Kurzfassung

Während sich die numerische Strömungssimulation durch *Computational Fluid Dynamics (CFD)* als mächtiges Werkzeug in der Flugzeugentwicklung erwiesen hat, ist diese für die Evaluierung aller notwendigen Strömungsbedingungen zu rechenaufwendig. Datengetriebene aerodynamische Ersatzmodelle, welche auf vollständigen CFD Lösungen basieren, können Vorhersagen für Strömungsbedingungen treffen, für welche keine CFD Berechnungen vorliegen. Die SMARTy Toolbox implementiert solche Erstzmodelle welche auf einer Dimensionsreduzierung der Trainingsdaten durch POD mit anschließender Interpolation auf dem dadurch erzeugen Unterraum basieren. Dabei kann es allerdings zu abweichenden Vorhersagen unter nichtlinearen, transsonischen Strömungsbedingungen kommen. In dieser Arbeit wurden Autoencodernetzwerke evaluiert, welche die POD Methode ersetzen und eine nichtlineare Dimensionsreduzierung ermöglichen. Diese wurden erneut mit TPS Interpolation kombiniert und konnten für die getesteten Oberflächendruckverteilungen auf einem zweidimensionalen Flügelprofil genauere Vorhersageergebnisse liefern. Dabei erwiesen sich vor allem Convolutional Autoencoder als geeignet. In allen Modellen konnten die, gegenüber POD genaueren Vorhersagen, durch eine für die Interpolation geeignete Einbettung im Unterraum erreicht werden. Während die Methode gegenüber POD zwar über deutlich höhere Offlinekosten verfügt, sind die Onlinekosten für eine Strömungsvorhersage vergleichbar geblieben.

Abstract

While numerical fluid simulation (CFD) proved to be a powerful tool in aircraft design, it is far to expensive to compute for every configuration that needs to be considered. Datadriven surrogate models, based on full order CFD solutions, can make predictions for flow conditions which are not present in the underlying CFD solutions. The SMARTy Toolbox implements surrogate models based on a dimensionality reduction of the training data using the POD Method. The created subspace can further be used to interpolate between the samples. The predictions generated by this sometimes differ from the original, especially in highly nonlinear transsonic flows. This work evaluates the use of Autoencodernetworks, which replace the dimensionality reduction through POD and make a nonlinear reduction possible. The method was combined with TPS interpolation and could offer more precise predictions on a two dimensional wingprofile. Especially convolutional autoencoders were found suitable for the task. All models could improve the prediction over POD by finding a better embedding in the subspace that is used for interpolations. While the method has higher offline costs compared to POD, the onlinecosts for a prediction remained comparable.

Entwicklung eines mit Hilfe von CFD Simulationen trainierten Autoencoders zwecks schneller Vorhersage aerodyamischer Daten für Flügelprofile

Bachelorarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik an der
Dualen Hochschule Baden-Württemberg Mannheim

von

Nils Hoffmann

Bearbeitungszeitraum:	24. Juni 2019 - 15. September 2019
Matrikelnummer, Kurs:	2512439, TINF16ITIN
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e. V.
Institut:	Institut für Aerodynamik und Strömungstechnik
Abteilung.	Abteilung: $C^2A^2S^2E$
Betreuer:	Stefan Görtz

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Braunschweig, der 16. September 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Numerische Strömungssimulation (im folgenden kurz *CFD, Computational Fluid Dynamics*) hat sich als mächtiges Werkzeug im Bereich des Flugzeugentwurfs erwiesen. Ziel von CFD ist die Berechnung von sogenannten Strömungsgrößen, dabei handelt es sich zum Beispiel um vektorwertige Größen wie den dynamischen Luftdruck im Raum um das Testobjekt. Ziel sind aber auch daraus abgeleitete Größen, etwa Auftrieb oder Luftwiderstand, welche die Performance des Flugzeugs kennzeichen. Diese Rechnungen werden für festgelegte Ausgangsbedingungen angefertigt - dabei handelt es sich um Parameter wie die Anströmgeschwindigkeit oder den Anstellwinkel (im folgenden auch α) des Flügelprofils zur Anströmrichtung. Die Berechnung der Strömungsgrößen basiert auf Lösungen des sogenannten *full order models*, welches das physikalische Verhalten durch partielle Differentialgleichungen abbildet, für das Lösen dieses Modells wurde in der DLR Abteilung $C^2A^2S^2E$ der numerische Strömungslöser *TAU* entwickelt [37]. Eine solche Lösung des Full Order Modells geschieht für eine Konfiguration an Ausgangsbedingungen, neben den bereits erwähnten Parametern der Anströmung auch etwa das geometrische Modell des Flugzeugs oder Flügels. Diese Lösung wird im folgenden als *Snapshot* bezeichnet, ein solcher Snapshot bildet also eine Reihe an Parametern auf eine Reihe an Strömungsgrößen, etwa Druckverteilung, ab.

1.1 Motivation der Ersatzmodellierung

Während CFD Simulation zwar im Vergleich zu physischen Windkanalversuchen deutlich schnelleren und günstigeren Design Prozess ermöglicht, handelt es sich immer noch um extrem zeitaufwendige Berechnungen, welche bei hinreichender Genauigkeit Wochen an Berechnungszeit kosten können. Problematisch ist dabei, dass im Designprozess eines Flugzeugs eine große Zahl an Konfigurationen evaluiert werden müssen, für welche jeweils eine neue CFD Rechnung notwendig ist. Diese Konfigurationen ergeben sich durch die Kombination verschiedener Szenarien des Flugzeugbetriebs, dazu gehört die Simulation verschiedener Massenkonfigurationen, verschiedener Flugbedingungen wie Start oder Landung oder auch instationäre Vorgänge wie Manöversimulationen. Zusätzlich sind Optimierungsmethoden von Interesse, bei welchen etwa die Flügelform verändert wird um einen möglichst geringen Luftwiderstand zu erreichen. Mit der Anzahl an Szenarien, welche im Designprozess evaluiert werden sollen, steigt auch exponentiell die Anzahl an notwendigen CFD Rechnungen, welche für die Simulation dieser geänderten Ausgangsbedingungen notwendig sind. Dabei ergibt sich eine Zahl im Bereich von $\mathcal{O}(10^6)$ Rechungen, welche nicht realistisch machbar sind, ohne

eine Geschwindigkeitssteigerung der Rechungen um mehrere Größenordnungen zu erreichen. Zu diesem Zweck beschäftigt sich die Abteilung $C^2A^2S^2E$ unter anderem mit datenbasierten Ersatzmodellen für CFD Rechnungen. Ziel dieser ist es Strömungslösungen in verminderter, aber ausreichender Genauigkeit abzubilden, diese aber mit stark reduziertem Rechenaufwand zu liefern. Die Modelle basieren dabei selber auf CFD Rechnungen, mit welchen die Modelle in einem sogenannten *Offline* Schritt trainiert werden. Im Designprozess können diese dann genutzt werden, um Aussagen über Parameterkombinationen zu treffen, für welche zuvor keine CFD Rechnungen angefertigt wurden. Tools für die Ersatzmodellierung sind in der DLR SMARTy (Surrogate Modeling for AeRo data Toolbox in Python) Toolbox implementiert. [23]

1.2 Einordnung

Diese Arbeit beschäftigt sich mit der Nutzung von Modellen reduzierter Ordnung, kurz *ROM (Reduced Order Model)*, zur Ersatzmodellierung. Für die Bildung der Modelle werden die hochdimensionalen CFD Trainingsdaten zunächst durch eine Dimensionsreduzierungsmethode in einen niedrigdimensionalen Unterraum überführt. Dabei können Informationen verloren gehen. Ziel des Unterraums ist es, mit möglichst wenig Dimensionen die Snapshots möglichst genau wiederzugeben.

In der Ersatzmodellierung wird der Unterraum nun genutzt, um auf diesem mit verringertem Rechenaufwand Interpolationen durchzuführen. Der interpolierte Unterraumvektor korrespondiert durch die Interpolationsmethode mit der gewünschten Parameterkombination. Daraufhin wird aus dem niedrigdimensionalen, durch die Interpolationsmethode produzierten, Punkt die hochdimensionale Lösung rekonstruiert. Bislang wurde für die Dimensionsreduzierung in SMARTy vor allem die POD (*Proper Orthogonal Decomposition*) Methode, auch Hauptkomponentenanalyse, verwendet, welche eine Singulärwertzerlegung nutzt, um den Datensatz in eine Reihe an Orthogonalvektoren zu zerlegen. Dabei handelt es sich um eine lineare Dimensionsreduzierung, welche sich durch eine Reihe an Matrixmultiplikation beschreiben lässt. Die Einbettung der Snapshots in den Unterraum ist demnach ebenfalls eine lineare Kombination der Snapshots selber. Dies ist problematisch, da das Strömungsverhalten, vor allem im transsonischen Bereich, sich stark nichtlinear im Bezug auf dessen Ausgangsparameter (etwa Machzahl der Anströmung, Anstellwinkel α) verhält. Die resultierende Einbettung im Unterraum der Hauptkomponentenanalyse bleibt somit zwangsläufig ebenfalls nichtlinear in Bezug auf diese Parameter. Dies macht eine Interpolation über die Ausgangsparameter in diesem Raum schwierig, da hier die Interpolationsmethode diese Nichtlinearitäten des Strömungsverhaltens bezüglich Ausgangsparametern abbilden muss. Die in SMARTy

implementierte und in Kombination mit POD genutzte Thin Plate Spline (TPS) Interpolationsmethode hat oft Schwierigkeiten im transsonischen Bereich akkurate Vorraussagen zu liefern, was sich durch die Einschränkungen der Interpolation im durch POD erzeugten Unterraum erklären lässt.

In dieser Arbeit wurde deshalb die Performance von Autoencodernetzwerken zur Ersatzmodellierung evaluiert. Diese ersetzen dabei die Singulärwertzerlegung als Methode der Dimensionsreduzierung. Dabei kommt erneut die selbe Interpolationsmethode im Unterraum zum Einsatz. Der Vorteil des Autoencodernetzwerks gegenüber der linearen Hauptkomponentenanalyse ist dessen Möglichkeit einen nichtlinearen Ansatz zu bieten, die Daten in den Unterraum einzubetten.

1.3 Umfeld

In den letzten Jahren haben Neuronale Netzwerke vielfach ihre Möglichkeit demonstriert komplexe nichtlineare Probleme zu modellieren. Diese Fähigkeiten sind auch für die Ersatzmodellierung im Bereich der Aerodynamik interessant. Zudem existieren für die Nutzung von Neuronalen Netzwerken eine Vielzahl an hochperformanten Opensource Bibliotheken, welche Definition, Training und Vorhersage dieser Netze ermöglichen. Dazu gehört auch die in dieser Arbeit verwendete Tensorflow Library [30], welche für Training und Vorhersage der Dimensionsreduzierungsmethode verwendet wird. Für die Verwendung von Autoencodernetzwerken wurden bereits erste Versuche am DLR unter eigener Implementierung der Berechnungen für Neuronale Netzwerke unternommen. Dabei hatten diese zwar konkurenzfähige Vorhersageergebnisse geliefert, konnten allerdings keine akzeptable Geschwindigkeit erreichen, weshalb das Thema auch aufgrund mangelnder Skalierbarkeit auf industrielle Problemgrößen zunächst nicht weiter verfolgt wurde.

1.4 Vorgehen

Im folgenden, theoretischen Teil dieser Arbeit wird zunächst zur Einordnung auf die Grundlagen des maschinellen Lernens eingegangen, auf welchem datengetriebene Modelle basieren. Daraufhin wird die Funktionsweise der hier eingesetzten neuronalen Netzwerke im allgemeinen beschrieben wobei deren Aufbau und Trainingsprozess dargestellt werden. Dabei wird auch die Unterkategorie der *Convolutional Neural Networks (CNNs)* behandelt, welche aufgrund ihres Aufbaus in vielen Bereichen gute Ergebnisse erzielen konnten und auch in dieser Arbeit eingesetzt wurden. Der letzte Abschnitt des Theorieteils behandelt die hier verwendeten

Autoencodernetzwerke und wie diese zur Dimensionsreduzierung und als generative Modelle eingesetzt werden können. Dabei wird auch auf den Einsatz der vorgestellten Convolutional Neural Networks in Autoencodernetzwerken eingegangen, ein Aufbau welcher in dieser Arbeit verwendet wurde. Da diese Modelle mit der bereits vorhandenen TPS Interpolation kombiniert wurden und mit einer Dimensionsreduzierung durch POD verglichen werden, wird die Funktionsweise dieser im Kapitel 3 hergeleitet.

Im praktischen Teil der Arbeit wurden drei verschiedene Autoencodermodelle zur Dimensionsreduzierung implementiert und für die Strömungsvorhersage getestet. Dabei werden jeweils die Rekonstruktions- und Vorhersageergebnisse eines flaschen, tiefen und Convolutional Autoencoders betrachtet. Alle drei Modelle wurden unter Verwendung der Tensorflow Library [30] implementiert, welche über dessen Keras API angesteuert wurde. Die API erlaubt eine Definition der verwendeten Netzwerke auf Basis der im Theorieteil beschrieben Ebenen und Aktivierungsfunktionen. Die Netzwerke wurden darauf trainiert, vor der Rekonstruktion die einzelnen Snapshots auf einen niedrigdimensionalen Unterraum zu reduzieren. Zwischen diesen Unterraumvektoren wurde mit der vorgestellten *Thin Plate Spline* Methode interpoliert, um Vektoren für neue Kombinationen von Ausgangsparametern zu erzeugen. Diese werden durch den Decoderteil des Netzwerks rekonstruiert, um neue Snapshots vorherzusagen. Die Vorhersagen durch die Kombination von TPS und Autoencoder wurde mit dem Orignal des Full Order Models und der in SMARTy bereits vorhandenen Kombination aus TPS und POD verglichen.

Für Training und Vergleich der Vorhersage kamen zwei verschiedene Testdatensätze zum Einsatz. Es handelte sich bei beiden um die Oberflächendrücke auf einem zweidimensionalen NACA64A010 Profil, für welches Snapshots mit dem TAU Strömungslöser berechnet wurden. Für beide Datensets wurden die Ausgangsparameter über verschiedene Anstellwinkel (α) und Anströmggeschwindigkeiten (Machzahlen) gesampelt. Der erste Datensatz besteht aus 30 Snapshots, gesamplet im transsonischen Bereich mit stark varriierenden Anstellwinkeln. Davon wurde 26 Snapshots für das Training des Autoencoders und des Interpolationsmodells verwendet, die übrigen vier wurden zum Vergleich der Vorhersagegenauigkeit genutzt. Der zweite Datensatz ist mit 250 Snapshots deutlich größer und deckt einen größeren Machbereich ab, hier ist subsonisches und transsonisches Strömungsverhalten vorhanden.

Abschließend wird exemplarisch für ein vereinfachtes Autoencodernetzwerk dessen Einbettung des ersten Testdatensatzes in einen zweidimensionalen Unterraum betrachtet. Dabei wird vor allem auf die Eignung dieser für den Interpolationsvorgang eingegangen. Auch in diesem Kapitel dient erneut die Einbettung durch die POD Methode als Vergleich zum Autoencodernetzwerk.

2 Theoretischer Hintergrund

Datenbasierte Modelle unterscheiden sich von herkömmlichen dadurch, dass diese in ihren Einzelheiten nicht manuell aufgebaut werden, sondern ihre Eigenschaften direkt aus bereits vorhandenen Daten ableiten. Die Eigenschaften herkömmlicher Modelle ergeben sich durch die manuelle Definition dieser, etwa basierend auf physikalischen Gegebenheiten. Datenbasierte Modelle im Gegensatz dazu lernen aus einer Reihe an Trainingsdaten, welche dem Modell als Beispiel dienen. Aus diesen kann das Modell idealerweise allgemeiner gültige Zusammenhänge ableiten. Dieser Vorgang der Modellbildung wird als Maschinelles Lernen bezeichnet.

2.1 Maschinelles Lernen

Maschinelles Lernen ist ein Begriff für Algorithmen, welche in der Lage sind aus Daten zu lernen. Mitchel definiert den Begriff wie folgt:

'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.' [31, Kapitel 1.1]

Maschinelles Lernen wird meist für Probleme genutzt, welche sich nicht durch herkömmliche, von Menschen definierte, Algorithmen und Modelle lösen lässt. Für diese Probleme T lassen sich lernende Algorithmen einsetzen, welche durch bereits bekannte Daten, *experience* E, ihre Genauigkeit, gemessen durch eine Metrik P, verbessern.

Dieser Bereich des maschinellen Lernens lässt sich grob in zwei Bereiche unterteilen: beaufsichtigtes und unbeaufsichtigtes Lernen.

2.1.1 Unbeaufsichtigtes Lernen

Ziel des unbeaufsichtigtem Lernen ist es in vorhandenen, unbekannten Daten 'interessante' Muster zu finden. Dabei sind keine bereits bekannten Beispiele für Eingabe-Ausgabe Paare des Modells notwendig. Stetig zunehmende Mengen an Daten, für welche keine Beschreibungen verfügbar sind und die mit deren Erstellung verbundenen Kosten macht das Feld des unbeaufsichtigten Lernens zunehmend interessant.

Ein Beispiel für Techniken um aus unbeschrifteten Daten Eigenschaften zu extrahieren ist etwa die Einteilung dieser in Gruppen, auch *Clustering* genannt. Eine solche Clusteringmethode (*k-Means*) ist in Abbildung 1 dargestellt. Eine andere Anwendung für unbeaufsichtigtes Lernen ist Dimensionsreduzierung, bei welcher hochdimensionale Daten auf eine geringere

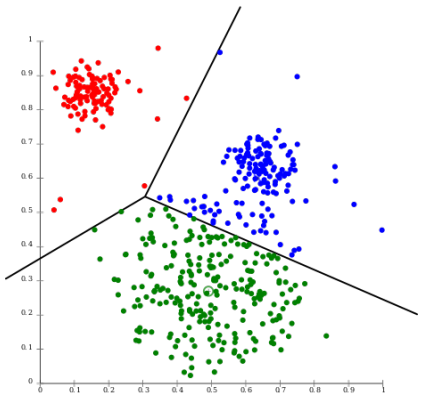


Abbildung 1: K-Means Clustering als ein Beispiel für eine Methode Daten in Gruppen einzuteilen. In diesem Fall wurden zweidimensionale Datenpunkte in drei Cluster geteilt. Quelle: [9]

Anzahl an Dimensionen reduziert werden, wobei möglichst viel Informationen und Struktur der Daten erhalten bleiben sollen. Eine Technik zur Dimensionsreduzierung ist etwa die Hauptkomponentenanalyse (*auch Proper Orthogonal Decomposition (POD)*), bei welcher Daten in lineare Hauptkompopnenten reduziert werden. Die Methode ordnet dieser nach absteigendem Wichtigkeit für die Reproduktion des Datensatzes, die Beschränkung der Daten auf die ersten n Hauptkomponenten eignet sich also für die Abbildung dieser in einem n dimensionalen Unterraum. [32, Kapitel 1.3.2]

Die in dieser Arbeit betrachteten Autoencoder Netzwerke sind ebenfalls ein Beispiel für eine solche Methode zur Dimensionsreduzierung.

2.1.2 Beaufsichtigtes Lernen

Ziel des beaufsichtigten Lernens ist es eine Funktion oder Abbildung der Eingabe x auf eine Ausgabe y zu bilden. Im einfachen Fall handelt es sich der Eingabe x um einen n Elementigen Vektor aus reellen Zahlen, welcher eine Reihe an Features kennzeichnet, etwa eine Serie an Messwerten. Eingaben mit komplexeren Zusammenhängen, etwa Bilder, Text, Audiodaten oder Beschreibungen eines Graphen werden aber ebenso verwendet.

Dies geschieht durch Lernen von Beispielen, dabei handelt es sich um Wertepaare bei welchen die gewünschte für Ein- und Ausgabe des Modells bekannt ist. Diese Datenpunkte werden deshalb Trainingsdaten genannt. Beim Training werden die Parameter des Modells

so lange angepasst, bis es die bekannten Trainingsdaten mit der gewünschten Genauigkeit abbilden kann. Ziel ist es, dass das Modell aus den vorhandenen Beispielen allgemeiner gültige Zusammenhänge erkennt und diese auf unbekannte Eingabedaten anwenden kann.

Die Ausgabe des Modells vor allem hängt von dem gewünschten Anwendungsfall ab. Im Folgenden sind die zwei hauptsächlichen Kategorien vorgestellt, in welche bewachtes Lernen unterteilt wird.

Klassifikation, bei welcher ein Eingabewert einer von k Kategorien zugeordnet wird. Das Modell lernt dabei eine Funktion der Form $f : \mathbb{R}^n \to \{1, ..., k\}$. Die Art der Encodierung der Kategorie kann unterschiedlich sein, möglich ist etwa eine direkte numerische Codierung oder eine Warscheinlichkeitsverteilung über mehrere Kategorien. Klassifikationsaufgaben sind in der Objekterkennung häufig, dabei entspricht Eingabe x dann z.B. einem Bild, dargestellt durch einen Vektor an Helligkeitswerden und die Ausgabe y des Modells eine Kategoriezuordnung des im Bild zu erkennenden Objekts. [32, Kapitel 1.2.1], [21]

Regression, bei welcher der Eingabewert auf einen kontinuierlichen Ausgabewert abgebildet wird, Das Modell lernt also nun eine Funktion der Art $f : \mathbb{R}^n \to \mathbb{R}^m$. Regression wird genutzt wenn Aussage über den Einfluss von Einflussgrößen auf eine Zielgröße getroffen werden soll. Einfaches Beispiel für eine Regressionsmethode ist lineare Regression bei welcher eine Grade durch ein vorhandenes Datenset gelegt wird. Diese, gekennzeichnet durch Steigung und Startpunkt bildet nun ein lineares Modell mit welches genutzt werden kann um Ausgabewerte y für Eingabewerte x, welche nicht im Trainingsdatensatz vorhanden waren, vorauszusagen.

Ein Vergleich dieser Kategorien ist in Abbildung 2 dargestellt. Das Feld des maschinellen Lernens beschränkt sich allerdings nicht nur auf diese Einteilungen, in der praktischen Anwendug wird es auch für komplexere Aufgaben genutzt, etwa für automatische Übersetzung, Annotation von Strukturen oder Synthese von Bildern. [12, Kapitel 20.11]

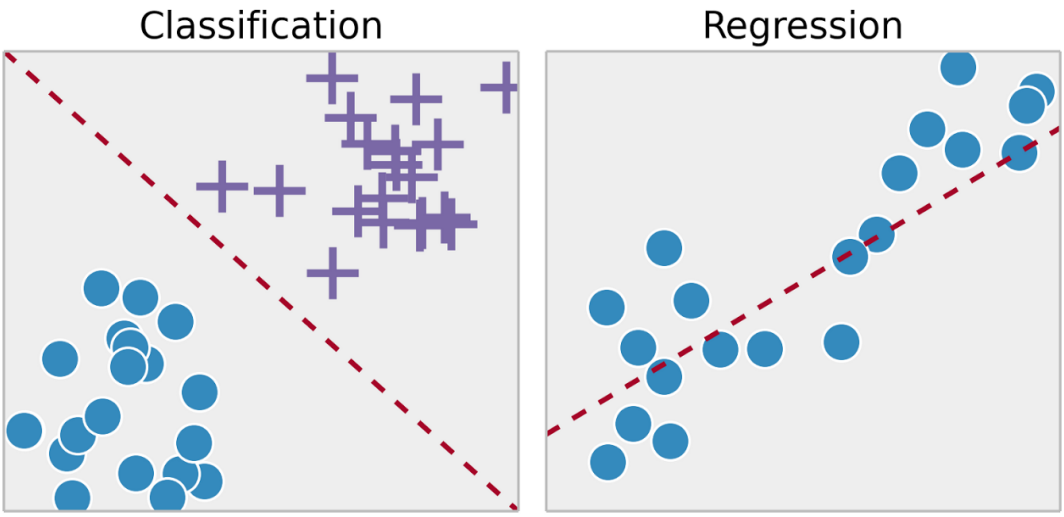


Abbildung 2: Vergleich zwischen Klassifikations und Regressionsaufgaben: Bei der Klassifikation ist es Ziel Kategorieren innerhalb eines Datensatzes voneinander abzugrenzen, bei der Regression geht es darum Zusammenhänge innerhalb eines Datensatzes zu definieren. Quelle: [39]

2.2 Neuronale Netzwerke

(Künstliche) Neuronale Netzwerke sind in ihrer Struktur inspiriert von biologischen Neuronen, woher sie ihren Namen erhalten haben. Moderne Forschung im Bereich Neuronaler Netzwerke richtet sich allerdings vor allem nach mathematischen und statistischen Erkenntnissen und sind nur loose mit Forschung der Neurobiologie verwandt. [12, Kapitel 6]

Ein Künstliches Neuronales Netzwerk besteht dabei aus einzelnen Ebenen von künstlichen Neuronen, welche miteinander verknüpft sind. Jedem Neuron ist ein Wert zugeordnet, welcher im folgenden als Aktivierung des Neurons bezeichnet wird. Diese Aktivierung ergibt sich dabei aus durch Aufsummierung der Werte aller externen Eingaben in das Neuron. Dabei wird jeder Eingabe zusätzlich ein Gewicht zugeordnet, mit welcher diese vor Aufsummierung multipliziert wird. Zusätzlich verfügen Neuronen noch über einen einzelnen Offset Wert b, welcher auf die gewichtete Summe aufaddiert wird. Der Gesamtwert dient als Eingabe in die dem Neuron zugeordnete Aktivierungsfunktion f. Für die Aktivierung out des Neurons ergibt sich also folgende Gleichung:

$$out = f(x^T w + b) \tag{2.1}$$

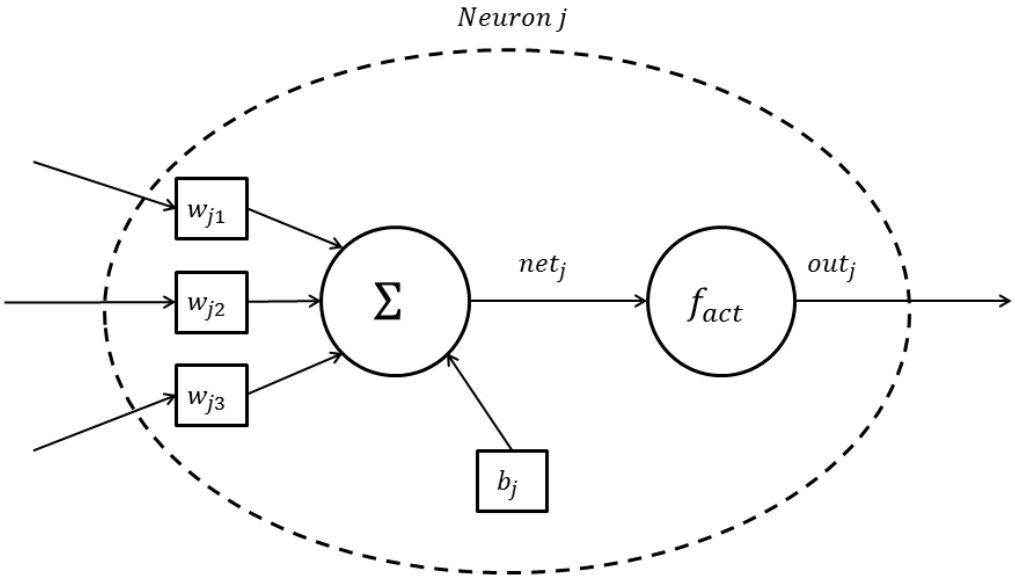


Abbildung 3: Darstellung eines künstlichen Neurons. Die Verbindungen zur vorherigen Ebene werden mit Gewichten multipliziert und zusammen mit einem Offsetwert b aufsummiert. Auf die Summe wird eine Aktivierungsfunktion f_{act} angewendet.

