6.15 kann sofort festgestellt werden, dass die Vorhersagekraft des Netzes beim Training der Forderung nach Massenerhaltung geopfert wird. Bei kleineren Werten für w steigt die Vorhersagekraft nicht nennenswert, während sie bei größeren Werten noch stärker einbricht, da der Lernalgorithmus versucht, den Term $\left|\sum_{i=1}^{N} \tilde{R}_i\right|$ in der Fehlerfunktion zu minimieren, indem die Netzvorhersagen $\tilde{R}_i$ minimiert werden.

6.5.2 KNN-7: Hard Constraint

Um einen *Hard Constraint* zu realisieren, wurde der Datensatz B so adaptiert, dass die Ausgangsgröße der Netze nicht mehr die Reaktionsrate R_i war, sondern der Massenanteil Y_i. Damit konnte auf die Netzeingaben net_i der Output-Neuronen des KNN die `Softmax`-Aktivierung [41] (siehe Gleichung 6.4) angewandt werden, die garantiert, dass



(a) Vorhersagen Datensatz A

(b) Vorhersagen Datensatz B

Abbildung 6.18: Vorhersagen des KNN-6 auf Basis unterschiedlicher Datensätze

die Summe aller Ausgabe-Werte (in diesem Fall also aller Massenanteile Y_i) gleich 1 ist. Damit war sichergestellt, dass die Gleichung 4.2 stets erfüllt war.

$$Y_i(net_i) = \frac{e^{net_i - \max(net)}}{\sum_{i=1}^{N}(e^{net_i - \max(net)})} \tag{6.4}$$

Für das KNN-7 wurde nachfolgend wieder eine Netzstruktur von 4 versteckten Schichten mit jeweils 256 Neuronen gewählt. Für die Aktivierung der versteckten Schichten wurde `ReLU` beibehalten, während auf die Neuronen der Ausgabeschicht die `Softmax`-Aktivierung angewandt wurde. Abbildung 6.19 zeigt die Vorhersagekraft des Netzes auf den Eingabedaten von Datensatz B beispielhaft anhand des Massenanteils von Wasserdampf Y_{H_2O}. Die Massenanteile Y_i konnten mithilfe dieser Netzstruktur sehr genau vorhergesagt werden, aber es durfte nicht vernachlässigt werden, dass dieser Fehler bei Berechnung der jeweiligen Reaktionsrate R_i um ein Vielfaches vergrößert wird. Man betrachte beispielsweise einen Fall einer tatsächlichen Reaktionsrate von $R = 0\,\mathrm{kg/m^3s}$ bei einer Verweildauer von $\tau^* = 5 \cdot 10^{-5}\,\mathrm{s}$ und einer Dichte des Gemischs $\rho_0 = 1\,\mathrm{kg/m^3}$. Der Massenanteil der betreffenden Spezies wird nun vom neuronalen Netz gemäß des RMSE aus Abbildung 6.19 um $1.6 \cdot 10^{-3}$ falsch abgeschätzt. Das würde zu einem Fehler von $\Delta R = \rho_0 \frac{\Delta Y}{\tau^*} = 32\,\mathrm{kg/m^3s}$ führen, was dem 6-fachen des RSME aus Abbildung 6.15b entspricht.

Abbildung 6.20 zeigt schließlich den Vergleich der Ergebnisse der CFD-Simulationen mit KNN-7 und mit dem SCM. Beim Vergleich von Abbildung 6.20c mit Abbildung

