Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Bachelorthesis

zur Erlangung des akademischen Grades

Bachelor of Engineering (B. Eng.)

an der

Hochschule Konstanz
für Technik, Wirtschaft und Gestaltung
Fakultät Bauingenieurwesen

Untersuchung der Anwendbarkeit von KI-Modellen in Rhino/Grasshopper für die Berechnung eines Holzfachwerkträgers

---

Jonas Hilberer, 2023

Hochschule Konstanz
Fakultät Bauingenieurwesen

Die folgende Arbeit wurde der Hochschule Konstanz, Hochschule für Technik Wirtschaft und Gestaltung vorgelegt

Untersuchung der Anwendbarkeit von KI-Modellen in Rhino/Grasshopper für die Berechnung eines Holzfachwerkträgers

Bachelor-Thesis

Zur Erlangung des akademischen Grades eines

Bachelor of Engineering (B. ENG)

vorgelegt von

Hilberer, Jonas

Matr.-Nr: [redacted]

Studiengang: Bauingenieurwesen (BIB)

Am 31.07.2023

Prüfer: Prof. Dr. Alexander Michalski

Ehrenwörtliche Erklärung

„Hiermit versichere ich, Jonas Hilberer, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: "Untersuchung der Anwendbarkeit von KI-Modellen in Rhino/Grasshopper für die Berechnung eines Holzfachwerkträgers" selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.
Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen."

Konstanz, 31.07.2023 Jonas Hilberer

Zusammenfassung

Die zunehmende Allgegenwart der Künstlichen Intelligenz (KI), die bereits zur Vereinfachung verschiedener Aufgaben in vielen Bereichen des Lebens eingesetzt wird, bietet auch Möglichkeiten zur Anwendung im konstruktiven Ingenieurholzbau. Die Tragwerksplanung, als eine der Hauptaufgaben im Ingenieurbau, bringt durch Ihren Arbeitsaufwand und komplexe Herausforderungen, ein Potential für die Verwendung von KI mit sich.
Diese Bachelorarbeit untersucht die Anwendbarkeit von KI in der parametrisierten Strukturmodellierung mit dem CAD-Programm Grasshopper, einer häufig verwendeten Software im Planungsprozess.
Dazu wird erklärt, wie eine KI funktioniert und eine Übersicht über relevante Programme und nützliche Software vorgestellt. Es werden Herausforderungen und Einschränkungen diskutiert, die sich aus der Verwendung von Grasshopper und dessen Abhängigkeiten von den Plugins für den Zugang zu einer, bisher noch nicht fest in Grasshopper integrierten, KI ergeben.
Dazu dient ein Beispiel, an dem vergleichend evaluiert wird, welches Plugin für das Training auf komplexe, umfangreiche Systeme brauchbar ist. Die Auswertung führt zur Anwendung von 'Pug', bereitgestellt durch Diego Apellániz, in einem weiteren, größeren Beispiel eines Fachwerkträgers. Die Erstellung des Datensatzes, den Aufbau neuronaler Netzwerke in Pug und das KI-Training bis zum fertigen Model werden dokumentiert und dienen als Einstiegspunkt für weitere Umsetzungen dieser Art.
Trotz der bei der Bearbeitung identifizierten Hürden, wurde erfolgreich ein funktionsfähiges KI-Modell in Grasshopper implementiert und auf eine exemplarische Planungsaufgabe im Holzbau angewandt. Die Arbeit bestätigt die Anwendbarkeit von KI-Modellen im gegebenen Kontext, diskutiert dabei die sinnhafte Verwendung dieser und identifiziert Herausforderungen für die zukünftige Forschung.

Abstract

The growing ubiquity of Artificial Intelligence (AI), already deployed to simplify various tasks in many areas of life, also offers potential for application in structural timber engineering. The process of structural design, one of the main tasks in engineering construction, brings significant scope for the use of AI due to the complexity and effort involved.

This Bachelor's thesis investigates the applicability of AI in the parametrized structural modelling with Grasshopper, a frequently used software in the planning process. This thesis provides an explanation of how AI operates and an overview of relevant programs and useful software. The challenges and limitations that arise from the use of Grasshopper and its dependencies on plugins for access to AI, which is not yet firmly integrated into Grasshopper, are discussed.

A comparative evaluation is performed based on an example, to assess which plugin is suitable for training complex, extensive systems. The evaluation leads to the application of 'Pug', provided by Diego Apellániz, in another larger example of a truss carrier. The creation of the dataset, the construction of neural networks in Pug, and the AI training to the finished model are documented and serve as a starting point for further implementations of this kind.

Despite the hurdles identified during the work, a functional AI model was successfully implemented in Grasshopper and applied to an exemplary planning task in timber construction. The paper confirms the applicability of AI models in the given context, discusses their meaningful use, and identifies challenges for future research.

Inhaltsverzeichnis

Ehrenwörtliche Erklärung **i**

Zusammenfassung **ii**

Abstract **iii**

Inhaltsverzeichnis **iv**

Tabellenverzeichnis **vi**

Abbildungsverzeichnis **vii**

1 Einleitung **1**

2 Planung mit Künstlicher Intelligenz **2**

2.1 Grundlagen und Motivation 2

2.1.1 Tragwerksbemessung im Holzbau 2

2.1.2 Möglichkeiten Künstlicher Intelligenz in der Baubranche 3

2.1.3 Schnittstelle und Software 5

2.2 Programmierung 7

2.2.1 Die Programmiersprachen 7

2.2.2 Visuelle Programmierung 10

2.2.3 Algorithmen 10

2.2.4 Fuzzy Logic 12

2.2.5 Künstliche Intelligenz 13

2.2.6 Neuronale Netzwerke 15

2.2.7 ML-Frameworks 19

2.3 Software - CAD, visuelles Programmieren, Bemessung 20

2.3.1 Rhinoceros 3D 20

2.3.2 Grasshopper 21

2.3.3 weitere Programme 22

3 Voraussetzungen für den Einsatz von KI **24**

3.1 Entwicklungsumgebung Grasshopper 24

3.1.1 Workflow 24

3.1.2 Python in Grasshopper 26

3.1.3 Karamba3D 27

3.1.4 Algorithmen in Grasshopper 27
3.1.5 KI in Grasshopper - Plugins 28
3.2 Alternative ohne Grasshopper 32

4 KI für Planungsaufgaben - Auswahl und Umsetzung 35
4.1 Pluginauswahl 35
4.1.1 Datensatz Einfeldträger 36
4.1.2 Lunchbox 36
4.1.3 Pug 39
4.2 Fachwerk KI 43
4.2.1 Praxisbeispiel 44
4.2.2 Datensatzerstellung 44
4.2.3 KI-Training 46
4.2.4 Modelltest und Auswertung 48
4.2.5 KI-Anwendung 52

5 Diskussion 54
5.1 Ergebniszusammenfassung 54
5.2 Kritik 54
5.3 Ausblick 55

6 Fazit 57

Anhang 58

Literatur IX

Tabellenverzeichnis

Tab. 2.1 Übersicht KI Frameworks 19

Tab. 3.1 Komponenten KI-Modell Entwicklung 24
Tab. 3.2 Algorithmen in Grasshopper 28
Tab. 3.3 KI-Plugins in Grasshopper 29

Tab. 4.1 Datensatz Querschnittshöhe Einfeldträger 36
Tab. 4.2 Datensatz EFT Lunchbox 37
Tab. 4.3 Aufbau NN + Trainigseinstellungen 40
Tab. 4.4 Datensatz Fachwerkträger 45
Tab. 4.5 Aufbau NN + Trainigseinstellungen Fachwerkträger 48

Abbildungsverzeichnis

Abb. 2.1 Übersicht Eurocodes 2
Abb. 2.2 Ausblick - KI im Bauwesen 4
Abb. 2.3 Hauptprogrammiersprachen 7
Abb. 2.4 Codeverarbeitung Python 8
Abb. 2.5 Framework und Library element 9
Abb. 2.6 Codeverarbeitung C 9
Abb. 2.7 Optimierungsalgorithmen 12
Abb. 2.8 KI, Machine Learning und Deep Learning 14
Abb. 2.9 Neuronales Netzwerk mit Eingabeschicht (blau), versteckte/n Schicht/en (grün), Ausgabeschicht (violett) 15
Abb. 2.10 Visualisierung Tensoren 16
Abb. 2.11 Vektor, R/G/B-Matrix, 3D-Tensor 16
Abb. 2.12 KI-Bibliotheken Nutzung 2019-2023 20
Abb. 2.13 Beispiel Skript und Ergebnis 21

Abb. 3.1 Schema Datensatzerstellung 25
Abb. 3.2 Schema Implementierung KI 26
Abb. 3.3 Umfang LunchboxML 30
Abb. 3.4 Umfang Pug 31
Abb. 3.5 Umfang Owl 32
Abb. 3.6 Schema Datensatzerstellung 33

Abb. 4.1 Ergebnisse Lunchbox EFT Berechnung 38
Abb. 4.2 Tensorvisualisierung EFT 39
Abb. 4.3 Regressionstrainer, SL-Agent 40
Abb. 4.4 Komponenten Neuronales Netzwerk 41
Abb. 4.5 Ergebnisse Pug EFT Berechnung 43
Abb. 4.6 Fachwerkträger Aufgabenstellung 44
Abb. 4.7 Inputdaten - links: Datensatz ”gegraftet”, rechts: Datensatz ”geflattet” 47
Abb. 4.8 Tensorvisualisierung Fachwerkträger 47
Abb. 4.9 Vergleich MAE 49
Abb. 4.10 Vergleich MAE 51
Abb. 4.11 KI Anwendung - Beispiel 1 52

Abb. 4.12 KI Anwendung - Beispiel 2 53

1 Einleitung

Künstliche Intelligenz (kurz: KI) hat in den letzten Jahren enorm an Bedeutung gewonnen und ist zu einem zentralen Forschungsgebiet in der Informatik geworden. Durch die fortschreitende Digitalisierung und die Verfügbarkeit großer Datenmengen ergeben sich immer mehr Anwendungsbereiche für KI-Technologien.
KI-Systeme können beispielsweise für Bild- und Spracherkennung, zur Prozesssteuerung und unterstützend bei Anwendungen eingesetzt werden, lösen dabei Probleme, treffen Entscheidungen oder automatisieren komplexe Aufgaben. Sie finden bereits Anwendung in vielen Bereichen unseres Lebens wie bspw. der Medizin, im Verkehr, Bankingprogrammen oder unseren Email Spam-Filtern.
Auch die moderne Bauingenieursarbeit entwickelte sich mit dem technischen Fortschritt weiter, weg von Rechenschieber und simplen Modellen zu Statikprogrammen, Formfindung und Strukturoptimierung, wo nun auch die KI ihren Einzug feiert. Gebäude die immer größer werden, Bauvorhaben die immer teurer werden und Aufgaben die immer komplexer werden, bieten ein Potenzial für die Einbindung von KI in den Aufgabenbereich der Baubranche und so auch der Planung.

Diese Bachelorarbeit beschäftigt sich mit dem Thema Künstliche Intelligenz und untersucht wie diese Technologie für die Tragwerksplanung in Grasshopper, einem typischen CAD-Programm, genutzt werden kann. Im Mittelpunkt stehen dabei die Grundlagen der KI, wie diese zugänglich gemacht wird und die Anwendungen eines KI-Modells auf eine exemplarische Planungsaufgabe im Holzbau.
Ziel dieser Arbeit ist es, ein Verständnis für die Grundlagen und Anwendungen von KI in Grasshopper zu vermitteln und gleichzeitig kritisch zu reflektieren, wo die Potenziale und Herausforderungen, für die zukünftige Umsetzung dieser neuen Planungsperspektive liegen, was zur Frage von Anwendbarkeit von KI-Modellen für die parametrisierte Strukturmodellierung in Grasshopper und im konstruktiven Ingenieurholzbau führt.

2 Planung mit Künstlicher Intelligenz

Dieses Kapitel widmet sich der Erläuterung der Grundlagen für Künstliche Intelligenz (kurz: KI) und der Tragwerksplanung im Holzbau. Dazu wird der Planungsprozess erklärt, bisherige Verwendungen von KI in der Baubranche aufgegriffen und die Funktionsweise von Programmiersprachen als Werkzeug für die Erstellung von KI erläutert. Das Kapitel wird mit einer Übersicht über gängige Programme in der Tragwerksplanung abgeschlossen.

2.1 Grundlagen und Motivation

Künstliche Intelligenz wird bei Arbeitsprozessen zunehmend eingesetzt. Dieses Kapitel geht der Frage nach, ob es sich lohnt, für Tragwerksplanungen KI einzusetzen, wenn die aufwendige Anwendung dieser sogar Berufe wie den eines Machine Learning Engineers erfordert, und gibt eine Übersicht wie KI bereits eingesetzt wird. Der Umfang konventioneller Methoden wird verglichen mit dem Umfang, der sich beim Einsatz von KI ergäbe.

2.1.1 Tragwerksbemessung im Holzbau

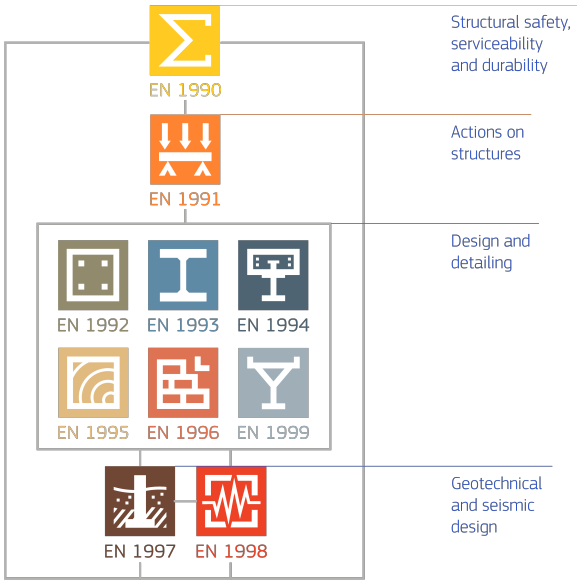


Abbildung 2.1: Übersicht Eurocodes

Quelle: https://eurocodes.jrc.ec.europa.eu/

Die Grundlage der Tragwerksplanung in Deutschland, sowie in weiten Teilen Europas, ist der Eurocode. Aufbauend darauf, bleibt jedem Land die Möglichkeit, die geltenden Vorschriften nationale Anhänge zu ergänzen oder zu ändern. So entsteht ein umfangreiches Regelwerk für den Bau und Betrieb baulicher Anlagen. Eine Übersicht lässt sich Abbildung [2.1] entnehmen.
Für den Holzbau bildet die DIN EN 1995 bzw. der Eurocode 5 (EC5) den Hauptteil der Bemessungsnormen ab. Der Bauteilnachweis umfasst eine Bemessung im Grenzzustand der Tragfähigkeit (Tragfähigkeitsnachweis - GZT) und einen Nachweis im Grenzzustand der Gebrauchstauglichkeit (Gebrauchstauglichkeitsnachweis - GZG). Ein Tragfähigkeitsnachweis besteht aus Lastannahmen mit daraus resultierender Schnittgrößenermittlung für das Gesamtsystem, sowie aus Spannungsnachweisen für die einzelnen Bauteile, wobei hierbei verschiedene Versagensarten zu untersuchen sind ([39], S. 17ff). Bei dieser Nachweisführung findet eine Iteration über verschiedene Querschnittsdimensionen mit nachfolgender Nachweisberechnung statt, die bis zum Erfüllen aller Nachweise angepasst werden. Abhängig vom statischen System des Tragwerks ergeben sich zwischen Quersschnittswahl, Schnittgrößenverteilung und geometrischen Einschränkungen, komplexe Abhängigkeiten bei der Nachweisführung, die mit automatisierten Berechnungsprogrammen nicht vollumfänglich abgedeckt werden können. Dieser Sachverhalt führt zu einem erhöhten Bemessungsaufwand, der zusammen mit der Vielzahl an Normen und Vorschriften, die bei einer Bemessung berücksichtigt werden müssen, und individueller Problemstellungen, ingenieurmäßiges Arbeiten erfordert.

2.1.2 Möglichkeiten Künstlicher Intelligenz in der Baubranche

Eine Internetrecherche nach "*Künstliche Intelligenz*" oder "*Artificial Intelligence*" offenbart eine Fülle an Informationen und verschiedener Anwendungen. KI-Modelle übernehmen zahlreiche Aufgaben[1], beginnend bei der Generierung von Bildern und Programmcodes bis hin zu Grundrissplänen[2].
Überträgt man diese Gedanken auf Gebiete der Baubranche ergeben sich ebenfalls viele neue Möglichkeiten, bspw. auf diesen Gebieten:

- Materialauswahl und Herstellung
- Formfindung und Architektur
- HLS-Planung und Betrieb

[1] Übersicht KI-Tools: https://topapps.ai
[2] Finch, KI generierte Grundrisse: https://www.finch3d.com

- Smart Cities

- Tragwerksplanung

([3], S. 2f). Dies ist in folgender Abbildung 2.2 ergänzend dargestellt.

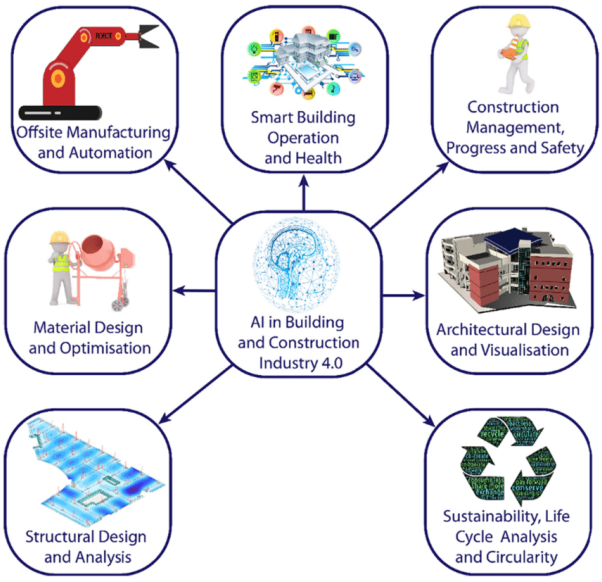


Abbildung 2.2: Ausblick - KI im Bauwesen

Quelle: entnommen aus *Automation in Construction* [3]

Mit diesem Anreiz und dem Wissen KI-Modelle auf multiple Szenarien trainieren zu können, ergeben sich viele denkbare Anwendungsmöglichkeiten für derartige Systeme in der Tragwerksplanung. Der Stand der Entwicklung zeigt, dass eine Künstliche Intelligenz automatisiert Pläne[3] und Objektstrukturen[4] zeichnen oder statische Berechnungen lösen kann, wobei besonders bei der Formfindung[5] bereits großes Potenzial entsteht ([38], S. 15ff).

Bezogen auf die oben genannten Herausforderungen und Aufgaben der ingenieurmäßigen Arbeit bei einer Bemessung eines Tragwerks oder einzelner Querschnitte darin, lassen sich auch hier die Implementierungsmöglichkeiten einer KI-Lösung abschätzen. Ein KI-Training auf Planungsaufgaben kann Aufwand reduzieren und den Planungsprozess beschleunigen und so zu einer optimierten Bemessung beitragen. Der iterative Prozess der Querschnitts- und Geometriefindung vereinfacht sich über Anwendung einer KI, die ein Problem erlernt hat und das Ergebnis dieser Problemlösung wiedergeben kann. Dies lässt sich auf verschiedene Problemstellungen der Bemessung übertragen. Für komplexe Aufgaben, die keinen regelbasierten, erklärbaren Zusammenhang haben, wie bspw.

[3] Beitrag über ArchiGAN: https://developer.nvidia.com/blog/archigan-generative-stack-apartment-building-design/
[4] Autodesk Add-On: https://www.autodesk.com/products/spacemaker/overview
[5] Bericht über Strukturfindung: https://news.mit.edu/2023/integrating-humans-ai-structural-design-0302

mechanisch nicht erklärbare Problemstellungen, ist die Anwendung von KI für viele Bereiche und ebenfalls der Baubranche und der Tragwerksplanung zukunftsweisend.
Es gibt es jedoch einen Grundsatz zu beachten, der den Anfang eines jeden Projekts mit Künstlicher Intelligenz bilden sollte:

"Don't be afraid to launch a product without machine learning"
Martin Zinkevich

Mit dieser ersten Regel des Google-Handbuchs "Rules of machine Learning[6]" fordert Zinkevich auf, die Erforderlichkeit eines KI-Modells zu hinterfragen. Es gilt zu klären ob, KI zwingend eingesetzt werden muss oder ob die Aufgabe auch mit einem simpleren, regelbasierten oder ggf. manuellen Ansatz gelöst werden kann, für den es keinen Datensatz braucht [20]. Dieser Sachverhalt wird an einem Bemessungsbeispiel gut erkenntlich.
Gesucht ist die erforderliche Querschnittfläche eines Holzstabes auf beliebigen, zentrischen Druck ohne Knickversagen:

- gegeben: Kraft F [N]; Druckwiderstand fck [N/mm²]
- gesucht: Querschnittsfläche A [mm²]

- Die schnellste Lösung stellt die manuelle Berechnung dar:

$$A = \frac{F}{f_{ck}} \tag{2.1}$$

- Der Lösungsansatz mittels KI fällt dabei deutlich aufwendiger und umfangreicher aus.