Dabei ist x^T der transponierte Eingabevektor und w der Vektor mit Gewichten, welcher beschreibt wie stark eine Eingabe den Ausgabewert beeinflusst. Eine schematische Darstellung eines solchen Neurons ist in Abbildung 3 dargestellt. Sowohl der Vektor der Gewichte w als auch der Offset Wert b wird dabei während des Trainings angepasst. Ein Neuron stellt also eine Funktion der Form $f(x, w, b) : \mathbb{R}^n, \mathbb{R}^n, \mathbb{R} \to \mathbb{R}$ dar, welche Eingabevektor, Gewichtvektor und Offset Wert auf einen Ausgabewert abbildet.

Ein Neuronales Netzwerk entsteht durch die Verknüpfung von mehreren dieser Neuronen. Das Netz lässt sich durch einen gerichteten Graphen beschreiben dessen Knoten Neuronen mit ihren Gewichten, Offsetwerten und Aktivierungsfunktionen sind. Die Kanten des Graphen beschreiben die Aktivierungswerte welche an die verbundenden Neuronen weitergegeben werden.

Die Netzwerke sind dabei meist in Ebenen der Breite m angeordnet wobei jedes Neuron der

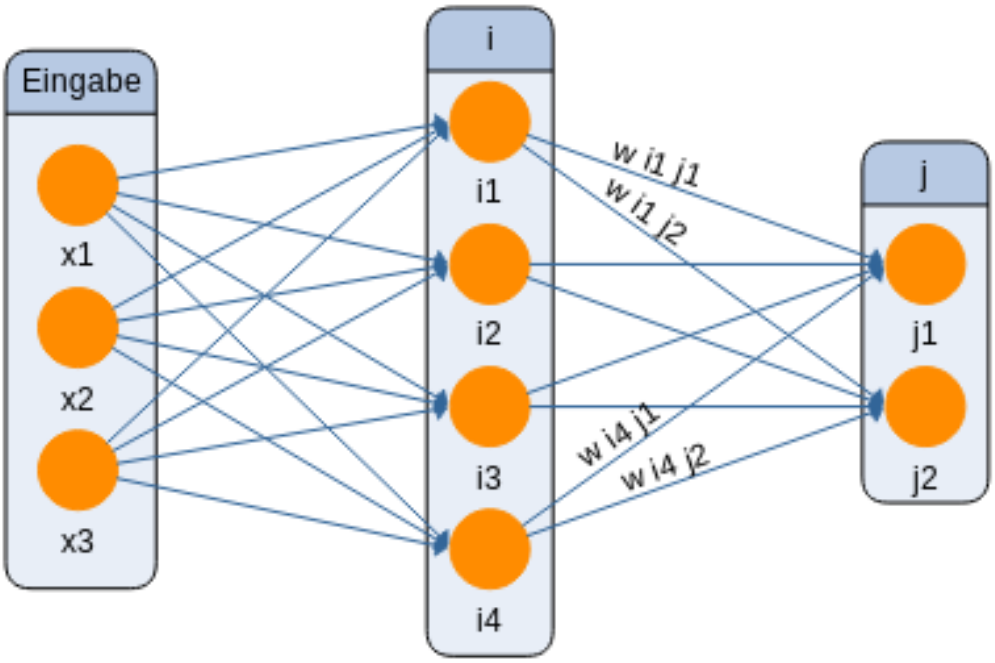


Abbildung 4: Beispiel für ein *Feedforward* Netzwerk mit dreielementigem Eingabevektor und zweielementiger Ausgabe. Das Netzwerk verfügt über eine versteckte Ebene i der Breite vier. Exemplarisch dargestellt sind vier Werte der Gewichtsmatrix W_{ij}. Die letzte Ebene j dient als Ausgabe des dargestellten Modells.

Ebene $i + 1$ mit allen Neuronen der Ebene i verknüpft ist. Dieser Aufbau wird als *vollständig verbunden* oder *dense* bezeichnet. Eine Darstellung eines solchen ist in Abbildung 4 gezeigt.

Jede Ebene des Netzwerks kann dabei als vektorielle Funktion $f_i : \mathbb{R}^n \to \mathbb{R}^m$ angesehen werden, wobei m die Breite der Ebene und n die der vorherigen ist. Eingabe der Funktion ist der Vektor x aller Aktivierungen der vorangehenden Ebene mit der Länge n. Statt des Gewichtungsvektors w des einzelnen Neurons verfügt die Ebene j über die Gewichtsmatrix W_j welche die Verbindung mit der Vorgängerebene i beschreibt und spaltenweise aus den Vektoren w der einzelnen Neuronen besteht. Die Gewichtsmatrix der Ebene j des in Abbildung

4 dargestellten Netzwerks sähe also wie folgt aus:

$$W_{ij} = \begin{bmatrix} w_{i1,j1} & w_{i1,j2} & \dots \\ w_{i2,j1} & w_{i2,j2} & \dots \\ w_{i3,j1} & w_{i3,j2} & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

Selbes gilt für die Offset Werte welche als Vektor b der Länge m vorliegen. Damit ergibt sich für jede Ebene f_i die Funktion

$$f_i(x) = f_{akt}(xW + b)$$

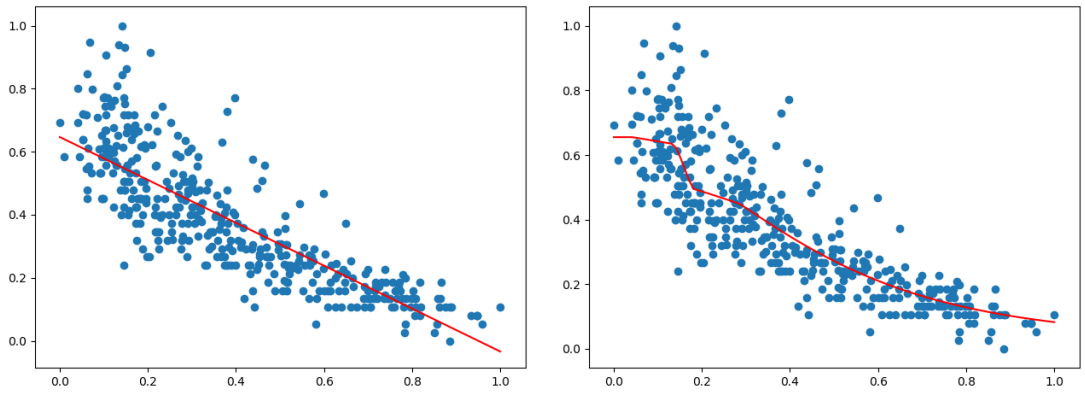
wobei f_{akt} die Aktivierungsfunktion der Neuronen ist, welche elementweise auf den Vektor $xW + b$ angewendet wird.

Das gesamte Netzwerk lässt sich also durch Komposition der Funktionen aller Ebenen beschreiben. Bei dem in Abbildung 4 dargestellten Netzwerk entspricht dies $y = f_j(f_i(x))$ oder allgemein $y = f_1 \circ f_2 \circ \dots \circ f_n(x)$.

Da die erste Ebene des Netzwerks keine Verbindungen zum vorherigen hat, sind diesem auch keinerlei trainierbare Gewichte zugeordnet. Stattdessen dient die Ebene lediglich als Eingabe für Daten in das Netzwerk. Dies ist auch Abbildung 4 dargestellt, in diesem Fall besteht die Eingabe aus dem dreielementigen Vektor x. Für diese existiert auch keine Offset Werte oder Aktivierungsfunktionen. Stattdessen wird der Wert jedes Eingabeneurons auf das korrespondierende Element des Vektors der Eingabedaten gesetzt.

Von der Eingabe an laufen Aktivierungen zur folgenden Ebene, in Abbildung 4 also zur vier Elemente breiten Ebene i. Diese wird als *hidden layer* bezeichet, da die Aktivierungen dieser weder Eingabe noch Ausgabe des Modells darstellen und somit nicht direkt sichtbar sind. Modelle mit mindestens einer versteckten Ebene werden auch als *Multi Layer Perceptron*, kurz *MLP* bezeichnet. Mit der Anzahl von versteckten Ebenen steigt auch die Fähigkeit des Netzwerks komplizierte Probleme abzubilden, befinden sich mehrere Ebenen zwischen Ein- und Ausgabe, bei also mindestens zwei *hidden layers* spricht man von *deep neural networks*, also tiefen Netzwerken. Besteht das Modell aus zu wenigen Ebenen, werden für die Modellierung derselben Funktion bis zu exponentiell mehr Elemente in den einzelnen Ebenen benötigt [2].

Da sich Aktivierungen im Netzwerk lediglich nach vorne bewegen, wird dieser Vorgang entsprechend als *Feedforward* Schritt bezeichnet und das dazugehörige Modell als *Feedforward* Netzwerk.



(a) **(b)**

Abbildung 5: Regression (rot) eines Datensatzes (blau) durch ein Netzwerk bestehend aus ausschließlich linearen Aktivierungsfunktion (a) und durch ein Netzwerk mit nichtlinearen Funktionen (b).

2.2.1 Aktivierungsfunktionen

Aktivierungsfunktionen sind wesentliche Bestandteil von Neuronalen Netzwerken. Letztendlich bestimmen diese wie Eingaben auf die Aktivierungen von Neuronen übertragen werden und welchen Randbedingungen diese unterliegt, etwa inwiefern diese begrenzet ist. Die Fähigkeit von Neuronalen Netzwerken beliebige Funktionen abzubilden [14] entsteht dabei durch die Nutzung von nichtlinearen Aktivierungsfunktionen - werden lediglich lineare Aktivierungsfunktionen genutzt ist das gesamte Model letztendlich nur eine lineare Kombination der Eingabe. Damit unterscheidet sich ein Netzwerk mit vielen Ebenen was in jeder eine lineare Aktivierungsfunktion verwendet nicht von einer einzelnen Ebene. Ein Beispiel hierfür ist im Bild 5a dargestellt. Regression desselben Datensatzes durch ein Netzwerk mit nichtlinearen Funktionen zum Vergleich ist in Abbildung 5b gezeigt. Diese Linearität des Modells tritt lediglich auf wenn keine einzige Aktivierungsfunktion eine Nichtlinearität aufweist. Die Nutzung einzelner Ebenen mit linearen Aktivierungsfunktionen hingegen ist nicht ungewöhnlich. Im Folgenden werden einige häufig verwendete Aktivierungsfunktionen vorgestellt, dies ist jedoch keinesfalls eine vollständige Darstellung aller in neuronalen Netzwerken eingesetzten Aktivierungsfunktionen.

Ein Beispiel ist etwa die **Sigmoid Funktion**, eine logistische Funktion. Diese ist definiert durch:

$$f_{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Diese bildet reale Eingaben auf ein Interval $[0, 1]$ ab, dargestellt in Abbildung 6a. Die begrenzte

Ausgabe verhindert das Auftreten stark wachsender Aktivierungswerte im Netzwerk. Zudem hat die Funktion im Bereich um $x = 0$ hohe Gradienten, wodurch kleine Wertänderungen der Eingabe die Ausgabe schnell Richtung 0 oder 1 wandern lassen. Mit steigender Gewichtung der Eingabe nimmt der Gradient weiter zu, wobei die Funktion immer mehr einer Heavyside-Funktion (auch Sprungfunktion) ähnelt. Somit modelliert die Funktion näherungsweise binäre Aktivierungen, weshalb sich die Sigmoid gut für Klassifizierungsaufgaben eignet, wo die Ausgabe die Zuordnung der Eingabe in eine Kategorie kennzeichnet.

Der Sigmoid Funktion sehr ähnlich ist der oft verwendete **Tangens Hyperbolikus (tanh)**. Dieser ist definiert durch und dargestellt in Abbildung 6b :

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

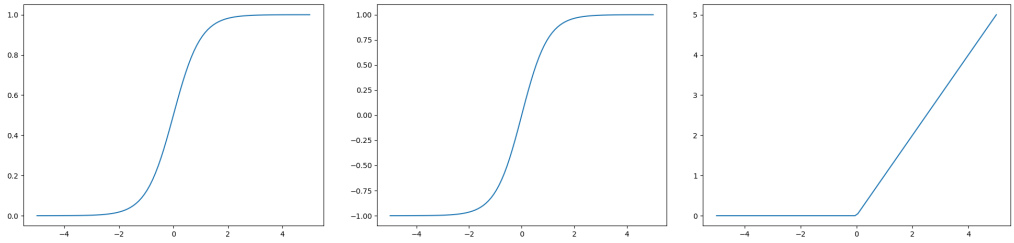
Die Funktion ist begrenzt im Bereich $[-1, 1]$ und weist ansonsten Eigenschaften sehr ähnlich der Sigmoid Funktion auf. Tatsäch ist tanh äquivalent einer skalierten und verschobenen Sigmoid Funktion:

$$\begin{aligned} tanh(x) &= \frac{1 - e^{-2x} + 1 + e^{-2x}}{1 + e^{-2x}} - 1 \\ &= \frac{2}{1 + e^{-2x}} - 1 \\ &= 2\sigma(x) - 1 \end{aligned}$$

Deutlich von den bisherigen Beispielen unterscheidet sich die **Rectified Linear** Aktivierungsfunktion, auch als *ReLU (Rectified Linear Unit)* bezeichnet. Diese ist dargestellt in Abbildung 6c und definiert durch:

$$f_{relu}(x) = max(0, x) = \begin{cases} 0 & : x \leq 0 \\ x & : x > 0 \end{cases}$$

Die Funktion setzt sich also aus zwei linearen Bereichen zusammen, welche zusammen eine nichtlineare Funktion ergeben. Anders als Sigmoid und tanh ist diese in der Ausgabe im positiven Bereich nicht begrenzt. Dies wirkt sich vorteilhaft auf den Trainingsprozess von vielen Ebenen tiefen Netzwerken (*deep neural networks*) aus, mehr dazu in 2.2.4. Nachteilhaft ist, dass durch die nach oben unbegrenzte Ausgabe sehr hohe Aktivierungswerte im Netzwerk auftreten können. Ist dies Problematisch kann dem mit zusätzlichen Regularisierungstermen in der Kostenfunktion entgegengewirkt werden [7]. ReLU ist zudem weniger komplex als die Sigmoid Funktion und damit auch entsprechend schneller zu berechnen was Feedforward und



(a) Die Sigmoid Aktivierungsfunktion **(b)** Die Tangens Hyperbolikus Aktivierungsfunktion **(c)** Die Rectified Linear Aktivierungsfunktion

Trainingszeiten entsprechend verkürzt.

Die Wahl der verwendeten Aktivierungsfunktionen hängt stark vom Problemfall und von der verwendeten Netzwerkarchitektur ab. Vor allem für versteckte Ebenen in Tiefen Neuronalen Netzwerken hat sich die Rectified Linear Unit durchgesetzt, da diese das Training des Netzwerks vereinfacht ohne die Genauigkeit des Modells einzuschränken [42]. Die Aktivierungsfunktion in der Ausgabeebene hängt stark vom Problemfall ab. Sigmoidfunktionen eignen sich etwa für Klassifizierungsaufgaben, sind für Regressionsprobleme aber nur bedingt sinnvoll. Genauso wie tanh ist diese im Bereich welcher abgebildet wird begrenzt, was Voraussagen außerhalb dieses verhindert.

2.2.2 Bewertung der Genauigkeit des Netzwerks

Die Eigenschaften des Modells ergeben sich durch die Gewichts- und Offsetwerte der Neuronen in den einzelnen Ebenen des Netzes. Die Ausgabe des Netzes ist also äquivalent zu einer Funktion der Form $\hat{y} = f(\theta, x)$, der Parameter θ ist dabei die Menge aller Gewichte und Offset Werte.

Diese werden während des Trainingsprozesses optimiert um die Genauigkeit des Modells verbessen. Dafür muss diese quantifiziert werden, was durch sogenannte Kostenfunktionen passiert, dabei handelt es sich um die in 2.1 beschriebene Metrik P. Diese wird während der Parameteroptimierung ausgewertet um zu beurteilen inwiefern sich die Performance des Modells verbessert hat.

Ähnlich wie bei Aktivierungsfunktionen hängt die Wahl der Kostenfunktionen erneut von der Problemstellung ab. Für Regressionsprobleme eignet sich oft ein direkter Vergleich der erwarteten Werte (der Label y) mit der Ausgabe des Netzwerks $\hat{y}$. Dafür kann die $L2$ Norm verwendet werden, welche die über alle Elemente gemittelten Fehlerquadrate zwischen y und $\hat{y}$ darstellt. Die Kostenfunktion ist auch unter dem Namen **Mean Squared Error (MSE)** bekannt und für Modellparameter θ und Eingabe x gegeben durch:

$$\mathcal{L}(\theta, x)_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\theta, x)_i)^2$$

In den in Kapitel 4.2 betrachteten Modellen wurde neben dem *Mean Squared Error* auch die **High Frequency Error Norm (HFEN)** verwendet [4, S. 4.4]. Die Berechnung dieser geschieht ähnlich zum vorgerigen MSE, allerdings werden in diesem Fall nicht die Ausgabewerte und Labels direkt verglichen, sondern die zweite Ableitung dieser. Diese wird berechnet durch einen *Laplacian of Gaussian* Filterkernel [45], welcher den Laplace Operator mit einem zuvor angewendeten Gaussian Weichzeichner kombiniert. Der Weichzeichner ist notwendig, um den Einfluss von feinem Rauschen in der Eingabe zu vermindern. Die zweite Ableitung des Laplacian ermöglicht es dann Ecken im Signalverlauf zu finden unter Ausschluss des Rauschverhaltens. Die Kombination von Laplace und Gaussian Kernel, bezeichnet als *Laplacian of Gaussian* ist definiert durch

$$LoG(x) = -\frac{1}{\pi\sigma^4}(1 - \frac{x^2}{2\sigma^2})e^{-\frac{x^2}{2\sigma^2}}$$

im (hier verwendeten) eindimensionalen Fall. σ bezeichnet hier die verwendete Standardabweichung des Gaussian Kernels. Die Anwendung dieses Kernels auf einen hier betrachteten Verlauf ist in Abbildung 7 dargestellt.

Für Klassifizierungsprobleme wird oft **Kreuzentropy** (oder Cross Entropy Error) als Kostenfunktion verwendet, welche die Differenz zwischen zwei Warscheinlichkeitsverteilungen kennzeichnet. Dabei wird die bekannte Verteilung der Zuordnungen zu Kategorien in den Trainingsdaten mit den Zuordnungen des Netzwerks verglichen. Der Fehlerwert ist gegeben durch:

$$\mathcal{L}(\theta, x)_{CEE} = -\sum_{i=1}^{N} y_i log(f(\theta, x)_i)$$

Diese Kostenfunktion eignet sich gut für den Einsatz in Netzwerken mit Sigmoid Aktivierungen in der Ausgabeebene. Hintergrund ist, dass der Logarithmus der Kostenfunktion den Effekt der von e^{-x} im der Sigmoid Funktion aufhebt. [12, Kapitel 6.2.2.2]

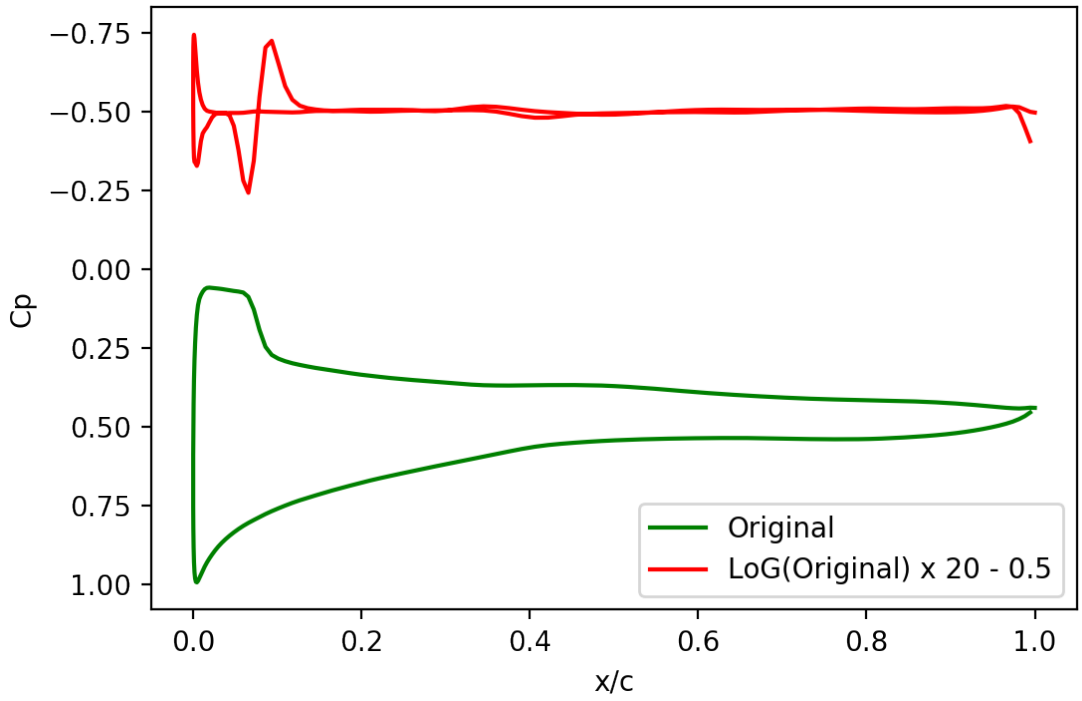


Abbildung 7: Expemlarischer Verlauf der Druckverteilung auf einer Flügeloberfläche und der LoG dieser, der besseren Sichtbarkeit skaliert und verschoben.

2.2.3 Training mit Backpropagation

Während des Trainingsprozesses wird durch einen Optimierungsalgorithmus versucht die verwendete Kostenfunktion zu minimieren. Dafür passt dieser θ, also Gewichte und Offsetwerte aller Neuronen an.

Die Anzahl der Parameter θ während der Optimierung ist üblicherweise extrem hoch, das für Bildklassifizierung eingesetzte AlexNet [22] etwa verfügt über 60 Millionen trainierbare Parameter. Dies macht die Optimierung aufwendig, da mit der Anzahl der Dimensionen des Problems die notwendige Anzahl von Evaluierungen exponentiell wächst. Dies wird auch als *Curse of Dimensionality* bezeichnet. [5, S. 25]

Auch aus diesem Grund werden gradientenbasierte Optimierungsmethoden verwendet, welche im Gegensatz zu gradientfreien Methoden zusätzlich die Ableitung der Kostenfunktion verwenden, um schneller zu einem Optimum zu konvergieren. Für den Einstaz dieser Methoden ist es notwendig, die partiellen Ableitungen des Modells mit Respekt zu den einzelnen trainierbaren Parametern zu kennen. Dieser Vektor für eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ wird als

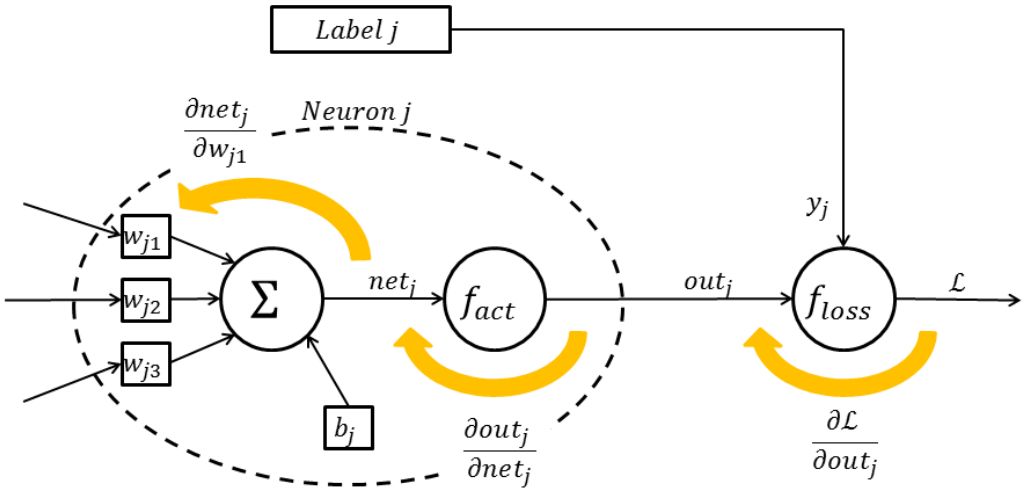


Abbildung 8: Backpropagation Vorgang: Der Gradient der Kostenfunktion f_{loss} wird durch das Netzwerk zurückgeführt.