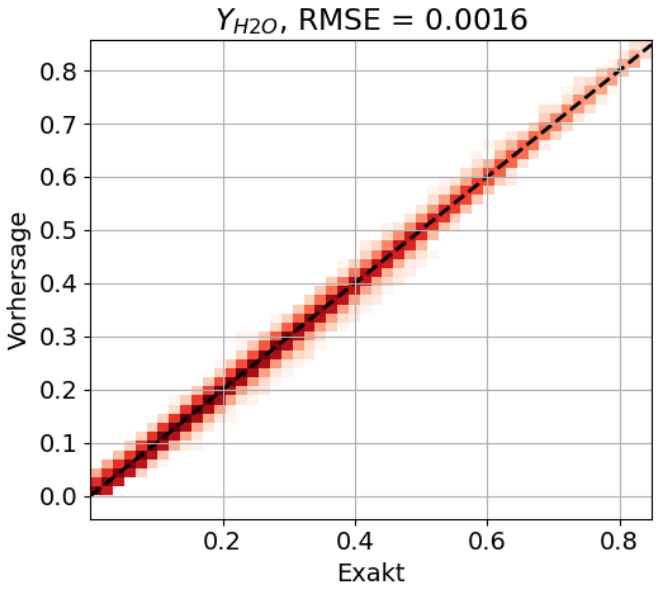
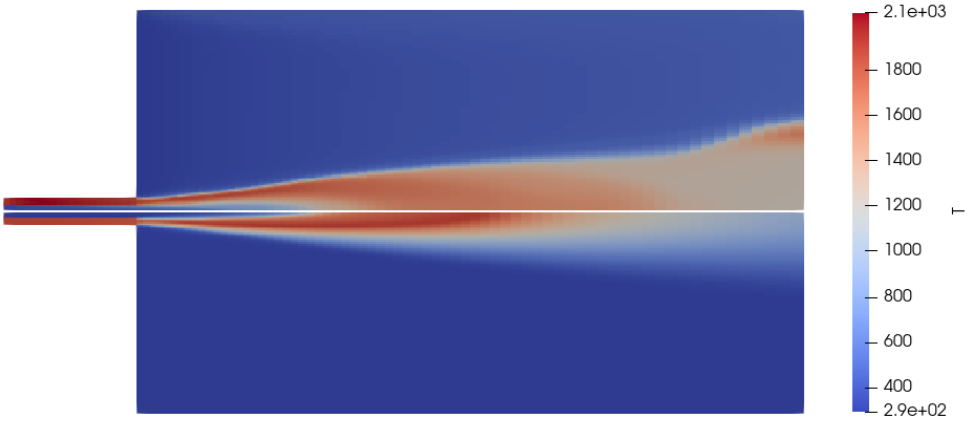
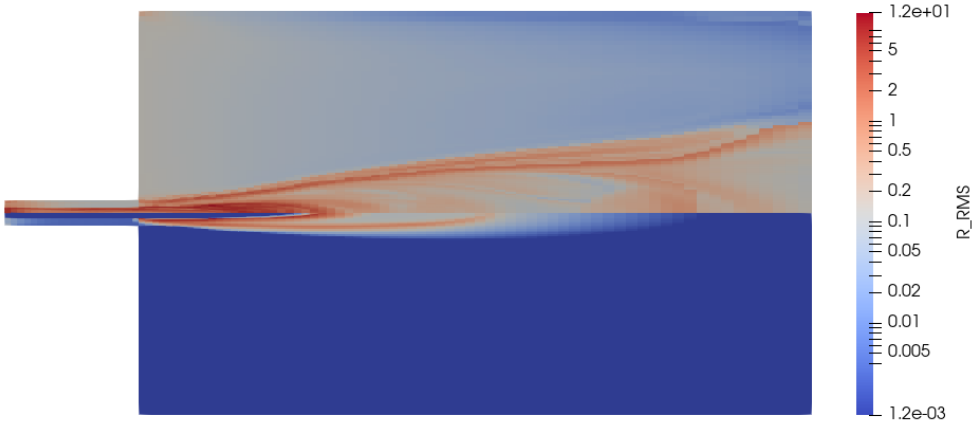


Abbildung 6.19: Vorhersageperformance des KNN-7 auf Basis von Datensatz B

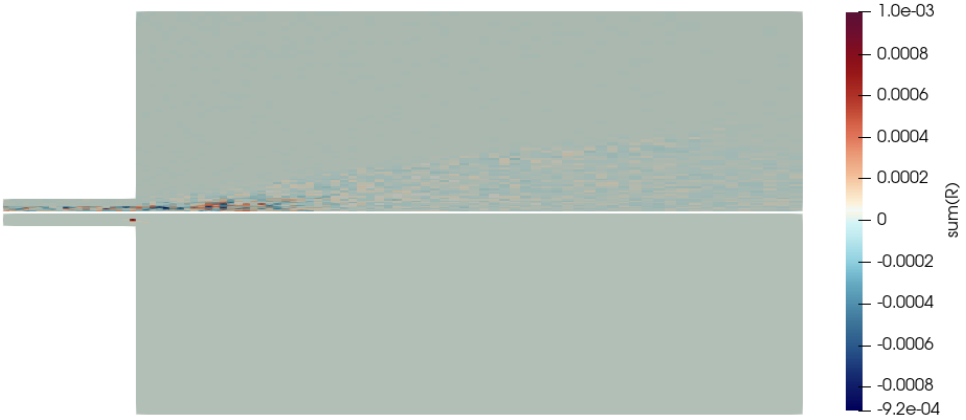
6.17 kann festgestellt werden, dass bei Berechnung mit KNN-7 die Masse erhalten blieb und lediglich Fehler in der Größenordnung von maximal $10^{-3}\,\mathrm{kg/m^3s}$ auftraten, im Vergleich dazu bis zu $2.8\,\mathrm{kg/m^3s}$ bei KNN-5. Abbildung 6.20b zeigt, dass die Abschätzung der Größenordnung der Reaktionsraten mit KNN-7 in etwa stimmte, aber zu hohe Reaktionsraten in der Nachlaufzone der Flamme vorhergesagt wurden.