Der Quellcode ist dem Anhang [A01] zu entnehmen. Bei dieser Aufgabenstellung liegt ein lineares Problem vor. Eine direkte Berechnung ist im Vergleich zur Berechnung mittels KI genauer (da sich diese im angegeben Code dem Ergebnis nur annähert) und in der praktischen Anwendung wesentlich zielführender.
Solche Berechnungen erfordert jede Tragwerksplanung. Trotz der Fülle an denkbaren Anwendungsmöglichkeiten, ist es demnach nicht immer hilfreich, KI einzusetzen.

2.1.3 Schnittstelle und Software

Um die Arbeit mit KI praxistauglich zu machen, erfordert es eine Schnittstelle für den Informationsaustausch. Gängig sind hierbei Chatbox-Eingaben, die eine geforderte Rück-

[6] Rules of Machine Learning: https://developers.google.com/machine-learning/guides/rules-of-ml

meldung der KI zur Folge haben. Eine Alternative ist ebenfalls durch grafische Benutzeroberflächen oder der Eingabe in einer Entwicklungsumgebung der KI gegeben. Entscheidend ist, dass die Eingabedaten korrekt übermittelt werden und die Ergebnisdarstellung nachvollziehbar ist.

Die Schnittstelle zwischen KI und Anwendenden bildet in dieser Arbeit das Plug-In Grasshopper für die CAD-Software Rhino. Grasshopper bietet eine grafische Programmieroberfläche für Softwarenutzende und verfügt über eine eingebaute Schnittstelle zu Python und zahlreichen anderen Programmen, welche für die weitere Bearbeitung entscheidend sind.

2.2 Programmierung

Bei der Arbeit mit Künstlicher Intelligenz spielt die Programmierung eine zentrale Rolle. Programmieren bildet die Grundlage der Datenverarbeitung und Entwicklung sämtlicher Software. Diese Arbeitsweise zu verstehen, ist Voraussetzung für die Implementierung eines KI-Modells. Programmiert wird in verschiedenen Programmiersprachen. Das sind künstliche Sprachen, die dazu dienen dem Computer Anweisungen zu vermitteln ([45], S. 26).

2.2.1 Die Programmiersprachen

Die Liste verschiedener Programmiersprachen ist zahlreich und jede findet bei unterschiedlichen Anwendungen ihre Vor- und Nachteile.

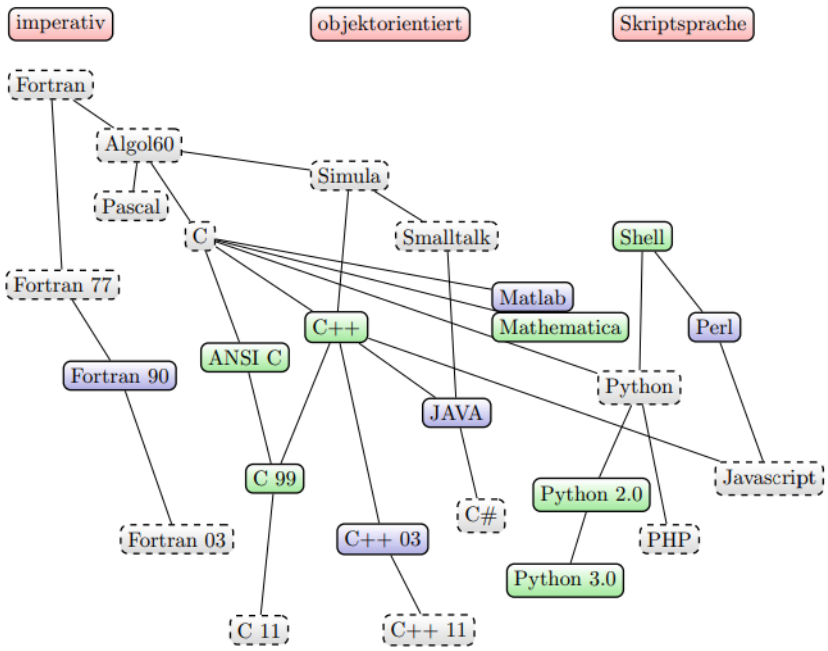


Abbildung 2.3: Hauptprogrammiersprachen

Quelle: entnommen aus *Computerphysik* [19]

Von den in Abbildung [2.3] abgebildeten Hauptprogrammsprachen und deren Entwicklungszusammenhängen, wird im Weiteren auf C, C#, C++ und Python eingegangen. Diese unterscheiden sich in der Codeverarbeitung, Skriptsprachen (Python) verwenden dazu einen *Interpreter* und kompilierte Sprachen (C, C#, C++) einen *Compiler*. Zusätzlich lässt sich in imperative und objektorientierte Sprachen differenzieren, Zuordnungen können Abbildung [2.3] entnommen werden. Imperative Sprachen funktionieren mit Anweisungen und Prozeduren die nacheinander abgearbeitet werden. Die objektorientierte

Programmierung basiert auf einer Objekten und Methoden, die mit einander interagieren um komplexe Systeme besser strukturieren zu können ([19], S. 32f).

Python

Python[7] gilt als höhere Programmiersprache, da nicht in Maschinensprache programmiert wird und bei der der Quellcode eines Programms interpretiert wird.

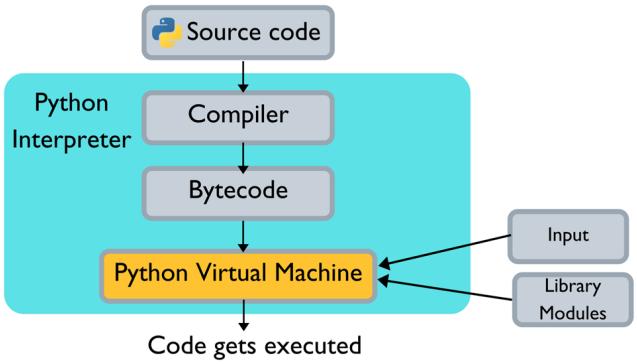


Abbildung 2.4: Codeverarbeitung Python

Quelle: https://www.c-sharpcorner.com/blogs/importance-of-python-programming1

Abbildung [2.4] ist die zugrundeliegende Struktur zu entnehmen. Es wird ein Interpreter verwendet, der bei Programmausführung die Zeilen des Codes nacheinander in Maschinencode übersetzt, mit dem der Computer weiter arbeitet ([41], S. 2). Für die Interpreter gibt es verschiedene Implementierungen wie bspw. CPython ("Hauptvariante"), Jython, IronPython oder PyPy. Die Ursprünge von Python finden sich in den 1990er-Jahren beim Entwickler Guido van Rossum, die erste Vollversion ist auf das Jahr 1994 datiert. Seit 2008 gilt Python 3 als aktuellste Version der Programmiersprache ([41], S. 4f). Ziel der Entwicklung war eine gut lesbarer, knapp gehaltener Programmierstil. Einfachheit und Übersichtlichkeit stellen einen großen Vorteil gegenüber anderer Programmiersprachen dar. Zusätzlich bietet sich die Option, auf eine umfangreiche Bibliothek (engl.: library, library element) von bereits geschriebenen Pythonprogrammen zu zugreifen und Elemente aus dieser im eigenen Code anzuwenden (siehe auch Abbildung [2.4]). Größere und komplexere Strukturen die ebenfalls Bibliothekselemente verwenden und dem Anwenden ein Grundgerüst für seine eigene Programmierung geben, werden auch **"*frameworks*" (engl.) genannt. Dieser Zusammenhang lässt sich grafisch besser verstehen und ist im Folgenden dargestellt.

[7] Dokumentation Python: https://docs.python.org/3/

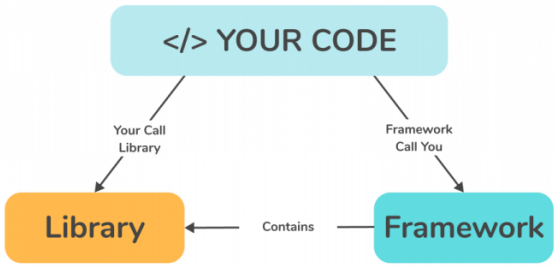


Abbildung 2.5: Framework und Library element

Quelle: https://www.interviewbit.com/blog/framework-vs-library/ (2023.04.21)

Ohne diese Importfunktionen wäre die Umsetzung dieser Arbeit nicht möglich. Die verwendeten Bibliothekselemente bilden Schnittstellen zwischen den Programmen und vereinfachen die Erstellung einer Künstlichen Intelligenz auf ein praktikables Niveau.

C, C++ und C#

C[8] ist ebenfalls eine höhere Programmiersprache, verfolgt jedoch eine imperative Softwareentwicklung sowie eine kompilierte Codeverarbeitung.

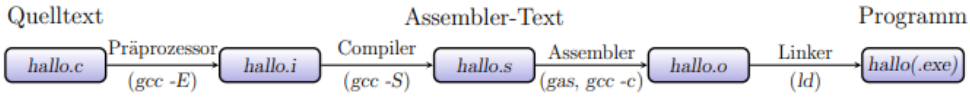


Abbildung 2.6: Codeverarbeitung C

Quelle: entnommen aus *Computerphysik* [19]

Ein Compiler übersetzt den gesamten Quellcode in ein ausführbares Programm, dass danach direkt ausgeführt werden kann und bei einem erneuten Aufruf nicht mehr kompiliert werden muss. Wie auch bei Python gibt es unterschiedliche Arten von Compilern, ein gängiger C-Compiler ist *GCC (GNU compiler collection)*. Die Entwicklung von C reicht zurück bis in die 1970er- Jahre, ehe sie 1973 von Brian Kernighan und Dennis Ritchie fertiggestellt wurde. C gilt als Muttersprache vieler Programmiersprachen, so auch von Python (siehe Abb.: [2.3]). Häufige Anwendungen von C sind Betriebssysteme und Systemprogramme, besonders in Linux aber auch Windows und MacOSX ([19], S. 34f). Auch in C bietet sich die Möglichkeit Bibliothekselemente bzw. -programme zu verwenden.

Auf dieser Grundlage wurde ab 1980 die Programmiersprache C++[9] von Bjarne Stroustrup entwickelt und die Programmiersprache C auf eine objektorientierte Programmierung geändert/ergänzt ([5], S. 15f).

[8] Community-Dokumentation C: https://en.cppreference.com/w/c/language
[9] Get Started with C++: https://isocpp.org/get-started

Ab 1998 begann Microsoft mit der Entwicklung von C#[10] und lieferte so eine weitere Möglichkeit für objektorientiertes Programmieren auf Grundlage von C ([27], S. XI).

2.2.2 Visuelle Programmierung

Visuelles Programmieren (engl.: visual programming *oder* visual programming language; kurz: VPL) beschreibt die Erstellung von Programmen indem grafische Elemente statt textbasiertem Code verwendet werden. Diese Systeme nutzen die visuelle Darstellung von Programmiertechniken, um die Komplexität der Programmsyntax zu reduzieren und damit das Verständnis der Programmlogik zu verbessern ([6], S. 275).
Für die Kategorisierung von visuellen Programmierungen und deren Funktionsweise gibt es verschiedene Einteilungen. Abseits dieser verschiedenen Taxonomien funktionieren visuelle Programmierungssysteme häufig durch das ”Zusammenstecken” oder ”Verbinden” von Blöcken, Icons (engl. für Symbol) oder Komponenten, die verschiedene Funktionen oder Prozeduren repräsentieren. Diese Blöcke können Schleifen, Entscheidungsstrukturen, mathematische Operationen, Logikoperationen und viele andere, bekannte Dinge aus der textbasierten Programmierung sein. Zusammen ergibt diese vernetzte Struktur ein funktionierendes Programm. Die grafische Darstellung der Programmlogik kann den VPL Programmierenden helfen, die Struktur und Funktionsweise des Programms besser zu verstehen, insbesondere in komplexen Systemen, in denen viele verschiedene Funktionen miteinander interagieren ([26], S. 14183).
Die Art und Weise, wie der ”Programmcode” in visuellen Programmierungssystemen dargestellt wird, unterscheidet sich demnach grundlegend von textbasierten Programmiersprachen. Anstelle des Schreibens von Codezeilen, wird der Code generiert, indem grafische Elemente auf einer Benutzeroberfläche arrangiert und miteinander verbunden werden. Die Komponenten, die in der visuellen Programmierung verwendet werden, lassen sich abstrahierte Versionen von Code verstehen. Wenn bspw. eine Addition-Komponente in einer visuellen Programmierungsumgebung platziert wird, generiert das System im Hintergrund den entsprechenden Code, der die Addition in einer textbasierten Programmiersprache darstellen würde ([6], S. 275ff).

2.2.3 Algorithmen

Algorithmen sind Anleitungen oder Ausführungsregeln für die Lösung eines Problems oder der Bearbeitung einer Aufgabe. Ein Algorithmus umfasst demnach konkrete Anweisungen, die ein Computer ausführen kann, um eine bestimmte Aufgabe zu erledigen.

[10] Dokumentation C# : https://learn.microsoft.com/en-us/dotnet/csharp/

Neben der Mathematik und Informatik finden Algorithmen in vielen Disziplinen der Wissenschaft Anwendung ([30], S. 3ff).
Dabei werden Sie für einfache mathematische Berechnungen bis hin zur komplexen Datenanalyse, der Erkennung von Mustern und Zusammenhängen oder bei Künstlicher Intelligenz verwendet. Die Vielzahl an Algorithmen, die in verschiedenen Bereichen eingesetzt werden, lassen sich in Arbeitsmethoden unterteilen, von denen hier einige wichtige aufgelistet sind:

- Numerische Algorithmen: Dienen der Lösung mathematischer Probleme, wie bspw. grundlegenden Gleichungssystemen, der Bestimmung eines größten gemeinsamen Teilers zweier natürlicher Zahlen(Euklid Algorithmus) oder begrenzt komplexen Optimierungsaufgaben.

- Iterative Algorithmen: Suchen durch eine iterative Prozedur eine Lösung für ein Problem. Durch wiederholtes Anwenden einer bestimmten Operation oder eines bestimmten Arbeitsschritts auf eine Startlösung, werden Zwischenlösungen erzielt, um eine sich stetig verbessernde Lösung zu erhalten. Der Prozess wird wiederholt, bis eine akzeptable Lösung gefunden wurde oder ein Abbruchkriterium erfüllt ist [30], S. 4). Ein Beispiel ist das Gradientenabstiegsverfahren, bei dem das Minimum einer Funktion gesucht wird (Optimierungsaufgabe).

- Heuristische Algorithmen (auch metaheuritsch): Bieten eine schnelle Lösung für Optimierungsprobleme, ohne eine Garantie für ein optimales Ergebnis zu liefern, was sie von numerischen Algorithmen unterscheidet. Für die Lösung komplexer Aufgaben durchsuchen Sie nur einen Teil des Lösungsraums, anstatt des gesamten. So finden Sie Anwendung in der KI und dem Maschinellen Lernen. Beispiele für heuristische Algorithmen sind genetische Algorithmen und der Ameisenalgorithmus. Die Fuzzylogic lässt sich hier ebenfalls einordnen ([4], S. 5f; [30], S. 4).

Neben vieler numerischer Algorithmen, dem Gradientenabstiegsverfahren (Iterativer Algorithmus) oder dem Neuronalen Netzwerk, was selbst als Algorithmus angesehen werden kann, werden bei der Verwendung von, für die KI-Anwendung unumgänglichen Optimierungsfunktionen, zusätzlich zwei Arten von Algorithmen eingesetzt:

- Genetische Algorithmen (GA): Ein GA ist ein evolutionärer Algorithmus (heuristisch/iterativ), der auf der Idee der natürlichen Selektion basiert und verwendet wird, um Lösungen für Optimierungsprobleme zu finden. Er wird oft in der Formoptimierung und der Materialoptimierung verwendet.

- Particle Swarm Optimization (PSO): PSO ist ein heuristischer Algorithmus, der auf der Schwarmintelligenz basiert. Er wird oft in der parametrischen Modellierung eingesetzt, um eine Gruppe von Partikeln zu simulieren, die durch den Lösungsraum fliegen und versuchen, die beste Lösung für ein bestimmtes Problem zu finden.

Zur Verdeutlichung der Vielzahl an Algorithmen und ihrer anwendungsabhängigen Funktionsweise, ist mit Abbildung [2.7] eine Übersicht zur Auswahl eines Optimierungsalgorithmus gegeben.

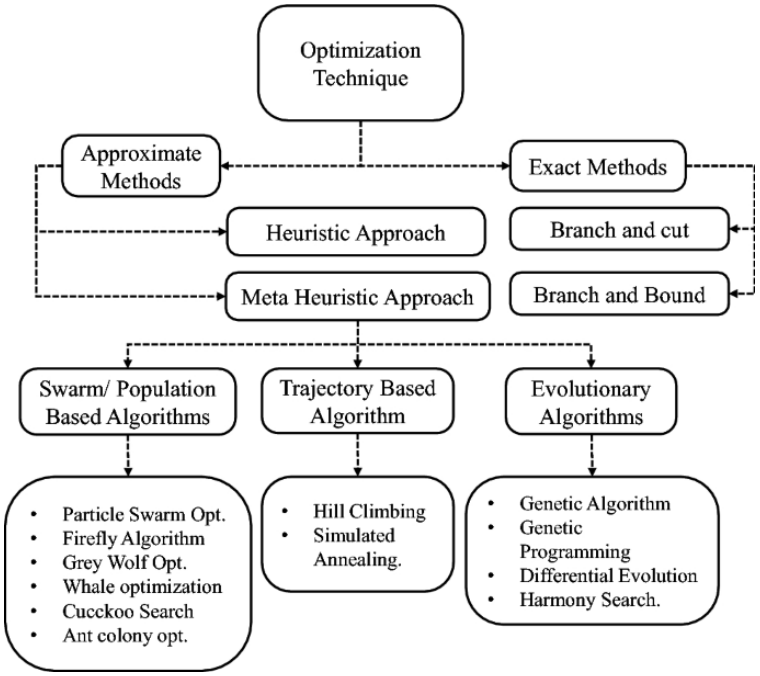


Abbildung 2.7: Optimierungsalgorithmen

Quelle: entnommen aus *Künstliche Intelligenz* [32]

Algorithmen bilden also das Grundgerüst einer Künstlichen Intelligenz, da diese (wie in nachfolgenden Kapiteln erläutert) lediglich Befehle ausführt und vorprogrammierte Anweisungen befolgt. Bei der Arbeit mit KI kommt Ihnen demnach eine große Bedeutung zu.

2.2.4 Fuzzy Logic

Den Gegensatz zu Algorithmen bildet die unscharfe Logik (engl.: fuzzy logic), die als Grundlage von *Fuzzy computing* (unscharfe Berechnung) dient. Unscharfe Logik ist ein alternativer Lösungsansatz für Aufgaben die nicht streng mathematisch lösbar sind und einen Umgang mit Teilwahrheiten oder unscharfen Begriffen erfordern, den klassische Rechenalgorithmen jedoch oft nicht umfassen [7].

Dieser Sachverhalt lässt sich mit Beispielen wie ”noch etwas nach links” oder ”schärfer

Einschlagen", besser verstehen. Derartige Anweisungen sind nicht präzise, für den Menschen jedoch trotz der sprachlichen Unschärfe verständlich. Grund dafür ist die Reduzierung eines Problems auf eine verbale und für die Anwendung, angemessene Komplexität. Sätze wie "-1 Grad, 2 Minuten und 1 Sekunge einschlagen" oder "noch 14 Grad nach rechts" sind zwar präzise, für den Menschen aber wenig praktikabel. Die Übersetzung unscharfer Begriffe in einen für Maschinen verständlichen Code, birgt Schwierigkeiten, für die auch eine Verwendung von Wahrscheinlichkeiten nicht zielführend ist.
Diese Arbeit umfasst eine Problemstellung die mathematisch adäquat formuliert und gelöst werden kann, da eine Nachweisführung aus Rechnungen besteht und sich keinesfalls auf unscharfen Definitionen stützen sollte [44]. Daher stellt die Verwendung von fuzzy logic-Modellen keine große Relevanz für diese Arbeit dar und wird im Weiteren nicht genauer ausgeführt.

2.2.5 Künstliche Intelligenz

Für die generelle Intelligenz gibt es viele unterschiedliche Beschreibungen. Wiederkehrende Schlagwörter sind dabei

- Anpassung an unbekannte Situationen bzw. zur Lösung neuer Probleme
- höhere mentale Prozesse, Entscheidungsfindung, abstraktes Denken und Repräsentation

die alle etwas mit Lernen und Verstand zu tun haben [43]. Menschliche Intelligenz äußert sich allerdings nicht nur in der Lösung von Aufgabenstellungen, sondern auch in anderen Bereichen wie Musik, Logik oder Sozialverhalten ([32], S. 2).
Die Entwicklung Künstlicher Intelligenz verfolgt das Ziel, ein lernfähiges Programm zu entwickeln. Anhand eines Datensatzes kann dieses Computerprogramm auf eine gewünschte Funktionalität trainiert werden, mit dem Ziel, auf neue Problemstellungen reagieren zu können. Es entsteht der Anschein das Programm erlernt und versteht den Sachverhalt([32], S. 3). Bei KI-Modellen kann zusätzlich wie folgt unterschieden werden:

- schwache/schmale KI (engl.: narrow AI)
  Training einer Maschine auf eine bestimmte Aufgabe/ein bestimmtes Aufgabenfeld. Lässt sich nicht auf nicht auf neue/andere Bereiche übertragen. Beispiel: Apps zur Schrifterkennung, Chatbots wie OpenAI
- starke/allgemeine KI (engl.: Artificial General Intelligence (AGI))
  Die Maschine ist in ihrer Anwendung umfangreicher und kann Aufgaben übernehmen die dem menschlichen Handeln nahe kommen. Es entsteht der Eindruck die

KI trifft 'bewusste' Entscheidungen. Die Entwicklung einer AGI stellt eine große Aufgabe der bisherigen Forschung an KI dar und ist bisher noch nicht gelungen.