Gradient bezeichnet und ist definiert durch:

$$\nabla_{f(x_1,x_2,x_3,\ldots,x_n)} = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \frac{\partial f}{\partial x_3} \quad \cdots \quad \frac{\partial f}{\partial x_n}\right]$$

Die partiellen Ableitungen des Gradients können automatisch aus den einzelnen Elementen des Netzwerks mit dem Backpropagation Algorithmus berechnet werden [36]. Die Funktionsweise wird im Folgenden am Beispiel des in Abbildung 4 dargestellten Netzwerks erklärt, dabei wird exemplarisch die Berechnung der partiellen Ableitung der Kostenfunktion mit Respekt zum Gewichtwert $w_{i1,j1}$ genutzt. Für die Berechnung dieser wird die Kettenregel angewendet:

$$\frac{\partial \mathcal{L}}{\partial w_{i1,j1}} = \frac{\partial \mathcal{L}}{\partial out_{j1}} * \frac{\partial out_{j1}}{\partial net_{j1}} * \frac{\partial net_{j1}}{\partial wi1,j1}$$

Dabei ist $\mathcal{L}$ die Kostenfunktion der Optimierung, out_{i1} die Ausgabe der Aktivierungsfunktion

für das Neuron i_1 und net_i die Ebene i, also Multiplikation der Eingabe mit Gewichtsmatrix mit anschließender Addition des Offsetwertes. Die Bedeutung dieser einzelnen Teilableitungen im Kontext eines einzelnen Neurons sind exemplarisch in Abbildung 8 illustriert.

Im Folgenden wird die Herleitung der einzelnen Teilableitung für das genutzte Beispiel gezeigt. Dabei wird *MSE* als Kostenfunktion und *Sigmoid* als Aktivierungsfunktion der letzten Ebene genuzt.

Der erste Teil kennzeichnet, in wie fern sich die Kostenfunktion bei Veränderung der Ausgabe von Neuron $j1$ ändert:

$$\begin{aligned} \mathcal{L} &= 0.5(out_{j1} - y_{j1})^2 + 0.5(out_{j2} - y_{j2})^2 \\ \frac{\partial \mathcal{L}}{\partial out_{j1}} &= out_{j1} - y_{j1} \end{aligned}$$

Die zweite Teilableitung definiert die Änderung der Aktivierungsfunktion von Neuron $j1$ in Abhänigkeit von deren Eingabe, also der gewichteten Summe des vorangehenden Netzwerks:

$$\begin{aligned} out_{j1} &= \frac{1}{1 + e^{-net_{j1}}} \\ \frac{\partial out_{j1}}{\partial net_{j1}} &= out_{j1}(1 - out_{j1}) \end{aligned}$$

Die dritte Teilableitung kennzeichnet, inwiefern sich die Summe der vorangehenden Neuronen ändert, wenn für dieser der Gewichtswert w_{i1j1} ändert:

$$\begin{aligned} net_{j1} &= w_{i1j1}i_1 + w_{i2j1}i_2 + w_{i3j1}i_3 + w_{i4j1}i_4 + b_{j1} \\ \frac{\partial net_{j1}}{\partial w_{i1j1}} &= w_{i1j1}i_1 \end{aligned}$$

Für den Backpropagation Schritt ist es also notwendig, dass Kosten- und Aktivierungsfunktionen des Netzwerks vollständig differenzierbar sind. In der Praxis ist diese Anforderung allerdings etwas eingeschränkt, wie das Beispiel der vorgestellten *ReLU* Aktivierungsfunktion zeigt: Diese ist für $x = 0$ nicht stetig und somit auch nicht differenzierbar, für die numerische Anwendung ist allerdings die Definition des Gradienten an dieser Stelle zu 0 oder 1 ausreichend für die Anwendung.

Dieser Vorgang wird für alle trainierbaren Parameter des Netzwerks durchgeführt. Daraus ergibt sich der Gradient ∇. In einem gradientenbasierten Optimierungsverfahren wird dieser

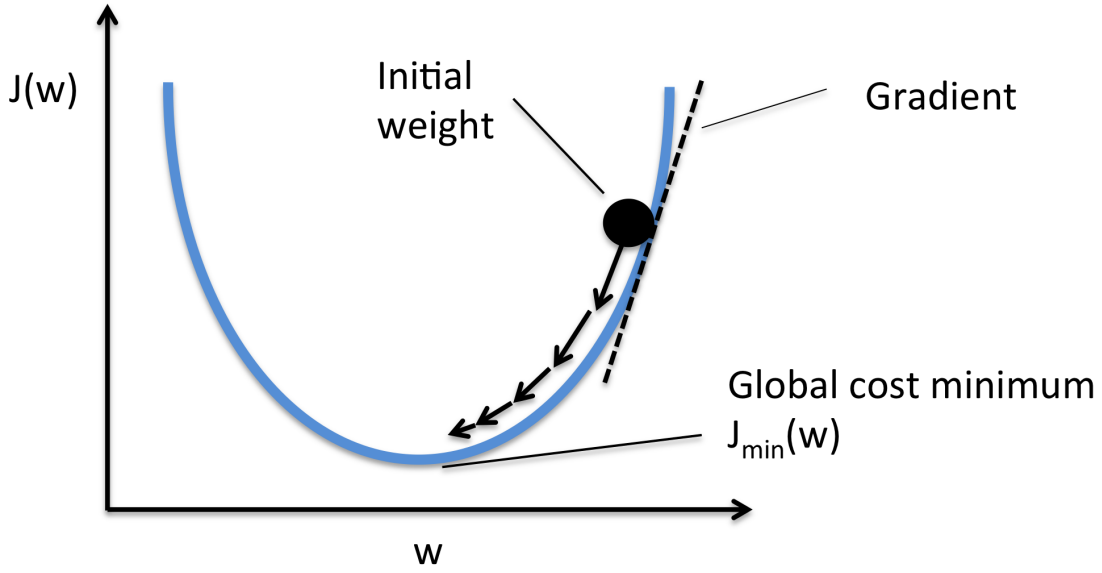


Abbildung 9: Gradientenbasierte Optimierung für eine eindimensionale Funktion. Jeder Schritt, durch Pfeil gekennzeichnet, entspricht einer Optimierungsiteration. Quelle: [35]

nun verwendet um die trainierbare Parameter des Netzwerks anzupassen und somit die Kostenfunktion zu minimieren. Parameter nach einer Iteration der Optimierung ergeben sich dabei wie folgt:

$$\theta_{n+1} = \theta_n - \eta\nabla(\theta_n)$$

Die aktuellen Parameter θ_n werden also in Richtung ihres steilsten Gradienten angepasst, weshalb der Algorithmus als *Gradient Descent* bezeichnet wird. Exemplarisch für einen eindimensionalen Fall ist dies in Abbildung 9 dargestellt. Dabei bestimmt der Hyperparameter η die Rate des Lernens, also wie stark innerhalb einer Iteration das Netzwerk angepasst wird. Zu niedrige Werte verlangsamen stark die Konvergenz und können dazu führen, dass die Optimierung einem lokalen Minimum einfach stecken bleibt während zu hohe Werte ein Überschwingen über das eigentlich Minimum hinweg verursachen können. Die kann zur schlechen Konvergenz oder gar Divergenz der Optimierung führen. Analog zu Abbildung 9 sind diese Effekte in Abbildung 10 dargestellt. Diese Berechnung der neuen Parameter θ_{n+1} wird für jedes Element des Datensatzes nacheinander durchgeführt. Dieser Optimierungsvorgang über alle Elemente des Datensatzes wird als Trainingsepoche bezeichnet, üblich ist es ein Training des Netzwerks über mehrere Epochen wobei die konkrete Anzahl dieser stark von der

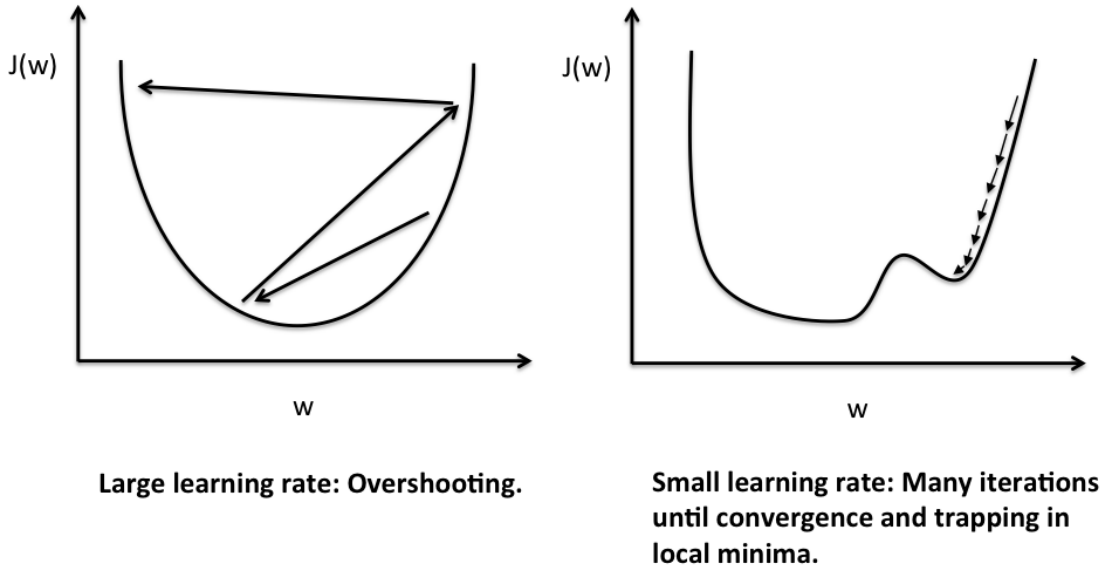


Abbildung 10: Effekte zu hoher und zu niedriger Lernrate bei Gradientenoptimierung. Quelle: [1]

Problemstellung und prinzipbeding der Größe des Trainingsdatensatzes abhängt. In der Praxis werden spezialisierte Optimierer eingesetzt, welche zwar auf demselben Prinzip basieren, dieses allerdings durch adaptive Lernraten oder stochastische Methoden erweitern um robustere und schnellere Optimierung zu erreichen. Ein Beispiel hierfür ist der auch in dieser Arbeit eingesetzte Adam Optimierer [19].

Die hohe Parameterdimension zusammen mit der Nichtlinearität des Netzwerkes sorgt für stark nicht konvexe Kostenfunktionen, ein Bespiel für solch eine Oberfläche ist in Abbildung 11 darstellt. Dies bedeutet, dass es prinzipiell möglich ist, während des Trainingsprozesses in einem lokalen Minimum hängen zu bleiben, welches vom globalen weit entfernt ist und dafür sorgt, dass das Modell die Trainingsdaten schlecht abbildet. Choromanska et. al. haben allerdings gezeigt, dass in den meisten Fällen das vom Optimierungsalgorithmus gefundene Optimum sehr nah dem globalen liegt [6], bedeutet das Finden des globalen Optimums in der Praxis ist eher irrelevant. Tatsächlich ist das absolute Minimum nicht unbedingt erwünscht, dies bedeutet, dass das Netzwerk die Trainingsdaten bestmöglich abbildet. Dies ist allerdings nicht das Ziel, schließlich sind hierfür die Eingabe-Ausgabe Kombinationen bereits bekannt. Ziel des Modells ist es vielmehr verallgemeinernde Zusammenhänge zu finden, um sinnvolle Aussagen über nicht in den Trainingsdaten erhaltende Eingaben zu treffen. [40]

Trotzdem führt dies, zusammen mit stochastischen Optimierungsalgorithmen, zur oft einge-

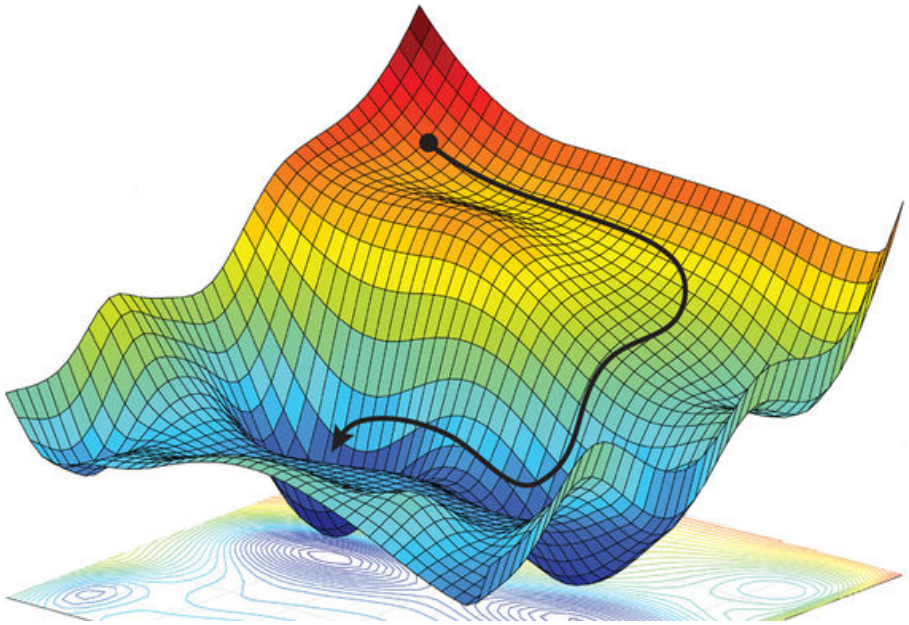


Abbildung 11: Gradientenbasierte Optimierung auf einer zweidimensionalen Parameterraum. Die dargestellte Oberfäche verdeutlicht das Problem, dass der Optimierungsalgorithmus in lokalen Minima hängen bleibt. Quelle: [16]

schränkten Reproduzierbarkeit von Trainingsergebnissen. [16]

2.2.4 Vanishing Gradients und ReLU

Die Nutzung von klassischen Aktivierungsfunktionen wie tanh oder Sigmoid hat sich beim Trainieren von sehr tiefen Netzwerken mit mehreren versteckten Ebenen als problematisch erwiesen. Der Grund dafür wird als *vanishing gradients*, also verschwindende Gradienten beschrieben. Das Problem ist, dass die problematischen Aktivierungsfunktionen sowohl im positiven, als auch im negativen Bereich sehr schnell saturieren, bedeutet das selbst große Werteänderungen der Eingabe nur sehr kleine Änderungen der Ausgabe verursachen. Der Gradient der Kostenfunktion geht also in den äußeren Bereichen dieser gegen Null. Dies ist problematisch im Backpropagation Schritt in tiefen Netzwerken, da sich hier die kleinen Gradienten mehrerer solcher Aktivierungsfunktionen aufmultiplizieren, so dass der Gesamtgradient für Parameter vor mehreren solcher Ebenen nahezu verschwindet, sobald in einem oder mehreren der nachfolgenden Ebenen Saturierung auftritt.

Diese fehlenden Gradienten machen das Training solcher Netzwerke deshalb sehr ineffizient oder verhindern gar komplett Fortschritt während der Optimierung. Abhilfe gegen das Problem

schafft die ReLU Aktivierungsfunktion, welche im positiven Bereich überhaupt nicht saturiert. Die Nutzung dieser in den versteckten Ebenen von Neuronalen Netzen hat generell zu besseren und schnelleren Trainingsergebnissen geführt [11], ohne dabei negative Auswirkungen auf die Möglichtkeit des Netzwerks komplexe Funktionen zu modellieren zu haben [14]. Aus diesem Grund gilt die Nutzung von ReLU als Aktivierungsfunktion für versteckte Ebenen, also allen außer der Ausgabe, als beste Praxis in der Architektur von Neuronalen Netzwerken [42]. Ein Problem das mit ReLU auftreten kann, ist dass Neuronen mit Aktivierungen im negativen Bereich nicht weiter zum Training beitragen, da hier der Gradient Null ist und während des Trainings nicht mehr in den positiven Bereich wandert. Das Problem wird deshalb als *Dead ReLU* bezeichnet. Verallgemeinerungen dieser Aktivierungsfunktion, etwa durch *Exponential Linear Units (ELU)* welche im negativen Bereich durch eine Exponentialfunktion $e^x - 1$ definiert sind, welche auch hier über einen Gradienten ungleich Null verfügt, können das Problem verhindern. [43]

2.3 Convolutional Neural Networks

Wie bereits beschrieben führt die steigende Anzahl von Parametern zum exponentiell aufwendigerem Trainingsprozess [12, Kapitel 9.1]. Vor allem in der Bildverarbeitung mit hohen Eingabedimensionen im Bereich von potentiell Millionen von Neuronen allein in der Eingabeebene stößt die bisher vorgestellte Architektur mit vollständig verbundenen Ebenen aus diesem Grund an ihre Grenzen. In Verbindung mit tiefen Netzwerken mit breiten Ebenen kommt es schnell zu nicht mehr beherrschbaren Parameteranzahlen.

In diesen Bereichen werden vor allem *Convolutional Neural Networks (CNNs)* verwendet. Diese machen sich die Lokalität der vorliegenden Daten zunutze, welche in Bildern wesentlicher Bestandteil der enthaltenen Informationen ist - durch Features wie Linien welche durch nebeneinanderliegende Pixel ähnlicher Farbe oder Helligkeit gekennzeichnet sind. Dazu kommt die räumliche Invarianz dieser, die Erkennung eines solchen Features hängt also nicht davon ab, wo innerhalb des Bildes es sich befindet. Dieser Unterschied zu vollständig verbundenen Ebenen ist in Abbildung 12 dargestellt. Um Features auszuwerten werden aus der Bildverarbeitung bekannte Filterkernel verwendet, welche im Feedforward Schritt über die Eingabe der Ebene gefaltet (englisch: *convolution*) werden. Diese Filterkernel sind in der Lage Features wie etwa Kanten innerhalb ihrer Eingabe zu erkennen, ein Beispiel dafür ist in Abbildung 13 zu sehen. Anders als in der Bildverarbeitung werden diese Kernels allerdings nicht manuell während des Designs definiert sondern während des Trainings erlernt. [24]

Hierbei ist der Prozess nicht anders als der bisher vorgestellte, welcher für vollständig

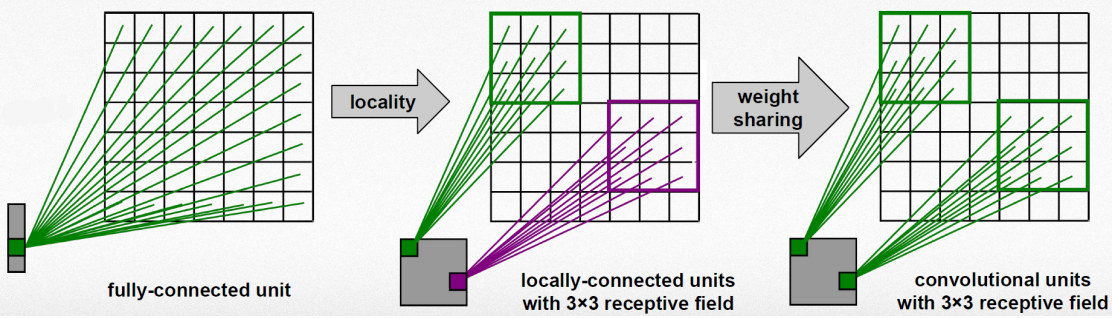


Abbildung 12: Ausnutzung von Lokalität und lokaler Invarianz in Convolutional Neuroal Networks im Vergleich zu Vollständig verbundenden Ebenen. Quelle: [26], modifiziert

verbundende Netzwerke genutzt wurde. Dies ist möglich, da sich CNNs grundsätzlich nicht von der bisherigen Architektur unterscheiden - Der Wert eines Elements der Ausgabe ergibt sich durch die Multiplikation der Nachbarwerte mit den korrespondierenden Kernelwerten und der Aufsummierung dieser. Dieser Vorgang ist prinzipiell sehr ähnlich mit der Multiplikation von Eingabeverbindungen mit Gewichtswerten in einem vollständig verbundenden Netzwerk, in diesem Fall werden allerdings nicht alle Ausgabewerte der vorangehenden Ebene berücksichtigt, sondern lediglich die benachbarten des Pixels. Dazu kommt die, bereits beschriebene, gemeinsam genutzte Gewichtsmatrix in Form des Faltungskernels. Erhalten bleibt auch in CNNs die Addition eines Offsetwertes und die darauf folgende Anwendung einer (meist nichtlinearen) Aktivierungsfunktion. Verändert ist lediglich die Anzahl der trainierbaren Parameter, diese ist drastisch reduziert. CNNs werden meistens in Kombinationen mit herkömmlichen, vollständig verbundenden Netzwerken eingesetzt, wobei vor diesen meist mehrere Convolutional Layer zur Feature Erkennung verwendet werden. Ein Beispiel für den solchen Einsatz ist das in Abbildung Abbildung 14 dargestellte Netzwerk welches in den US zur Erkennung von handschriftlichen Ziffern auf Bankchecks konzipiert wurde [24].

Um unterschiedliche Features zu erkennen, werden üblicherweise mehrere Kernel auf die Eingabe angewendet, wobei jeder eine eigene Ausgabe produziert. Aus einem Bild mit Breite und Höhe entsteht nach Anwendung von n Filtern also ein Tensor mit der Tiefe n. Die Anwendung von weiteren Faltungsoperationen auf diese Ausgabe in darauf folgenden Ebenen nutzt dann Filterkernel der Größe $n*k*k$, es werden also alle Nachbarwerte über die einzelnen Featureebenen berücksichtigt.

Die Anzahl und Größe der trainierten Kernels sind, anders als deren tatsächliche Werte, nicht trainierte Hyperparameter und werden in der Architektur des Netzwerks bestimmt. Größere Filterkenel berücksichtigen entsprechend größere Bereiche, haben allerdings wieder

Abbildung 13: Beispiel für die Anwendung eines Filters zur Kantenerkennung (*Sobol edge detection*. In diesem Fall handelt es sich um einen manuell definierten Filterkernel und nicht um erlernte Parameter eines CNNs.) Quelle: [34]

mehr Parameter welche trainiert werden müssen und sind aufgrund der höheren Anzahl an Rechenoperationen auch ansonsten in der Ausführung langsamer.

Um die Größe der resultierenden Featuremaps zu verkleinern und mit der resultierenden niedrigeren Größe der Daten besser mit vollständig verbundenen Ebenen zusammen arbeiten zu können, werden in CNNs Pooling Layer eingesetzt. Diese verringern die Größe der Eingabe, in dem sie lediglich einen Anteil dieser ohne Veränderung weitergeben, weshalb der Vorgang in Abbildung 14 auch als Subsampling bezeichnet wird. Oft wird dabei ein Max-Pooling Verfahren genutzt, bei welchen aus einem definierten Bereich lediglich der größte Aktivierungswert weitergegeben wird.

Der Einsatz von CNNs hat zu rasanten Fortschritten im Bereich der Bilderkennung geführt. Kennzeichnend dafür ist die sich drastisch verbessernde Klassifizierungsgenauigkeit im *Large Scale Visual Recognition Challenge* Wettbewerb, bei welchen es um Klassifizierung eines gegebenen Bilddatensatzes geht [17]. Die Nutzung von unterschiedlichen CNN Architekturen führte hier zu einer Verbesserung von zuvor 25% Fehlerrate im Jahr 2011 zu 3.5% im Jahr 2015, eine Genauigkeit welche mit menschlicher Erkennungsrate auf dem selben Datensatz vergleichbar ist. [22], [15]

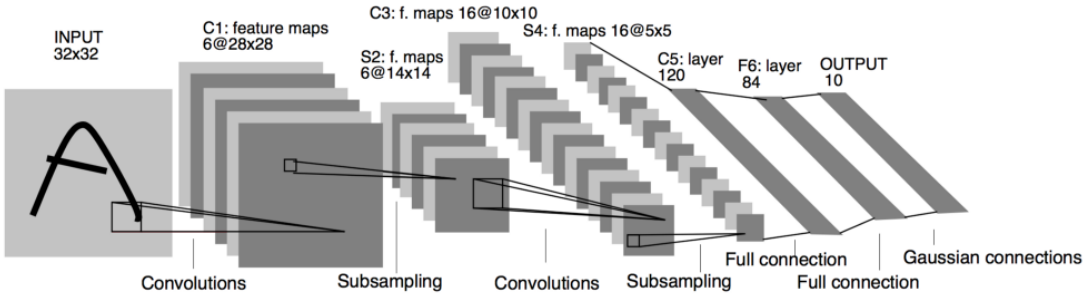


Abbildung 14: LeNet, ein CNN konzipiert für die Erkennung handgeschriebener Ziffern auf Bankchecks in der US. Deutlich sichtbar ist die Generierung mehrere Featuremaps durch Filterkenel. Quelle: [24]

2.4 Dimensionsreduzierung

Das Ziel von Dimensionsreduzierung ist es, eine kleinere Anzahl an Variablen zu finden, welche ein üblicherweise hochdimensionales Problem möglichst adäquat beschreiben. Der dimensionsreduzierte Raum kann daraufhin genutzt werden um Klassifizierungs- oder Regressionsprobleme auf diesem zu lösen. Die Abbildung in weniger Dimensionen kann die Probleme stark vereinfachen da hier der mit der Dimension generell exponentiell steigende Aufwand vermieden wird. [5]

In Abbildung 15 ist ein Beispiel für eine solche Dimensionsreduzierung gegeben. Dabei wurde das *MNIST Handwritten Digit* [25] verwendet, welches aus Schwarz-Weiss Bildern von handgeschriebenen Ziffern der Größe 28x28 Pixeln besteht. Die Vektoren dieser mit je 784 Elementen wurden mit dem t-SNE Algorithmus [29] auf einen zweielementigen Vektor abgebildet, welcher sich als Punkt im dargestellten Diagramm visualisieren lässt. Der Algorithmus versucht dabei die euklidischen Abstände der Daten in der dimensionsreduzierten Abbildung zu erhalten. Sichtbar ist die starke Gruppierung nach Labels (geschriebener Ziffer) welche den möglichen Einsatz von Dimensionsreduzierung für Klassifikationsprobleme demonstriert.