(a) Contour-Plot der Temperatur T für die Simulation mit KNN-7 (oben) und mit dem SCM (unten)



(b) Contour-Plot der durchschnittlichen Reaktionsraten R_{RMS} für die Simulation mit KNN-7 (oben) und mit dem SCM (unten)



(c) Contour-Plot der Summe der Reaktionsraten $\sum(R_i)$ für die Simulation mit KNN-7 (oben) und mit dem SCM (unten)

Abbildung 6.20: Contour-Plots als Vergleich der Simulationsergebnisse mit KNN-7 und mit dem SCM für die Sandia Flamme D

7 Zusammenfassung und Ausblick

Mittels künstlicher neuronaler Netze konnte die Rechenaufwand für die Simulation einer turbulenten Flamme bei Berechnung mit einem CPU-Kern um bis zu 46% reduziert werden. Das Netzgitter war dabei mit 5170 Rechenzellen äußerst grob und der gewählte Reaktionsmechanismus (Jones-Lindstedt, $N_S = 10$) sehr klein. Es wird vermutet, dass die Beschleunigung, die durch den Gebrauch von KNN erzielt werden kann, für feinere Netzgitter und größere Reaktionsmechanismen noch bedeutend größer sein könnte.

Die neuronalen Netze hatten hierbei die Aufgabe, die rechnerisch aufwändige numerische Integration der Reaktionskinetik in den Rechenzellen zu umgehen, und die Reaktionsraten direkt abzuschätzen. Die Spitzentemperatur der Flamme konnte dabei im Vergleich zur Berechnung mit OpenFOAM's Standard Chemistry Model (SCM) bis auf 9.5 K genau abgeschätzt werden, was bei einer Soll-Temperatur von $T_{max} = 1880\,\mathrm{K}$ einem relativen Fehler von lediglich 0.5% entsprach.

Das Training der KNN wurde auf Basis zweier Datensätze durchgeführt: Datensatz A wurde aus Daten aufgebaut, die direkt aus der CFD-Simulation der Sandia Flamme D gewonnen wurden. Datensatz B wurde mit der Monte Carlo Methode generiert, indem eine große Anzahl an Rohrreaktoren in Cantera simuliert wurde, deren Anfangsbedingungen zufällig erzeugt wurden.

Der Datensatz A konnte zwar vom Netz sehr gut approximiert werden, aber kleine Fehler in der Vorhersage führten schnell dazu, dass das Netz bei der CFD-Simulation extrapolieren musste, was in der Folge eine Konvergenz der Simulation verhinderte. Durch Zugabe einiger Datenpunkte aus Datensatz B konnten die Grenzen des Parameterraums der Trainingsdaten erweitert werden, um die anschließende Simulation zu stabilisieren.

Das beste Ergebnis für die Sandia Flamme D wurde schließlich erreicht, indem dieser kombinierte Datensatz AB auf 4 Netze aufgeteilt wurde, die die Reaktionsraten in der Flamme beinahe identisch zur Berechnung mittels `StandardChemistryModel` abschätzen konnten. Dieses KNN hatte aber den Nachteil, dass es hauptsächlich auf Daten

aus der Sandia Flamme D trainiert wurde, und diese Vorhersagekraft daher nur bei Berechnung dieser einen Flamme erreichen konnte.

Weitere Netze wurden nur auf Basis des Datensatzes B trainiert. Das Ziel dabei war, einen Datensatz zu erstellen, der weitestgehend unabhängig von der zu untersuchenden Flamme ist, um ein möglichst allgemein einsetzbares KNN trainieren zu können. Aufgrund der großen Unterschiede in der Größenordnung der auftretenden Reaktionsraten zwischen den Datensätzen A und B konnte auf diese Weise allerdings kein zufriedenstellendes Ergebnis erzielt werden. Künftige Arbeiten sollten sich darauf fokussieren, den Trainingsdatensatz besser an die Verteilung der auftretenden Werte in der Flamme anzupassen.