- Künstliche Superintelligenz (engl.: Artificial Superintelligence (ASI))
  Höher entwickelte Intelligenz die auch die menschliche Intelligenz bei Weitem übersteigt.

Bisher befindet sich der Stand der Entwicklung bei schwacher KI ([42], S. 6).
Vor diesen Hintergründen und in Anbetracht der Tatsache, dass ein Computer nichts anderes tut als Programme auszuführen, lässt sich Künstliche Intelligenz nicht mit dem eigentlichen Sinne des Wortes erklären ([31], S. 8). Die 'Intelligenz' hängt also davon ab wie gut das System auf eine oder mehrere Aufgaben/Gebiete trainiert wurde und wie umfangreich der Datensatz des Wissens ist.

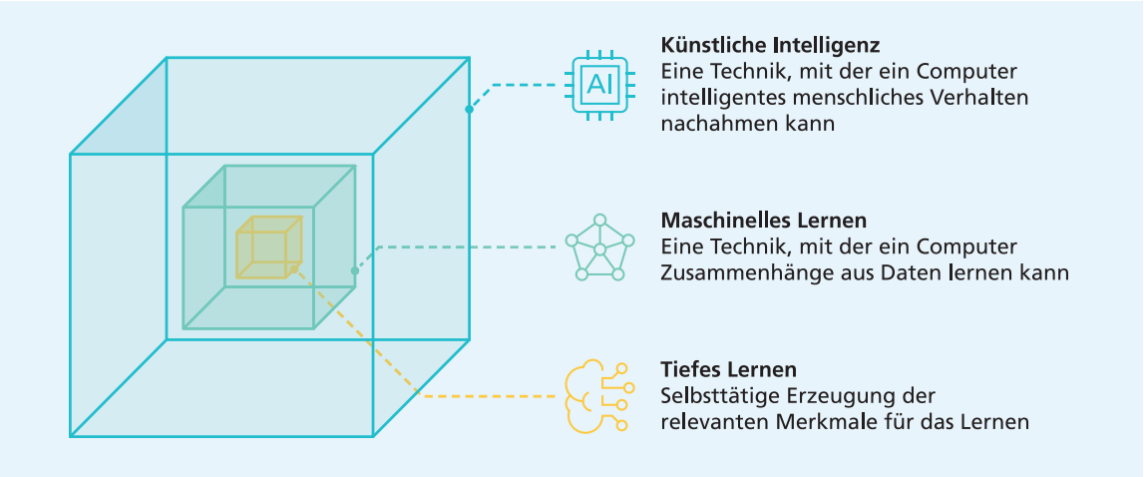


Abbildung 2.8: KI, Machine Learning und Deep Learning

Quelle: entnommen aus *Künstliche Intelligenz* [32]

Künstliche Intelligenz ist ein Oberbegriff für Maschinelles Lernen (engl.: machine learning, Abk.: ML), welcher sich wiederum in Tiefes Lernen (engl.: deep learning, Abk.: DL) untergliedern lässt (siehe Abbildung [2.8]). Während Maschinelles Lernen als Oberbegriff eine Technik beschreibt, wie Computer mit Algorithmen und Modellen aus komplexen Daten Schlussfolgerungen ziehen, erweitert Tiefes Lernen die Begrifflichkeit durch die Anwendung Neuronaler Netzwerke zur Lösung großer, komplizierter Probleme ([42], S. 13f). Dies ermöglicht es einem Programm selbstständig relevante Merkmale für das Erlernen anzulegen und nimmt so den Softwareentwickelnden die Erstellung dieser und weiterer Anweisungen, wie beim ML erforderlich, ab. Ein typisches Beispiel hierfür ist die Bildidentifikation. Modelle ohne Tiefes Lernen kommen bspw. bei der Spracherkennung zur Anwendung, anhand eines großen Datensatzes lernt ein Computer Sprache in Text

umzuwandeln.

Das Maschinelle Lernen lässt sich in drei Arten unterscheiden

- Überwachtes Lernen (engl.: supervised learning)
  Das System trainiert mit Input und Output. Das Ergebnis ist bekannt

- Unüberwachtes Lernen (engl.: unsupervised learning)
  Das System trainiert nur mit Input. Maschinelles Lernen wird genutzt um Regelmäßigkeiten zu erkennen

- Bestärkendes Lernen (engl.: reinforcement learning)
  Das System trainiert indem es Aktionen durchführt und darauf ein Ergebnis erhält ([32], S. 47f). Maschinelles Lernen wird genutzt um eine aufgabenabhängige Belohnung zu maximieren ([42], S. 13)

von denen das überwachte Lernen den Hauptteil dieser Arbeit bildet.

2.2.6 Neuronale Netzwerke

Neuronale Netzwerke (engl.: artificial neural network (ANN)) bilden die Grundlage des Tiefen Lernens. Die Eigenschaft, abstrakte oder 'verstecke' Merkmale und Zusammenhänge in Daten zuerkennen ermöglicht die Lösung komplexer Problemstellungen.

Aufbau

Dem biologischen Gehirn nachempfunden, lassen sich Neuronale Netzwerke ebenfalls in miteinander verbundene Neuronen zerlegen. Wie in Abbildung [2.9] erkenntlich, erfolgt die Anordnung in Schichten (engl.: layer), die mindestens ein Neuron umfassen und sich in eine Eingabeschicht, mehrere versteckte Schichten und eine Ausgabeschicht aufteilen ([42] S. 27).

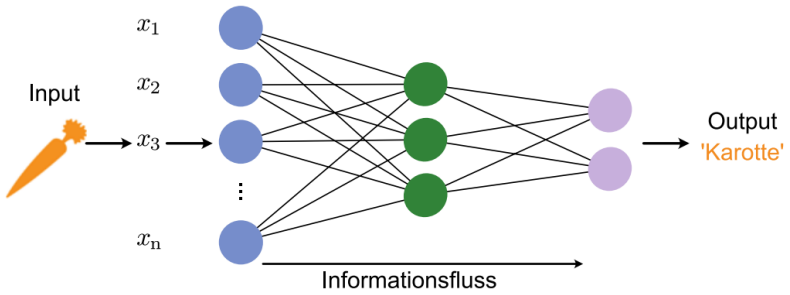


Abbildung 2.9: Neuronales Netzwerk mit Eingabeschicht (blau), versteckte/n Schicht/en (grün), Ausgabeschicht (violett)

Quelle: entnommen aus *Machine Learning kompakt* [8]

Informationfluss, Informationsabbildung

Das Bild einer Karotte ist der Input für eine Anzahl *n* an Neuronen der Eingabeschicht. Entlang des Informationsflusses werden durch Filtervorgänge verschiedene Neuronen aktiviert, was in einem Output endet, hier das Wort ”Karotte” ([42] S. 146f). Die Daten eines Inputs, hier der RGB-Farbcode und die Anzahl der Pixel für das Bild der Karotte, werden als Information in Tensoren gespeichert.

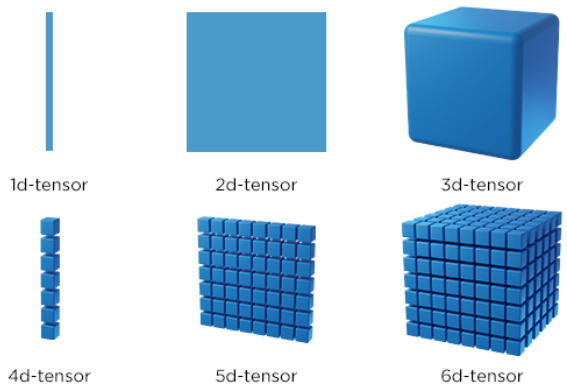


Abbildung 2.10: Visualisierung Tensoren

Quelle: https://visagetechnologies.com/tensors-in-computer-vision/

Wie Abbildung [2.10] zeigt, sind Tensoren ein Zahlenpaket, welches Skalare, Vektoren und Matrizen mehrdimensional speichern kann. Sie sind unabhängig von der Dimension, da sowohl ein Skalar als auch eine Matrix als Tensor bezeichnet wird. Bei einem grafischen Input lässt sich jeder Pixel durch eine Rotwert-, Grünwert- und Blauwert-Matrix beschreiben, die übereinander gelegt die richtigen Farben ergeben, hier die orangene Farbe der Karotte. Abbildung [2.11] veranschaulicht wie dieses dreidimensionale Zahlenpaket in einem dreidimensionalen Tensor gespeichert wird ([32], S. 51f).

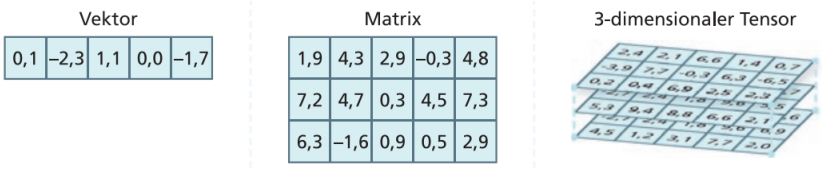


Abbildung 2.11: Vektor, R/G/B-Matrix, 3D-Tensor

Quelle: entnommen aus *Künstliche Intelligenz* [32]

Grundlage des Informationsfluss, Funktionen als Neuronen

Jedem Layer werden lineare oder nicht-lineare Funktionen zugewiesen, die für alle Neuronen des jeweiligen Layers gelten. Ein Neuron ist mit *j* Neuronen der vorherigen Schicht verbunden und gibt die Daten an *k* Neuronen der nachfolgenden Schicht weiter. Im Neu-

ron selbst wird/werden die Information bzw. der/die Tensor/en durch die hinterlegte lineare oder nicht-lineare Funktion verändert. Aus der Mathematik sind lineare Funktionen in folgender Form bekannt:

$$y = mx + c \tag{2.2}$$

Übertragen auf die Neuronen Neuronaler Netzwerke ergibt sich:

$$y(z_1, ..., z_k) = \sum_{j=1}^{k} w_j z_j + b \tag{2.3}$$

Die Steigungskoeffizienten *w* werden Gewichte (engl.: weights) genannt und geben die Stärke der Verbindung der jeweiligen Neuronen an. Ergänzt durch die reele Zahl *b* (engl.: bias) ergeben sich zwei Parameter die während des Trainings der KI optimiert werden. Den Output *y* gibt das Neuron an *k* Neuronen der nachfolgende Schicht weiter. Layer mit nicht-linearen Funktionen manipulieren diese linearen Zusammenhänge der Neuronen. Es werden Schwellenwerte abgespeichert, unter denen die Neuronen nicht aktiviert werden. Daher werden sie auch Aktivierungsfunktionen genannt ([8], S. 26f). Aus mathematischer Sicht ist die Grundoperation einer KI immer eine Tensormanipulierung wie bspw. eine Matrizenmultiplikation.

Training

Die iterative Anpassung der Parameter in den Neuronen wird auch Training des Neuronalen Netzwerkes bezeichnet ([8], S. 33). Die Anzahl der Iterationen in dieser Optimierungsschleife werden vorab festgelegt und Epochen genannt. Um sich dem Ergebnis anzunähern beginnt die KI mit zufälligen Zahlenwerten, die sehr wahrscheinlich nicht richtig sind. In jedem Trainingsschritt (engl.: training step) wird durch eine festgelegte Verlustfunktion (engl.: loss function) der Grad der Unterschiedlichkeit (engl.: degree of dissimilarity) zum Zielwert ausgewertet und darauffolgend durch Optimierungsalgorithmen (engl.: optimizer) angepasst. Neben der Wahl der Netzwerkstruktur und trainingsrelevanter Funktionen, lässt sich das KI-Training zusätzlich durch *Hyperparameter* steuern[11].

Üblich ist eine Datensatzaufteilung in Trainingsdaten (80%) und Testdaten zur Evaluation des Traingingsfortschritts (20%) [34].

Die Struktur des Neuronalen Netzwerks und die optimierten Parameter lassen sich exportieren, um das Ergebnis des Trainings zu speichern.

[11] Die Auswahl an Verlustfunktionen, Optimierungsalgorithmen und Hyperparametern ist sehr groß weshalb von einer ausführlichen Erläuterung an dieser Stelle abgesehen wird. Nähere Ausführungen folgen bei Verwendung der jeweiligen Funktion oder Festlegung der Einstellungen. Das Wissen über diese Möglichkeiten ist elementar und sollte immer in der Dokumentation des Frameworks nachgelesen werden.

Abhängig vom Zweck der KI und dadurch Aufbau des Neuronalen Netzwerks, wird eine KI auf unterschiedliche Zwecke trainiert. Beispiele sind

- Regressionsaufgaben - Ansatz um eine kontinuierliche Beziehung zwischen Eingabevariablen und einer Zielvariablen zu modellieren. Das bereits geführte Beispiel der Stützenbemessung zählt als Regressionsaufgabe.

- Klassifikation - Dient der Einordnung von Daten in Gruppen. Dabei kann in zwei oder mehrere Klassenunterschieden werden. Gekoppelt mit Bildverarbeitungsprogrammen können diese KI-Modelle bspw. zur Ermittlung von Hautkrebs verwendet werden [[22], S. 1ff].

- Datengenerierung (GAN) - KI wird darauf trainiert Daten zu genieren wie bspw. Bilder oder Text. Ein populäres Beispiel hierfür ist chatGPT von openAI [[18], S. 2ff].

Ergebnisinterpretation

Die Abbildung von Informationen in Tensoren ist sehr abstrakt, weshalb oft nicht nachvollziehbar ist wie eine KI zu ihrem Ergebnis kommt. Ein ausgebildetes Neuronales Netzwerk ähnelt einer Black-Box, die mehrere Anwendungen benötigt um verstanden zu werden ([29], S. 5ff). Dies ist je nach Aufgabenfeld sehr problematisch. Für den Fall einer Bemessungsaufgabe die ohnehin von menschlicher Hand bearbeitet wird, bereitet die erschwerte Ergebnisinterpretation jedoch keine Schwierigkeiten, weil Ergebnisse sehr gut kontrolliert werden können und der Fehler bzw. das falsch Erlernte experimentell schnell erkannt werden kann.

2.2.7 ML-Frameworks

Für die Erstellung von KI-Modellen existieren verschiedene Frameworks, die den Anwendenden als Grundgerüst dienen. Das bedeutet die erforderlichen Komponenten und Funktionen für Maschinelles Lernen oder Neuronale Netzwerke und den Umgang mit diesen, wurden bereits programmiert und ermöglichen die Nutzung dieser für die eigene KI. Verschiedene Entwicklerteams haben sich so einer Aufgabe bereits angenommen, von den einige in Tabelle [2.1] aufgelistet sind.

Name	**Entwicklungs-sprache**	**Entwickler**	**Version**	**Referenz**
PyTorch	Python	Linux Foundation	2023	pytorch.org
Tensorflow	Python	Google	2023	tensorflow.org
SciKit	Python	Scikit-learn, Community-Projekt	2023	scikit-learn.org
Pandas	Python	finaziert durch NumFOCUS Inc.	2023	pandas.pydata.org
Accord.NET	überwiegend C#	Community-Projekt	2020	accord-framework.net
ML.NET	C#, .NET	Microsoft	2023	dotnet.microsoft.net - machine learning

Tabelle 2.1: Übersicht KI Frameworks

Wie Abbildung [2.12] zeigt, unterscheidet sich die Popularität der KI-Frameworks. Pytorch und Tensorflow decken diesem Diagramm zu Folge, den größten Anteil der KI-Modelle ab.

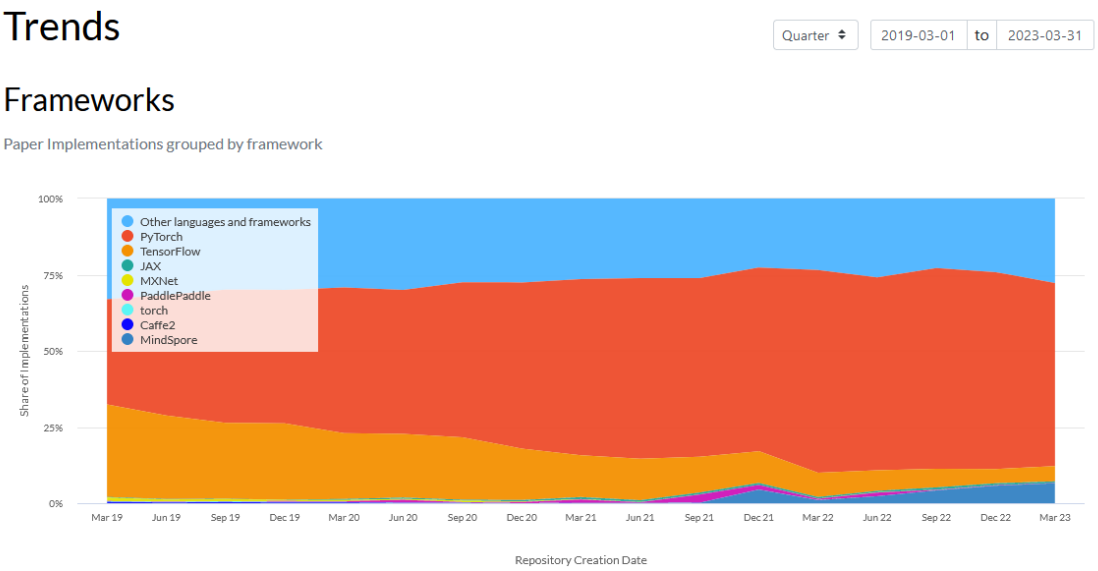


Abbildung 2.12: KI-Bibliotheken Nutzung 2019-2023

Quelle: https://paperswithcode.com/trends (2023.04.20)

2.3 Software - CAD, visuelles Programmieren, Bemessung

Eine gute Tragwerksplanung erfordert Zeichnungen. Sie sind die Sprache des/r Ingenieurs/in und halten alle Überlegungen der Planung fest. In der Praxis gibt es viele verschiedene CAD-Programme, die sich in ihrer Handhabung unterscheiden und anwendungsspezifische Vor- und Nachteile haben. Im Folgenden wird auf die Software Rhinoceros 3D eingegangen und die wesentlichen Komponenten, in Bezug für die Erstellung einer KI, erklärt.

2.3.1 Rhinoceros 3D

Rhinoceros 3D (auch Rhino genannt) ist ein CAD-Programm mit Stärken in der 3D-Freiform-Modellierung. Die Software wird für Design, Rendering und Analyse verwendet und findet so Anwendung in der Gestaltung von Produkten, der Architektur, dem Industriedesign und verschiedenen Disziplinen des Ingenieurswesen. Durch eine breite Auswahl von Werkzeugen für die Erstellung komplexer Geometrien, ist Rhino besonders für Tragwerksplanende eines Freiformbaus geeignet [37]. Die Entwicklung startete im Mai 1992 und erreichte einen großen Meilenstein im Oktober 1998 mit der Veröffentlichung von Rhino 1.0. Das zuständige Entwicklungsstudio Robert McNeel & Associates vertreibt die Software seit November 2020 in der aktuellen Version Rhino 7.0. Die Entwicklung ist noch nicht abgeschlossen [23].

Das Rhino-Paket umfasst weitere Programme, wie bspw. Grasshopper, die bereitgestellt

werden und aus der CAD-Oberfläche heraus verwendbar sind. Neben der Verwendung zahlreicher Entwicklertools, steht es der Community zu, Plugins für die Software zu entwickeln und diese zu veröffentlichen[12]. Die Verwendung dieser ist essenziell für diese Arbeit.

2.3.2 Grasshopper

Grasshopper ist ein in Rhino integrierter, grafischer Algorithmeneditor, der kaum Programmierkenntnisse benötigt. Das Programm wird aus Rhino heraus gestaret und ergänzt so die 3D-Modellierung [10]. Diese Synergie ermöglicht die Definition präziser, parametrischer Beziehungen in Modellen und somit einen generativen Entwurfsablauf, weshalb Grasshopper besonders bei der Formfindung und Freiformmodellierung sehr beliebt ist.
Diese grafisch basierte Programmierart wird auch visuelle Programmierung genannt (siehe Kap. [2.2.2]). Die logischen Elemente und Operationen werden durch miteinander verbundene funktionale Blöcke übernommen, die eine Sequenz von Aktionen ergeben, die in Grasshopper zu einer Geometrie oder Programmierungen führen ([12], S. 6f).