2.4.1 Autoencoder Netzwerke für Dimensionsreduzierung

Autoencoder sind Neuronale Netzwerke welche das Ziel haben, ihre Eingabe in der Ausgabe zu reproduzieren. Dies geschieht über eine Zwischenebene welche eine Abbildung der Daten erlernt, diese wird auch als *Coding Layer* bezeichnet. Da eine direkte Projektion der Daten als eins zu eins Abbildung der Eingabe auf die Ausgabe nicht sonderlich interessant ist, wird die Coding Ebene meist stark in ihrer Breite eingeschränkt, was das Netzwerk zwingt eine effiziente Repräsentation der Daten innerhalb dieser zu definieren. Die Codierung wird auch

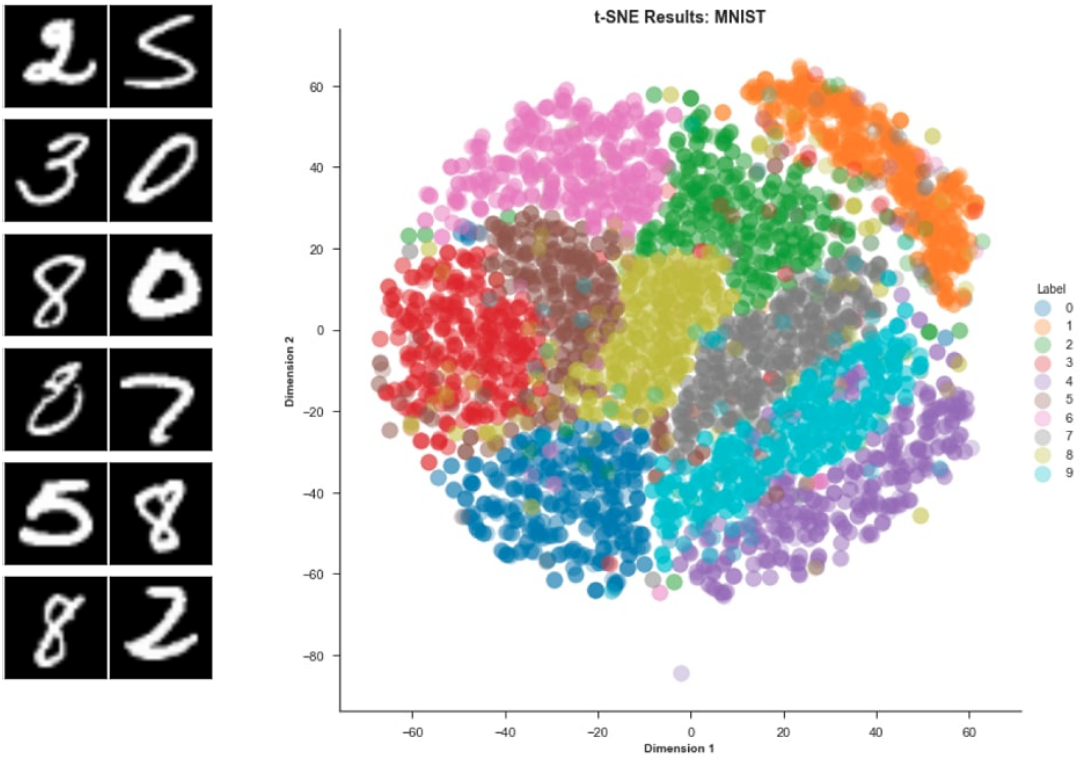


Abbildung 15: Dimensionsreduzierung am Beispiel von t-SNE: Der Algorithmus versucht die 28x28 Pixel (=784 dimensionalen) Samples der MNIST [25] Datensets (links) in einem einfach darstellbaren zweidimensionalen Raum abzubilden (rechts) wobei er versucht Abstände der Daten zu erhalten. [29]. Quelle: [44], modifiziert

als Latente Abbildung bezeichnet, idealerweise bildet die Ebene Variablen ab, welche in den Daten nicht direkt sichtbar sind, sondern sich aus einem durch das Netzwerk definiertes Modell ergeben. [12, Kapitel 14.1]

Autoencoder bestehen dabei aus einem Encoder und einem Decoder Netzwerk. Ausgabe des ersteren ist die niedrig dimensionale Coding Ebene, welche gleichzeitig als Eingabe für den Decoder dient. Der Decoder wird darauf trainiert aus der Coding Ebene die originalen Daten möglichst genau zu rekonstruieren. Dieser Aufbau ist in Abbildung 16 dargestellt.

Da Autoencoder sich prinzipiell nicht von den bisher vorgestellten Neuronalen Netzwerken unterscheiden, ist auch in diesen die Anwendung von *Convolutional* Ebenen möglich. Da im Decodeteil dieser die Dimension wieder erhöht wird, kommen hier keine Pooling, sondern

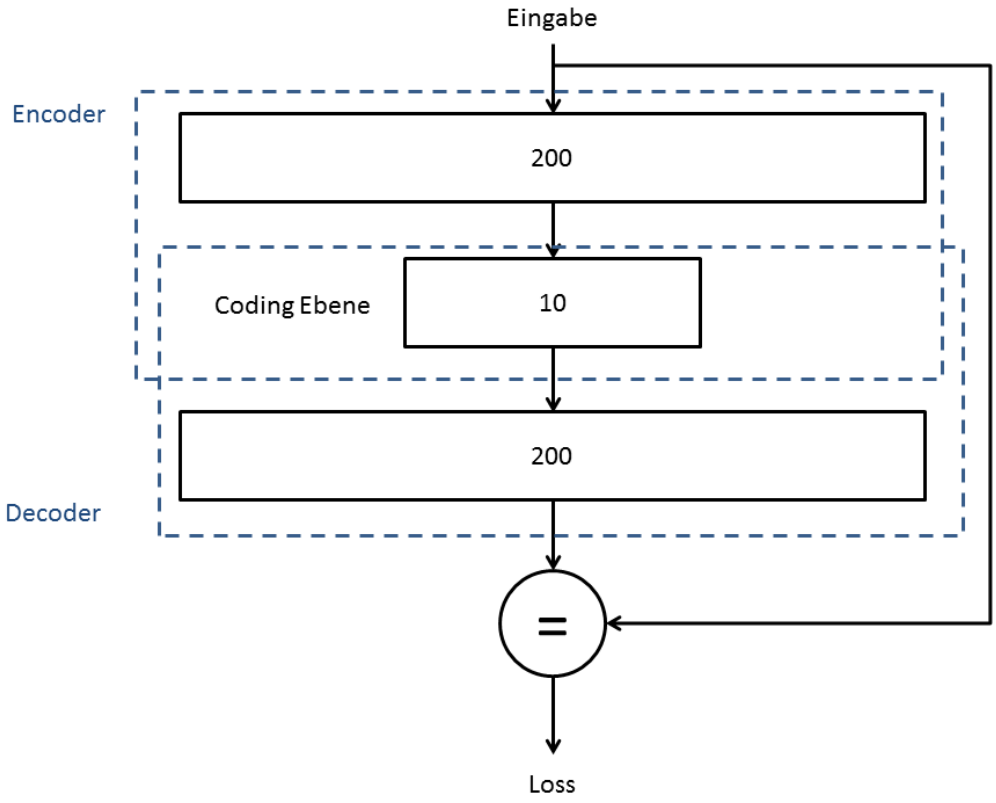


Abbildung 16: Darstellung eines einfachen Autoencoders mit drei Ebenen. Ziel des Netzwerks ist es, die Eingabe in der Ausgabe wieder abzubilden. Erschwert wird dies durch die reduzierte Coding Ebene, welche eine direkte Abbildung verhindert und das Netzwerk zwingt eine Beschreibung für die wichtigen Features der Eingabe zu definieren.

sogenannte Unpooling oder Upsampling Ebenen zum Einsatz. Diese vergrößern die Eingabedimension, indem sie jeden Eingabewert auf mehrere Ausgabewerte abbilden. Neben dem einfachen Kopieren der Eingabe auf mehrere Ausgaben sind auch Interpolationsverfahren für die neuen Ausgabeelemente möglich.

2.4.2 Dimensionsreduzierung für Generative Modelle

Die dimensionsreduzierte Repräsentation von Daten kann auch genutzt werden um durch Interpolationsmethoden auf diesem Unterraum neue Repräsentationen zu erzeugen, welche durch die Dimensionsreduzierungsmethode rekonstruiert werden können. Es handelt sich dabei also um ein Regressionsmodell, welches sich die potientiell besser zur Interpolation

geeignete Abbildung auf dem reduzierten Unterraum zu Nutze macht. Da diese Art der Modelle Daten generiert, welche den Originalen des Trainingsdatensatzes ähneln, aber nicht in diesem enthalten waren, spricht man auch von generativen Modellen.

Autoencoder, als Methode zur Dimensionsreduzierung, lassen ebenso für die Generierung neuer Daten nutzen. Dabei wird dieser wie bisher mit den verfügbaren Trainingsdaten darauf trainiert, diese möglichst gut abzubilden. Nach dem Training des gesamten Autoencoders wird der Encoder Teil verwendet, um die latenten Repräsentationen des Autoencoders innerhalb der Coding Ebene für das Trainingsdatenset zu bestimmen. Mit diesen Paaren von Eingabeelementen zu latenten Vektoren lässt sich mit einer Interpolationsmethode ein neuer latenter Vektor bilden, welcher z.B. zwischen zwei Eingabedaten liegt. Dieser dient dann als Eingabe des Decoderteils, welcher mit diesem ein Datum mit der Ausgangsdimension erzeugt.

3 Ersatzmodellierung aerodynamischer Daten mit Modellen reduzierter Ordnung

Die in SMARTy implementierte Ersatzmodellierung auf Basis von POD und TPS basiert auf der Interpolation zwischen vorhandenen Trainingssnapshots in dem durch die POD Methode erzeugten Unterraum. Dabei wird die TPS Interpolationsmethode genutzt, um für die entsprechende Parameterkombination einen abweichenden Unterraumvektor zu generieren. Die Rekonstruktion des Unterraumvektors durch die POD Methode erzeugt eine näherungsweise Vorhersage für diese Parameterkombination. Im diesem Kapitel werden die POD und TPS Methoden beschrieben und wie sich der Vorgang durch die Nutzung von Autoencodernetzwerken zur Dimensionsreduzierung anstelle von POD ändert.

3.1 Dimensionsreduzierung mit POD

Proper Orthogonal Decomposition (POD), auch bekannt als PCA (Principal Component Analysis, dt.: Hauptkomponentenanalyse) ist eine Methode um einen Datensatz in orthogonale Hauptkomonenten zu zerlegen. Die Berechnung dieser Komponenten geschieht durch die Singulärwertzerlegung der Matrix des Datensatzes. Im Kontext der aerodynamischen Ersatzmodellierung besteht diese Matrix zeilenweise aus den m Trainingssnapshots, aus welchen das Ersatzmodell gebildet wird. Bei diesen handelt es sich um Vektoren mit n Elementen, dabei kann es sich etwa um Volumen oder Oberflächendruckwerte in definierter Reihenfolge handeln. Diese Snapshotmatrix, im folgenden $W \in \mathbb{R}^{n \times m}$ wird mit einer Singulärwertzerlegung in zwei orthogonale Matritzen und eine Diagonalmatrix mit Singulärwerten zerlegt:

$$W = U \Sigma V^T$$

U und V sind orthogonale Matritzen mit den Größen $n \times n$ und $m \times m$. Bei $\Sigma \in \mathbb{R}^{n \times m}$ handelt es sich um eine Diagonalmatrix der Singulärwerte $\sigma_1, \sigma_2, ...\sigma_n$. Diese sind alle ≥ 0 und kennzeichnen die Skalierung der jeweiligen Achsen im Datenset. Zusätzlich sind die Singulärwerte ihrer Größe nach absteigend geordnet, so dass $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{min(n,m)}$ gilt. Diese Tatsache kann für die Dimensionsreduzierung genutzt werden - Dabei werden lediglich die letzten k Singulärwerte $\sigma_1 ... \sigma_k$ verwendet und alle überigen Werte auf 0 gesetzt, wodurch die Dimension der Darstellung auf effektiv k Elemente reduziert wird. [38], [47].

Durch die Singulärwertzerlegung hängt jeder einzelnen Snapshot W_i direkt von einer Zeile

der Matrix V^T ab:

$$W^i = U\Sigma(V^T)^i$$

Die Multiplikation mit der Diagonalmatrix Σ mit dem Vektor $(V^T)^i$ lässt sich durch die folgende Summe über alle Elemente des Vektors, multipliziert mit den einzelnen Singulärwerten σ abbilden:

$$W_i = \sum_{j=1}^{k} (\sigma_j V_i^j) U^j \tag{3.1}$$

Ein einzelner Snapshot hängt also von allen Elementen der Matrix U ab, die Elemente dieser werden als POD Moden bezeichnet. $(\sigma_j V_i^j) = a_i^j$ wird als POD Koeffizient bezeichnet, der Vektor $a^i = (a_1^i, a_2^i \dots a_m^i)$ ist der Koeffizientenvektor, von welchem der Snapshot i abhängt. Er besteht zwar aus m Elementen, also der vollständigen Snapshotdimension, allerdings durch die Multiplikation mit den Singulärwerten maximal die ersten n Werte dieses ungleich 0, eine Dimensionsreduktion auf die ersten k Singulärwerte reduziert den Koeffizientenvektor ebenfalls auf k Werte ungleich 0.

Da ein Snapshot direkt vom Koeffizientenvektor abhängt, kann das Verfahren für die Generierung einer Vorhersage außerhalb der Snapshotmatrix W genutzt werden. Dabei wird mithilfe eines Interpolationsverfahrens, etwa mit der im Folgenden vorgestellten TPS Methode, ein neuer Koeffizientenvektor a erzeugt. Für die Interpolationsmethode kann die Tatsache genutzt werden, dass für jeden Vektor a^i der vorhandenen Snapshots W^i eine Ausgangsparameterkombination bekannt ist. Diese Zuordnungen aus Berechnungsparametern zu POD Koeffizientenvektoren können als Stützstellen für eine Interpolationsmethode genutzt werden, welche es dann ermöglicht zwischen den bereits bekannten Parameterkombinationen neue Koeffizientenvektoren zu berechnen. [10, Kapitel 3.3]

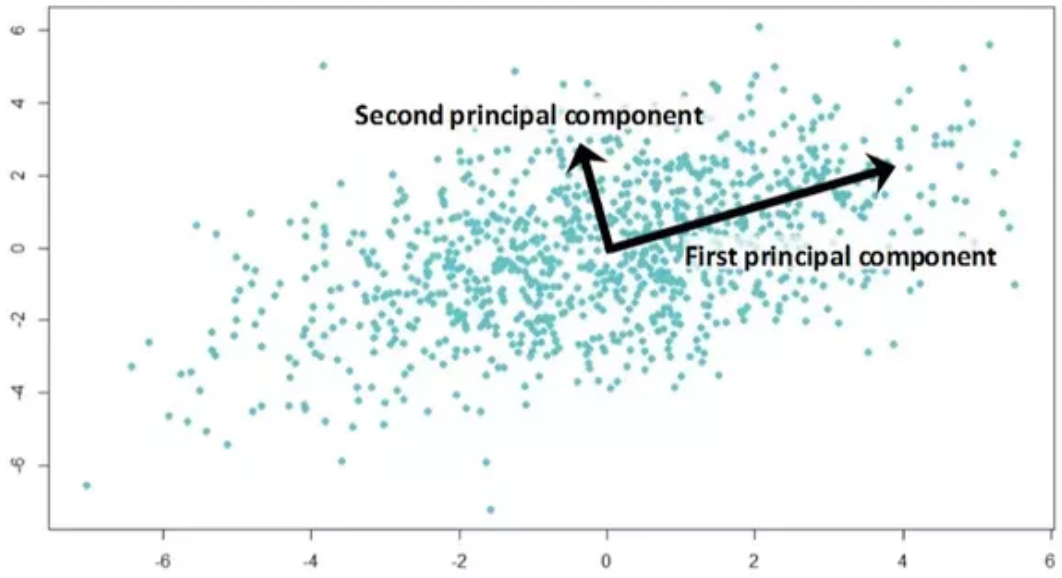


Abbildung 17: Hauptkomponentenanalyse, eingezeichnet sind die beiden orthogonalen Hauptachsen des Datensets. Quelle: [33]

3.2 Interpolationsmethode TPS

Für die Interpolation der Koeffizientenvektoren des POD Verfahrens ist in SMARTy die Thin Plate Spline (TPS) Methode implementiert. Die Methode wurde in dieser Arbeit nicht nur in Verbindung mit POD eingesetzt, sondern auch für die Interpolation innerhalb der latenten Repräsentation des Autoencodernetzwerks.

TPS basiert darauf eine minimal gebogene Oberfläche durch eine Menge an Kontrollpunkten zu legen. Ein Beispiel für diese Methode ist in Abbildung 18 dargestellt, hier wird ein eindimensionaler Wert, welcher von einem zweidimensionalen Parametervektor abhängig ist, interpoliert.

TPS verfügt über einen Regularisierungsterm, welcher der Methode ermöglicht nicht sämtliche Kontrollpunkte genau abzubilden, sondern Abweichungen zugunsten einer geringeren Biegung der Oberfläche erlaubt. Dies ist vor allem bei verrauschten Daten mit eventuellen Ausreißern interessant und ermöglicht für diese eine glattere Oberfläche, welche idealerweise in der Lage ist die zugrundeliegende Struktur der Daten besser abzubilden. Diese Regularisierung ist durch einen Parameter λ steuerbar, Werte gegen 0 sorgen für eine vollständige Abbildung aller Kontrollpunkte, Werte Richtung ∞ resultieren in einer graden

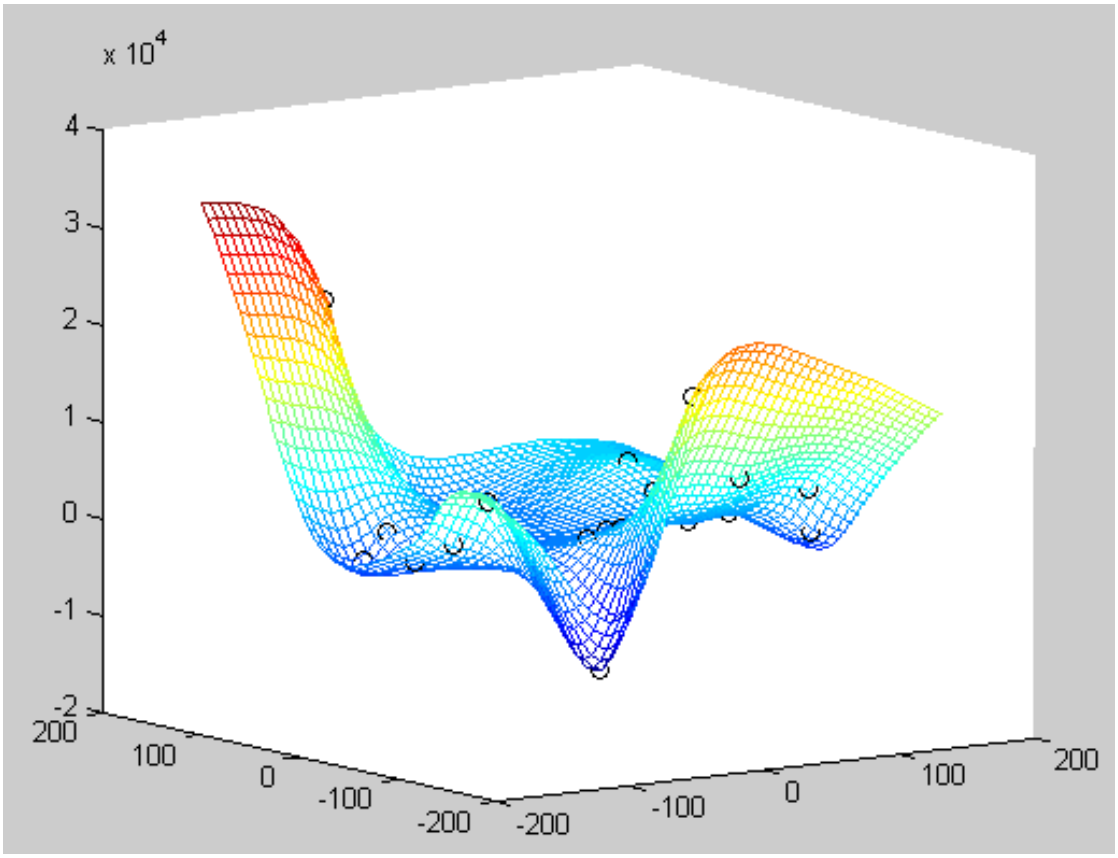


Abbildung 18: Interpolationsoberfläche welche zweidimensionale Parameter auf einen skalaren Wert abbildet. Die Kontrollpunkte welche die Fläche definieren sind als schwarze Kreise gekennzeichnet. Quelle: [48]

Ebene ohne jegliche Biegung. [8]

Ziel der Methode ist es, eine Abbildung von m dimensionalen Parametern zu n dimensionalen Vektoren zu bilden. Im Einsatz für die Ersatzmodellierung bedeutet dies die Abbildung von m Parametern auf die auf n Dimensionen reduzierte Snapshotdarstellung. Der Spline, welcher eine Ebene zwischen p Kontrollpunkten $c_i, i \in [1..p]$ der Dimensions $\mathbb{R}^m$ mit den Werten $d_i \in \mathbb{R}^n, i \in [1..p]$ ist durch folgende Gleichung definiert:

$$f(x) = a_1 + x^T A_r + \sum_{i=1}^{p} w_i U(|c_i - x_i|)$$
$$\text{mit der Radialen Basisfunktion } U(r) = \begin{cases} r^2 log(r) & r > 0 \\ 0 & r = 0 \end{cases} \tag{3.2}$$

Dabei sind $A \in \mathbb{R}^{(m+1)\times n}$ und $W \in \mathbb{R}^{p\times n}$ Kontrollmatritzen der für die Interpolationsoberfläche, welche Lage und Krümmung der Ebene definieren. a_1 bezeichnet die erste Zeile von A während A_r den Rest der Matrix ohne diese Zeile bezeichnet. Die Kontrollmatritzen ergeben sich aus der Lösung des folgenden linearen Gleichungssystems:

$$\begin{bmatrix} K & P \\ P^T & 0 \end{bmatrix} \begin{bmatrix} W \\ A \end{bmatrix} = \begin{bmatrix} V \\ 0 \end{bmatrix} \tag{3.3}$$

Die Matrix $K \in \mathbb{R}^{p\times p}$ besteht aus der, in 3.2 definierten, Radialen Basis Funktion U der euklidischen Distanzen zwischen den Elementen des Vektors der Kontrollpunkte C:

$$K_{ij} = U(\|c_i - c_j\|)$$

Für den Regularisierungsparameter $\lambda \neq 0$ beinhaltet diese Matrix zusätzlich noch folgenden Regularisierungsterm welcher eine Abweichung der Oberfläche von den Kontrollpunkten ermöglicht:

$$K_{ij} = U(\|c_i - c_j\|) + \alpha^2\lambda \quad i,j \in [1..p]$$
$$\text{mit } \alpha = \frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{p}\|c_i - c_j\|$$

$P \in \mathbb{R}^{p\times(m+1)}$ beinhaltet die Kontrollpunkte C:

$$P = \begin{bmatrix} 1 & c_{11} & c_{12} & ... & c_{1m} \\ 1 & c_{21} & c_{22} & ... & c_{3m} \\ 1 & c_{31} & c_{22} & ... & c_{3m} \\ ... & & & & \\ 1 & c_{p1} & c_{p2} & ... & c_{pm} \end{bmatrix}$$

und $V \in \mathbb{R}^{p\times n}$ die Zugehörigen Kontrollwerte der jeweiligen Kontrollpunkte:

$$V = \begin{bmatrix} d_{11} & d_{12} & .. & d_{1n} \\ d_{21} & d_{22} & .. & d_{2n} \\ d_{31} & d_{32} & .. & d_{2n} \\ .. & .. & .. & \\ d_{p1} & d_{p2} & .. & d_{pn} \end{bmatrix} \tag{3.4}$$

Die Lösungen W und A des Gleichungssystems in 3.3 definieren die Interpolationsoberfläche.

Die Methode ist in SMARTy implementiert, wo für die Lösung des Gleichungssystems eine LU-Dekomposition [46] verwendet wird. [27, Kapitel 3.1]

Bei der Nutzung der Interpolationsmethode in Kombination mit POD sind die Kontrollwerte in Gleichung 3.4 durch die einzelnen Werte der in Gleichung 3.1 definierten POD Koeffizienten $a_i, i \in [1..p]$ gegeben. Die Kontrollpunkte der in 3.2 definierten Matrix sind die Ausgangsparameter der Snapshots, etwa Machzahlen und Anstellwinkel. Um nun eine neue Vorhersage zu treffen, wird die in 3.2 definierte Funktion auf die Vorhersageparameter angewendet und damit ein neuer POD Koeffizientenvektor $a_{prediction}$ erzeugt. Dieser wird, wie in Gleichung 3.1 definiert, mit dem bekannten POD Moden $U_j, \quad j \in [1..k]$ multipliziert, was den Vorhersagesnapshot ergibt.

3.3 Kombination von TPS mit Autoencodernetzwerken

In dieser Arbeit wurde die POD Methode durch ein Autoencodernetzwerk ersetzt und für die Vorhersage von neuen Snapshots mit der TPS Interpolationsmethode kombiniert, welche dabei im latenten Unterraum des Netzwerks interpoliert. Das Netzwerk wird in einer Offlinephase zunächst mit den vorhandenen Snapshots trainiert. Dabei ergibt sich folgneder Ablauf:

1. Das gesamte Autoencodernetzwerk wird darauf trainiert, die gewünschte Strömungsgröße in den m Snapshots $W_i, i \in [1..m]$ möglichst akkurat wiederzugeben, so dass $f_{decoder}(f_{encoder}(W_i)) \approx W_i, i \in [1..m]$ gilt. Dieser Schritt entspricht in der ersetzten POD Methode der Singulärwertzerlegung, wobei Enocder und Decoderteil des Netzwerks, ähnlich der POD Moden U, über alle Snapshots verwendet werden und die latenten Vektoren, ähnlich den POD Koeffizienten a_i jeweils einem Snapshot W_i zugeordnet sind.
2. Der Encoderteil des Netzwerks wird genutzt um für jeden Snapshot W_i den latenten Vektor aus der mittleren Ebene des Autoencoders zu berechnen welcher sich durch $latent_i = f_{enocder}(W_i)$ ergibt. Da für jeden Snapshot Ausgangsparameter der Strömungsberechnung bekannt sind, ergibt sich, ähnlich wie bei den vorgestellten POD Koeffizienten, je Snapshot ein Paar von zusammenhängendem latenten und Parametervektor.
3. Mit diesem Paaren von Kontrollpunkten und Werten wird die Interpolationsmethode definiert. Im Fall von TPS bedeutet dies das Aufstellen und Lösen des in 3.3 beschriebenen Gleichungssystems, wobei die in 3.4 beschriebene Matrix die m latenten Vektoren des Netzwerks beinhaltet. Dabei ergeben sich hier erneut die Matritzen $A \in \mathbb{R}^{(m+1)\times n}$ und $W \in \mathbb{R}^{p\times n}$, welche die Interpolationsoberfläche definieren.