Ein weiteres Problem, das in dieser Arbeit adressiert wurde, war die Erhaltung der Masse in den chemischen Reaktionen. Bei Abschätzung der Reaktionsraten mittels KNN war die Massenerhaltung nicht automatisch gegeben. Es wurde versucht, die Erhaltung der Masse mittels *Soft-Constraint* oder *Hard-Constraint* zu erzwingen. Für den *Soft-Constraint* wurde die Kostenfunktion des neuronalen Netzes angepasst und um das Residuum der Massenerhaltung ergänzt. Dies führte jedoch nur dazu, dass von dem Netz kleinere Reaktionsraten vorhergesagt wurden, um die Kostenfunktion zu minimieren. Ein *Hard-Constraint* wurde so realisiert, dass anstelle der Reaktionsraten die resultierenden Massenanteile abgeschätzt wurden, die mittels `Softmax`-Aktivierung normiert wurden, um die Summe von 1 zu erhalten. Diese Variante scheint vielversprechend, erfordert aber noch weitere Anpassung, um die Vorhersagekraft zu verbessern.

Um den vollen Nutzen für die Reduktion der Rechenzeit aus der Performance der KNN ziehen zu können, sollten sich Folgearbeiten weiters damit beschäftigen, den Code für CPU-Parallelisierung zu optimieren.

Abbildungsverzeichnis

Literatur

[1] J. Ahrens, Berk Geveci und Charles Law. »ParaView: An End-User Tool for Large Data Visualization«. In: *Visualization Handbook* (Jan. 2005) (siehe S. 21).

[2] Robert Barlow und Jonathan Frank. *Sandia/TUD Piloted CH4/Air Jet Flames*. 2003. URL: https://tnfworkshop.org/data-archives/pilotedjet/ch4-air/ (siehe S. 20–22).

[3] Markus Bösenhofer u. a. »The Eddy Dissipation Concept — Analysis of Different Fine Structure Treatments for Classical Combustion«. In: *Energies* 11.7 (2018). ISSN: 1996-1073. DOI: 10.3390/en11071902. URL: https://www.mdpi.com/1996-1073/11/7/1902 (siehe S. 5, 7, 9, 10, 26).

[4] *Chemical-Kinetic Mechanisms for Combustion Applications*. http://combustion.ucsd.edu. Mechanical und Aerospace Engineering (Combustion Research), University of California at San Diego (siehe S. 4).

[5] François Chollet u. a. *Keras*. https://keras.io. 2015 (siehe S. 2, 22).

[6] Pedro Domingos. *Every Model Learned by Gradient Descent Is Approximately a Kernel Machine*. 2020. arXiv: 2012.00152 (siehe S. 11).

[7] Alessio Frassoldati u. a. »Simplified kinetic schemes for oxy-fuel combustion«. In: (Jan. 2009) (siehe S. 4, 25, 26).

[8] Peter Gerlinger. *Numerische Verbrennungssimulation. Effiziente numerische Simulation turbulenter Verbrennung*. Springer Berlin, Heidelberg, 2005. DOI: 10.1007/3-540-27535-5 (siehe S. 7).

[9] David G. Goodwin u. a. *Cantera: An Object-oriented Software Toolkit for Chemical Kinetics, Thermodynamics, and Transport Processes*. https://www.cantera.org. Version 3.0.0. 2023. DOI: 10.5281/zenodo.8137090 (siehe S. 6, 25).

[10] Christopher Greenshields. *OpenFOAM v11 User Guide*. London, UK: The OpenFOAM Foundation, 2023. URL: https://doc.cfd.direct/openfoam/user-guide-v11 (siehe S. 2, 19).

[11] Carl Heinz Hamann, Dirk Hoogestraat und Rainer Koch. *Grundlagen der Kinetik. Von Transportprozessen zur Reaktionskinetik*. Springer Spektrum Berlin, Heidelberg, 2018. DOI: 10.1007/978-3-662-49393-9 (siehe S. 3).

[12] Kaiming He u. a. »Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification«. In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852 (siehe S. 30).

[13] Tobias Holzmann. *Mathematics, Numerics, Derivations and OpenFOAM*. Holzmann CFD, 2021. URL: https://holzmann-cfd.de (siehe S. 19).