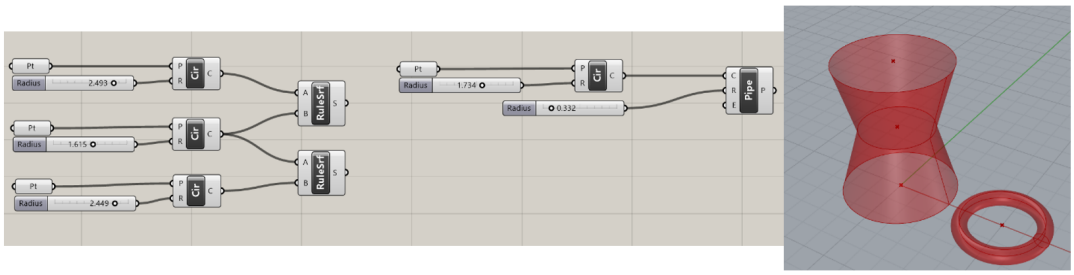


Abbildung 2.13: Beispiel Skript und Ergebnis

Quelle: eigene Darstellung

Ein Beispiel des Workflows in Grasshopper ist in Abbildung [2.13] dargestellt, die Programmierung erzeugt hier 3D-Objekte in Rhino. Die funktionalen Blöcke werden Komponenten genannt. Sie beinhalten die Definitionen für verschiedene Funktionen oder Objekte und werden auf der Leinwand platziert. Durch die Verbindung der Komponenten entsteht eine Datenweitergabe und somit die Abfolge von Aktionen, die in den sichtbaren Objekten resultieren.
Grasshopper hat sich von der reinen Anwendung zur parametrisierten Objekterstellung zu einem umfangreichen Tool diverser Programmieraufgaben weiterentwickelt. Es können Datenbrücken zu anderen Programmen, wie bspw. Sofistik oder Excel erstellt wer-

[12] Offizielle Community-Website: https://www.food4rhino.com/en

den, um die Modellierung automatisiert an eine Statikanalyse o.Ä. anzuknüpfen. Durch Built-In Programmiermodule wie GHPython kann in Grasshopper selbst textbasiert programmiert werden, was die Anwendbarkeit von Grasshopper immens steigert. Zusätzlich werden in Community-Foren Lösungen und Plugins bereitgestellt, um Grasshopper um viele Funktionen, wie auch welche zur Erstellung von KI, zu erweitern.

2.3.3 weitere Programme

Dlubal RSTAB/RFEM

Dlubal Software wurde 1987 gegründet und ist ein international tätiges Unternehmen, das sich auf die Entwicklung von Statik- und Dynamik-Software spezialisiert hat. Ihre Hauptprodukte sind RFEM und RSTAB, welche weltweit ihre Anwendung finden [15].
Das am häufigsten vertriebene RFEM ist eine leistungsfähige Finite-Elemente-Software, die für eine Strukturmodellierung, statische und dynamische Analyse sowie die Designoptimierung in 3D konzipiert ist. Für die Berechnung können beliebigen Strukturen Materialien zu gewiesen werden und die Ergebnisse dank Schnittstellen zu vielen gängigen CAD-Programmen oder einer anderen Software weitergegeben werden [16].
Änhlich zu RFEM aber in der Anwendung und Berechnung unterschiedlich, ist RSTAB für die Analyse und Bemessung von Stabwerken. Es bietet eine schnelle und effiziente Berechnung von Stabmodellen wie Träger-, Rahmengerüst- und Gitterstrukturen im Zwei- und Dreidimensionalen. Ähnlich wie RFEM bietet RSTAB eine Vielzahl von Schnittstellen für den Datenaustausch mit anderen Programmen [17].
Beide Programme, RFEM und RSTAB, bieten umfangreiche Funktionen für die Modellierung, Lastgenerierung und -kombination, Berechnung sowie die Ausgabe von Ergebnissen. Durch Zusatzmodule werden beide Programme um viele Möglichkeiten von Nachweisberechnung bis Knicklängenermittlung ergänzt [15].

SOFiSTiK

SOFiSTiK ist ein Softwarehersteller für Aufgaben wie Berechnung, Bemessung und Konstruktion im Bauingenieurbereich. Seit den anfänglichen Jahren 1974 an der TUM München als Forschungsprojekt, ist das Unternehmen für seine leistungsstarken und flexiblen Softwarelösungen bekannt.
Die SOFiSTiK Software bietet verschiedene Module und Anwendungen, die für die Modellierung, Analyse und das Design von Bauwerksstrukturen in 2D sowie 3D. Mit dem Einsatz von Finite-Elemente-Analysen wird ermöglicht, komplexe Strukturen zu analysieren und deren Verhalten unter verschiedenen Lasten und Bedingungen zu bestimmen.

Konträr zu Dlubal, findet die Arbeit in Sofistik oft textbasiert statt grafisch statt. Ein zentrales Element der Software ist die SOFiSTiK Structural Desktop (SSD) Benutzeroberfläche, welche als Plattform für die meisten Anwendungen und Module dient. Sie ermöglicht die Verwaltung der Projekte, die Kontrolle der Workflows und die Integration von Fremdprogrammen. Auch Sofistik stellt Datenbrücken zu anderen Programmen her und kann so in den Planungsprozess mit CAD-Programmen und Nachweiserstellungen eingebunden werden [40].

Excel

Microsoft Excel ist ein Tabellenkalkulationsprogramm, das von der Firma Microsoft entwickelt wurde. Es ist ein wichtiger Bestandteil der Microsoft Office-Suite und findet weltweit Anwendung. Excel-Tabellen sind in Matrixform, in der jede Zelle einer Spalte und Zeile zugeordnet ist. Bei der Bearbeitung können Zellen mit Funktionen manipuliert und auf den Inhalt anderer Zellen verwiesen werden. Excel dient so zum Daten organisieren, berechnen und analysieren. Durch Visual Basic for Applications (engl., kurz: VBA) lassen sich in der Programmiersprache der Microsoft-Office Programme, Excel-Dateien zusätzlich beeinflussen. Dank Schnittstellen zu anderen Programmen können die Vorteile der Datenverarbeitung in Excel optimal in einen Arbeitsprozess eingebracht werden und so bspw. eine Nachweisberechung oder Kostenkalkulationen ergänzen [28].

3 Voraussetzungen für den Einsatz von KI

In vorherigen Abschnitten wurde bereits die Struktur von Künstlicher Intelligenz erarbeitet und erklärt wie diese für verschiedene Anwendungsfelder zugänglich gemacht wird. Im Folgenden wird ermittelt, was es erfordert, eine KI in eine bei der Tragwerksplanung übliche Softwareumgebung zu integrieren und darin zu trainieren. Es werden verschiedene Möglichkeiten für die Verwendung von KI mit unterschiedlichen Programmen erarbeitet. Dabei wird grundlegend unterschieden zwischen Modellen die Grasshopper als Entwicklungsumgebung haben und solche, die Software getrennt funktionieren und damit eher einem konventionellen, textbasierten Ansatz für KI-Modelle ähneln. Die Workflows werden schematisch dargestellt und die verwendeten Komponenten erläutert.
Unabhängig von der Entwicklungsumgebung des Gesamtmodells bzw. des Programms, sind verschiedene Komponenten zur Geometrieabbildung, Nachweisberechnungen und KI-Entwicklung erforderlich, die dazu miteinander interagieren müssen. Tabelle [3.1] gibt eine Übersicht.

Aufgabe	**Programm**	**Programmalternative**	**Ziel**	
Geometrie	Grasshopper	Numerisch generiert mit Excel/Python	-Parametrisierte Modellierung	-Datensatz
Bemessung	Dlubal, Sofistik	Excel, Karamba, Python	-Nachweiserstellung zu Statik, Anschlüssen, etc.	
KI-Framework	Grasshopper	Python	-Erstellung KI	

Tabelle 3.1: Komponenten KI-Modell Entwicklung

3.1 Entwicklungsumgebung Grasshopper

Durch Plugins und Built-In Module kann Grasshopper selbst als Entwicklungsumgebung für den gesamten Prozess verwendet werden. Es können KI-Frameworks und Nachweismodule importiert und durch Python-Komponenten prozessunterstützende Programme geschrieben werden. Durch Datenbrücken findet ein Datentransfer zwischen verschiedener Software statt. Von der Auswahl an Möglichkeiten wird eine Übersicht gegeben und die jeweiligen Komponenten in Grasshopper erläutert.

3.1.1 Workflow

Datensatz

Grasshopper wird als Basis der gesamten Prozessschleife verwendet. Abbildung [3.6] gibt Aufschluss über das grundlegende Schema.

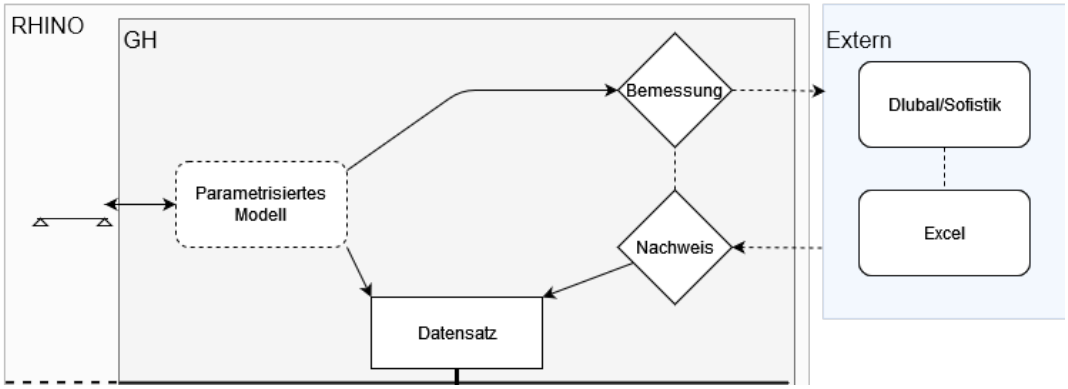


Abbildung 3.1: Schema Datensatzerstellung

Quelle: eigene Darstellung

Durch parametrisierte Tragwerksmodellierung kann ein ausreichend großer Datensatz für das Training einer KI entstehen. Das variable Modell entsteht durch den grafischen Code in Rhino und kann mittels integrierter Datenbrücken und dementsprechenden Komponenten in Grasshopper, an weitere Programme gegeben werden. Fehlt es an Schnittstellen, können diese mit Hilfe des GHPython-Modules geschaffen werden. Durch weitere Programme wie RSTAB/RFEM oder Excel folgen automatisiert Ergebnisse zu Statik und Nachweisen, die zurück in die Grasshopper Umgebung gespielt und dann in einer Tabelle abgespeichert werden, ehe sich dieser Prozess neuen Eingangsgrößen wiederholt. Es entsteht ein geschlossener Kreislauf der einen großen, gut zugänglichen Datensatz generiert. Hier wird festgelegt worauf die KI trainiert werden soll, da dass überwachte Lernen nur mit Eingangs- und Ausgangswerten funktioniert, die nach Abschluss des Trainings wiedergegeben werden können. Der ganze Workflow profitiert von der parametrisierten Tragwerksbildung, der gut verständlichen Syntax von Grasshopper und der Vielfalt an Plugins und Datenbrücken.

KI

Die KI soll ebenfalls in Grasshopper entwickelt werden. Durch Plugins werden Funktionen zur Erstellung Neuronaler Netzwerke in die Entwicklungsumgebung importiert und der Zugang zu den in Tabelle [2.1] dargestellten Frameworks ermöglicht. Die KI erhält Zugriff auf den generierten Datensatz und wird mit diesem trainiert.

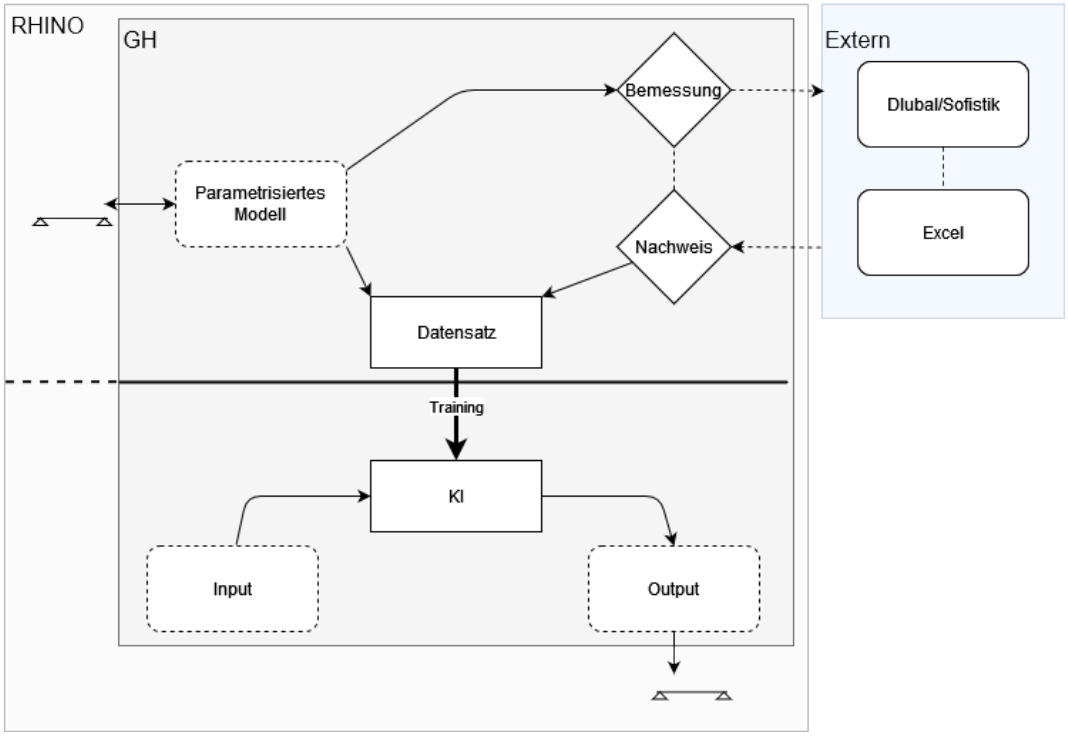


Abbildung 3.2: Schema Implementierung KI

Quelle: eigene Darstellung

In Schema [3.2] ist zu erkennen, wie die KI den Kreislauf der Datensatzerstellung ergänzt und nach abgeschlossenem Training eigene Vorhersagen treffen soll. Es werden die Zusammenhänge bei der Ergebnisfindung erlernt, ohne dabei Nachweise oder Statik selbst zu rechnen. Die Genauigkeit der Ergebnisse ist demnach begrenzt.

3.1.2 Python in Grasshopper

Der Zugang zu Python wird in Rhino und Grasshopper jeweils durch vorinstallierte Add-Ons ermöglicht. Dazu muss in Grasshopper die GHPython Komponente auf der Leinwand platziert und mit entsprechenden Inputs und Outputs verbunden werden. GHPython enthält ein Skripteditor der Python 2.7 verwendet, kann aber entsprechend konfiguriert werden. So erhält man direkten Zugang zu Python-, Rhino- und Grasshopperfunktionen [11]. Interpretiert wird der Code durch IronPython, eine Implementierung von Python die auf der .NET-Plattform läuft. Der direkte Zugang zu vielen CPython Bibliotheken wie PyTorch und Tensorflow ist dadurch erschwert [1].
Hierfür gibt es folgende Lösungsmöglichkeiten:

- GHPython remote

GHPython Remote ermöglicht das Nutzen von Standart-Python Code in der GHPython

Komponente [14]. Dazu verbindet sich GHPython mit einer externen Instanz von Python, auf der die gewöhnlichen Programme funktionieren [13]. Die Installation erfolgt in Form von Code in der Windowseingabeaufforderung, erfordert aber die Installation von Python 2.7 und optional das Plugin Anaconda [9].

- GH CPython

Durch GH CPython wird ein Plugin für die Implementierung von CPython gegeben, mit dem gleichen Ziel, die gewöhnlichen Pythonprogramme in Grasshopper zugänglich zu machen [35]. Dieses Plugin wurde für frühere Rhinoversionen entwickelt und ist nicht mehr aktuell. Für die weitere Bearbeitung spielt GH CPython keine Rolle [36].

3.1.3 Karamba3D

Karamba3D ist ein Plugin, das eine Finite-Elemente-Analyse in Grasshopper ermöglicht. So lässt es sich gut in die Parametrik von Grasshopper einbauen und wird so für FE-Berechnungen, Optimierungen und Tragwerksanalysen verwendet. Für weitere Anwendungen ist Karamba mit weiteren Plugins und Algorithmen kompatibel und bildet eine Alternative zu Dlubal, Sofistik und anderen Statikprogrammen, auch wenn es nicht so umfangreich wie eine Statiksoftware ist. Karamba3D wird von einem Arbeitskreis um Clemens Preisinger in Kooperation mit Bollinger und Grohmann ZT GmbH entwickelt und vertrieben [24][25].

3.1.4 Algorithmen in Grasshopper

Algorithmen sind die Kernelemente jeder KI. Durch erforderliche Optimierungsprozesse im Workflow wird ihre Relevanz für diese Arbeit zusätzlich verdeutlicht. Grasshopper verfügt über die Möglichkeit Algorithmen-Komponenten in das visuelle Programmieren aufzunehmen. So profitieren Anwendende von den verschiedenen Arten an Algorithmen (siehe Kapitel 2.2.2) und können diese für spezifische Aufgaben einsetzen. Galapagos ist der Name eines bekannten, bereits implementierten Algorithmus in Grasshopper, der für Optimierungsfunktionen eingesetzt wird. Von der breiten Vielfalt an Algorithmen werden einige relevante, mit Bezug auf ihre Funktion für KI, in Tabelle [3.2] dargestellt. Frameworks für KI zeichnen sich dadurch aus, dass sie in der Vielzahl der integrierten Funktionen u.A. bereits über die erforderlichen Algorithmen verfügen, weshalb bei der Verwendung von KI-Plugins die unten aufgeführten Algorithmen nicht zwingend erforderlich sind und als Zusatz dienen.

Name	Art/Aufgabe	Umsetzung	Version	Referenz
Galapagos	Optimierung	Evolutionäre Lösung	2019	grasshopperdocs.com - Galapagos
Octopus	Optimierung	Evolutionäre Lösung mit mehreren Zielen	2018	food4rhino.com - octopus
Quelea	Optimierung	Schwarmintelligenz, Ergebnisbewertung	2015	food4rhino - Quelea
Boid	Simulation von Schwarmverhalten	Schwarmintelligenz, Ergebnisbewertung	2015	food4rhino - boid-library
Nursery	Schwarmverhalten, Modellierung	Schwarmintelligenz	2017	food4rhino - nursery
Peacock	jewelry design/modeling	algorithmic jewelry modeling	2020	food4rhino - peacock
Wallacei	Optimierung	Evolutionäre Lösung mit mehreren Zielen	2022	food4rhino - wallacei wallacei.com
Opossum	Optimierung	Evolutionäre Lösung mit mehreren Zielen +Schwarmintelligenz	2022	food4rhino - opossum
Zebra	Schwarmverhalten, Modellierung	Schwarmintelligenz	2019	food4rhino - Zebra github.com - Zebra Sourcecode

Tabelle 3.2: Algorithmen in Grasshopper

3.1.5 KI in Grasshopper - Plugins

Die Implementierung von KI in Grasshopper wird durch verschiedene Community-Plugins ermöglicht. In Tabelle [3.3] sind einige davon aufgelistet. Die Plugins funktionieren mit bekannten KI-Frameworks und machen diese für die Grasshopper Programmierung zugänglich. So kann auf visuelle Weise ein Neuronales Netzwerk in Grasshopper programmiert und verwendet werden.

Name	Framework	Entwickler	Version	Referenz
Crow	NeuronDotNet	Benjamin Felbrich	2020	food4rhino - crow
Dodo	Neural Network on C# by Andrew Kirillov	Lorenzo Greco	2019	food4rhino - dodo raphos.com Codeproject.com
Lunchbox	Accord.NET	Nathan Miller und Proving Ground	2023	food4rhino - Lunchbox provingground.io bitbucket.org - Sourcecode
Owl	Accord.net	Mateusz Zwierzycki	2022	food4rhino - Owl github.com - Sourcecode + Update
Pug	Tensorflow.NET	Diego Apellániz	2023	food4rhino - Pug
Angora Angora2	AForge.NET	Fred Becquelin	2021	food4rhino - Angora 2 grasshopper3d.com - Angora
ANT	Scikit learn	Mahmoud Abdel Rahman	2017	github.com - ANT github.com - -ANT-

Tabelle 3.3: KI-Plugins in Grasshopper

Teilweise sind diese Plugins nicht mehr aktuell und scheiden für die weitere Bearbeitung aus. Auf aktuellem Stand sind:

- Lunchbox
- Pug
- Owl

Lunchbox

Das von Provinggground entwickelte Plugin Lunchbox, bringt durch LunchboxML ein Paket mit verschiedenen Anwendungen für Maschinelles Lernen in Grasshopper. Wie Abbildung [3.3] zeigt, bilden Accord.NET und ML.NET die Grundlagen des Plugins.