Damit ist der Offlineschritt der Methode abgeschlossen, womit die Kombination aus TPS und Autoencoder für Vorhersagen von Strömungsgrößen für neue Parameter genutzt werden kann. Dabei ergibt sich folgender Ablauf:

1. Der TPS Interpolator berechnet für die gewünschte Parameterkombinationen einen latenten Vektor $latent_{prediction}$ mit der in 3.2 definierten Gleichung. Dabei wird die im Offlineschritt zuvor berechneten Kontrollmatritzen A und W genutzt.
2. Mit dem Dekoderteil des Netzwerks wird durch $W_{prediction} = f_{decoder}(latent_{prediction})$ ein Vorhersagesnapshot berechnet.

4 Anwendung

Im Folgenden werden mehrere Autoencodermodelle getestet und dabei ihre Genauigkeit für die Vorhersage beurteilt. Dabei wird, wie zuvor beschrieben, das jeweilige Autoencodernetzwerk mit der vorhandenen TPS Interpolationsmethode kombiniert.

4.1 Testdatensätze

In diesem Abschnitt werden die zwei Datensätze vorgestellt, welche im Offlineschritt zum Trainieren der Methode und im Onlineschritt zur Beurteilung der Vorraussagequalität genutzt wurden.

Bei beiden verwendeten Datensätzen handelt es sich um TAU Rechnungen eines zweidimensionalen NACA64A010 Profils, das Rechengitter des Profils, welches für die TAU Berechnungen der Snapshots genutzt wurde, ist in 19 gezeigt. Wie bereits erwähnt wurden für die Modelle dieser Arbeit lediglich die Druckbeiwerte auf der Profiloberfläche in das Modell einbezogen - das bedeutet, dass alle hier betrachteten Autoencodernetzwerke mit einem 200 elementigen Vektor der einzelnen Oberflächendruckwerte des Snapshots trainiert wurden. Diese Werte können in einem zweidimensionalen Plot dargestellt werden, dabei wird der Druckbeiwert in y Richtung gegenüber der normalisierten Position auf der Oberfläche dargestellt. Ein Beispiel dieser Darstellung ist, im Vergleich mit dem dazu gehörigen Druckverlauf im Volumen, in Abbildungen 20 gezeigt. Da Druckbeiwerte für Unter- und Oberseite des Profils exisitieren, ergeben sich zwei Linien im Plot. Weitere Eigenheit ist die Umkehrung der Richtung für die Y Achse. Dies ist für die Darstellung einer Druckverteilung gebräuchlich, da hierbei die Unterdruckwerte auf der Oberseite des Profils im oberen Bereich des zweidimensionalen Plots dargestellt werden. In der folgenden Auswertung der Vorhersageergebnisse für Oberflächendruckwerte wird auf diese zweidimensionale Darstellung zurückgegriffen werden.

Beide Datensätze wurden für verschiedene Machzahlen und Anstellwinkel (α) berechnet, die konkreten Parameterkombinationen ergaben sich durch Halton Sequenzen, eine Methode für die Generierung von quasizufälligen Punkten im mehrdimensionalen Parameterraum [13]. Die Datensätze unterscheiden sich in eben diesem Parameterraum, in welchem die Parameterkombinationen der Berechnungen gesampelt wurden. Der erste Datensatz wurde in einem engen, transsonischen Machbereich von $[0.74, 0.82]$ mit einem Anstellwinkel α im Bereich von $[4, 10]^\circ$ gesampelt. Die einzelnen Parameterkombinationen dieses Datensatzes sind in Abbildung 21 dargestellt, blaue Punkte stellen dabei Snapshots dar, welche für das Training im Offline Schritt verwendet wurden. Die vier in rot markierten Punkte kennzeichnen

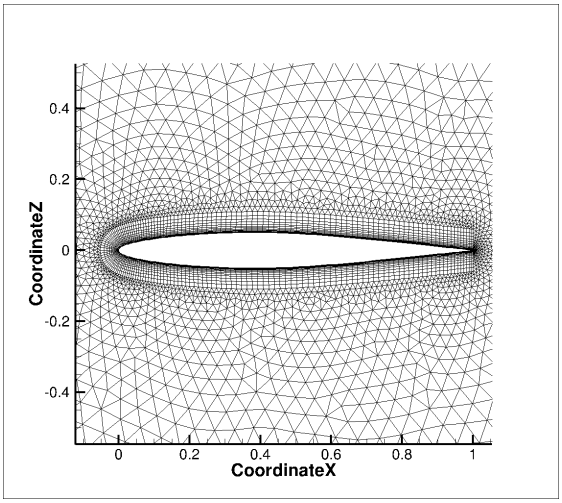


Abbildung 19: Rechengitter des NACA64 Profils welches für die vorgestellten Datensätze verwendet wurde.

vorhandenen TAU Rechnungen, welche zur Validierung der Vorhersagen des Ersatzmodells an diesen Stellen genutzt werden. Insgesamt besteht der Datensatz aus 30 Snapshots von denen 26 zum Training verwendet wurden. In allen Punkten dieses Bereiches kommt es zu transsonischem Verhalten mit einem nichtlinearen Schock, an dem der Unterdruck auf der Profiloberseite stark abfällt. Position und Ausprägung dieses varriieren zwar, insgesamt weist die Druckverteilung über alle Punkte aber einen ähnlichen Verlauf auf.

Der zweite Datensatz unterscheidet sich sowohl in Snapshotanzahl als auch Sampelbereich. Es handelt sich dabei um 250 Snapshots, von denen 240 für Trainingsprozess verwendet wurden, die übrigen zehn erneut zum Vergleich der Vorhersagen. Der Sampelbereich für die Anströmgeschwindigkeit ist in diesem Fall bedeutent größer mit Machzahlen im Bereich von $[0.63, 0.85]$. Ebenso unterscheiden sich die betrachteten Anstellwinkel stark, diese sind im Bereich von $[-4, 6]^\circ$ gesampelt worden. Dieses Sampling ist in Abbildung 22 dargestellt. Im Gegensatz zum Vorherigen unterscheiden sich die einzelnen Druckverteilungen der Snapshots des Datensatzes stark voneinander. Im oberen Machbereich tritt erneut transsonisches Verhalten mit Stößen in der Druckverteilung auf, im niedrigeren ist dies überhaupt nicht vorhanden (vgl. Abbildung 23). Zusätzlich ist die Druckverteilung im Bereich der negativen Anstellwinkel umgekehrt, so dass Unterdruckbereiche sich auf der Flügelunterseite befinden, dargestellt etwa in Abbildung 24. Der Datensatz stellt also bedeutent höhere Anforderungen an das Autoencodernetzwerk bezüglich Verallgemeinerung und der, für die Interpolation genutzten, Einbettung in den Unterraum.

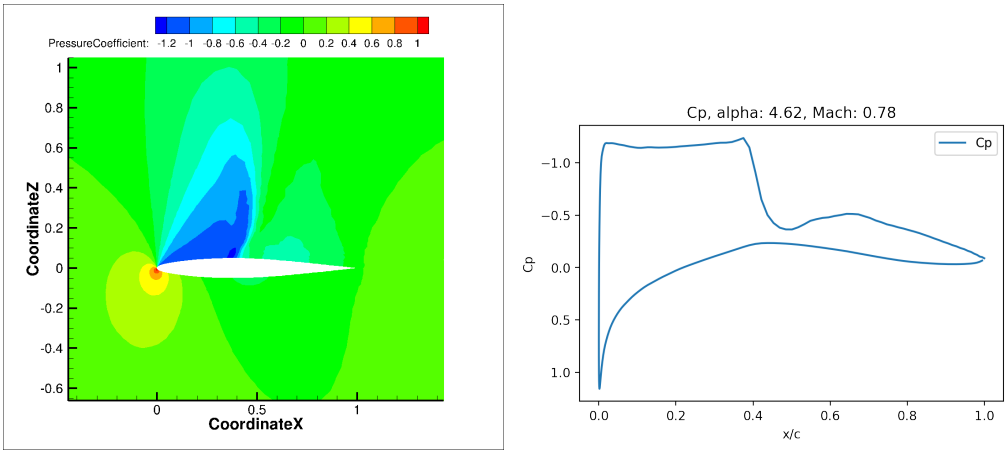


Abbildung 20: Druckverteilung um das Profil im Volumen (links) und der selbe Verlauf auf der Profiloberfläche (rechts). Da Werte auf Ober- und Unterseite des Flügels aufgetragen sind, sind jedem Punkt auf der Oberfläche x/c zwei Druckwerte zugeordnet.

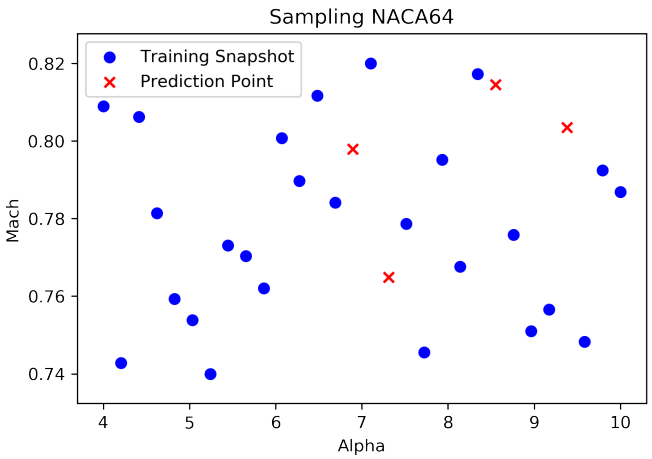


Abbildung 21: Parameter für verwendete Snapshots im Bereich Mach und Alpha des ersten Datensatzes. In blau markierte Snapshtos wurden für den Trainingsprozess verwendet, die in rot markierten zum Vergleich mit der Vorhersage.

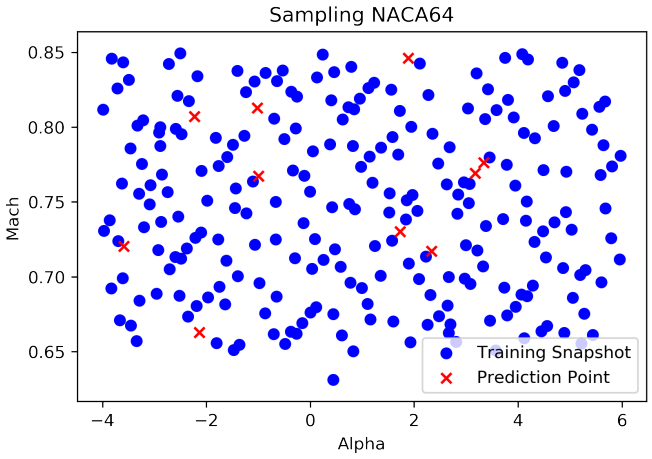


Abbildung 22: Sampling des zweiten Datensatzes mit 250 Snapshots im Bereich Mach und Alpha. Im Gengensatz zum ersten ist der Bereich der Machzahlen und Anstellwinkel deutlich größer, wodurch die Snapshots unterschiedlichere Strömungsverhalten abbilden.

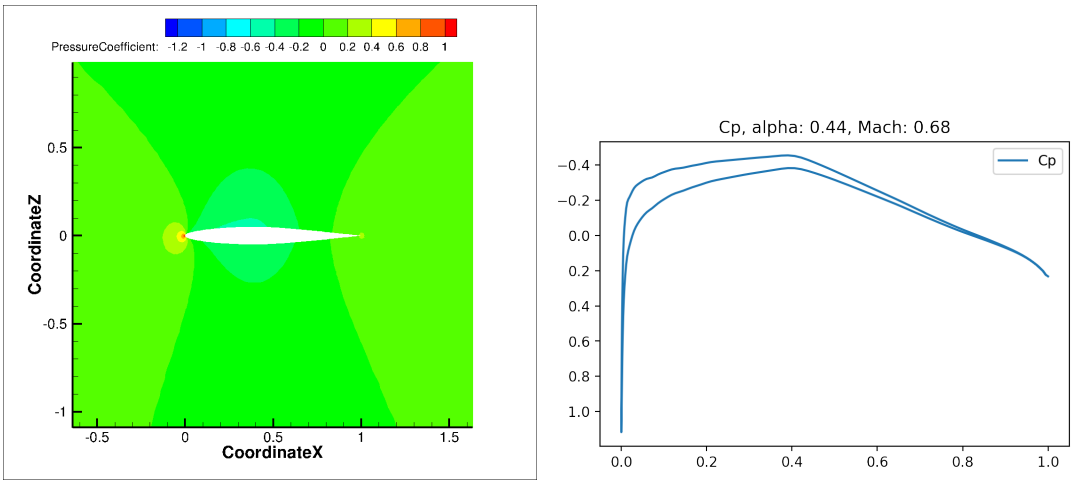


Abbildung 23: Strömungsverhalten um das betrachtete Profil im Bereich niedriger Machzahl von 0.68 - Als Volumen- (links) und Oberflächendarstellung (rechts). Es tritt, anders als bei höheren Machzahlen, kein Bereich mit niedrigem Druck und Stoßverhalten auf der Oberfläche auf.

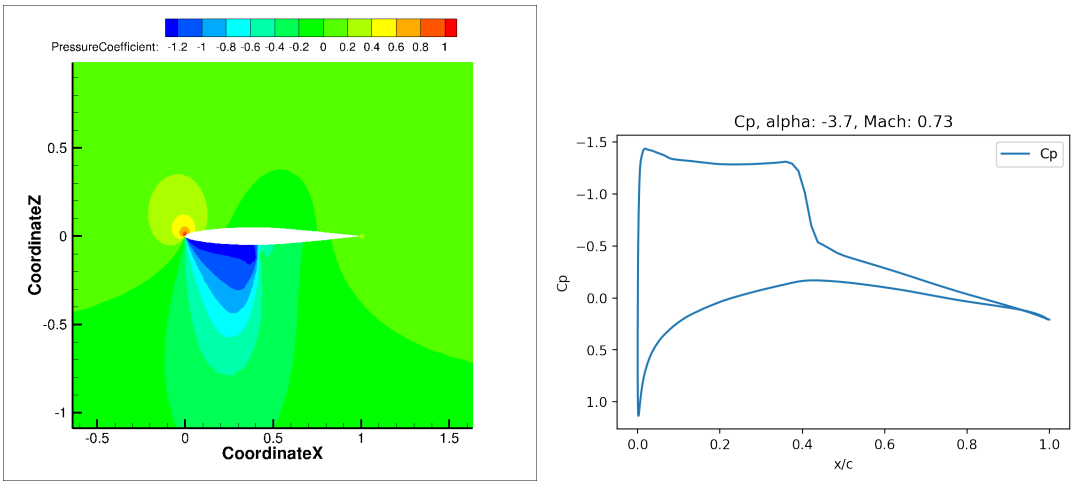


Abbildung 24: Im Bereich von negativen Machzahlen befindet sich der Bereich niedrigen Drucks und des Stoßes auf der Flügelunterseite.

4.2 Getestete Autoencodermodelle

Im Folgenden werden die drei betrachteten Autoencodernetzwerke vorgestellt - Ein flaches, vollständig verbundenes mit insgesamt 5 Ebenen, ein tiefes vollständig verbundenes mit insgesamt 11 Ebenen und ein Convolutional Neural Network mit effektiv 9 Ebenen. Alle Modelle wurden in Tensorflow, basierend auf der Keras API definiert, über welche die Modelle durch ihre einzelnen Ebenen beschrieben werden. Ebenfalls identisch ist für alle Modelle die Ein- und Ausgabegröße des 200 elementigen Vektors, welcher die einzelnen Druckbeiwerte (Cp) der Oberfläche beinhaltet. Der Vergleichbarkeit halber wurde auch die Breite der latenten Ebene, also der dimensionsreduzierten Darstellung welche für die Interpolation genutzt wurde, über alle Modelle hinweg mit einer Dimension von 25 identisch gelassen. Ebenfalls wurde für das Training aller Modelle der in Tensorflow implementierte ADAM Optimierer verwendet [19], [41], wobei auf Batchtraining aufgrund der im ersten Datensatz sehr kleinen Menge an Snapshots verzichtet wurde. Das Training erfolgte für beide Datensätze über 1000 Epochen, die gesamte Anzahl der Trainingsiterationen des größeren Datensatzes ist also entsprechend der Anzahl Snaphots um einen Faktor von ≈ 10 höher.

Alle Cp Daten wurden zur Normalisierung auf einen Bereich von $[0, 1]$ skaliert. Dies ist für den Trainingsprozess notwendig, hohe Werte können zur sofortigen Saturierung der Aktivierungsfunktion im Netzwerk führen, die somit verschwindenden Gradienteninformationen verhindern ein effektives Training der Netzwerkparameter. Zudem können Netzwerke mit $Sigmoid$ Aktivierungsfunktionen in der Ausgabeebene lediglich diesen Wertebereich wiedergeben. Die spätere Auswertung der Ergebnisse erfolgt unter Denormalisierung, so dass die originale Verteilung der Cp Daten wiederhergestellt ist.

4.2.1 Kostenfunktion

Die Wahl der Kostenfunktion beeinflusst direkt die Qualität der Reproduktion und damit auch der vorhergesagten Druckverteilung. Dabei ist die Beurteilung der Qualität nicht eindeutig, die Genauigkeit der Wiedergabe kann nach unterschiedlichen Kritierien beurteilt werden. Am einfachsten ist dabei die gemittelte paarweise Differenz zwischen den einzelnen Cp Werten, dies entspricht direkt dem Mean Squared Error (MSE) als Kostenfunktion. Anderes Kriterium zur Berwertung der Vorhersagequalität kann auch die Position des Stoßes mit starker Druckzunahme auf der Flügeloberfläche sein. Um zu Wiedergabe der Position zu quantifizieren wurde die *High Frequency Error Norm (HFEN)* mit $\sigma = 1.5$ verwendet. Die hier für alle Modellle genutzte Gesamtkostenfunktion besteht kombiniert HFEN und MSE im Verhältnis 0.4 zu 0.6.

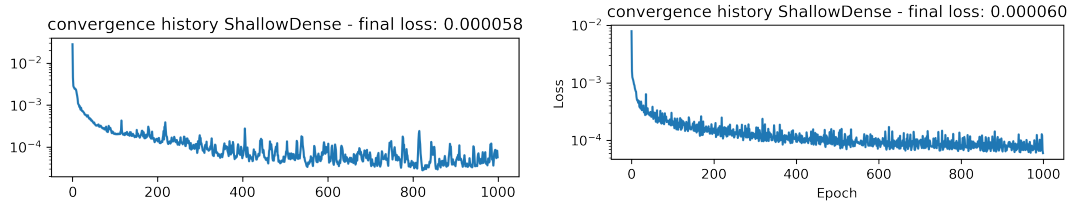


Abbildung 25: Konvergenz des flachen, vollständig verbundenden Netzwerks für ersteren Datensatz (links) und zweiteren (rechts) über je 1000 Epochen.

4.2.2 Flaches, vollständig verbundenes Netzwerk

Das flache, vollständig verbundenes Netzwerk besteht aus insgesamt fünf Ebenen, so das sowohl Encoder als auch Decoder aus einer Eingabe-, einer versteckten und einer Ausgabeebenene bestehen. Der Encoder bestand dabei aus Ebenen der Breiten 200 (Eingabe), 50 (versteckte Ebene) und 25 (Ausgabe). Selbes gilt für den Decoderteil, hier hat die Ausgabeebene entsprechend eine Breite von 200 und die Eingabe die Breite 25. Als Aktivierungsfunktionen wurden *Exponential Linear Units (ELU)* in allen versteckten Ebenen und *Sigmoid* in der Ausgabeebenene verwendet.

Dadurch ergeben sich für das Autoencodernetzwerk zusammen $22'825$ trainierbare Parameter. Die Konvergenz der Optimierung ist in Abbildung 25 gezeigt, dabei ist der Fehlerwert der Kostenfunktion über die Epochen geplottet, links für Optimierung auf ersterem, kleineren Datensatz, rechts für den zweiten, größeren. Auffällig ist dabei, dass die Größe des Datensatzes in diesem Fall Einfluss auf die Stabilität des Optimimums hat - die Konvergenz ersteren weist starke Varianz im Fehlerwert in späteren Epochen auf, ein Abbruch der Optimierung zu einem ungünstigen Zeitpunkt kann zu einer um eine halbe Größenordnung schlechtere Konvergenz führen. Dieses Problem tritt mit deutlich mehr Trainingssnapshots nicht mehr auf (Abbildung 25, rechts).

4.2.3 Tiefes, vollständig verbundenes Netzwerk

Tiefere Netzwerke haben theoretisch ein höheres Potential das Strömungsverhalten im Testfall abzubilden. Je nach Breite der genutzten versteckten Ebenen steigt bei vollständig verbundenen Netzwerken allerdings auch die Parameteranzahl stark an. In diesem Fall bestand das tiefere Netzwerk aus elf Ebenen welche im Encoderteil die Breiten 200 (Eingabe), 150, 100, 75, 50, 25 (latente Ausgabe) hatten. Der Decoder ist erneut umgekehrt mit den selben Ebenenbreiten aufgebaut. Ebenso wie beim flacheren Netzwerk werden für versteckte Ebenen ELU und für die Ausgabeebene $Sigmoid$ Aktivierungsfunktionen genutzt. Insgesamt ergeben sich aus

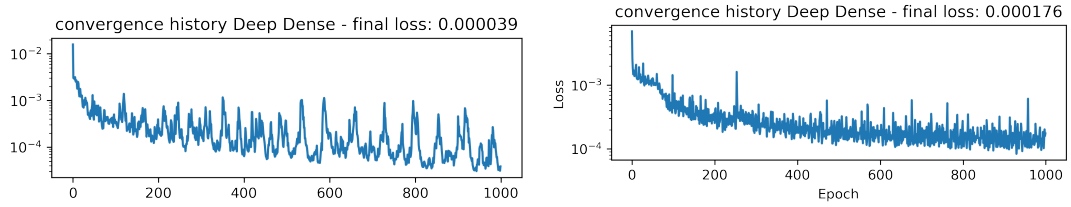


Abbildung 26: Konvergenz des tiefen, vollständig verbundenden Netzwerks für ersteren Datnesatz (links) und zweiters (rechts) über je 1000 Epochen.

dieser Konfiguration von vollständig verbundenen Ebenen $115'975$ trainierbare Parameter, im Vergleich zum vorherigen Modell also um den Faktor ≈ 5 mehr. Der Konvergenzverlauf für beide Datensätze ist in Abbildung 26 dargestellt. Trotz der höheren Parameteranzahl ist der Endwert nach selber Anzahl an Epochen in der gleichen Größenordnung wie der des vorherigen, flacheren Modells. Allerdings sind die Schwankungen der Konvergenz im Vergleich noch deutlicher und auch beim größeren Datensatz vorhanden - Dies lässt vermuten, dass für ein Modell mit dieser Parameteranzahl für sichere Konvergenz eine noch größere Menge an Trainingsdaten notwendig ist.

4.2.4 Convolutional Neural Network

Da die Druckverteilung auf dem Flügel vor allem von benachbarten Werten abhängt, sollte sich das Verhalten gut mit Convolutional Neural Networks (CNNs) abbilden lassen. Die Tatsache, dass lediglich Nachbarwerte betrachtet werden und die Gewichtung dieser invariant zur Position ist, reduziert die Anzahl an traininerbaren Parametern stark. Im Anwendungsfall ist dies besonders interessant, da in vielen Fällen die Menge an vorhandenen Snapshots eher klein ist, was das Training von Netzwerken mit vielen Parametern erschwert. Ebenso ähnelt die Erkennung und Reproduktion des Stoßes im Strömungsverlauf der Featureerkennung, welche in der Bildverarbeitung auftritt, ein Bereich in dem CNNs große Erfolge erziehlen konnten. In dieser Arbeit kam ein Convolutional Network mit den Tabelle 1 beschriebenen Ebenenparametern zum Einsatz. Nachteilig ist hier die Tatsache anzumerken, dass die Dimensionsreduzierung lediglich durch die Maxpooling Ebenen passiert - dadurch ist für den 200 elementigen Eingabevektor nur eine Reduzierung auf 25 Dimensionen mit anschließender Reproduktion möglich. Insgesamt verfügt das Netzwerk über $27'906$ trainierbare Parameter auf effektiv 9 Ebenen. Damit ist es ähnlich tief wie das zuvor beschriebene, vollständig verbundene Netzwerk, verfügt allerdings nur über ein Viertel der Parameter. Die Effekte davon sind schon während des Optimierungsprozesses in Abbildung 27 sichtbar, anders als bei den beiden bisherigen Netzwerken verläuft die Konvergenz bedeutend stabiler ohne die

Encoder				
Ebene	Filteranzahl	Kernelgröße	Aktivierungsfunktion	Ausgabegröße
Eingabe	/	/	/	200×1
Convolutional 1D	8	3	ELU	200×8
Maxpooling	/	2	/	100×1
Convolutional 1D	32	7	ELU	100×32
Maxpooling	/	2	/	50×64
Convolutional 1D	64	5	ELU	50×64
Maxpooling	/	2	/	25×64
Convolutional 1D	1	3	Linear	25×1
Decoder				
Ebene	Filteranzahl	Kernelgröße	Aktivierungsfunktion	Ausgabegröße
Eingabe	/	/	/	25×1
Convolutional 1D	64	5	ELU	25×64
Upsampling	/	2	/	50×64
Convolutional 1D	32	7	ELU	50×32
Upsampling	/	2	/	100×32
Convolutional 1D	8	3	ELU	100×8
Upsampling	/	2	/	200×8
Convolutional 1D	1	3	Linear	200×1

Tabelle 1: Ebenen des Convolutional Autoencoder Netzwerks mit ihren Aktivierungsfunktionen, Kernelgrößen und Filteranzahlen. Um die Dimensionsreduzierung im Netz zu verdeutlichen ist zudem die Ausgabegröße der einzelnen Ebenen beschrieben. Die Ausgabe des Encoders der Größe 25×1 ist gleichzeitig Eingabe des Decoders und entspricht der latenten Repräsentation welche auch für Interpolation verwendet wird.

bisher auftretenden Sprünge im Fehlerwert. Zudem verläuft die Optimierung in Bezug auf die Epochenanzahl schneller, so dass ein ähnlicher Fehlerwert bereits nach der Hälfte der Epochen erreicht wird.