[14] Kurt Hornik, Maxwell Stinchcombe und Halbert White. »Multilayer feedforward networks are universal approximators«. In: *Neural Networks* 2.5 (1989), S. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208 (siehe S. 15).

[15] A. Imren und D.C. Haworth. »On the merits of extrapolation-based stiff ODE solvers for combustion CFD«. In: *Combustion and Flame* 174 (2016), S. 1–15. ISSN: 0010-2180. DOI: https://doi.org/10.1016/j.combustflame.2016.09.018. URL: https://www.sciencedirect.com/science/article/pii/S001021801630267X (siehe S. 19).

[16] Sergey Ioffe und Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG] (siehe S. 30).

[17] Wenzel Jakob, Jason Rhinelander und Dean Moldovan. *pybind11 – Seamless operability between C++11 and Python*. https://github.com/pybind/pybind11. 2017 (siehe S. 2, 22).

[18] Nitish Shirish Keskar u. a. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. arXiv: 1609.04836 (siehe S. 33).

[19] Diederik P. Kingma und Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (siehe S. 33).

[20] Alex Krizhevsky, Ilya Sutskever und Geoffrey Hinton. »ImageNet Classification with Deep Convolutional Neural Networks«. In: *Neural Information Processing Systems* 25 (Jan. 2012). DOI: 10.1145/3065386 (siehe S. 34).

[21] Raphael Langbauer u. a. »Modelling of thermal shrinkage of seamless steel pipes using artificial neural networks (ANN) focussing on the influence of the ANN architecture«. English. In: *Results in Engineering* 17 (März 2023). Publisher

Copyright: © 2023 The Authors. ISSN: 2590-1230. DOI: 10.1016/j.rineng.2023.100999 (siehe S. 1).

[22] Yann Lecun u. a. »Efficient BackProp«. In: *Neural Networks: tricks of the trade* (1998) (siehe S. 30).

[23] Lisha Li u. a. »Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization«. In: *Journal of Machine Learning Research* 18.185 (2018), S. 1–52. URL: http://jmlr.org/papers/v18/16-558.html (siehe S. 34).

[24] Tianfeng Lu und Chung K. Law. »Toward accommodating realistic fuel chemistry in large-scale computations«. In: *Progress in Energy and Combustion Science* 35.2 (2009), S. 192–215. ISSN: 0360-1285. DOI: https://doi.org/10.1016/j.pecs.2008.10.002. URL: https://www.sciencedirect.com/science/article/pii/S036012850800066X (siehe S. 1).

[25] B.F. Magnussen und B.H. Hjertager. »On mathematical modeling of turbulent combustion with special emphasis on soot formation and combustion«. In: *Symposium (International) on Combustion* 16.1 (1977), S. 719–729. ISSN: 0082-0784. DOI: https://doi.org/10.1016/S0082-0784(77)80366-4. URL: https://www.sciencedirect.com/science/article/pii/S0082078477803664 (siehe S. 8).

[26] Bjørn F. Magnussen. »THE EDDY DISSIPATION CONCEPT A BRIDGE BETWEEN SCIENCE AND TECHNOLOGY«. In: *ECCOMAS Thematic Conference on Computational Combustion*. 2005. URL: https://api.semanticscholar.org/CorpusID:67818474 (siehe S. 7–9, 19).

[27] Martín Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/ (siehe S. 2, 22).

[28] Markus Mayrhofer u. a. »CFD investigation of a vertical annealing furnace for stainless steel and non-ferrous alloys strips – A comparative study on air-staged MILD combustion«. In: *Thermal Science and Engineering Progress* 28 (2022), S. 101056. ISSN: 2451-9049. DOI: https://doi.org/10.1016/j.tsep.2021.101056. URL: https://www.sciencedirect.com/science/article/pii/S2451904921002171 (siehe S. 1).

[29] Reza Moradi, Reza Berangi und Behrouz Minaei. »A survey of regularization strategies for deep models«. In: *Artificial Intelligence Review* 53 (2020), S. 3947–3986. ISSN: 1573-7462. DOI: https://doi.org/10.1007/s10462-019-09784-7. URL: https://link.springer.com/article/10.1007/s10462-019-09784-7#citeas (siehe S. 16).

[30] Felix Mütter u. a. »Artificial intelligence for solid oxide fuel cells: Combining automated high accuracy artificial neural network model generation and genetic algorithm for time-efficient performance prediction and optimization«. English. In: *Energy Conversion and Management* 291 (Sep. 2023). ISSN: 0196-8904. DOI: 10.1016/j.enconman.2023.117263 (siehe S. 1).