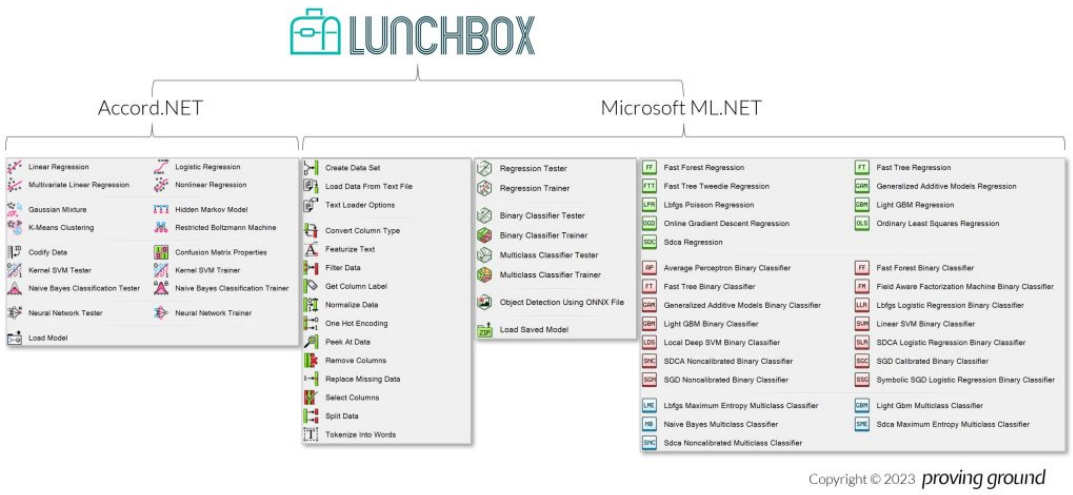


Abbildung 3.3: Umfang LunchboxML

Quelle: Dokumentation LunchboxML [33]

Die Accord.NET Implementierungen teilen sich in verschiedene Regressions-, Klassifizierungs- und SVM-Anwendungen. Zusätzlich wird die Möglichkeit gegeben, ein Neuronales Netzwerk zu erstellen, zu trainieren und zu laden. Die Algorithmenauswahl und die Netzwerkkomplexität sind dabei nur beschränkt möglich. Der Datensatz muss für die KI nicht konvertiert werden und ist jederzeit einfach kontrollierbar.

Für ML.NET werden verschiedene Komponenten zur Datensatzbearbeitung gegeben, die den Datensatz aus Grasshopper in ein ML.NET verständliches Format konvertieren. Mit diesem Datensatz werden Regressions, Klassifizierungs- und Objekterkennungsaufgaben trainiert. Trainierte Modelle können ebenfalls importiert werden. Für das KI-Training stehen verschiedene Algorithmen[1] zur Auswahl, welche die Brauchbarkeit des Modells maßgebend beeinflussen [33].

Ein fertiges Modell lässt sich in einem .zip-Ordner abspeichern, welcher den gleichen Aufbau wie ein in ML.NET programmiertes und gespeichertes KI-Modell hat. Es besteht demnach die Möglichkeit konventionell programmierte KI-Modelle aus ML.NET in Grasshopper zu laden.

Weil die beiden Frameworks kaum bzw. keine Funktionen für Neuronale Netzwerke importieren, werden die KI-Modelle nach hinterlegten Algorithmen trainiert und können damit Zusammenhänge im Datensatz finden. D.h. die Anwendenden legen fest wie die KI lernt und welche Zusammenhänge erlernt werden sollen. Für ein nicht-lineares Problem bedeutet das die Angabe einer nicht-linearen Beziehung der Inputvariablen für Training und Anwendung der KI oder der Verwendung von Kernel-Tricks oder einer Transformation

[1] ML.NET Dokumentation: Algorithmen, https://learn.microsoft.com/de-de/dotnet/machine-learning/how-to-choose-an-ml-net-algorithm

der Daten (siehe auch Kapitel [2]).

Pug

Durch die Installation von Pug wird Grasshopper um einige Elemente des Frameworks Tensorflow erweitert und bringt so einen Zugang zu Neuronalen Netzwerken und Tiefem Lernen. Entwickelt wurde Pug von Diego Apellániz mit Unterstützung von Björn Pettersson [2].

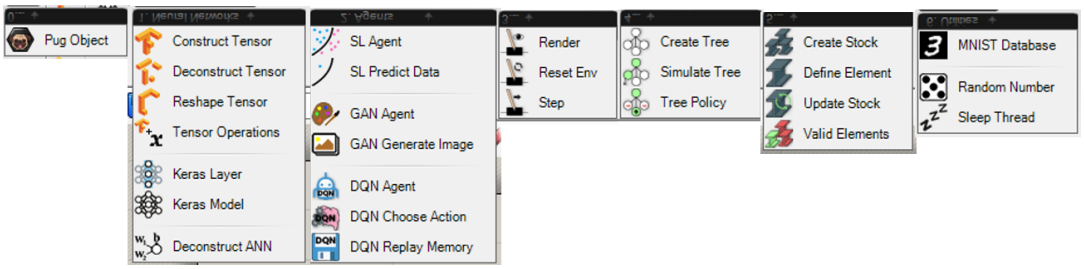


Abbildung 3.4: Umfang Pug

Quelle: eigene Darstellung

In Abbildung [3.4] ist der Umfang des Plugins zu erkennen. Um die Datenverarbeitung mit Tensorflow zu ermöglichen, stehen verschiedene Komponenten zur Tensorerstellung und Manipulierung zur Verfügung. Diese durchlaufen ein Neuronales Netzwerk, welches sich aus den Keras-'Bausteinen' bildet. Mit dieser Auswahl können innerhalb von Pug, Regressions- und Klassifizierungsprobleme, sowie verschiedene GAN und DQN-Agenten trainiert werden. Für die Anwendung und das Training der KI, gibt es noch weitere Elemente der Datenverarbeitung wie bspw. handschriftliche Zahlenbilder der MNIST-Database für das Training eines GAN Agents zur Erkennung und Zeichnung neuer handschriftlicher Zahlen.

Owl

Owl wurde von Mateusz Zwierzycki entwickelt und bringt eine Implementierung des Frameworks Accord.NET für Grasshopper.

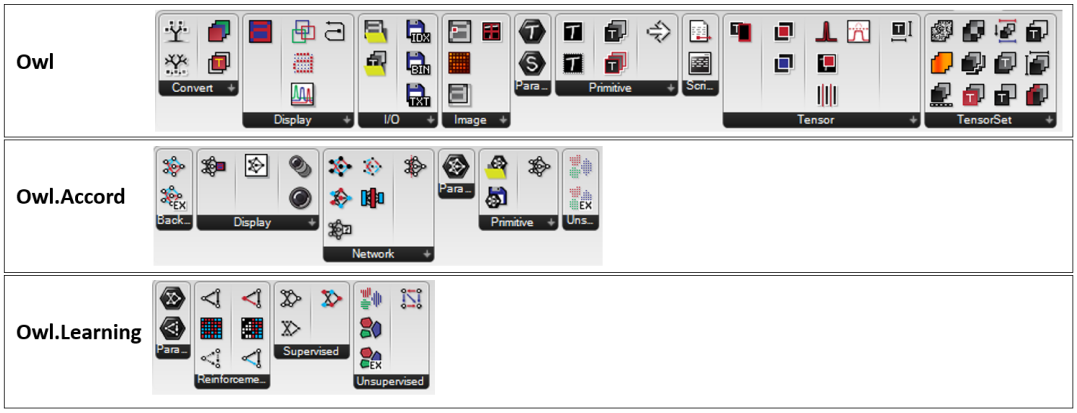


Abbildung 3.5: Umfang Owl

Quelle: eigene Darstellung

Wie in Abbildung [3.5] erkenntlich, teilt sich das Plugin in drei Bereiche auf. Mit Owl wird der Datentransfer zur KI durch Komponenten der Tensorerstellung und -manipulation ermöglicht. Mit Owl.Accord lässt sich ein Accord.NET Netzwerk bearbeiten. In Owl.Learning kann zwischen den drei Trainingsmöglichkeiten für KI gewählt werden. Mit diesen 3 Paketen werden Werkzeuge für die Datenverarbeitung und den Umgang mit Accord.NET gegeben. Dies ermöglicht Datennormalisierung, Verarbeitungs- und Serialisierungsmethoden, Visualisierungen und letztendlich die KI-Entwicklung [46][47].

3.2 Alternative ohne Grasshopper

Konträr zu den bisher betrachteten Lösungsansätzen, gibt es die Möglichkeit die KI-Komponenten des Workflows aus Grasshopper herauszulösen und extern über Python zu programmieren. Grasshopper als Entwicklungsumgebung für Datensatz und KI-Training kann, abhängig von der Problemstellung, hinderlich sein, weshalb im Folgenden ein von dieser Software getrenntes Modell, mit Datentransfers zwischen den Programmen, alternativ betrachtet wird.

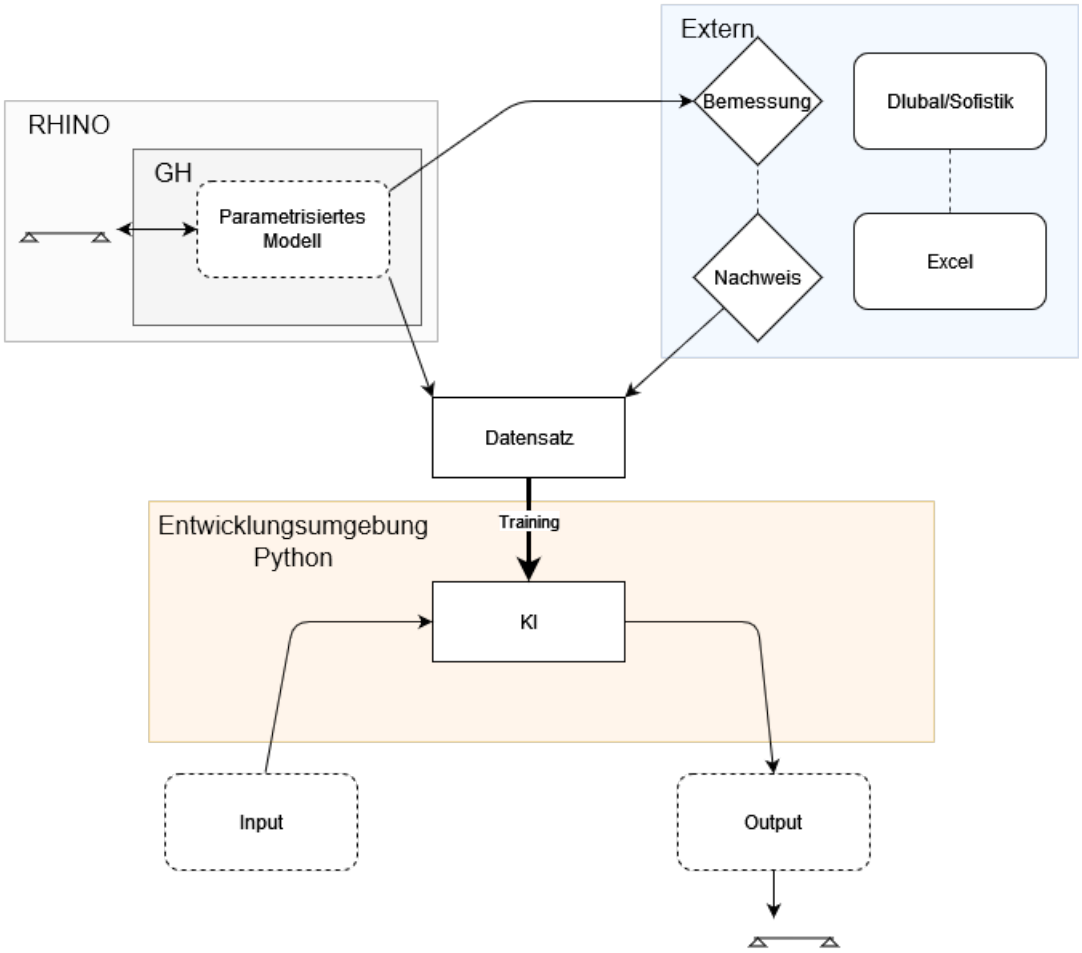


Abbildung 3.6: Schema Datensatzerstellung

Quelle: eigene Darstellung

Die Datensatzerstellung kann ebenfalls aus Grasshopper heraus erfolgen und beruht auf der Synergie zu weiteren Programmen für Statik und Bemessung. Die Ergebnisse dieser Berechnungen werden klar strukturiert und tabellarisch festgehalten. Da ein kompletter Datensatz in Grasshopper mit Listen abgebildet wird oder die Daten einer Berechnungsschleife in einer externen Textdatei abgespeichert werden müssen, müssen die Daten immer nach einer klar strukturierten Form abgelegt werden, um das KI Training überhaupt zu ermöglichen. Aufgrund dieser Datenstruktur, die die Eingangsdaten verschieden kategorisiert oder diesen gewisse Eigenschaften zuordnet, bietet sich für weniger komplexe Aufgaben eine alternative Datensatzerstellung mit Python für Geometriebildung und Statik- bzw. Nachweisrechnung an, die zum gleichen Ergebnis führt.

Die erhaltene Ergebnistabelle wird für das KI-Training über Python Code ausgelesen und in das Neuronale Netzwerk gespeist. Das KI-Modell wird mithilfe gängiger KI-Frameworks (siehe Tabelle [2.1]) programmiert und liegt in einer Pythonentwicklungsumgebung vor. Die KI bekommt ihren Input in Form von Programmiercode, für die bessere Anwendung

lässt sich die Eingabe in Python durch grafische Benutzeroberflächen oder in diesem Fall einer GHPython Komponente in Grasshopper ersetzen. Grasshopper dient dann der Visualisierung.
Nach abgeschlossenem KI-Training kann die KI entweder über Befehlseingaben in Python oder über eine Verbindung zu Grasshopper verwendet werden. In einer GHPython-Komponente kann nach dem Training entweder eine Zugriffsverbindung zur fertigen KI hergestellt werden, wenn diese auf einer Instanz außerhalb Grasshopper läuft oder der Quellcode des KI-Modells bzw. eine Initialisierung dieses Modells in eine GHPython remote ergänzte GHPython Komponente eingefügt werden. Letztere Lösung kommt den in Tabelle [2.1] aufgelisteten Plugins mit KI-Frameworks nahe.
Der zentrale Aspekt dieses Workflows besteht in einem KI-Training ohne Grasshopper, wobei alle anderen erforderlichen Schritte sehr gut mit Grasshopper umzusetzen sind.

Wenn umfangreiche Systeme mit großen Datenmengen und viel Rechenleistung erfordern, stellt die Programmierung in einer Pythonentwicklungsumgebung eine gute Alternative dar. Es besteht einerseits voller Zugriff auf Python und die dortigen KI-Frameworks, andererseits wird weniger Rechenleistung als in Grasshopper benötigt. Allerdings lässt sich eine KI in dieser Umgebung nicht so leicht wie Grasshopper Plugins, in einen Planungsprozess einbinden.
Ziel dieser Arbeit ist die Untersuchung der Anwendbarkeit einer KI-Implementierung in die Parametrik von Grasshopper für einen umsetzungsfreundlichen Workflow, was in der weiteren Bearbeitung eine Vernachlässigung dieser Möglichkeit zur Folge hat, da die KI-Plugins die gleiche Funktionalität bei geringerem Aufwand bieten und die Nutzung der KI-Plugins in Grasshopper, im Vordergrund steht. Es sei erneut angemerkt, dass alle Plugins in Grasshopper auf dem Programmiercode ihrer Frameworks basieren und somit die KI-Plugins genau gleich wie eine herkömmlich programmierte KI funktionieren, was die Kenntnis über die jeweiligen KI-Frameworks unumgänglich macht und der entscheidende Unterschied in der Verwendung unterschiedlicher Programmiersyntaxen besteht.

4 KI für Planungsaufgaben - Auswahl und Umsetzung

Dieses Kapitel widmet sich der Anwendung der zuvor behandelten Themen. Dazu soll eine KI auf die Vorhersage der erforderlichen Querschnittsabmessungen eines Fachwerkträgers trainiert werden. Damit dies gelingt, wird eine Analyse der verwendeten Software vorangestellt und aus diesem Ergebnis Schlussfolgerungen für die weitere Arbeit gezogen. Anhand von Beispielen wird erörtert, welche Probleme und Herausforderungen eine KI-Erstellung in Grasshopper mit sich bringt. Alle behandelten Aufgabenstellungen erfordern eine KI für ein Regressionsproblem, weshalb die Plugins nur darauf ausgewertet werden. Problemstellungen der Datenklassifizierung oder -generierung, denen mit manchen Plugins nachgegangen werden könnte, werden nicht untersucht.

4.1 Pluginauswahl

Für das Ziel eines KI-Trainings auf eine nicht-lineare Planungsproblemstellung, stellen Ergebnisgenauigkeit und Anwendbarkeit die wichtigsten Merkmale für den Erfolg des Modells dar. Vor Erstellung eines KI-Modells für das Hauptproblem, ist es ratsam, die verschiedenen Plugins auf diese Ergebniseigenschaften und ihre Fähigkeit diese zu erreichen, genauer zu untersuchen. Ziel des Vergleichs soll die Wahl eines Plugins für die weitere Verwendung und eine Darlegung des Skriptaufbaus sein.
Die Plugins werden mit dieser exemplarischen Querschnittsbemessung eines Einfeldträgers vergleichend evaluiert:

- Gegeben: Last: load, Länge: length, Breite: b; zul. Biegespannung: fmk

$$load \in [0, 20]; length \in [0, 50]; b = 0.2m; fmk = 24 \tag{4.1}$$

- Gesucht: erforderliche Höhe h aus

$$h = \sqrt{(6 * M)/(b * fmd)} \tag{4.2}$$

und

$$M = q * l^2/8 \tag{4.3}$$

Die KI wird darauf trainiert, die erforderliche Querschnitthöhe für eine Last und Spannweite eines Einfeldträgers anzugeben. Es sei angemerkt, dass sich hier ein Datensatz mit 1000 Einträgen ergibt, welcher ausreicht, eine KI auf eine brauchbare Aussage zu

trainieren. Der komplette Skriptaufbau der jeweiligen Modelle kann dem Anhang [B] entnommen werden.
Für die Untersuchung werden Lunchbox und Pug verwendet. Mit Lunchbox wird ein Framework für Maschinelles Lernen untersucht, mit Pug eines für Tiefes Lernen. Vom Plugin Owl wird abgesehen, da dieses, ähnlich wie Pug, ebenfalls Möglichkeiten zum Tiefen Lernen bietet, mit Accord.NET jedoch ein älteres KI-Framework, als Pug mit Tensorflow, benutzt (siehe auch Tab. [2.1]).

4.1.1 Datensatz Einfeldträger

Der Datensatz bildet sich über Listen in Grasshopper. Die Variablen *load* und *length* werden kombiniert und ergeben zusammen mit der Breite *b* und der zulässigen Biegespannung *fmk* nach obiger Formel die erforderliche Höhe *h*. Für die Kombination der Eingangsvariablen wird eine Liste load generiert, die jede Zahl aus der Reichweite von *load* (hier von 0 bis 19) für die Länge der Liste length (len(length) = 50) wiederholt. In der Liste length wird die Zahlenreihe der Elemente (von 0 bis *length*) für jeden Zahlenwert von *load* generiert. In Python lässt sich das durch diesen Code ausdrücken:

Listing 4.1: Python Code - KI Querschnittsermittlung

```
load = 20
length = 50
load_list = []
length_list = []

for load_value in range (0, load):
    for length_value in range (0, length):
        load_list.append(load_value)
        length_list.append(length_value)
```

Die Kombination der Listenwerte mit den selben Indizes ergeben den Datensatz dargestellt in Tabelle 4.1:

load [kN/m]	0	0	...	2	2	...	19	19	19
length [m]	0	1	...	5	6	...	47	48	49
req. h [m]	0	0	...	0.095015	0.114018	...	2.752834	2.811405	2.869976

Tabelle 4.1: Datensatz Querschnittshöhe Einfeldträger

Dieser nicht-lineare Datensatz dient als Grundlage der folgenden KI-Evaluation.