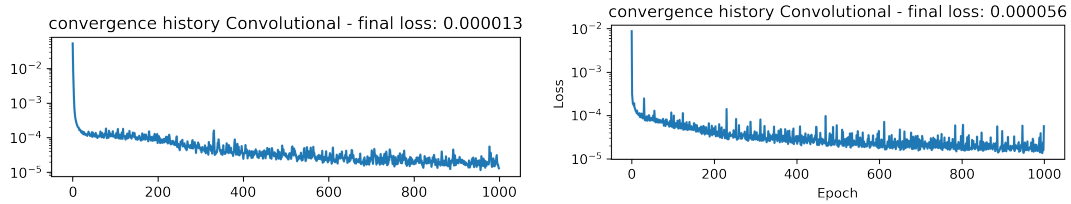


Abbildung 27: Konvergenz des CNNs für ersteren Datensatz (links) und zweiters (rechts) über je 1000 Epochen.

4.3 Ergebnisse - Rekonstruktion

Im folgenden ist die direkte Rekonstruktion eines Snapshots aus dem Trainingsdatensatz gezeigt. Dabei ergeben sich die Fehlerwerte der, für die Optimierung genutzten, Kostenfunktion aus den Abweichungen im Druckverlauf über alle diese Trainingssnapshots. Interessant ist dabei auch, ob die Möglichkeit des Autoencoders die Ausgangsdaten in einer nichtlinearen Weise zu reduzieren eine im Vergleich zur POD Methode genauere Wiedergabe der Trainingsdaten ermöglicht.

Diese Rekonstruktion für das erste vorgestellte **flache, vollständig verbundende Autoencodernetzwerk** ist für jeweils einen Snapshot der beiden betrachteten Datensätze in Abbildung 28 dargestellt. Zusätzlich zum Orginal (schwarz) ist hier auch die Rekonstruktion durch die POD Methode (rot) mit ebenfalls 25 Dimensionen aufgetragen. Sichtbar ist direkt, dass das Netzwerk nicht in der Lage ist, eine bessere Rekonstruktion als die POD Methode zu liefern, letztere trifft in beiden Datensätzen mit einer Einschränkung auf 25 Dimensionen das Original sehr gut. Das flache Netzwerk hat im transsonischen Fall deutliche Probleme die Stoßkante scharf abzubilden und vefügt auch in glatten Bereichen über größere Schwankungen in der Wiedergabe des Druckverlaufs, sehr ausgeprägt vor allem im zweiten Testfall (Abbildung 28, unten). Die Tatsache, dass das Netzwerk den Druckverlauf nicht glatt wiedergiebt, lässt sich direkt durch die genutzte Kostenfunktion erklären. Diese betrachtet die Abweichung nur für jeweils einzelne Vektorelemente, so dass eine Abweichung nach unten mit folgender Abweichung nach oben den selben Fehler wie glatterer abweichender Verlauf ergibt. Somit besteht während des Trainings kein Grund diesen gegenüber einem glatteren Verlauf mit gleichmäßigerer Abweichung zu erlernen.

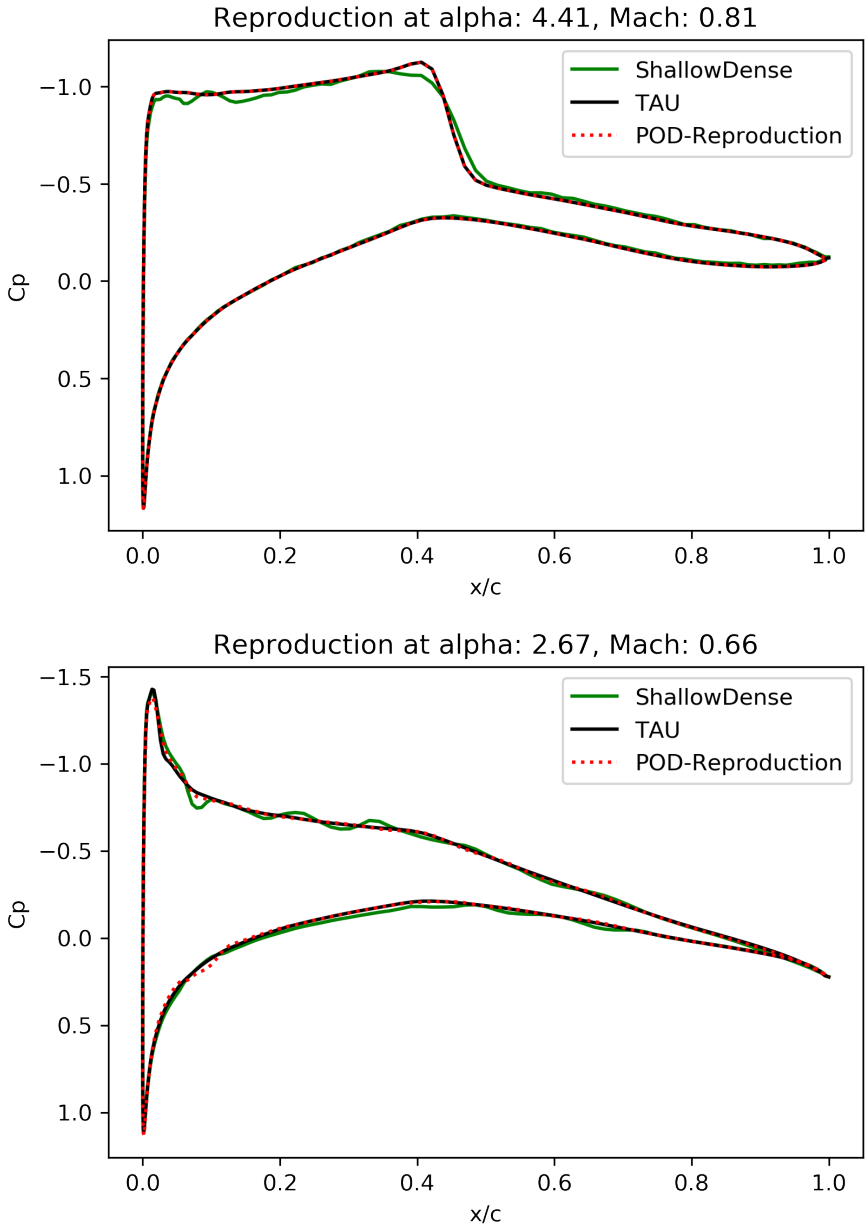


Abbildung 28: Reproduktion des flachen, vollständig verbundenden Autoencodernetzwerks (grün) für Snapshots, welche im Trainingsset enthalten waren. Dazu im Vergleich das Original (schwarz) und die Rekonstruktion der POD Methode (rot), welche auf die selbe Anzahl an Dimensionen reduziert. Oberer Plot ist Rekonstruktion des, mit dem ersten Datensatz trainierten Modells, unterer Plot selbes für zweiten Datensatz.

Ein **tieferes, vollständig verbundendes Netzwerk**, mit dem Potential komplexere Zusammenhänge abzubilden, könnte die, hinter POD mit selber Anzahl Dimensionen, zurückbleibenden Rekonstruktionsgenauigkeit verbessern. Diese Rekonstruktion des 4.2.3 beschriebenen Modells ist in Abbildung 29 dargestellt, hier im Vergleich mit de vorherigen, flacheren Modell. In diesem Fall gelingt es dem tieferen Modell im transsonischen Fall (29, oben) eine leicht bessere Rekonstruktion, während diese im zweiten Fall (29, unten) im Vergleich zum vorherigen Modell sogar schlechter ist. Auch das tiefere Netzwerk konnte also keine mit der POD Methode vergleichbare Genauigkeit erreichen.

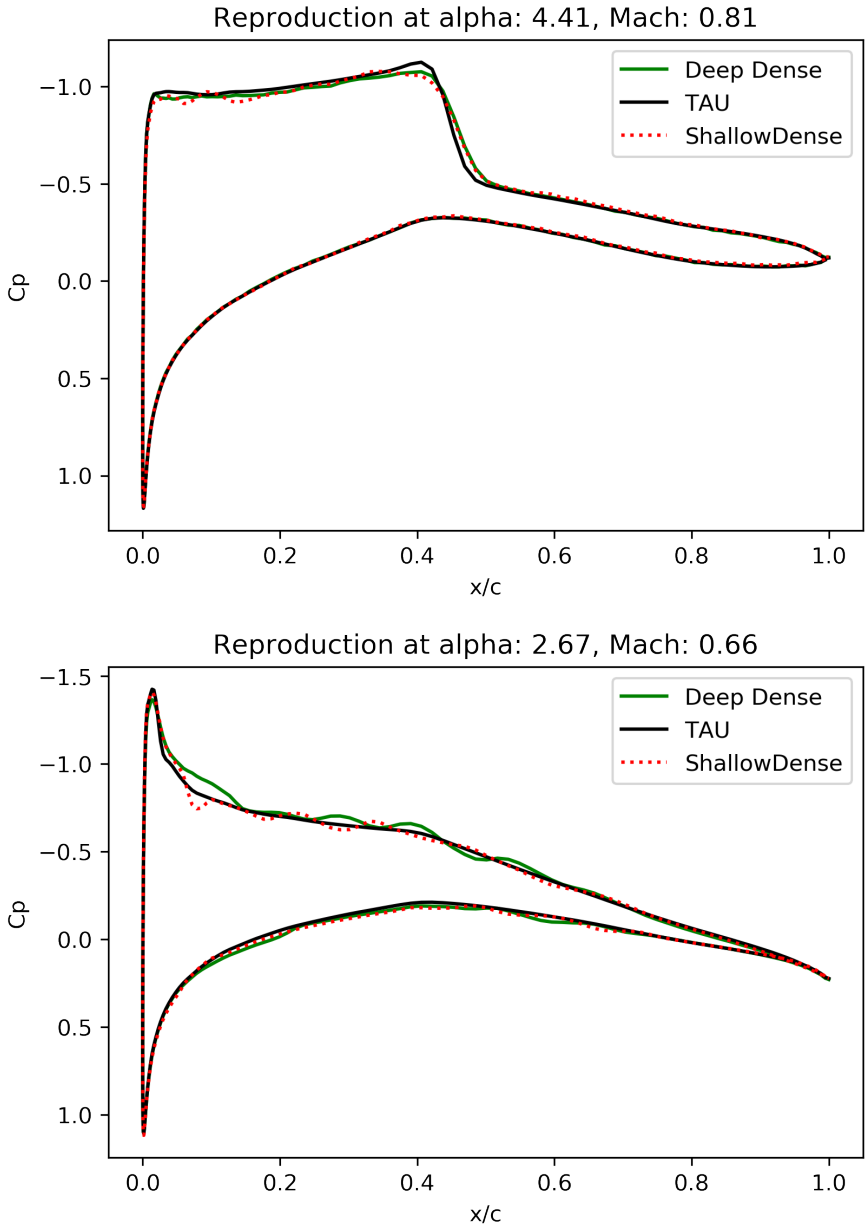


Abbildung 29: Reproduktion des tiefen, vollständig verbundenden Autoencodernetzwerks (grün) für Snapshots, welche im Trainingsset enthalten waren. Dazu im Vergleich erneut das Original (schwarz) und die Rekonstruktion des flachen Netzwerks (rot). Dargestellt ein Snapshot aus dem transsonischen, ersteren Datensatz oben und ein Snapshot des zweiten unten.

Der **Convolutional Autoencoder** verfügt über eine relativ geringere Anzahl an lernbaren Parametern bei einer mit dem vorherigen, tieferen Modell vergleichbarer Ebenenzahl. Dadurch ist das Netzwerk potentiall ebenso gut in der Lage den Verlauf wiederzugeben und hat sich im Optimierungsvorgang durch die geringe Parameterzahl als vergleichsweise stabil erwiesen (Abbildung 27). Die Rekonstruktionsergebnisse dieses Modells sind in Abbildung 30 dargestellt, erneut im Vergleich mit dem ersten, flachen Modell (rot). Hier zeigt sich, dass das Netzwerk die Stoßkante deutlich passender abbilden konnte und nicht zu einer geglätteten Wiedergabe dieser neigt. Insgesamt liegt das Convolutional Netzwerk für beide Datensätze nahezu genau auf dem Original und ist somit in der Rekonstruktion mit der POD Methode vergleichbar.

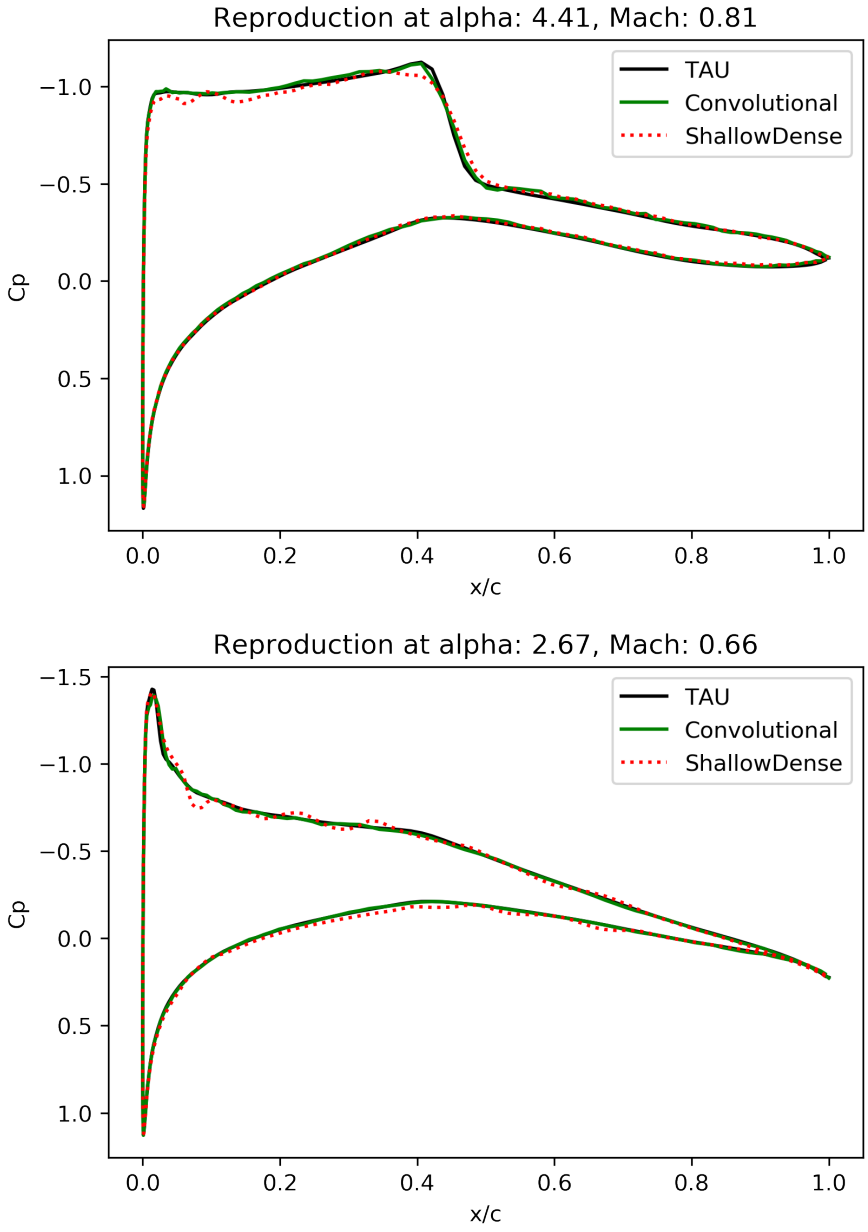


Abbildung 30: Reproduktion des Convolutional Autoencodernetzwerks (grün) für Snapshots, welche im Trainingsset enthalten waren. Dazu im Vergleich erneut das Original (schwarz) und die Rekonstruktion des flachen Netzwerks (rot). Dargestellt ein Snapshot aus dem transsonischen, ersteren Datensatz oben und ein Snapshot des zweiten unten.

4.4 Ergebnisse - Vorhersage

Dieses Kapitel stellt die Qualität der Vorhersagen aller drei Modelle dar. Erneut dient dabei die POD Methode in Kombination mit TPS als Vergleich. In diesem Fall werden hier für beide Datensätze des Umfangs halber jeweils zwei Snapshots betrachtet.

In Abbildung 31 dargestellt ist die Vorhersage des **flachen, vollständig verbundenden Netzkwerks** für einen Snapshot im Inneren des Samplingbereichs, in diesem Fall existieren für die Interpolatonsmethode Kontrollpunkte in beide Richtungen. Die Vorhersage durch die Kombination von flachem Autoencoder und Interpolation trifft den Druckverlauf in großen Bereichen sehr gut, scheitert allerdings bei der Wiederabe der Stoßkante, wobei die Vohersage der der POD Methode ähnelt. Dabei ist das Netzwerk in der Lage die Vorhersage zu konstruieren, sofern der korrekte latente Vektor (blau) verwendet wird. Die Vorhersage der genauen Stoßkante scheitert also vor allem an der Interpolationsmethode.

Dies ist anders in der in Abbildung 32 dargestellten Vorhersage. In diesem Fall scheitern sowohl POD+TPS als auch Autoencoder+TPS an der Vorhersage des Stoßes und liefern einen unphysikalischen Druckverlauf, wobei die Autoencodermethode letztendlich eine im Vergleich bessere Vorhersage liefert. Anders als im vorherigen Punkt liegt das Problem nicht nur an der Interpolationsmethode, wie die fehlerhafte Konstruktion der Vorhersage (blaue) zeigt.

Abbildung 33 und Abbildung 34 zeigen jeweils eine Vorhersage des Modells im zweiten, größeren Datensatz. In Abbildung Abbildung 33 liefert das Autoencodernetzwerk im Vergleich mit POD+TPS erneut eine minimal bessere Vorhersage welche das Verhalten an der Stoßkante leicht besser abbildet. Die Interpolationsmethode trifft in diesem Fall den realen latenten Vektor sehr gut, wie der lediglich minimal abweichende Verlauf der Rekonstruktion (blau) zeigt.

Das Vorhersageverhalten an der Stoßkante ist umgekehrt in Abbildung 34, hier liefert POD+TPS eine besseres Ergebniss. Auch wenn die Vorhersage im Außenbereich des Sampelbereichs stattfindet, erfolgt die Extrapolation hier sehr genau, die reale Rekonstruktion weicht nur minimal von der Vorhersage ab.

Insgesamt kann das flache, vollständig verbundende Autoencodernetzwerk bei der Vorhersage eine gegenüber POD+TPS leicht verbesserte Genauigkeit liefern. Damit sind die Vorhersageergebnisse bedeutent besser als die zuvor vorgestellte Reproduktion alleine, welche in der Wiederhabe deutlich hinter POD+TPS zurücklag. Für letztes Verfahren scheint die TPS Methode Probleme zu haben, die für die Vorhersage richtigen POD Koeffizienten zu finden. Für das Autoencodernetzwerk hingegen kann die TPS Interpolationsmethode in den

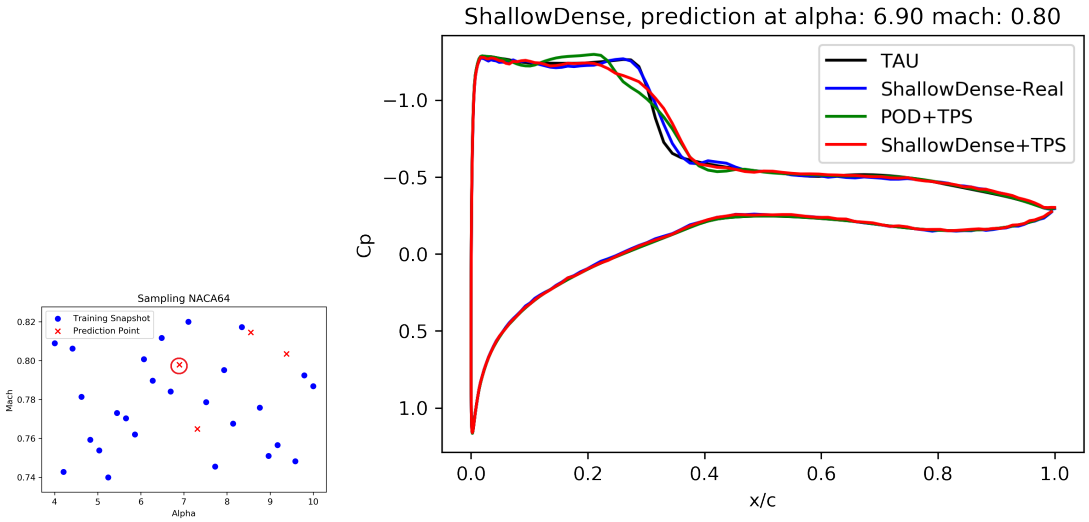


Abbildung 31: Druckverlauf (rechts) für einen Punkt im inneren Samplingbereich (links) in rot und reale Rekonstruktion des TAU Originals in grün. Im Vergleich dazu POD in Kombination mit TPS Interpolator (rot).

meisten Fällen den richtigen latenten Vektor bilden, lediglich im Außenbereich des ersten, transsonischen Datensatzes kommt es dabei zu Abweichungen.

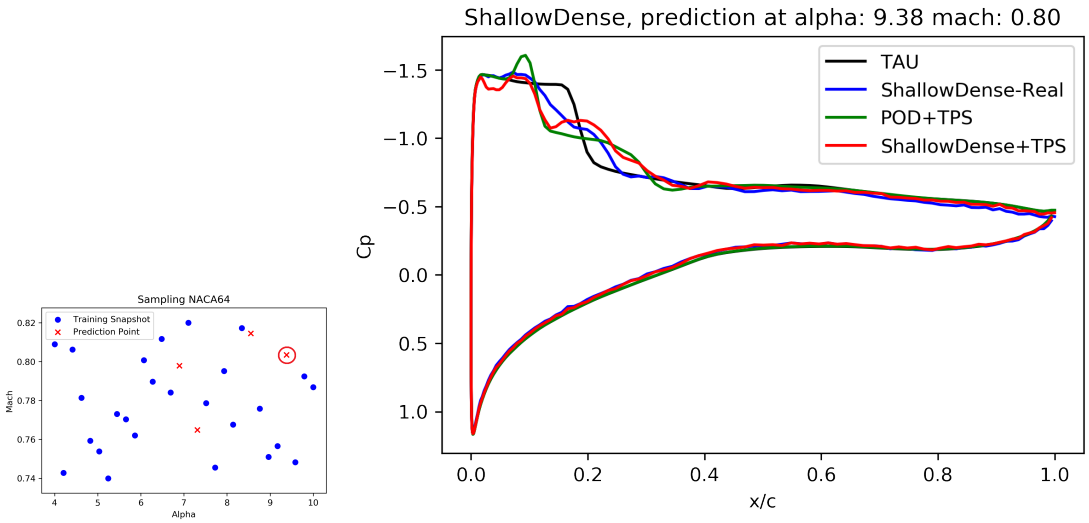


Abbildung 32: Vorhersage für einen Punkt im äußeren Samplingbereich (rot) und reale Reproduktion des TAU Originals (schwarz). Im Vergleich dazu POD in Kombination mit TPS Interpolator (grün). In diesem Fall findet durch TPS eine Extrapolation am Rand des Samplingbereichs statt.

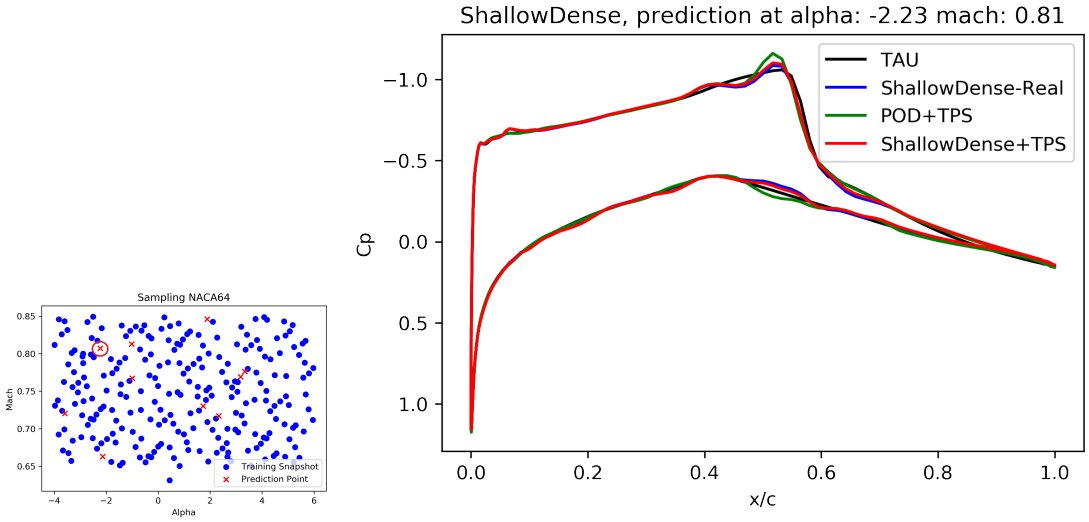


Abbildung 33: Vorhersage des flachen Autoencodernetzwerks Innenbereich des zweiten, größeren Datensatzes.

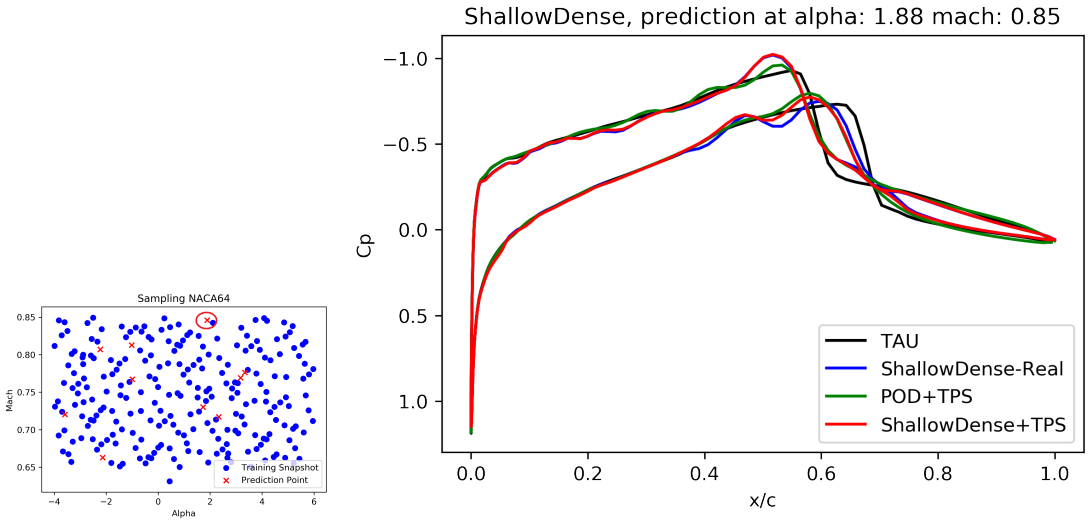


Abbildung 34: Vorhersage des flachen Autoencodernetzwerks im zweiten, größeren Datensatz. Hier findet erneut eine Extrapolation am Rande des Samplingbereichs statt.