[31] »Numerical simulation of a hydrogen fuelled gas turbine combustor«. In: *International Journal of Hydrogen Energy* 36.13 (2011). Hysydays, S. 7993–8002. ISSN: 0360-3199. DOI: https://doi.org/10.1016/j.ijhydene.2011.01.045. URL: https://www.sciencedirect.com/science/article/pii/S0360319911000875 (siehe S. 1).

[32] Tom O'Malley u. a. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019 (siehe S. 34).

[33] Rene Prieler u. a. »Machine learning techniques to predict the flame state, temperature and species concentrations in counter-flow diffusion flames operated with CH4/CO/H2-air mixtures«. In: *Fuel* 326 (2022), S. 124915. ISSN: 0016-2361. DOI: https://doi.org/10.1016/j.fuel.2022.124915. URL: https://www.sciencedirect.com/science/article/pii/S0016236122017574 (siehe S. 30).

[34] »Proposal and validation of a numerical framework for 3D-CFD in-cylinder simulations of hydrogen spark-ignition internal combustion engines«. In: *International Journal of Hydrogen Energy* 53 (2024), S. 114–130. ISSN: 0360-3199. DOI: https://doi.org/10.1016/j.ijhydene.2023.12.027. URL: https://www.sciencedirect.com/science/article/pii/S0360319923062663 (siehe S. 1).

[35] Mohammed I. Radaideh, Stelios Rigopoulos und Dimitris A. Goussis. »Characteristic time scale as optimal input in Machine Learning algorithms: Homogeneous autoignition«. In: *Energy and AI* 14 (2023), S. 100273. ISSN: 2666-5468. DOI: https://doi.org/10.1016/j.egyai.2023.100273. URL: https://www.sciencedirect.com/science/article/pii/S2666546823000459 (siehe S. 30).

[36] Sidney Radcliffe. *Reverse-mode automatic differentiation from scratch, in Python*. 2020. URL: https://sidsite.com/posts/autodiff/ (siehe S. 16).

[37] Maziar Raissi, Paris Perdikaris und George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017. arXiv: 1711.10561 [cs.AI] (siehe S. 17).

[38] *Sandia National Laboratories*. 2024. URL: https://www.sandia.gov/ (siehe S. 20).

[39] Gregory P. Smith u. a. *GRI-Mech 3.0*. http://www.me.berkeley.edu/gri_mech/ (siehe S. 4).

[40] M.D. Smooke und R.W. Bilger. *Reduced Kinetic Mechanisms and Asymptotic Approximations for Methane-air Flames: A Topical Volume*. Lecture notes in physics. Springer-Verlag, 1991. ISBN: 9783540542100. URL: https://books.google.at/books?id=2CDwAAAAMAAJ (siehe S. 4).

[41] *Softmax layer*. https://keras.io/api/layers/activation_layers/softmax/. Keras (siehe S. 54).

[42] Daniel Sonnet. *Neuronale Netze Kompakt. Vom Perceptron zum Deep Learning*. 1. Aufl. Springer Vieweg Wiesbaden, 2022. DOI: 10.1007/978-3-658-29081-8 (siehe S. 10, 12, 13).

[43] *stirred_reactor.ipynb*. 2024. URL: https://cantera.org/examples/jupyter/reactors/stirred_reactor.ipynb.html (siehe S. 5).

[44] TensorFlow. *Introduction to gradients and automatic differentiation*. 2023. URL: https://www.tensorflow.org/guide/autodiff (siehe S. 15).

[45] Inc. The MathWorks. *Überanpassung*. 2024. URL: https://de.mathworks.com/discovery/overfitting.html (siehe S. 17).

[46] *TNF Workshop*. 2024. URL: https://tnfworkshop.org/ (siehe S. 20).

[47] Serge Zakharian, Patricia Ladewig-Riebler und Stefan Thoer. »Regelungstechnische Anwendungen von KNN«. In: *Neuronale Netze für Ingenieure: Arbeits- und Übungsbuch für regelungstechnische Anwendungen*. Wiesbaden: Vieweg+Teubner Verlag, 1998, S. 107–144. ISBN: 978-3-663-07675-9. DOI: 10.1007/978-3-663-07675-9_4. URL: https://doi.org/10.1007/978-3-663-07675-9_4 (siehe S. 1).