4.1.2 Lunchbox

In Lunchbox werden die Komponenten von ML.NET verwendet, da hier mehr Möglichkeiten der Modellausbildung und Steuerung gegeben sind als bei den Komponenten von Ac-

cord.NET. Das Training dieses nicht-linearen Problems muss, bei Verzicht auf SVM oder Kernel-Tricks, um nicht-lineare Inputs ergänzt werden, um sich dem Ergebnis überhaupt nicht-linear annähern zu können. Dazu wird der Datensatz zur in Tabelle [4.2] gegebenen Form ergänzt. Anwendende haben dadurch keinen Mehraufwand in der Nutzung des KI-Modells, da bei weiteren Eingaben die zwei Eingangsgrößen um die unten dargestellten Formen ergänzt werden. Die Datensatzerstellung wird für Softwareentwickelnde jedoch verkompliziert und aufwändiger.

load [kN/m]	0	0	...	2	2	...	19	19	19
load**2	0	0	...	4	4	...	361	361	361
sqrt(load)	0	0	...	1.414	1.414	...	4.359	4.359	4.359
length [m]	0	1	...	5	6	...	47	48	49
length**2	0	1	...	25	36	...	2209	2304	2401
sqrt(length)	0	1	...	2.236	2.45	...	6.856	6.928	7
M [MNm]	0	0	...	0.00625	0.009	...	5.2464	5.472	5.702
req. h [m]	0	0	...	0.095015	0.114018	...	2.752834	2.811405	2.869976
(req. h)**2	0	0	...	0.009026	0.013	...	7.5781	7.904	8.2368
sqrt(req. h)	0	0	...	0.308244	0.033767	...	1.66	1.677	1.694

Tabelle 4.2: Datensatz EFT Lunchbox

Für den Zugang zu ML.NET müssen die Daten durch ML.NET Komponenten zu einem für dieses Framework verständlichen Datensatz konvertiert werden und eine Zuordnung von Features und Labels erfolgen. Eine *Regression Trainer*-Komponente erhält all diese Informationen zusammen mit einer Auswahl des Trainingsalgorithmus. Für eine derartige Regressionsaufgabe werden in ML.NET üblicherweise *Online Gradient Descent Regression* oder *Ordinary Least Squares Regression* Algorithmen verwendet.
In einem *Regression Tester* kann das Training mittels Testdaten kontrolliert werden. Für das zu untersuchende Beispiel führt dieser Sachverhalt zu folgendem Ergebnis:

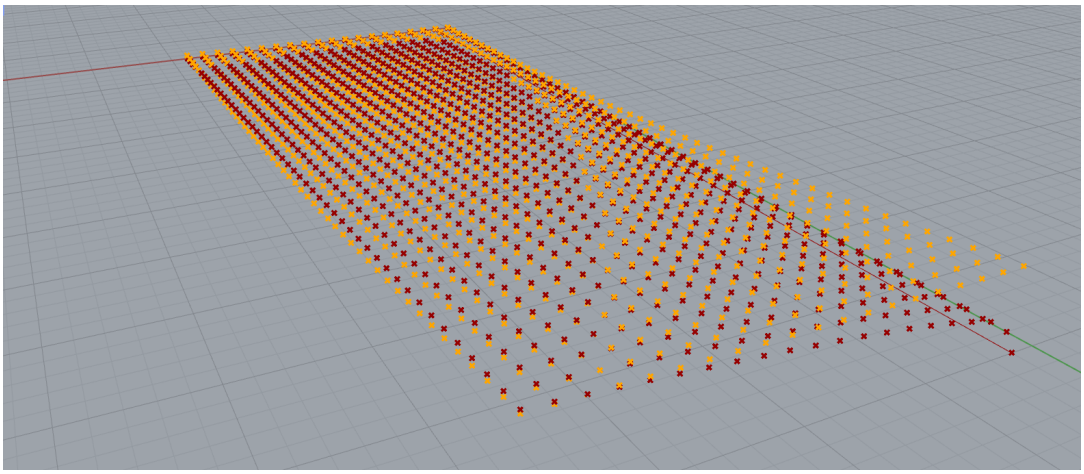


Abbildung 4.1: Ergebnisse Lunchbox EFT Berechnung

Quelle: eigene Darstellung

Die Annäherung an die Lösung erfolgt hier aufgrund der oben aufgeführten Inputs mit folgender grundlegenden Formel, worin die Gewichte w_i und das Bias b während des Trainings angepasst werden:

$$h = w_1 x + w_2 x^2 + w_3 \sqrt{x} + w_4 y + w_5 y^2 + w_6 \sqrt{y} + w_7 M + w_8 (req.h)^2 + \sqrt{req.h} + b \quad (4.4)$$

In Darstellung [4.1] entspricht die X-Achse dem Lastwert und die Y-Achse der Länge des Einfeldträgers. Auf der Z-Achse wird die erforderliche Querschnitthöhe dargestellt. Die roten Punkte bilden den Trainingsdatensatz ab, der wiedergegeben werden soll. Die KI-Ergebnisse (orange) weisen besonders bei höheren Werten für die Länge und niedrigeren Werten für die Last eine ansteigende Differenz zu den Testwerten auf. Das resultiert aus einer mangelhaften Erfassung des nicht-linearen Verhältnisses auf die Gesamtfunktion.

Es ist unüblich eine KI mit den exakt gleichen Daten zu trainieren und zu testen, wie es hier der Fall ist. Dieses Vorgehen kommt einer Schulklausur mit Bekanntgabe der Lösungen vor dem eigentlichen Test nahe. Es soll an dieser Stelle lediglich verdeutlicht werden, dass die KI unabhängig von ihrer Lernfähigkeit, für das Ziel einer Vorhersage unbekannter Daten, bereits einen bekannten, nicht-linearen Sachverhalt, nicht ausreichend genau wiedergeben kann.

Es zeigt sich zudem, welcher Aufwand damit verbunden wäre die KI mit genügend transformierten Inputvariablen zu trainieren, um eine ausreichende Genauigkeit zu erzielen. Für die gewünschte simple Anwendung in Grasshopper wirkt sich dies negativ aus. Daraus folgt, dass durch Lunchbox zwar der Zugang zu hilfreichen Funktionen der beiden Frameworks gegeben wäre, deren Nutzen für diese Arbeit jedoch nur gering ist. Ein KI-

Training mit .NET und ML.NET kommt zum gleichen Ergebnis, was eine Einschränkung durch Probleme mit dem Plugin Lunchbox, ausschließt.

4.1.3 Pug

In Pug kann der ursprüngliche Datensatz ohne ergänzende Inputvariablen verwendet werden. Für den Zugang zu Tensorflow werden die Listen *load* und *length* in einem Inputtensor der Form (engl.: shape) *2* dargestellt, die Liste der erforderlichen Querschnittshöhe in einem Outputtensor der Form *1*, siehe auch Abbildung [4.2]. Die Stapelgröße (engl.: batch size) entspricht der Größe des Datensatzes und kann als Anzahl der Zeilen der Matrizen verstanden werden. Der ”Stapel” wird im Training paarweise durchgegangen und der Zusammenhang zwischen den Paaren erlernt. Ein Paar besteht hier aus Input- und Outputtensor, wobei diese jeweils zwei bzw. einen Eintrag umfassen (Shape=2 und Shape=1). Die Batchsize muss für beide Tensoren die gleiche sein, weil sonst Einträge nicht zugeordnet werden können, und entspricht hier einem Wert von 20*50=1000.

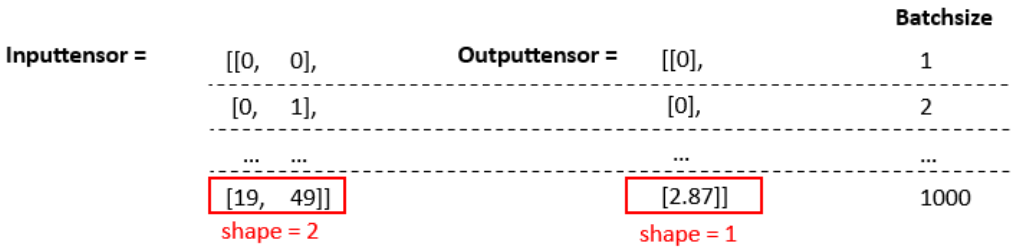


Abbildung 4.2: Tensorvisualisierung EFT

Quelle: eigene Darstellung

Um ein Auswendiglernen des Datensatzes der KI beim Training zu verhindern, werden die Inputlisten zufällig neu angeordnet. In Grasshopper ist dies mit der Komponente *Jitter* umsetzbar. Der Inputtensor (”x”) und Outputtensor (”Y”) werden mit einem *SL-Agent* verbunden.

An dieser Komponente werden zusätzlich Informationen zum Trainingsumfang (Batchsize, Epochen, Validationsplit), ein Pfad zum Speichern der Variablen des trainierten Neuronalen Netzwerks und die eigentliche Struktur dessen, eingesteckt. Das Training wird über eine Knopf (engl.: button) gestartet. Der genaue Aufbau sowie die getroffenen Trainingseinstellungen sind in Abbildung [4.3] dargestellt und können Tabelle [4.3] ergänzend entnommen werden:

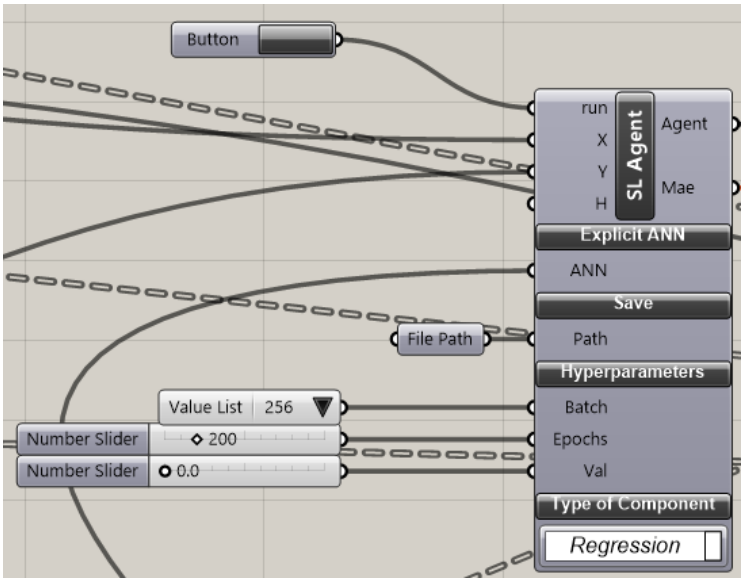


Abbildung 4.3: Regressionstrainer, SL-Agent
Quelle: eigene Darstellung

Aufbau NN	**Layer**	Dense/Input	Dense	Dense	Dense	Dense	Dense/Output
	Neuronen	shape=2	64	64	64	64	1
	Aktivierungs-funktion		ReLU	ReLU	ReLU	ReLU	
Batchsize		256	**Epochen**	200	**Validation Split**		80/20 %
Optimierungsfunktion		Adam	**Lernrate**	0.001	**Verlustfunktion**		Mae

Tabelle 4.3: Aufbau NN + Trainigseinstellungen

Die Daten werden in 80% Trainingsdaten und 20% Testdaten unterteilt, wobei der Datensatz um 20% mehr Daten ergänzt wird. Am SL-Agent wird ein *Validation-Split* von 0.0 eingestellt, damit der komplette Trainingsdatensatz eingelesen wird. Die KI trainiert demnach mit den exakt gleichen Daten wie bei Lunchbox, d.h. auch mit dem gleichen Umfang von Batchsize = 1000.

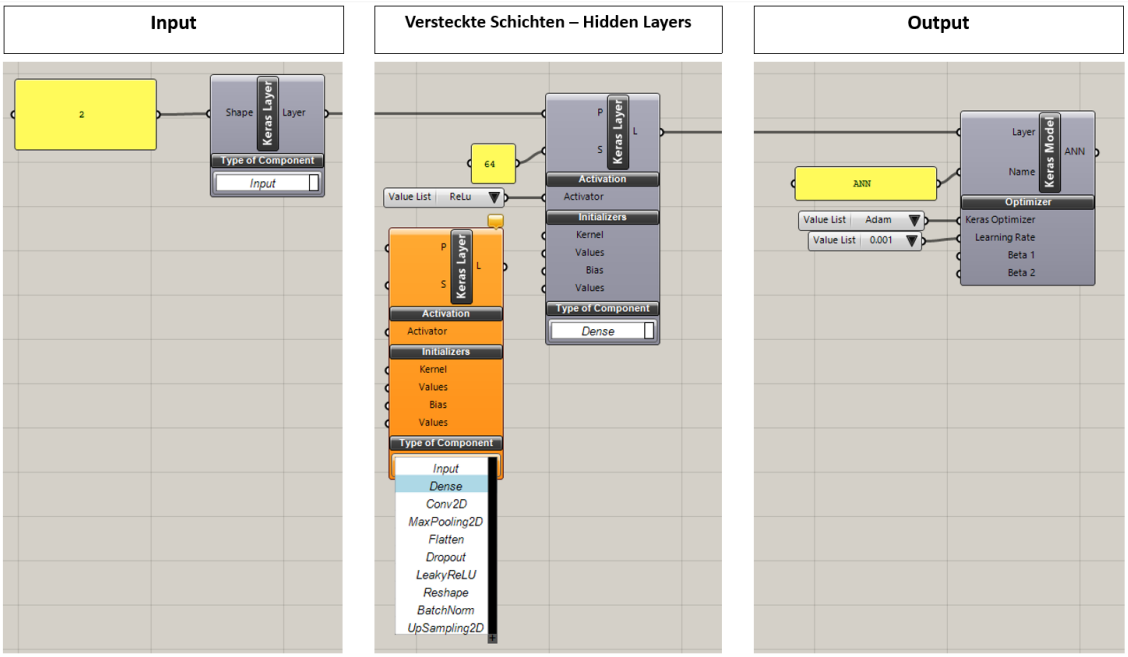


Abbildung 4.4: Komponenten Neuronales Netzwerk

Quelle: eigene Darstellung

Seit Tensorflow 2.0 ist Keras[1], was einst ein eigenes Framework war, standardmäßig fest in dieses integriert [21]. Keras dient als Anwendungsprogrammierschnittstelle (engl.: Application Programming Interface, kurz: *API*) zur Erstellung der Schichten des Netzwerks. Abbildung [4.4] zeigt welche Komponenten zur Verfügung stehen und welche Auswahlmöglichkeiten es gibt. Aus einem Vergleich mit Tabelle [4.3], gehen u.A. folgende Funktionen als relevant hervor:

- Input-Layer: 1x pro Neuronales Netzwerk, unabh. vom Aufbau des NN
  Dient als Eingangspunkt in das Neuronale Netzwerk. Erhält die Daten aus den Tensoren und gibt diese an das NN weiter.

- Dense-Layer[2]: variable Anzahl an Neuronen und Häufigkeit im NN.
  Verbindet jedes Neuron mit jedem Neuron in der vorherigen und der nachfolgenden Schicht und wird bei der Datenweitergabe von einer Aktivierungsfunktion bzw. einem ”Activator” beeinflusst. Dient als Grundbaustein eines Neuronalen Netzwerks.
  -> ReLU Aktivierung (engl.: rectified linear unit activation)[3]: Wählt für jeden Wert der berechnet wird das Maximum zwischen dem eigentlichen Wert und 0 (standardmäßig eingestellt) und gibt das Ergebnis weiter. So werden negative Werte gleich null gesetzt und ein nicht-linearer Sachverhalt bei der Datenweitergabe im Neuronalen

[1] Keras-offizielle Website- https://keras.io/
[2] Tensorflow Dokumentation: Dense - https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
[3] Tensorflow Dokumentation: ReLU - https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu

Netzwerk eingeprägt.

- Keras Model: 1x pro Neuronales Netzwerk, unabh. vom Aufbau des NN
  Erhält den letzten Layer als Eingabe und bietet Möglichkeiten zum Einstellen der Optimierungsfunktion, der Lernrate und Beta-Werten. Fasst das Neuronale Netzwerk unter einem Namen zusammen und gibt alle Informationen an den SL-Agent zurück.
  -> Adam Optimierer[4]: Benutzt eine stochastische Gradientenabstiegsmethode zur Annäherung der Daten während des Trainings.
  -> Lernrate: Gibt an in welchen Schritten sich die KI pro Iterationsschritt dem Ergebnis annähert.
  -> Beta Werte: Beeinflusst die Schätzung der Momente im Optimierer.

Als Verlustfunktion bietet PUG den *Mean Absolute Error* (engl., kurz: MAE). Der mittlere absolute Fehler/Unterschied zwischen den tatsächlichen Labels und den vorhergesagten Werten wird nach dieser Formel berechnet:

$$MAE = \frac{1}{n} * \sum_{j=1}^{n} \mid Y_j - Y_{pred,j} \mid \tag{4.5}$$

Worin *n* für die Anzahl der Datenpunkte steht und den Betrag der Summe der Subtraktion der tatsächlichen Werte (*Y*) mit den vorhergesagten Werten (*Ypred*) für jeden Datenpunkt, teilt. So entsteht eine mittlere durchschnittliche Differenz, die für ein gutes KI-Training möglichst gering ausfallen sollte. Der MAE wird für jeden Stapel berechnet, weshalb n der Batchsize entspricht[5]. Neben dem MAE gibt es noch weitere Fehlerfunktionen in Tensorflow, welche der SL-Agent für Regression in Pug nicht berechnen kann. Die Visualisierung des Fehlers dient der Trainingskontrolle und gibt eine anschauliche Auskunft über den Trainingsverlauf, was Grundlage für die Optimierung des Neuronalen Netzwerks und der Trainingseinstellungen ist. Bei der Verwendung von Grasshopper entsteht hierbei ein Vorteil in der Ergebnisdarstellung, da mit Rhino bereits ein Visualisierungsprogramm für die Abbildung dreidimensionaler Sachverhalte gegeben ist und sich dies, wie nachfolgend erklärt und bereits in Kapitel [4.1.2] angewendet, ebenfalls für die Ergebniskontrolle maximal dreidimensionaler Problemstellungen, nutzen lässt.

Mit dem oben beschriebenen Aufbau liefert Pug für das Untersuchungsbeispiel das in Abbildung [4.5] dargestellte Ergebnis.

[4] Keras.io: Adam - https://keras.io/api/optimizers/adam/
[5] Tensorflow Dokumentation: MeanAboluteError - https://www.tensorflow.org/api_docs/python/tf/keras/losses/MeanAbsoluteError

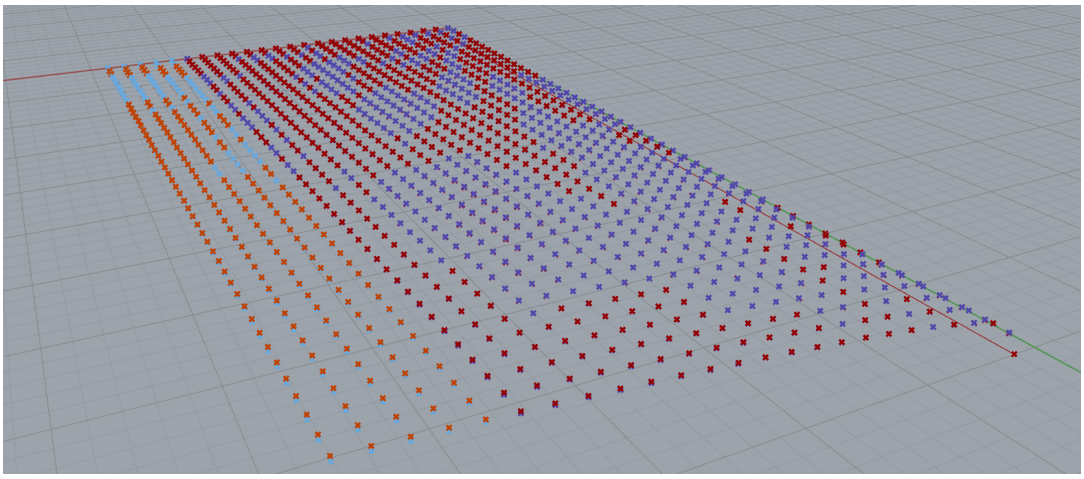


Abbildung 4.5: Ergebnisse Pug EFT Berechnung

Quelle: eigene Darstellung

Rote und orange Punkte entsprechen den Soll-Werten und dienen der Ergebnisevaluation. Die dunkel-blauen Punkte sind die Ergebnisse der KI auf Daten innerhalb des Trainingsbereichs (80% Trainingsdaten). Die geringe Differenz zwischen Soll- und Ist-Werten in diesem Bereich hat wenig Aussagekraft über ein erfolgreiches KI-Training, repräsentiert jedoch die Fähigkeit des Modells nicht-lineare Zusammenhänge zu verstehen sehr gut. Die hell-blauen Punkte sind Ergebnisse der KI auf neue Testdaten (20% Testdaten). Die ausreichend genaue Annäherung an die Ist-Werte (orange) deutet auf ein akzeptables KI-Modell hin.

Es wird erkenntlich, dass sich mit Pug nicht-lineare Zusammenhänge trainieren lassen und die Ergebnisse auch auf neue Problemstellungen übertragbar sind. Der Datensatz muss nicht erweitert werden und das fertig trainierte Netzwerk kann für erneute Verwendung abgespeichert werden.

In Punkten Brauchbarkeit und Ergebnisgenauigkeit liefert Pug demnach deutlich bessere Ergebnisse als Lunchbox, was die Eignung von Pug für den weiteren Verlauf dieser Arbeit bestätigt.

4.2 Fachwerk KI

Die Möglichkeit zur Erstellung brauchbarer Neuronaler Netzwerke mittels Pug wurde gezeigt. Erste Erläuterungen und Erkenntnisse über das Plugin sind vorhanden und sollen nun auf eine Planungsaufgabe des Bauingenieurwesens übertragen werden.

4.2.1 Praxisbeispiel

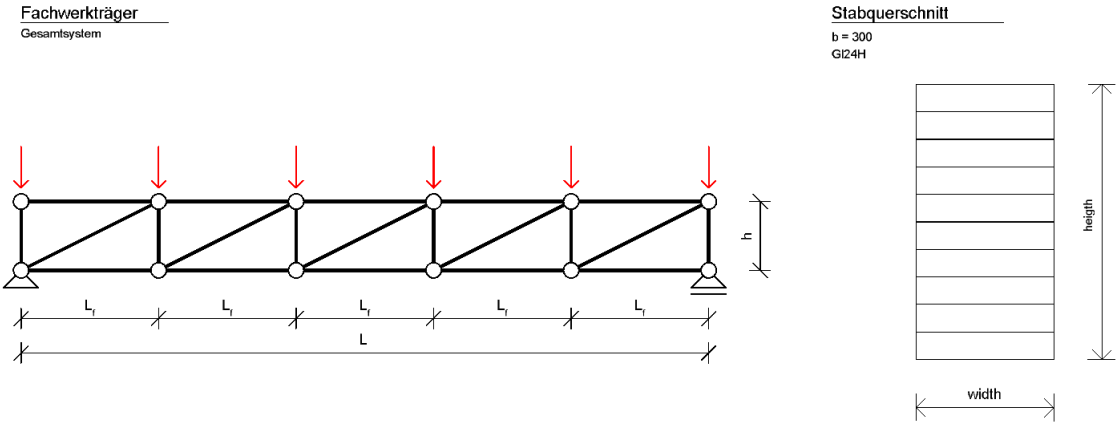


Abbildung 4.6: Fachwerkträger Aufgabenstellung

Quelle: eigene Darstellung

Das dargestellte System kommt in der Praxis bspw. in Form von Aussteifungsverbänden gegen Horizontallasten in Dächern oder anderen Scheiben zum Einsatz. Trifft Wind auf eine Fassade, muss diese die Last abtragen und gibt dazu die entstehenden Kräfte an dafür bemessene Bauteile weiter. In einem Dach ist es so oft der Fall, dass aus der vollflächig auftreffenden Windlast durch Bauteile wie Pfetten oder Sparren, die die Kraft weiterleiten, eine Punktlast am aussteifenden Verband wird.