Im folgenden wird analog zum vorgherigen Modell die Vorhersagegenauigkeit des **tiefen, vollständig verbundenden Autoencodernetzwerks** betrachtet. Dabei findet hier nicht mehr ein Vergleich mit POD+TPS sondern mit dem vorherigen, flachen Modell statt. Es zeigt sich, dass das aufwendigere, tiefere Modell die Vorgersagegenauigkeit nicht verbessern kann. Im besten Fall (Abbildung 35) erfolgt durch das Netzwerk eine leicht verbesserte Vorhersage der Stoßkante, in anderen Fällen erfolgt die Vorhersage dieser entweder ähnlich fehlerhaft (Abbildung 36) oder schlechter (Abbildung 37). Die größten Abweichungen sind in der Vorsage des komplexeren Strömungsverlauf in Abbildung 38 sichtbar, hier unterscheidet sich die Druckvorhersage am Profilende deutlich, zudem gibt es größere Ungenauigkeiten im vorderen Bereich niedrigeren Drucks. Dabei ist vor allem in diesem Beispiel die Genauigkeit des tieferen Netzwerks signifikant schlechter als die des einfachere, flacheren Modells. Erneut liegt hier, außer im Extrapolationsbereich in Abbildung 35, die interpolierte Vorhersage sehr nah an der tatsächlichen Rekonstruktion durch das Netzwerk, die Fehlerquelle der Vorhersage liegt also bei der unzureichenden Fähigkeit des Netzwerkes die Rekonstruktion über das Trainingsset hinaus zu verallgemeinern.

Letztendlich ist dieses Netzwerk trotz deutlich gestiegenden Berechnungskosten und dadurch größerem Trainingsaufwand nicht in der Lage eine bessere Genauigkeit als das einfachere, zuvor vorgestellte Netzwerk zu liefern. Die eingeschränkte Fähigkeit der Verallgemeinerung könnte sich allerdings durch eine größere Menge an Trainingssnapshots verbessern.

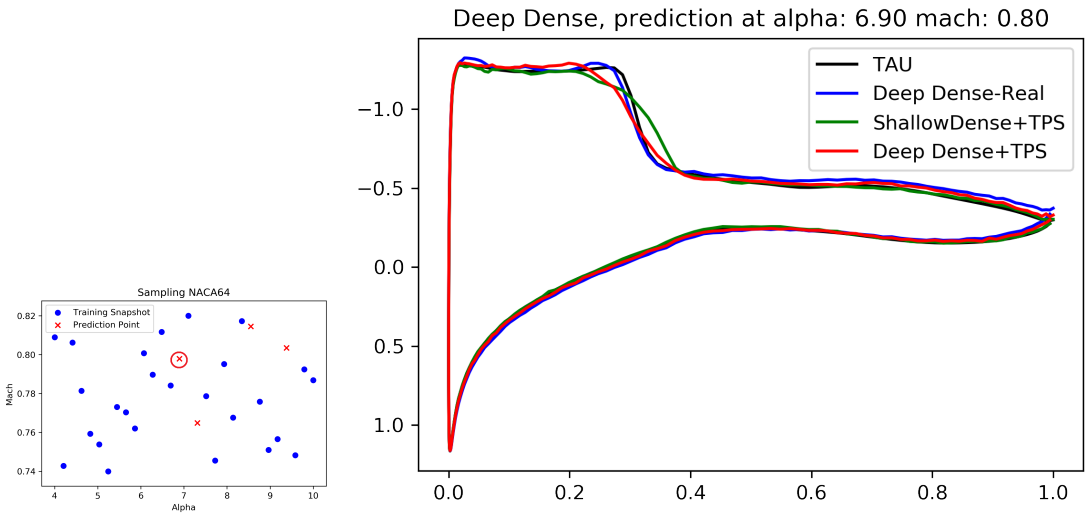


Abbildung 35: Vorhersage des tiefen Autoencodernetzwerks im Extrapolationsbereich des im ersten, transsonischen Datensatzes.

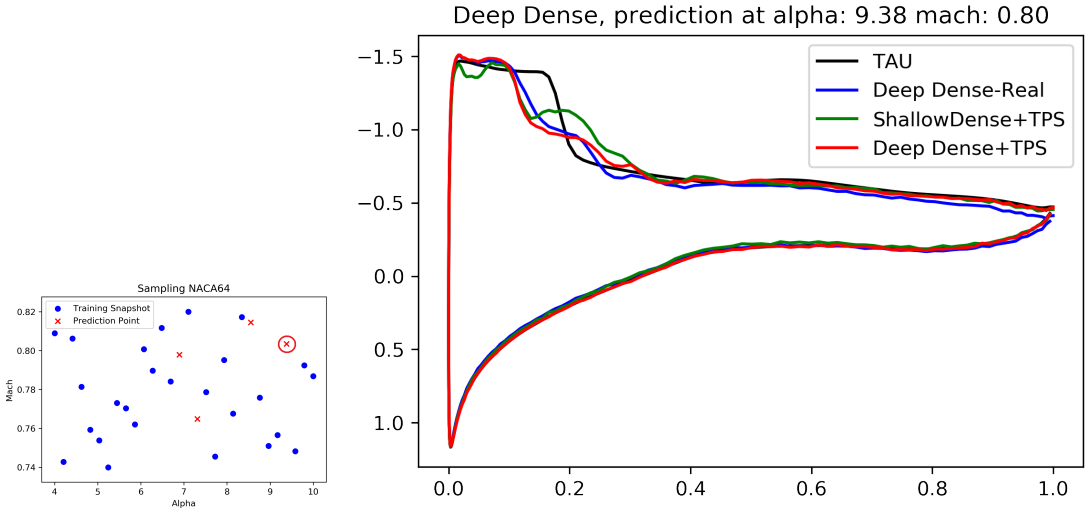


Abbildung 36: Vorhersage des tiefen Autoencodernetzwerks im Innenbereich des im ersten, transsonischen Datensatzes.

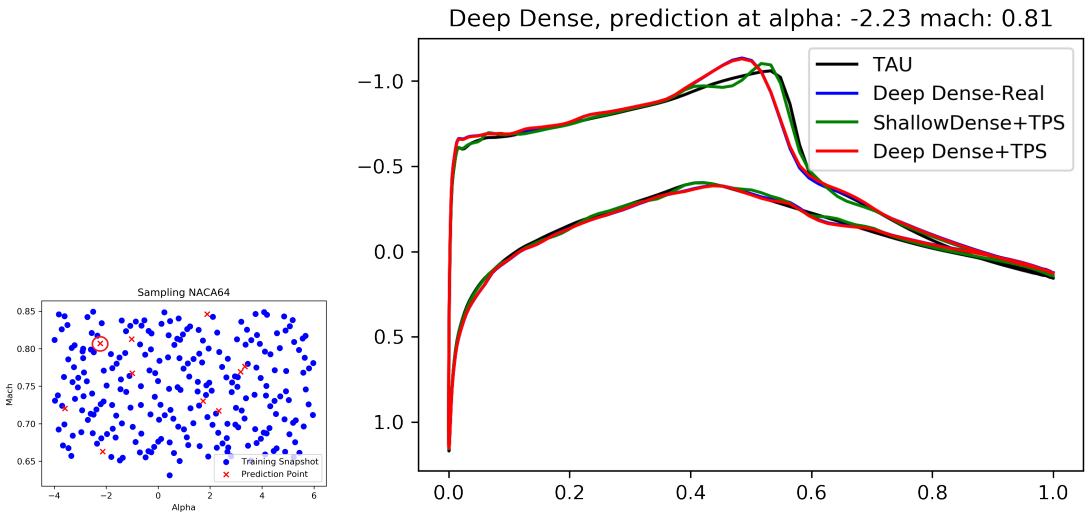


Abbildung 37: Vorhersage des tiefen Autoencodernetzwerks im Innenbereich des zweiten, größeren Datensatzes.

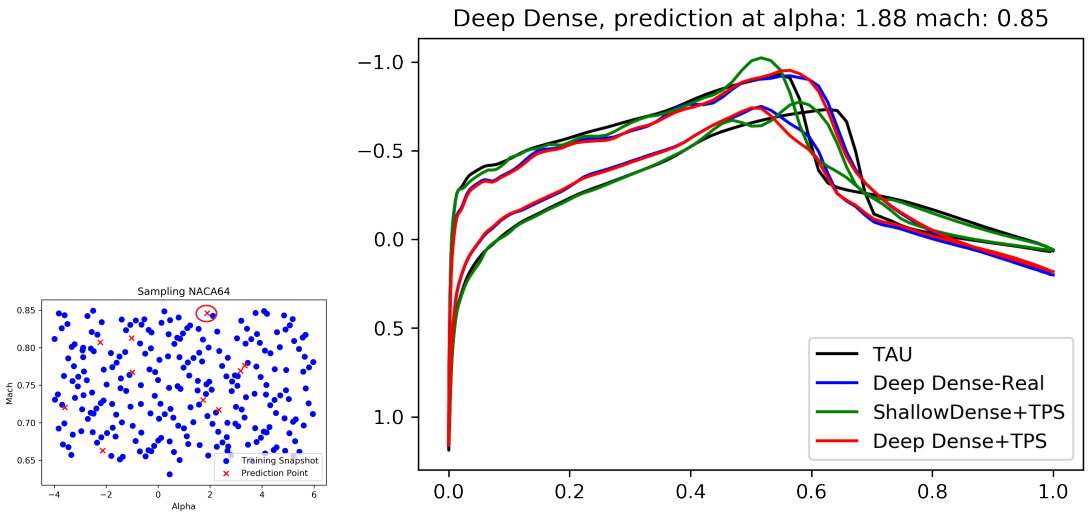


Abbildung 38: Vorhersage des tiefen Autoencodernetzwerks im zweiten, größeren Datensatz. Hier findet erneut eine Extrapolation am Rande des Samplingbereichs statt.

Im folgenden wird die Vorhersage des **Convolutional Autoencodernetzwerks** betrachtet, dabei dient erneut das erste, flache Modell als Vergleich. Das Convolutional Modell lieferte bereits bei der Rekonstruktion sehr gute Ergebnisse, und kann dies auch auf die Vorhersage übertragen. Für den erste, transsonische Datensatz kann das Modell eine Vorherage liefern, welche nahezu mit dem Original identisch ist (Abbildung 39). Im Außenbereich (Abbildung 40) bei Extrapolation weicht die Vorhersage im Stoßbereich stark vom Orginal ab. Wie an der realen Reproduktion des Netzwerks sichtbar ist, liegt dies allerdings an der Interpolationsmethode, anders als die bisherigen Modelle kann das CNN den Stoß prinzipell genau reproduzieren.

Ein ähnliches Verhalten tritt auch im zweiten, größeren Datensatz für den Strömungsverlauf in Abbildung 42 auf. Zwar ist hier die Vorhersage des Convolutional Netzwerks in Kombination mit Interpolation deutlich genauer als die vorherigen Modelle, weicht aber immer noch vom originalen Verlauf ab. Hier hat die Interpolationsmethode erneut Probleme den passenden latenten Vektor zu berechnen, ist dieser bekannt (Abbildung 42, blau), kann das Netzwerk prinzipell das Original mit deutlich kleinerne Abweichungen reproduzieren. Im Innenbereich des Samplings (Abbildung 41 treten keine Extrapolationsprobleme auf, hier ist die Vorhersage dem Original sehr nahe und mit der realen Reproduktion des Netzwerks praktisch identisch.

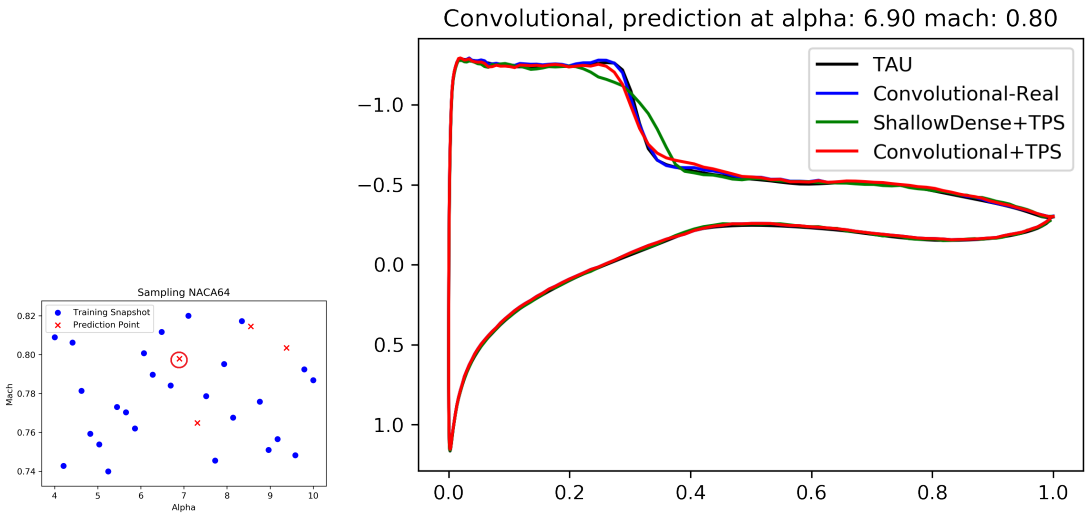


Abbildung 39: Vorhersage des Convolutional Autoencodernetzwerks im Samplinginnenbereich des ersten, transsonischen Datensatzes.

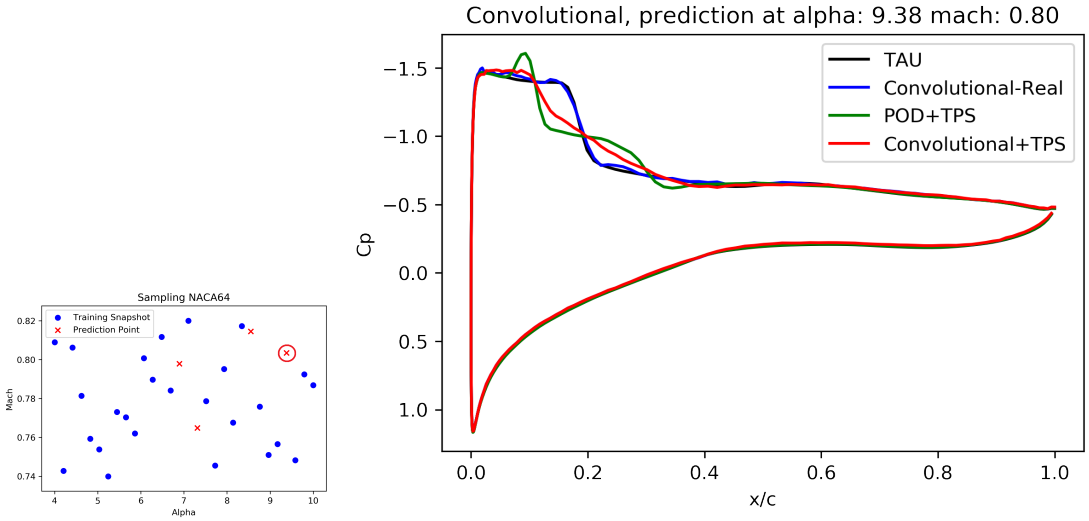


Abbildung 40: Vorhersage des Convolutional Autoencodernetzwerks im Extrapolationsbereich des im ersten, transsonischen Datensatz.

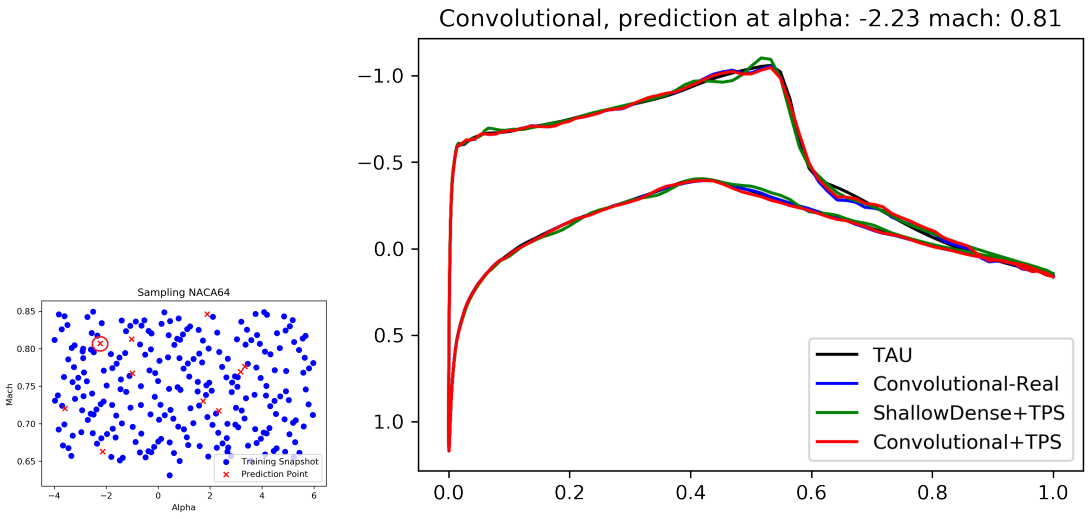


Abbildung 41: Vorhersage des Convolutional Autoencodernetzwerks im Innenbereich des zweiten, größeren Datensatzes.

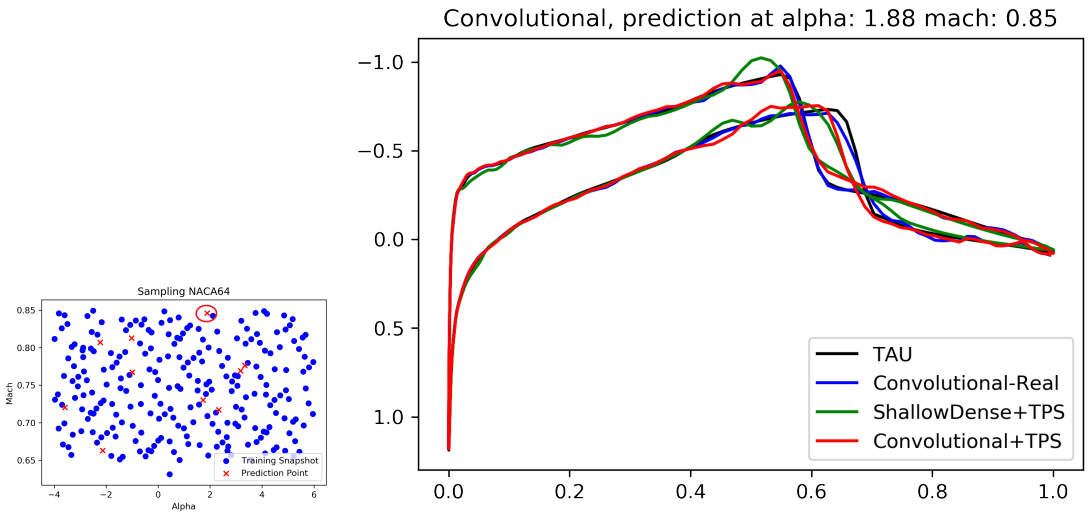


Abbildung 42: Vorhersage des tiefen Autoencodernetzwerks im zweiten, größeren Datensatz. Hier findet erneut eine Extrapolation am Rande des Samplingbereichs statt.

4.5 Berechnungsaufwand

Da das Ziel der Ersatzmodellierung die schnelle Vorhersage des Strömungsverlauf ist, spielt der Berechnungsaufwand neben der Genauigkeit eine große Rolle. Vor allem der Convolutional Autoencoder konnte in letzterer Kategorie sehr gute Ergebnisse liefern, welche dem Original deutlich näher als die Vorhersage mit POD+TPS waren, allerdings ist die Methode bedeutend rechenaufwendiger als die Singulärwertzerlegung der Snapshotmatrix in der POD Methode. Dabei muss, sowohl für Autoencoder als auch POD, zwischen Offline und Online Kosten unterschieden werden. Offlinekosten beeinhalten den Berechnungsaufwand für das Training des neuronalen Netzwerks, oder im Fall von POD, der Singulärwertzerlegung und die Berechnung der TPS Koeffizienten für beide Methoden. Dazu kommt gegebenenfalls die Berechnungszeit der TAU Snapshots, welche für das Training benötigt werden. Online Kosten bezeichnen die Rechenzeit, welche für eine Vorhersage notwendig ist. In diesem Fall bedeutet dies die Berechnung eines neuen latenten Vektors durch TPS und der Feedforward Schritt mit diesem durch das Decodernetzwerk. Für POD ist ebenfalls die Interpolation durch TPS notwendig, dazu kommt die in Gleichung 3.1 beschriebene Rekonstuktion eines Snapshots aus POD Moden und POD Koeffizienten.

Die Online- und Offlinekosten der einzelnen Modelle sind in den Tabellen 3 und 2 dargestellt. Dabei handelt es sich um die reale Zeit (auch *wall clock time*), gemessen durch die Differenz zwischen Start- und Endzeitpunkt. Alle Schritte wurden auf einer `Intel(R) Xeon(R) E3-1276 v3 @ 3.60GHz` CPU berechnet, es wurde also keine GPU für die Beschleunigung der in Tensorflow implementierten neuronalen Netzwerke verwendet, ebenso kamen keine `AVX2` Vektorinstruktionen [28] zum Einsatz. Praktisch die gesamten Offlinekosten entstehen dabei durch das Training des jeweiligen Autoencodernetzwerks, zudem ist die Trainingszeit des Convolutional Autoencoders merklich höher als die des tieferen, vollständig verbundenden Autoencoders mit mehr Parametern. Verglichen mit den Berechnungszeiten für POD sind die Offlinekosten des Autoencoders um mehrere Größenordnungen höher. Dies betrachtet allerdings nicht die Berechnungszeit der einzelnen TAU Snapshots, welche pro Snapshot etwa $180s$ in Anspruch nimmt. Damit machen die Trainingskosten der Modellbildung selbst für den betrachteten kleineren Datensatz einen Anteil an der gesamten Rechenkosten von weniger als 2% aus. Die Onlinekosten (Tabelle 3) der auf Neuronalen Netzwerken basierenden Modelle ist deutlich näher an der POD+TPS Methode und lediglich bis zu einem Faktor vier höher, insgesamt bewegen sich diese aber immer noch im Bereich von unter einer Millisekunde. Verglichen mit einer vollständigen TAU Rechnung, welche durch diese Vorhersage ersetzt wird, ensteht damit eine Zeiteinsparung um fünf Größenordnungen.

Kleiner Datensatz (26 Snapshots)				
Schritt	flacher AE	tiefer AE	Convolutional AE	POD
Dimensionsreduktion	$23.2s$	$36.9s$	$59.8s$	$1.32ms$
Berechnung TPS Koeffizienten	$0.75ms$	$0.83ms$	$1.96ms$	$0.58ms$
Großer Datensatz (240 Snapshots)				
Schritt	flacher AE	tiefer AE	Convolutional AE	POD
Dimensionsreduktion	$3.43min$	$6.2min$	$9.35min$	$15.9ms$
Berechnung TPS Koeffizienten	$6.17ms$	$8.65ms$	$23.5ms$	$9.5ms$

Tabelle 2: Offlinekosten der getesteten flachen, tiefen und Convolutional Autoencodermodelle im Vergleich mit POD. Neben der Dimensionsreduktion durch Training des Netzwerks oder im Fall von POD der Singulärwertzerlegung kommt die Berechnung der TPS Koeffizienten zu den Offlinekosten dazu.

Kleiner Datensatz (26 Snapshots)				
Schritt	flacher AE	tiefer AE	Convolutional AE	POD
Rekonstruktion	$397\mu s$	$407\mu s$	$642\mu s$	$109.9\mu s$
TPS Interpolation	$89.3\mu s$	$87.3\mu s$	$91.3\mu s$	$90.1\mu s$
Gesamtvorhersage	$486.3\mu s$	$489.3\mu s$	$733.3\mu s$	$200\mu s$
Großer Datensatz (240 Snapshots)				
Schritt	flacher AE	tiefer AE	Convolutional AE	POD
Rekonstruktion	$479\mu s$	$495\mu s$	$733\mu s$	$107.8\mu s$
TPS Interpolation	$95.0\mu s$	$94.1\mu s$	$95.0\mu s$	$94.2\mu s$
Gesamtvorhersage	$574\mu s$	$598.1\mu s$	$828\mu s$	$202\mu s$

Tabelle 3: Onlinekosten der getesteten flachen, tiefen und Convolutional Decodermodelle im Vergleich mit POD. Alle vier Modelle wurden mit TPS Interpolation genutzt.

4.6 Skalierbarkeit

Da der in dieser Arbeit betrachtete Testfall eines zweidimensionalen Flügelprofils mit lediglich 200 Oberflächendruckwerten nicht umbedingt im Flugzeugentwurf praxisnah ist, spielt die Skalierbarkeit der auf neuronalen Netzwerken basierenden Methoden eine große Rolle. Die 200 elementigen Eingabe kann für Volumenvorraussagen oder dreidimensionale Modelle bei entsprechend größeren Gittern um einen Faktor 1000 oder auch $1'000'000$ für Volumenvorraussagen steigen. Dies ist vor allem für die beiden vollständig verbundenden Netzwerke problematisch, da bei diesen die Parameteranzahl durch die individuelle Gewichtung jedes Vektorelements stark steigt - Ein hypothetisches Netzwerk mit einer $200'000$ elementigen Eingabeebene und einer, wie im vorgestellten kleinen Modell, darauf folgenden 100 elementigen Ebene hätte hier bereits 20 Millionen trainierbare Gewichtungsparameter. Wächst die Größe der zweiten Ebene, steigt damit auch linear die Anzahl der Parameter zwischen diesen. Bereits das mit $115'000$ Parametern deutlich kleinere, tiefere Netzwerk hat gezeigt, dass selbst mit der relativ großen Menge von 240 Snapshots Verallgemeinerungsprobleme entstehen.

Diese Skalierungsprobleme treten beim Einsatz des Convolutional Netzwerks nicht auf. Da hier die Gewichtsmatrix in Kernelgröße über die gesamte Ebene geteilt wird, ist die Anzahl an trainierbaren Parametern unabhänig von der Eingabegröße und wird durch die Anzahl an Ebenen und Filterkanälen definiert. Da die Dimensionsreduzierung lediglich durch Pooling Ebenen stattfindet, muss die Kernelgröße dieser entsprechend angepasst werden, um eine passende Anzahl an Ebenen im Netzwerk zu erhalten. Dabei gilt durch die Reduzierung im Netzwerk folgender Zusammenhang zwischen Ebenenanzahl n, Eingabedimension w_e, Kernelgröße der Poolingebenen k und reduzierter Dimension w_l:

$$w_e = w_l * k^n$$

womit sich eine Netzwerktiefe von

$$n = log_k(\frac{w_e}{w_l})$$

ergibt. Da das Abbildungspotential und die Trainingskomplexität des Convolutional Netzwerks stark von der Ebenenanzahl abhängt, macht es Sinn, diese je nach Problemfall zu wählen. Mit einer konstanten Ebenenzahl n ergibt sich die Kernelgröße:

$$k = (\frac{w_e}{w_l})^{\frac{1}{n}}$$

Da sowohl Ebenentiefen und Kernelgrößen nur als natürliche Zahlen sinnvoll sind, handelt es

sich bei beiden Gleichungen zunächst nur um Richtgrößen.

Dabei ist die Tiefe des Netzkwerks ein Hyperparameter, welcher manuell je nach Anwendungsfall bestimmt werden muss. Eine automatische Bestimmung oder Abstschätzung dieser erfordert weitere Versuche der Methode mit anderen Datensätzen.

5 Einbettung im Unterraum

Bei den in 4.4 dargestellten Ergebnissen trat wiederholt der Fall auf, dass das Autoencodernetzwerk zwar in der Lage war, bei korrekten latenten Vektor die passende Vorherage wiederzugeben, dieser allerdings nicht passend durch die TPS Interpolation gebildet werden konnte. Dies passierte vor allem im Randbereich, also bei Extrapolation im Samplingset (etwa in Abbildung 40). Für alle Vorhersagefälle muss die TPS Methode innerhalb des Unterraums auf der definierten Interpolationsfläche einen Punkt berechnen, welche mit den Ausgangsparametern (Mach und α) korrespondiert. Um zu verstehen, warum dies in vielen Fällen sehr genau funktioniert (etwa in Abbildung 41), aber im obigen Fall nicht gelingt, wird in diesem Kaptitel eben diese Einbettung im Unterraum betrachtet. Da die Darstellung eines 25 dimensionalen Unterraums ohne erneute Dimensionsreduzierung nicht visualisierbar ist, wurde hierfür ein weiteres Modell gebildet, welches die Trainingssnapshots auf zwei Dimensionen reduziert. Dieses wurde analog zu dem in 4.2.2 beschriebenen Modell definiert, einziger Unterschied ist die Breite der latenten Ebene welche nun aus zwei anstatt 25 Neuronen besteht. Die Einbettung des Modells für ersteren Datensatz ist in der Abbildung 43 dargestellt, dabei sind die auf x- und y-Achse die einzelnen Einbettungsvektoren der Snapshots. Diese sind farblich nach Ausgangsparameter markiert, rechts entspricht der Farbverlauf der Ausgangsmachzahl, links dem Anstellwinkel. In dem im linken Bild dargestellten Verlauf des Anstellwinkels ist ein sehr linearer Verauf von unten links nach oben rechts erkennnbar, zudem ist der Abstand der Einbettungen nahezu linear in Bezug auf den Ausgangsparameter. Allerdings sind bei hohen Anstellwinkeln um $\alpha \approx 10^\circ$ einige Ausreißer sichtbar. Ein ähnliches Verhalten ist im rechten Bild zu sehen, wo die selben Einbettungsvektoren mit korrespondierenden Machzahl aufgetragen sind. Hier ist ein Verlauf von unten rechts nach oben links erkennbar, also nahezu orthogonal zum Verlauf der Anstellwinkel. Allerdings gibt es in dieser Einbettung deutlich mehr Außreißer als bei der Einbettung der Anstellwinkel. Dabei sind es die selben Snapshots in der Einbettung, welche sowohl im Verlauf des Anstellwinkels, als auch der Machzahl Ausreißer darstellen.

Abseits dieser Ausreißer finden sich die Ausgangsvariablen der Rechnungen, also Machzahl und Anstellwinkel, in der latenten Ebene des Netzwerks in einem nahezu linearen Verlauf wieder. Das Netzwerk konnte also die, den Snapshots zugrundeliegenden latenten Ausgangsvariablen im Unterraum wiederherstellen, ohne das die Informationen zu diesen in das Training einflossen.

Diese Einbettung kann ebenso für die POD Methode visualisiert werden, dies ist in Abbildung 44 sichtbar. Diese ist, im Vergleich zum Autoencodernetzwerk, weniger gleichmäßig innerhalb des Raums verteilt. Für den Verlauf der Machzahl ist ein ähnlich glatter Verlauf

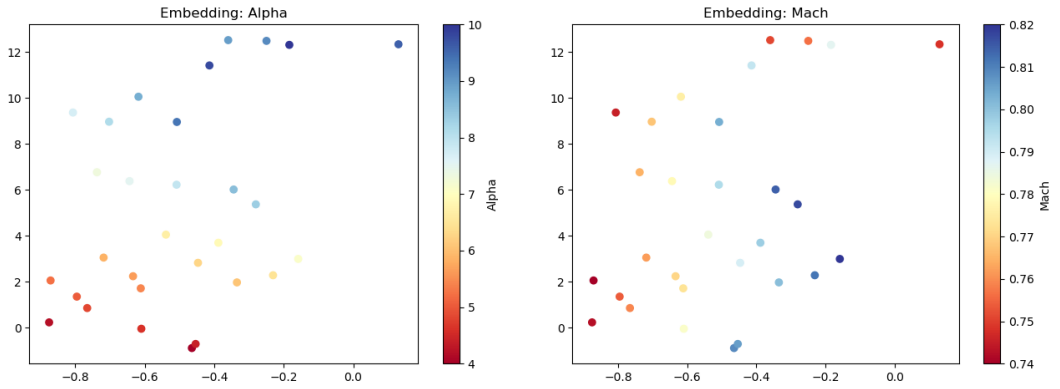


Abbildung 43: Einbettung des ersten Datensatzes aus 30 Snapshots in einen zweidimensionale Unterraum durch ein flaches, vollständig verbundendes Autoencodernetzwerk.

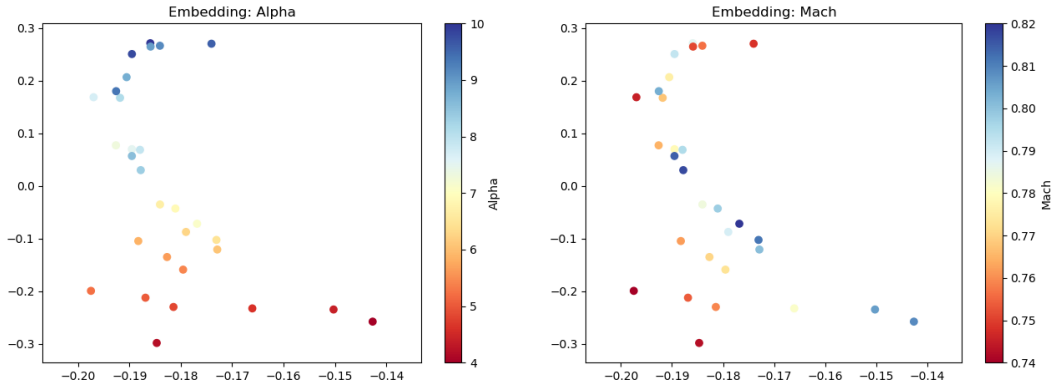


Abbildung 44: Einbettung des ersten Datensatzes aus 30 Snapshots in einen zweidimensionale Unterraum durch die POD Methode.

gegeben, für die Einbettung des Anstellwinkels im rechten Bild ist dies nicht mehr der Fall. Zwar sind stellenweise im unteren, linken Bereich Einbettungen mit stetig steigender Machzahl vorhanden, bei Snapshots mit steigendem Anstellwinkel ab $\geq 6^\circ$ ist ein Zusammenhang zwischen Einbettung und Machzahl nur noch bedingt gegeben. Diese Unterschiede der Einbettung machen deutlich, warum das flache Autoencodernetzwerk bei teils deutlich ungenauerer Rekonstruktion der Snapshots im Vergleich zu POD ähnliche gute bis bessere Vorhersageergebnisse liefern konnte.

Letztendlich ist eine Dimensionsreduzierung auf zwei Dimensionen eine untypische Verwendung der Methode, welche sich nicht für genaue Vorhersageergebnisse eignet. Die betrachteten Ergebnisse lassen allerdings einen ähnlichen Verlauf mit selbigen Interpolationsproblemen in einem höherdimensionalen Unterraum erwarten.

6 Fazit und Ausblick

Zusammenfassend wurde in dieser Arbeit die Anwendung von Autoencdernetzwerken in der Ersatzmodellierung in der Aerodynamik evaluiert. Ziel der Ersatzmodellierung ist es, eine Alternative zu vollständigen CFD Rechnungen zu liefern. Deren numerische Simulation bildet das aerodynamische Verhalten durch partielle Differentialgleichungen ab, was zwar genaue Ergebnisse liefert aber äußerst Rechenaufwendig ist. Ersatzmodelle ermöglichen es Strömungsverhalten zwar mit verminderter Genauigkeit, dafür aber mit deutlich niedrigerem Rechenaufwand vorherzusagen. Diese gesteigerte Geschwindigkeit ist vor allem im Flugzeugentwurf relevant, wo in der Entwurfsphase eine Vielzahl an Konfigurationen berechnet werden muss, was mit vollständigen CFD Berechnungen nicht machtbar ist.

In der DLR Toolbox SMARTy sind solche Methoden zur Ersatzmodellierung implementiert. Die gebräuchlichste davon ist die Kombination einer Dimensionsreduzierung durch eine Hauptkomponentenanalyse (*Proper Orthogonal Decomposition, POD*) mit der *Thin Plate Spline* Interpolationsmethode. Mit diesen Methoden werden datengetriebene Modelle reduzierter Ordnung gebildet, welche aus vollständigen CFD Lösungen bestehen. Diese Lösungen werden durch POD auf einen niedrigdimensionalen Unterraum übertragen, in welchem mit der TPS Methode interpoliert werden kann. Dadurch lassen sich Vorhersagen für Strömungsparameter treffen, welche nicht in der Menge der zugrunde liegenden CFD Rechnungen vorhanden sind.

Die hier betrachteten Autoencodernetzwerke ersetzten die in SMARTy vorhandende Dimensionsreduzierung durch POD mit einem neuronalen Netzwerk zur Dimensionsreduzierung. Der Vorteil von Neuronalen Netzwerken liegt in deren Möglichkeit nichtlineare Beziehungen in Daten abzubilden, während die Hauptkomponentenanalyse eine lineare Methode ist, welche sich durch Matrixmultiplikationen beschrieben lässt. Diese Möglichkeit nichtlineare Vorgänge zu modellieren ist vor allem für das stark nichtlineare aerodynamische Verhalten im transsonischen Bereich interessant. Dabei werden die Autoencodernetzwerke darauf trainiert, die Trainingsdaten der CFD Lösungen möglichst genau wiederzugeben. Um eine Dimensionsreduzierung zu erziehlen, ist die mittlere Ebene dieser Netzwerke in ihrere Breite stark eingeschränkt, so dass das Netzwerk gezwungen ist, in dieser eine effiziente Darstellung der Daten zu erlernen. Die Abbildung der Daten auf diesem Unterraum wird, ebenso wie bei der Hauptkomponentenanalyse, genuzt, um auf diesem zwischen einzelnen Lösungen zu interpolieren und darüber eine Vorhersage der Strömungsbedingungen für Parameter zwischen den bereits vorhandenden zu treffen. Dabei kam in dieser Arbeit die in SMARTy vorhandende TPS Interpolationsmethode zum Einsatz.

In dieser Arbeit wurde die Vorhersage durch die Kombination von Autoencodernetzwerken

und TPS Interpolation dabei, neben der originalen CFD Lösung, mit der bereits vorhandenden POD+TPS Methode verglichen. Dabei konnten alle drei Autoencodernetzwerke im Vergleich mit POD ähnlich gute bis deutlich bessere Vorhersageergebnisse liefern, wobei es deutliche Unterschiede zwischen den verschiedenen Modellen gab. Es hat sich gezeigt, dass im konkreten Problemfall der Einsatz von tieferen Netzwerken, trotz teils sehr guten Ergebnissen in anderen Feldern, hier nicht zu besseren Vorhersagen führt. Hauptgrund dafür scheint die hohe Anzahl an trainierbaren Parametern in Kombination mit der vergleichsweise kleinen Anzahl an Trainingsdaten zu sein. Dieses Problem konnte das getestete Convolutional Neuronal Network beheben, welches bei vergleichbar Tiefe eine stark veringere Zahl an Gewichtungswerten aufweist. Insgesamt konnte dieses Netzwerk die besten Vorhersagen liefern und hat von allen drei getesteten Modellen die beste Skalierbarkeit auf industriell relevantere Problemgrößen.

Die durchweg gute Vorhersagequalität, vor allem in Anbetracht der gegenüber POD teils schlechteren Rekonstruktion, lässt sich durch eine linearere Einbettung der einzelnen Snapshots im reduzierten Unterraum erklären, was die Interpolationsergebnisse in diesem verbesserte. Insgesamt steigen durch die Nutzung der aufwendigeren Neuronalen Netzwerke allerdings die Offlinekosten, also benötigte Rechenzeit zur Bildung des Modells, im Vergleich zu POD stark an. Dabei sind die Onlinekosten für die Generierung einer neuen Vorhersage ähnlich, was im Vergleich zur vollständigen CFD Berechnung eine um mehrere Größenordnungen schneller Vorhersage durch das Modell ermöglicht, welche gleichzeitig die realen Strömungsverhältnisse im Vergleich zu POD besser abbildet.

Da in dieser Arbeit die Autoencdernetzwerke lediglich auf ein zweidimensionales Profilbeispiel angewendet wurde, wäre die Validierung dieser an einem industriellen, dreidimensionalen Modell sinnvoll. Dabei stellen sich für einen solchen Testfall diverse Fragen zur Performance und Skalierbarkeit der Methode.

Da bei dem in dieser Arbeit betrachteten Modellen und Datensätzen Trainingszeiten des Neuronalen Netzwerks die Berechnungskosten dominierten, ist bei größeren Modellen und Testfällen der Einsatz von Beschleunigern hier sehr relevant. Durch die Nutzung der Tensorflow Library ist eine einfacher Ausführung der Modelle auf GPUs machbar, was, im Vergleich zur hier genutzten Berechnung auf der CPU, hohe Geschwindigkeitssteigerungen mit sich bringen kann. Ebenso wurde im betrachteten Fall eine Batchgröße von eins genutzt, höhere Werte können, vor allem auf GPUs, durch die Ausnutzung höherer Parallelität, die Berechnung erneut beschleunigen.

Neben der reinen Rechenzeit ist auch die Skalierbarkeit der Methode bezüglich Parameteranzahl für komplexere Fälle wichtig, wobei Convolutional Neuronal Networks sich hierfür prinzipbedingt gut eignen. Dabei ist die Übertragung der Methode, welche zunächst lediglich

für karthesische Strukturen wie Bilder definiert ist, auf die in der CFD Berechnung genutzten unstrukturierten Rechengitter notwendig. Möglichkeit dafür ist etwa eine Projektion des unstrukturierten Gitters auf eine karthesische Darstellung in einem Preprocessing Schritt oder die Nutzung von Graph Convolutional Networks [18] welche direkt auf beliebig definierten Graphen arbeiten.

Im Laufe der Arbeit hat sich zudem gezeigt, dass die Einbettung in den Unterraum für die Kombination mit einer Interpolationsmethode stark die Qualität der Vorhersagen beeinflusst. Im betrachteten Testfall war diese im latenten Vektor größtenteils gleichmäßig verteilt, obwohl keinerlei Regularisierungsmethoden hierfür zum Einsatz kamen. Fraglich ist dabei, ob dies der Fall ist, weil das Netzwerk zur besseren Reproduktion alle Samples mit möglichst hohem Abstand in diesem Raum verteilt. Die gleichmäßige Verteilung der Ausgangsparameter Machzahl und Anstellwinkel in der latenten Repräsentation,welche für genaue Interpolationsergebnisse sorgt, wäre somit nur Nebeneffekt des ebenfalls gleichmäßig verteilten Samplings der Snapshots über die Ausgangsparameter. Um dies zu testen, könnte das selbe Modell auf ein ungleichmäßiges Sampling, etwa mit mehreren dichteren Clustern an Snapshots, angewendet werden. Sind die einzelnen Snapshots erneut unabhänig dieser Ausgangsparameter gleichmäßig in der latenten Repräsentation verteilt, könnte dies zu ungenauerer Interpolation führen, da der ebene Verlauf der Ausgangsparameter nicht mehr gegeben ist.

Es existieren für Autoencodernetzwerke verschiedene Weiterentwicklungen, welche die Einbettung beeinflussen und damit prinzipiell geeigneter für die Generierung neuer Vorhersagen sind. Dazu zählen etwa Variational Autoencoder [20] oder die Nutzung eines gegensätzlichen Netzwerks zur Regularisierung der latenten Ebene [3]. Diese Weiterentwicklungen haben das Potential die in den Extrapolationsbereichen fehlerhaften Vorhersagen durch die Interpolation zu verbessern.

Literatur

[1] *3 Types of Gradient Descent Algorithms for Small & Large Data Sets*. [Online; accessed 26. Jul. 2019]. März 2017. URL: https://www.hackerearth.com/blog/developers/3-types-gradient-descent-algorithms-small-large-data-sets.

[2] Yoshua Bengio. "Learning deep architectures for AI". In: *Foundations and Trends in Machine Learning* (2009). DOI: 10.1561/2200000006.

[3] David Berthelot u. a. "Understanding and Improving Interpolation in Autoencoders via an Adversarial Regularizer". In: *arXiv* (Juli 2018). eprint: 1807.07543. URL: https://arxiv.org/abs/1807.07543.

[4] Chakravarty R Alla Chaitanya u. a. "Interactive reconstruction of Monte Carlo image sequences using a recurrent denoising autoencoder". In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), S. 98.

[5] Lei Chen. *Curse of Dimensionality*. Hrsg. von LING LIU und M. TAMER ZSU. Boston, MA: Springer US, 2009, S. 545–546. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_133. URL: https://doi.org/10.1007/978-0-387-39940-9_133.

[6] Choromanska, Anna and Henaff, Mikael and Mathieu, Michael and Arous, Gérard Ben and LeCun, Yann. "The Loss Surfaces of Multilayer Networks". In: *arXiv* (Nov. 2014). eprint: 1412.0233. URL: https://arxiv.org/abs/1412.0233.

[7] Sergey Demyanov. "Regularization methods for neural networks and related models". Diss. 2015.

[8] Gianluca Donato und Serge Belongie. "Approximation Methods for Thin Plate Spline Mappings and Principal Warps". In: *Transformation of Datasets in Linear-based Map Conflation Framework. Surveying and Land Information Systems*, S. 159–169.

[9] *File:KMeans-Gaussian-data.svg - Wikimedia Commons*. [Online; accessed 18. Jul. 2019]. Juli 2019. URL: https://commons.wikimedia.org/wiki/File:KMeans-Gaussian-data.svg.

[10] Thomas Franz. "Reduced-order modeling for steady transonic flows via manifold learning". Diss. DLR, Deutsches Zentrum für Luft-und Raumfahrt, 2016.

[11] Xavier Glorot, Antoine Bordes und Y Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011* 15 (Jan. 2011), S. 315–323.

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

[13] J. H. Halton. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". In: *Numer. Math.* 2.1 (Dez. 1960), S. 84–90. ISSN: 0029-599X. DOI: 10.1007/BF01386213.

[14] Boris Hanin. "Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations". In: *arXiv* (Aug. 2017). eprint: 1708.02691. URL: https://arxiv.org/abs/1708.02691.

[15] Kaiming He u. a. "Deep Residual Learning for Image Recognition". In: *arXiv* (Dez. 2015). eprint: 1512.03385. URL: https://arxiv.org/abs/1512.03385.

[16] Matthew Hutson. "AI researchers allege that machine learning is alchemy". In: *Science | AAAS* (Mai 2018). DOI: 10.1126/science.aau0577.

[17] *ImageNet Large Scale Visual Recognition Competition (ILSVRC)*. [Online; accessed 30. Jul. 2019]. Juli 2019. URL: http://image-net.org/challenges/LSVRC.

[18] Tobias Skovgaard Jepsen. "How to do Deep Learning on Graphs with Graph Convolutional Networks". In: *Medium* (Mai 2019). URL: https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780.

[19] Diederik P. Kingma und Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv* (Dez. 2014). eprint: 1412.6980. URL: https://arxiv.org/abs/1412.6980.

[20] Diederik P. Kingma und Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv* (Dez. 2013). eprint: 1312.6114. URL: https://arxiv.org/abs/1312.6114.

[21] Alex Krizhevsky, Ilya Sutskever und Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Hrsg. von F. Pereira u. a. Curran Associates, Inc., 2012, S. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[22] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (Mai 2017), S. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[23] N. Kroll u. a. "DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods". In: *CEAS Aeronaut. J.* 7.1 (März 2016), S. 3–27. ISSN: 1869-5582. DOI: 10.1007/s13272-015-0179-7.

[24] Y. Lecun u. a. "Gradient-based learning applied to document recognition". In: *Proc. IEEE* 86.11 (Nov. 1998), S. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[25] Yann LeCun und Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[26] Prof. Seungchul Lee. *Convolutional Neural Networks (CNN)*. [Online; accessed 30. Jul. 2019]. Juli 2019. URL: http://i-systems.github.io/HSE545/machine%20learning%20all/Workshop/180208_COSEIK/06_CNN.html.

[27] M. Loehndorf und J. M. Melenk. "On thin plate spline interpolation". In: *arXiv* (Mai 2017). DOI: 10.1007/978-3-319-65870-4. eprint: 1705.05178.

[28] Chris Lomont. *Introduction to Intel® Advanced Vector Extensions*. [Online; accessed 1. Sep. 2019]. Juni 2017. URL: https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions.

[29] Laurens van der Maaten und Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.Nov (2008), S. 2579–2605. ISSN: 1533-7928. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[30] Martıén Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[31] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673.

[32] Kevin P. Murphy. "Machine learning - a probabilistic perspective". In: *Adaptive computation and machine learning series*. 2012.

[33] Syed Sadat Nazrul. "The DOs and DON'Ts of Principal Component Analysis". In: *Medium* (Juli 2018). URL: https://medium.com/@sadatnazrul/the-dos-and-donts-of-principal-component-analysis-7c2e9dc8cc48.

[34] *OpenCV: Structured forests for fast edge detection*. [Online; accessed 5. Aug. 2019]. Aug. 2019. URL: https://docs.opencv.org/3.1.0/d0/da5/tutorial_ximgproc_prediction.html.

[35] Dr. Sebastian Raschka. *Fitting a model via closed-form equations vs. Gradient Descent vs Stochastic Gradient Descent vs Mini-Batch Learning. What is the difference?* [Online; accessed 26. Jul. 2019]. Juli 2019. URL: https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html.

[36] David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Okt. 1986), S. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[37] D. Schwamborn, T. Gerhold und R. Kessler. "DLR-TAU Code - an Overview". In: *1st ONERA/DLR Aerospace Symposium, Paris, 21.-24. Juni 1999*. LIDO-Berichtsjahr=1999, 1999, S4–2-S4-10. URL: https://elib.dlr.de/13547/.

[38] *Singular Value Decomposition (SVD) tutorial*. [Online; accessed 14. Aug. 2019]. Sep. 2002. URL: https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm.

[39] Devin Soni. "Supervised vs. Unsupervised Learning". In: *Medium* (Juli 2019). URL: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d.

[40] Nitish Srivastava u. a. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), S. 1929–1958.

[41] *tf.train.AdamOptimizer | TensorFlow Core r1.14*. [Online; accessed 26. Aug. 2019]. Aug. 2019. URL: https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer.

[42] *Training Neural Networks: Best Practices | Machine Learning Crash Course*. [Online; accessed 29. Jul. 2019]. März 2019. URL: https://developers.google.com/machine-learning/crash-course/training-neural-networks/best-practices.

[43] Ludovic Trottier, Philippe Giguère und Brahim Chaib-draa. "Parametric Exponential Linear Unit for Deep Convolutional Neural Networks". In: *arXiv* (Mai 2016). eprint: 1605.09332. URL: https://arxiv.org/abs/1605.09332.

[44] Andre Violante. *An Introduction to t-SNE with Python Example*. [Online; accessed 1. Aug. 2019]. Aug. 2019. URL: https://www.kdnuggets.com/2018/08/introduction-t-sne-python.html.

[45] Ruye Wang. *Laplacian of Gaussian (LoG)*. [Online; accessed 9. Sep. 2019]. Dez. 2018. URL: http://fourier.eng.hmc.edu/e161/lectures/gradient/node8.html.

[46] Eric W. Weisstein. *LU Decomposition*. [Online; accessed 13. Aug. 2019]. Aug. 2019. URL: http://mathworld.wolfram.com/LUDecomposition.html.

[47] Eric W. Weisstein. *Singular Value Decomposition*. [Online; accessed 21. Aug. 2019]. Aug. 2019. URL: http://mathworld.wolfram.com/SingularValueDecomposition.html.

[48] Gordon Wetzstein. *Gordon Wetzstein - MIT Media Lab*. [Online; accessed 22. Aug. 2019]. Feb. 2003. URL: http://web.media.mit.edu/~gordonw/private/index.php.