4.2.2 Datensatzerstellung

Die Ausbildung eines Fachwerkträger kann stark variieren. Geometrie, Baustoffeigenschaften und statisches System liefern so viele Variationen, dass für das KI-Training festgesetzte Randbedingungen getroffen werden müssen, um die Datensatzgröße einzuschränken:

- Die Geometrie und das statische System weichen nicht vom oben dargestellten ab, mit Ausnahme der Felderanzahl. Alle Felder haben die gleichen Abmessungen.
- Alle Stäbe sind Fachwerkstäbe
- Höhe (engl.: height) h = 1m
- Querschnittbreite (engl.: sectionwidth) b = 300mm
- Brettschichtholz GL24h: fck = 24 N/mm^2; ftk = 16,5 N/mm^2
- Teilsicherheitsbeiwert = 1,3; kmod = 0,9

- Kein Stab hat eine Knicklänge größer als Lcr= 3m

Als Variablen bleiben:

$$load \in [0, 20]; length \in [0, 50]; fields \in [1; 10] \tag{4.6}$$

Für andere Fachwerke müsste eine weitere KI mit einem anderen Datensatz trainiert werden, was nicht Ziel dieser Ausarbeitung ist.
Weil eine Regression auf der Modellierung einer kontinuierlichen Beziehung zwischen Eingabevariablen und einer Zielvariablen beruht, was das KI-Training generell auf eine Outputvariable pro Input bzw. pro Batch beschränkt, empfiehlt es sich den Datensatz stabweise zu generieren:

Gesamtsystem			**Stab**			
Last [kN]	**Länge [m]**	**Felder**	**Position**			**erforderliche Querschnittshöhe [mm]**
			Feld	**Stab Text**	**Stab Nummer**	
0	0	1	1	Obergurt	1	0
...	...	...	...	...	...	...
10	28	9	8	Pfosten	2	80
...	...	...	...	...	...	...
19	49	10	10	letzter Pfosten	5	150

Tabelle 4.4: Datensatz Fachwerkträger

Pro Zeile des Datensatzes wird das Verständnis über das gesamte Fachwerk (Last, Länge, Felder) gegeben und dann Auskunft über den betreffenden Stab (Feld, Position im Feld als Text und Nummer, erforderliche Querschnittshöhe) ergänzt. Die Position jedes Stabes im Feld (Feldernummerierung von links beginnend) wird der KI numerisch mitgeteilt (1=Obergurt, 2=Pfosten links im Feld, 3=Diagonale, 4=Untergurt, 5=letzter Pfosten des Fachwerks), die Textausdrücke dienen der Veranschaulichung und Kontrolle. So entsteht ein Umfang von 55000 Fachwerkfeldern, was 230000 Zeilen bzw. Stäben ergibt.
Prinzipiell lässt sich der Datensatz nach dem in Abbildung [3.6] beschriebenen Schema erstellen. In Grasshopper wird der Umfang des Trainingsatzes festgelegt und eine Prozess- und Optimierungsschleife gestartet. Für die Querschnitte des Fachwerks werden die Schnittgrößen pro Inputdatenvariation durch eine Instanz wie bspw. Dlubal RFEM berechnet und durch eine weitere Komponente wie bspw. ein Excelarbeitsblatt oder einer Python-Komponente die Tragfähigkeit nachgewiesen. Scheitert dieser Nachweis, wird der Querschnitt angepasst und die Berechnung mit unveränderten Inputdaten wiederholt

bis der Querschnitt nachgewiesen ist. Anschließend wird das Ergebnis in einer Textdatei oder einer Exceltabelle abgespeichert und neue Inputdaten für die Berechnung gewählt. Diese Schleife ist in der Erstellung und während der Berechnung sehr zeitaufwändig, weshalb an dieser Stelle dankend an die Arbeit von Paul Stosik verwiesen wird, der in ”Parametrische Modellierung im Holzbau mit Fokus auf integrierter Anschlussdetaillierung unter Verwendung Computergestützter Simulationsverfahren” das Vorgehen hierfür genauer beschreibt.
Möchte man sich den Datensatz wie in Kapitel [3.2] beschrieben über Python generieren, werden Programme für Statik und Nachweise auf Zug und Druck erforderlich. Die Schnittgrößen lassen sich über Gleichgewichtsbedingungen am Ritterschnitt in jedem Feld des Fachwerks ermitteln. Für Nachweise auf Zug kann nach der erforderlichen Querschnittshöhe aufgelöst werden, bei Drucknachweisen mit Knickversagen wird die Querschnittshöhe jedes Stabes iterativ erhöht bis die Stabausnutzung nichtmehr überschritten wird. Die Querschnittshöhen werden auf 10mm gerundet angegeben und in eine Text-Datei zeilenweise eingetragen. Aufgrund der einfachen Skripterstellung für Statik- sowie Nachweisrechnung und geringerer PC-Hardware Anforderung an die Rechenleistung, wurde diese Methode verwendet.
Für ein besseres KI-Training wird aus bereits genannten Gründen, die Zeilenanordnung dieser Datei durch ein weiteres Skript neu sortiert und eine zusätzliche Textdatei abgelegt. Diese Datei kann für das KI-Training in Grasshopper über eine GHPython Komponente eingelesen werden. Es zeigt sich welcher Aufwand eine Datensatzerstellung mit sich bringt und wie Eingangsdaten klar strukturiert werden können. Das entsprechende Skript zur Erstellung des Datensatzes und dem Training weiterer KI ist angehängt (siehe Anhang [C01]).

4.2.3 KI-Training

Vor Trainingsbeginn wird der Datensatz eingelesen. Eine GH-Python-Komponente erhält den Dateipfad zum Datensatz und liest von dieser 80% der Zeilen für das Training ein. Für die Tensorerstellung ist es erforderlich die Daten in der richtigen Reihenfolge und bereits in Stapeln angeordnet zu haben. In der GH-Python Komponente könnte so der Inhalt jeder Zeile an eine Liste angehängt werden und diese zu einem Tensor erstellt werden. Für die Übersichtlichkeit wurde hier jedoch der Inhalt jeder Zeile auf verschiedene Listen, entsprechend der Spaltennamen in Tabelle [4.4], aufgeteilt und diese in gegrafteter Form über eine *Merge*-Komponente zusammengeführt. Für die Erstellung dieser Datenstruktur empfiehlt sich die Verwendung eines sortierten Datensatzes, um die Eingabe der

Daten in die *Construct Tensor*-Komponente besser kontrollieren zu können. In Abbildung [4.7] lässt sich die Datenstruktur für die Tensorerstellung erkennen. Die Anzahl wie viele Einträge in einen Stapel (Batch) zusammengefasst werden, also was alles zusammen gehört, wird über die Angabe der Form (Shape) gesteuert. Für die Inputdaten enspricht dies einer Shape=5 und für die Outputdaten einer Shape=1.

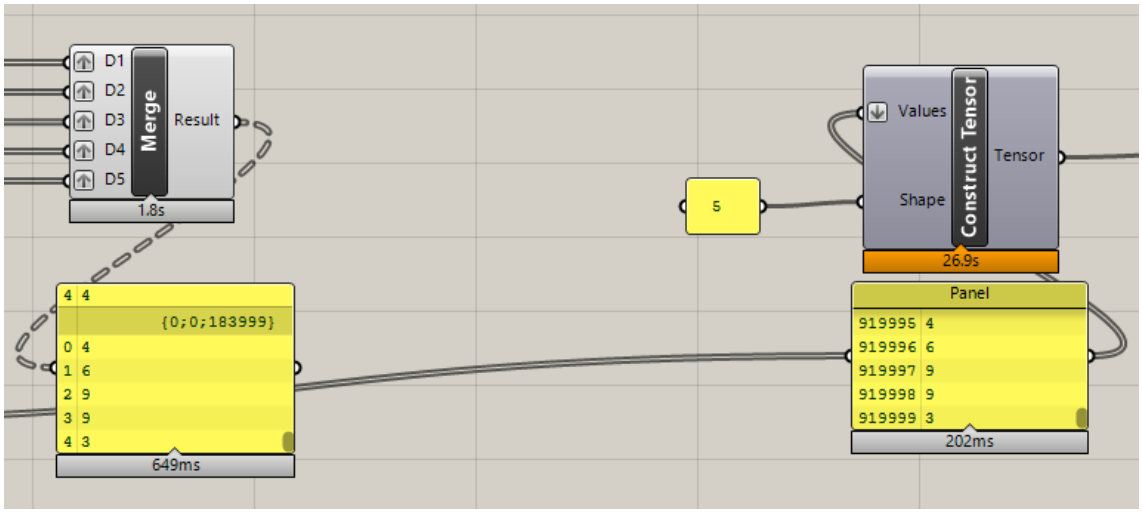


Abbildung 4.7: Inputdaten - links: Datensatz ”gegraftet”, rechts: Datensatz ”geflattet”

Quelle: eigene Darstellung

Die erstellten Tensoren entsprechen der in Abbildung [4.8] ersichtlichen Form.

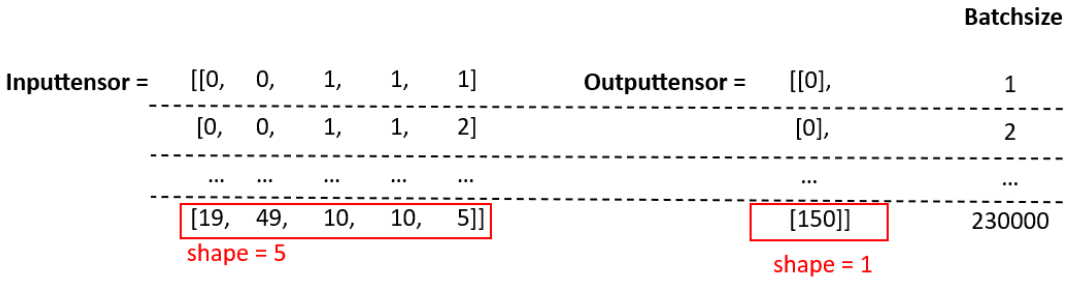


Abbildung 4.8: Tensorvisualisierung Fachwerkträger

Quelle: eigene Darstellung

Abbildung [4.8] dient der Veranschaulichung der Tensorstruktur. Auf das konkrete Beispiel bezogen, ist die Darstellung des gesamten Datensatzes in einem Tensor (bis Batch 230000) falsch, da für den Trainingssatz die Daten nur bis Zeile/Batch 0.8*230000=184000 (80%) ausgelesen werden. Der Testsatz bezieht sich auf die Zeilen [184001; 230000] und umfasst damit die letzten 20%.

Nach Kontrolle des Datensatzes auf seine Richtigkeit, sollte über die GH-Python Komponente der Dateipfad auf die zufällig angeordnete Textdatei gewechselt werden. Der Rest bleibt unverändert.

Input- und Outputtensor werden mit dem Regressionstrainer verbunden und nach Erstellung des Neuronalen Netzwerks folgt die Festlegung der erforderlichen Trainingseinstellungen.

Aufbau NN	**Layer**	Dense/Input	Dense	Dense	Dense	Dense	Dense	Dense/Output
	Neuronen	shape=5	64	64	64	64	64	1
	Aktivierungs-funktion		ReLU	ReLU	ReLU	ReLU	ReLU	
Batchsize		256	**Epochen**		10-100	**Validation Split**		80/20 %
Optimierungsfunktion		Adam	**Lernrate**		0.001	**Verlustfunktion**		Mae

Tabelle 4.5: Aufbau NN + Trainigseinstellungen Fachwerkträger

Die Struktur ähnelt der in Tabelle [4.3] dargestellten für den Einfeldträger mit Ausnahme der Ergänzung einer weiteren Schicht mit 64 Neuronen und ReLU-Aktivierung. Die Funktionsweise der Schichten oder des Trainings bleibt gleich und kann in Kapitel [4.1.3] nachgelesen werden.

Abhängig von PC-Hardware und Rechenleistung kann das iterativ erfolgende KI-Training bei einem Datensatz dieser Größe viel Zeit in Anspruch nehmen. Grasshopper verlangsamt den Trainingsprozess zusätzlich. Aus diesen Gründen bietet es sich an, bei einem klar strukturierten, numerischen Datensatz, wie es hier der Fall ist, externe Instanzen für das KI-Training zu verwenden. Im Umfang dieser Arbeit wurde der Web-Service *Colaboratory* von Google verwendet, was als Entwicklungsumgebung für Python genutzt werden kann und für die Skriptausführung kostenfreie Hardware über eine Remoteverbindung zur Verfügung stellt.
In dieser Umgebung lässt sich Tensorflow importieren und der Datensatz einlesen. So wird ein KI-Training mit den selben Einstellungen wie in Grasshopper ermöglicht, mit dem Vorteil der Zeitersparnis. In Grasshopper wird das optimierte Trainings-Setup übernommen und sollte durch dieses, mit einer entsprechenden Abweichung aufgrund der zufallsgenerierten Startwerte, nach bereits einem Training zu ähnlichen Ergebnissen führen. Das Vorgehen entspricht dem in Kapitel [3.2] erklärten Workflow, dient für diese Arbeit jedoch lediglich als Hilfestellung. Das Skript für die KI ist unter [C03] angehängt.

4.2.4 Modelltest und Auswertung

Das KI-Training wird über den Mae ausgewertet. Der Graph dieser Funktion des SL-Agents sinkt in den ersten 10 Epochen bereits stark und verbessert dieses Niveau bei

Erhöhung der Epochen kaum. Es sind Ausreißer des MAE zu erkennen, an denen ein Abbruch des KI-Trainings mit Speichern der Netzwerkparameter vermieden werden sollte, da die KI sonst weniger genaue Ergebnisse liefert. Dieser Sachverhalt ist in Abbildung [4.9] vergleichend dargestellt.

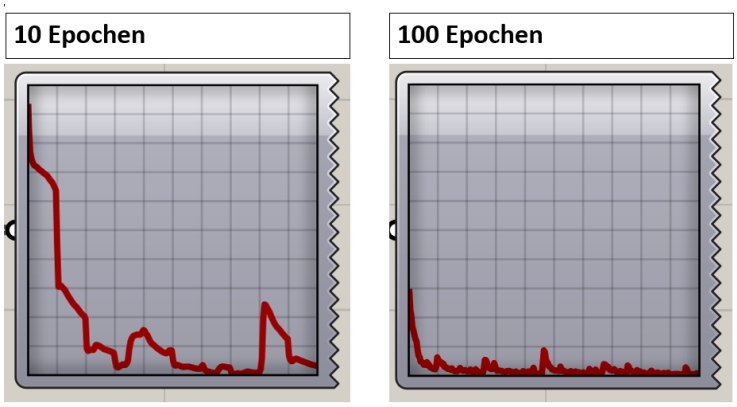


Abbildung 4.9: Vergleich MAE

Quelle: eigene Darstellung

Um Aussagen über den Erfolg des Trainings machen zu können, bedarf es weiterer Vergleiche. Dazu wird die *SL Predict Data*-Komponente verwendet. Diese erhält als Input den *Agent* (Ausgabe der *SL Agent*-Komponente), der die Struktur des Neuronalen Netzwerks beinhaltet, ”x”-Werte, die die Testdaten in einem Tensor in der gleichen Form wie beim Training beinhalten, und einen Dateipfad, der auf eine Datei mit gespeicherten Gewichten und Bias des Neuronalen Netzwerks verweist. Ausgegeben werden mit ”Val” (kurz für engl.: values) die Ergebnisse der Vorhersage (engl.: predictions). Für eine Ergebnisauswertung liegt es bei den Anwendenden, die Testvorhersagen durch Funktionen und Berechnungen für die Visualisierung und Trainingsbewertung vorzubereiten.
Es wird im Folgenden zur Veranschaulichung zwischen einem Training mit 10 und 100 Epochen differenziert. Die Tabelle in Abbildung [4.10] gibt Auskunft über den MAE bei Trainingsende (entspricht Abb. [4.9]), dem Verlauf der Ergebnisdifferenz des Testsatzes und einen MAE auf diese Werte. Der ähnliche Verlauf des Graphen deutet auf ein funktionierendes KI-Training beider Varianten hin, ohne ein Auswendiglernen der Daten oder ein nicht-erfassen des Zusammenhangs. Die Zahlenwerte zu dieser Darstellung ergeben im Vergleich ein besseres Trainingsergebnis bei 10 Epochen. Grund dafür könnte ein ”*Overfitting*” der KI an die Trainingsdaten sein. Das passiert bei einer langer Trainingsdauer (hier: 100 Epochen und mehr), bei der ein KI-Modell zu stark auf die Trainingsdaten angepasst wird und nicht mehr perfekt in der Lage ist, allgemeine Muster zu erkennen und auf neue Daten anzuwenden. Dabei erlernt die KI zu spezifische und detaillierte Muster in den Trainingsdaten, die in anderen Daten jedoch nicht vorhanden sind. Neben diesem

Auftreten kann es viele weitere Gründe für mangelnde Genauigkeit beim KI-Training geben, die für jedes Modell unterschiedlich ausfallen und überprüft werden sollten.
Für dieses Beispiel liefern beide Trainingsvarianten akzeptable Ergebnisse. In der letzten Zeile von Abbildung [4.10] wird ein Ausschnitt der Ergebnisse für ein Fachwerk mit 12 Metern Länge, 5 Feldern und 12 kN/m Belastung dargestellt und verglichen. Die erforderlichen Querschnittshöhen werden in Millimetern angeben, eine Fehlvorhersage von unter 10mm entspricht somit einem Fehler von keinem ganzen Zentimeter. Für eine Bemessung in diesen Dimensionen resultiert bei einer derartigen Abweichung keine dramatische Fehlabschätzung des Querschnitts und das Ziel der KI-Erstellung, der Angabe eines Anhaltspunkts für die Stabdimensionen, wird definitiv erreicht, wenn auch diese KI keinesfalls eine richtige Bemessung ersetzen kann.

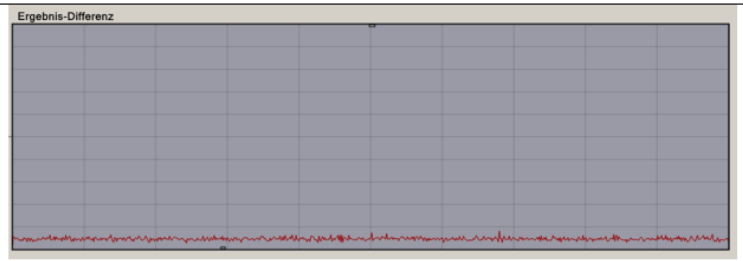
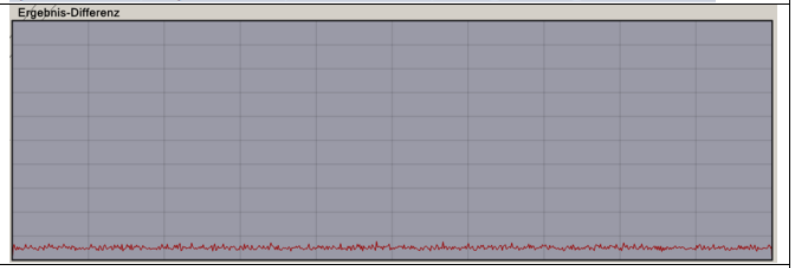
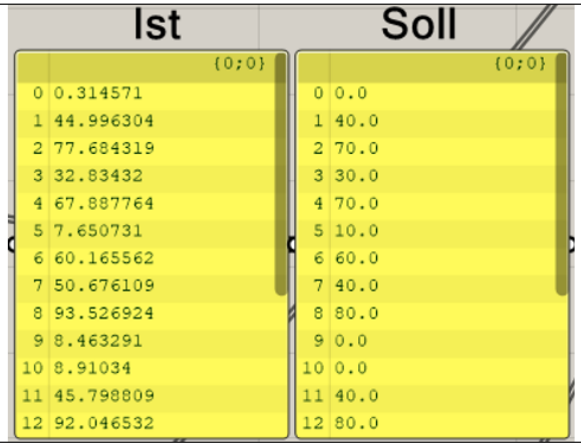
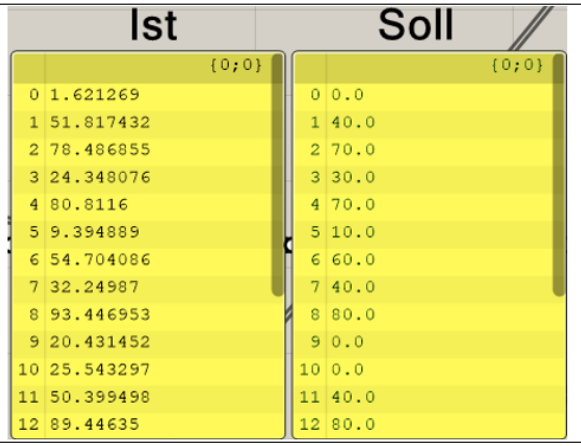
<table>
<tr><th></th><th>10 Epochen</th><th>100 Epochen</th></tr>
<tr><td>MAE bei Trainingsende</td><td>Epoch: 010/010, Step: 0715/0719, loss: 208.566400, mean_absolute_error: 8.713029
Epoch: 010/010, Step: 0716/0719, loss: 208.387300, mean_absolute_error: 8.709749
Epoch: 010/010, Step: 0717/0719, loss: 208.219400, mean_absolute_error: 8.706442
Epoch: 010/010, Step: 0718/0719, loss: 208.142600, mean_absolute_error: 8.704906
Epoch: 010/010, Step: 0719/0719, loss: 208.000000, mean_absolute_error: 8.702026</td><td>Epoch: 100/100, Step: 0715/0719, loss: 232.094100, mean_absolute_error: 9.619114
Epoch: 100/100, Step: 0716/0719, loss: 231.985800, mean_absolute_error: 9.616324
Epoch: 100/100, Step: 0717/0719, loss: 231.977300, mean_absolute_error: 9.615818
Epoch: 100/100, Step: 0718/0719, loss: 231.849600, mean_absolute_error: 9.613549
Epoch: 100/100, Step: 0719/0719, loss: 231.780000, mean_absolute_error: 9.612161</td></tr>
<tr><td>Ergebnisdifferenz Testsatz</td><td> 
</td><td> 
</td></tr>
<tr><td>MAE ges. Testsatz</td><td>6.549</td><td>9.176</td></tr>
<tr><td>Ergebnisvergleich</td><td> 
</td><td> 
</td></tr>
</table>

Abbildung 4.10: Vergleich MAE

Quelle: eigene Darstellung

4.2.5 KI-Anwendung

Für eine Integration der KI in den Planungsprozess benötigt es die *SL Predict Data*-Komponente mit Informationen über den Netzwerkaufbau und den Dateipfad zu den Werten des Neuronalen Netzwerks. Damit das Neuronale Netzwerk nicht bei jeder Anwendung mit jeder Information neu aufgebaut werden muss, kann die Auskunft darüber in einem ”Pug” gespeichert werden. Diese *Pug*-Komponente wird mit dem Output des SL-Trainers (Agent) verbunden und beinhaltet dadurch diese Werte, die an die *SL Predict Data*-Komponente weitergegeben werden. Die gewünschte Fachwerkstruktur wird in Grasshopper modelliert und in Rhino abgebildet, parallel dazu werden die Eingangsvariablen Last, Länge und Anzahl der Felder durch eine GHPython Komponente in die erforderliche Abbildung in mehreren Zeilen (siehe Tab.: 4.4) transformiert und danach durch ”Construct Tensor” in einem Tensor der Form 5 abgebildet. Parallel zur Formfindung werden so die erforderlichen Querschnittswerte ausgegeben.
Die drei Eingangsvariablen können frei gewählt werden. Beispiele zur Verdeutlichung dieser Anwendung sind ergänzend dargestellt, die praktische Sinnhaftigkeit sei außer Acht gelassen.

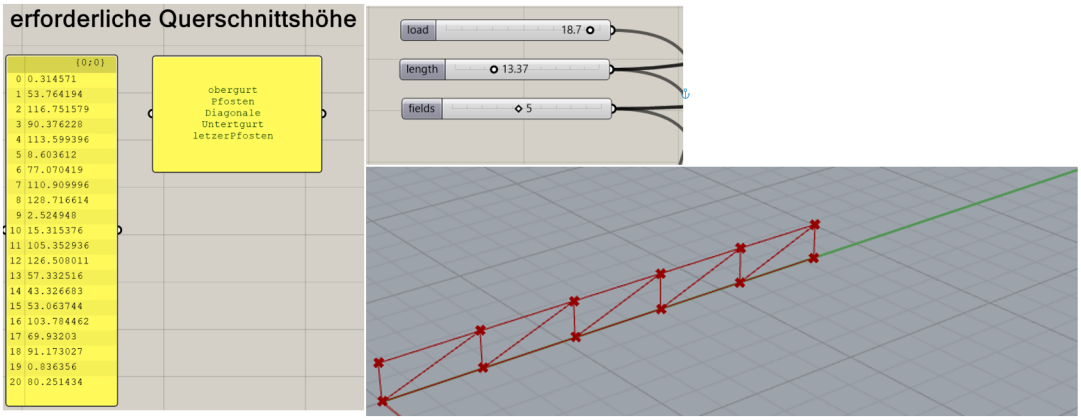


Abbildung 4.11: KI Anwendung - Beispiel 1

Quelle: eigene Darstellung

Die Ergebnisausgabe in Listenform wird bei vielen Einträgen unübersichtlich. Um dies vorzubeugen kann die Darstellung der Ergebnisse bspw. nach Bauteilen sortiert werden oder Querschnittsabmessungen vereinheitlicht werden, typisch wäre eine einheitliche Größe für Ober- und Untergurt des Trägers. Am besten lässt sich dies durch eine GHPython Komponente direkt nach der Ergebnisausgabe umsetzen, letzten Endes unterliegt jedoch auch diese Darstellung der Größe des Systems und den Ansprüchen der Anwendenden. Die Funktionalität der KI bleibt dabei immer die Gleiche.

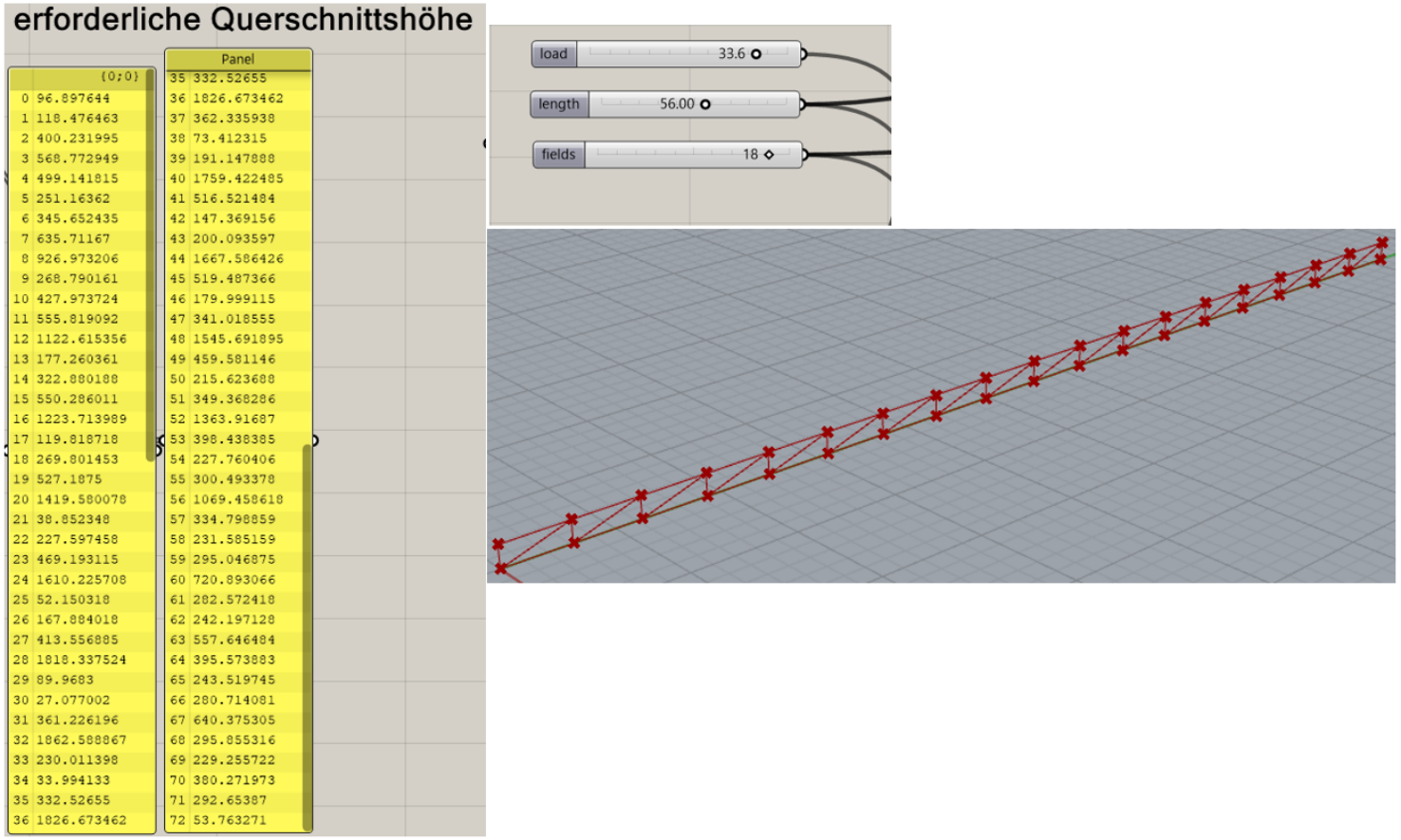


Abbildung 4.12: KI Anwendung - Beispiel 2

Quelle: eigene Darstellung

5 Diskussion

Die Erstellung eines KI-Modells ist abgeschlossen. Bisherige Ergebnisse aus vorangestellten Kapiteln werden zusammengefasst dargelegt und ihre Brauchbarkeit im Bezug auf die Zielsetzung dieser Arbeit erörtert. Es folgt eine Diskussion des Geleisteten mit beigefügtem Ausblick.

5.1 Ergebniszusammenfassung

Das Erarbeiten der Leitfrage dieser Arbeit erforderte eine Darlegung der grundlegenden Theorie über KI und das Verständnis über die Funktionsweise dieser Modelle. Aufbauend darauf konnten Übersichten zu KI-Frameworks, relevanter Software und Grasshopper Plugins erstellt werden, die zum Zeitpunkt der Veröffentlichung als Basis der Umsetzung für dieses und weitere Projekte genutzt werden können. Von diesen erwiesen sich nach einer Auswertung Pug und Lunchbox in Kombination mit weiteren Programmen für Statik und Bemessung als aktuell und brauchbar. Die gewählte Aufgabenstellung eines Einfeldträgers konnte auf praktische Art zeigen, welche Unterschiede zwischen Maschinellem Lernen und Tiefem Lernen bestehen und wie sich diese Funktionsweisen auf die Ergebnisse auswirken. Abhängig von der Anwendung bietet jedes Plugin seine Vorteile. Für die anschließend festgelegte Bemessung eines Fachwerkträgers konnten mit Pug bessere Ergebnisse erwartet werden. Die Bearbeitung dieser Aufgabenstellung zeigte welcher Aufwand eine Datensatzerstellung macht und wie relevant ein guter Datensatz für die KI ist. Geringere Rechenleistungen in Grasshopper führten zur Verwendung von alternativen KI-Trainingsumgebungen wie Google Colaboratory und dem Fazit der schnelleren Verwendbarkeit dieser, was für das Training von Grasshopper KI positiv genutzt werden kann. Die Umsetzung wurde mit Anwendung neuer Problemstellungen für die KI abgeschlossen.

5.2 Kritik

Die Anwendbarkeit von KI in Grasshopper wurde an einem einfachen Beispiel bewiesen, beschränkt sich dabei jedoch nur auf diesen spezifischen Fall und kann so keinen umfangreichen Ausblick auf alle Möglichkeiten und Anwendungen einer KI-Implementierung in Grasshopper liefern. Die Erstellung einer KI unterliegt vielen Faktoren, für die keine ganzheitlichen Lösungen oder eine einzige, perfekte Lösung formuliert werden können. So kann auch in dieser Arbeit nur auf die verwendeten Elemente, beginnend bei

Grasshopper-Komponenten, dem Umfang der Plugins und bis zu KI-Funktionen reichend, eingegangen werden. Die Darlegung dieser KI-Implementierung soll anregen, über weitere Möglichkeiten nachzudenken, fordert an dieser Stelle jedoch auf, die Sinnhaftigkeit eines jeden KI-Modells vor der eigentlichen Erstellung dessen zu hinterfragen. Zu Beginn dieser Arbeit wurde erklärt, dass sich eine KI den richtigen Ergebnissen nur annähert und die aufwendige Erstellung eines brauchbaren Modells durch eine Programmierung von Anweisungen und Rechnungen ersetzt werden sollte, wenn dies einfacher möglich ist. Die Anwendung von KI zur Berechnung eines Fachwerkträgers für den eine bestimmte Lösung mechanisch bestimmbar ist, ist aus praktischer Sichtweise weder nötig noch sinnvoll und verfehlt das Potenzial von KI. Für die Datensatzerstellung der Fachwerk-KI wurde ein Programm geschrieben welches Fachwerke berechnen und dimensionieren kann. Tragwerksplanende sind an eben dieser Lösung interessiert und würden sich bei Verwendung des Rechenprogramms nicht nur den Aufwand der KI-Erstellung und -Verwendung sparen, sondern hätten zusätzlich eine treffend genaue Angabe für die Querschnittsabmessungen. Die Entscheidung über das bevorzugte Vorgehen muss je nach Aufgabenstellung gut gewählt werden und kann mit und ohne KI sehr kompliziert werden. Für diese Ausarbeitung spielte die Bewertung dieses Aspektes keine Rolle, es galt die Anwendbarkeit von KI in Grasshopper darzustellen, auch wenn der praktische Nutzen des erstellten Modells nicht gegeben ist. Für kompliziertere Problemstellungen, die nicht über einfache Pythonprogramme gelöst werden können und deshalb eine Berechnung durch ein Statikprogramm erfordern oder sogar mechanisch nicht genau bestimmbar sind, kann die KI-Erstellung jedoch sehr hilfreich sein. Für solche Probleme liefert diese Arbeit einen Einsteig in die KI-Anwendung in Grasshopper und präsentiert erste Probleme und Herangehensweisen.

5.3 Ausblick

Auch wenn Künstliche Intelligenz in anderen Bereichen der Baubranche bereits umfangreicher eingesetzt wird, erweist sich ihr Nutzen auch in der Tragwerksplanung als bedeutend. Unter Beachtung der Sinnhaftigkeit einer KI Verwendung lässt sich das Potenzial auf viele planerische Aufgabenteile übertragen, beginnend bei einer Bemessung mit Modell- und KI-Entwicklung in Grasshopper. Was in dieser Arbeit exemplarisch an einem einfachen System gezeigt wurde, kann in weiteren Forschungen auf komplexere Aufgabenstellungen angewendet und überprüft werden. Die Möglichkeiten der Plugins können weiter untersucht werden und es könnte ein Projekt zur Betreuung eines KI-Plugins initi-

iert werden, welches Entwicklern Unterstützung bietet und das Plugin auf dem neuesten Stand eines KI-Frameworks hält. Die vorliegende Arbeit demonstriert den vergleichsweise einfachen Einstieg in die KI-Programmierung in Grasshopper und soll so Tragwerksplanende und Entwickler ermutigen, die Erkenntnisse dieser Arbeit zu nutzen und auf neue Problemstellungen zu übertragen. Wie jeder andere Beruf wird auch die Ingenieursarbeit durch den technologischen Fortschritt geprägt und passt sich diesem an um davon zu profitieren. Was ursprünglich bei der Berechnung mit einem Rechenschieber begann, hat sich bis zur Modellierung und Auswertung komplexer FE-Modelle entwickelt. Die Entwicklung und das Potenzial von Künstlicher Intelligenz lassen erahnen, dass schon jetzt und in Zukunft, dieser Fortschritt eine Vielzahl neuer, spezifischer KI-Programme mit sich bringen wird, die Aufgaben abnehmen und die Arbeit der Tragwerksplanenden effizient unterstützen können.

6 Fazit

Diese Bachelorthesis klärte über die Anwendbarkeit von KI-Modellen in Grasshopper anhand konkreter Beispiele auf und ging dazu auf die Theorie hinter der Arbeit mit Künstlicher Intelligenz, zuerst im Allgemeinen und anschließend in Grasshopper ein, was eine folgende, aufgabenbezogene Übersicht relevanter Programme und nützlicher Software beinhaltete. Damit wurde nicht nur die Anwendung in Grasshopper gezeigt, sondern auch die leichte Zugänglichkeit einer KI beim visuellen Programmieren.
Auftretende Einschränkungen durch die Verwendung von Grasshopper ergänzt mit Abhängigkeiten von Plugins und die Herausforderungen der Abschätzung einer sinnvollen KI-Anwendungen bleiben aufgabenabhängig zu beurteilen. Probleme mit der Rechenleistung von Grasshopper und vorhandener Hardware verlangsamen die KI-Entwicklung gegenüber einer herkömmlichen Entwicklung ohne visuelle Programmierung. Es wurde erkenntlich, dass sich KI-Plugins für Grasshopper den eigentlichen KI-Frameworks bedienen, bei der Implementierung allerdings einige Funktionen des Frameworks nicht übernommen werden, was zu Einschränkungen bei der KI-Ausbildung führen kann.
Unabhängig von diesen Schwierigkeiten wurde ein brauchbares KI-Modell in Grasshopper erstellt und exemplarisch angewandt, was auf andere Problemstellungen übertragen werden kann, da bereits dieses demonstriert, wie KI effektiv zur Lösung spezifischer Problemstellungen der Tragwerksplanung und weiterer Ingenieursaufgaben eingesetzt werden kann.
Die Anwendbarkeit von KI-Modellen für die parametrisierte Strukturmodellierung in Grasshopper und im konstruktiven Ingenieurholzbau wurde bewiesen und lässt sich mit Blick auf künftige Entwicklungen, als nutzbaren Beitrag für die Effizienz und Eröffnung neuer Möglichkeiten in der Branche, beurteilen.

Anhang

A01 - Programmcode: Stützenbemessung durch KI

siehe Kap. [2]

Listing 1: Python Code - KI Querschnittsermittlung

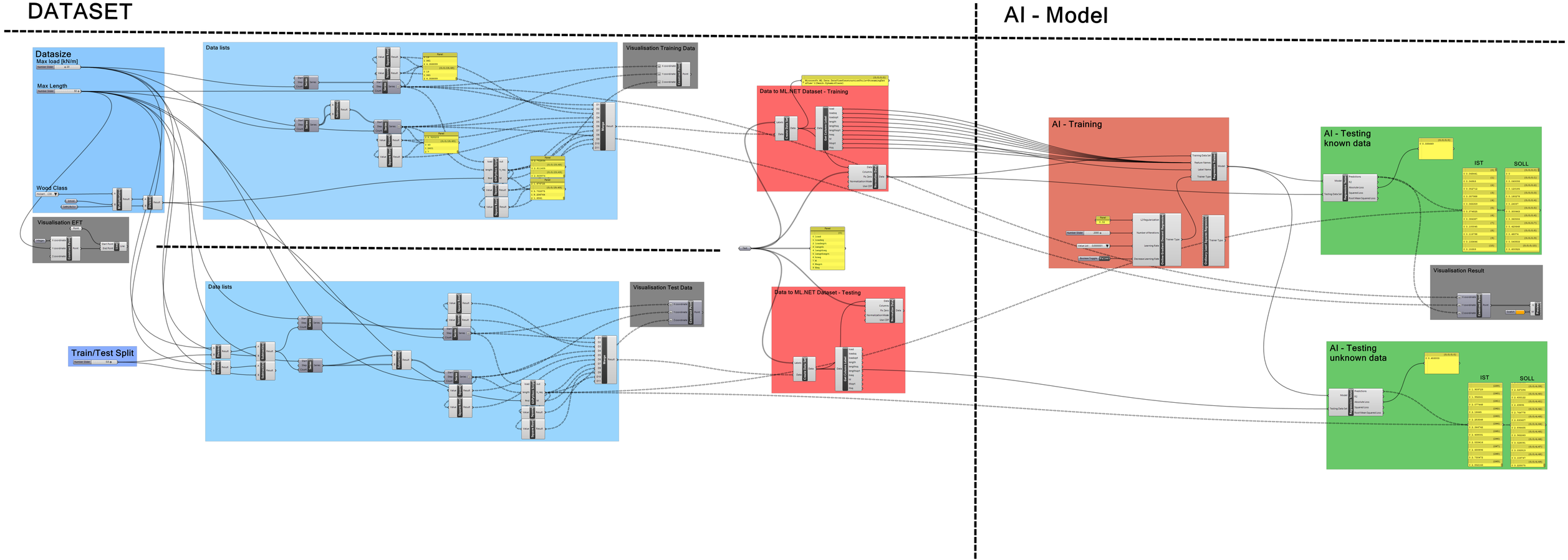
```
#Klasse erstellen
class DatacreatorT1():

  #Initialisieren
  def __init__(self, force, fck, step):
    self.force = force
    self.fck = fck
    self.step = step

  def calculate_area(self):
    a_dict ={}
    for force in range(0, self.force, self.step):
      a = force * 10 / self.fck
      a_dict[force] = a

    return(a_dict)


max_force = 100
step = 1          #in welchen Schritten soll die Belastung erhoeht werden
Datensatz = DatacreatorT1(max_force, 24, step).calculate_area() #24 = GL24h festigkeitswert

#Datensatz

#weigth = 0
#bias = 0.416

import torch
from torch import nn
import matplotlib.pyplot as plt

#Datenauslesen
#zuerst features
X = torch.arange(0, max_force, step).unsqueeze(1)
#was soll rauskommen
y = torch.tensor([Datensatz[i] for i in range(0, max_force, step)]).reshape(-1, 1)

#Create train/test_split, 80/20
train_split = int(0.8*len(X)) #80%
X_train, y_train = X[:train_split], y[:train_split] #:train_split nimmt die ersten 80%
X_test, y_test = X[train_split:], y[train_split:]   #train_split: nimmt was nach den 80% noch uebrig ist (20%)

torch.manual_seed(42)

#Class fuer neuronales Netzwerk
##Versuch es mit nur einem linearen Layer
class LinearRegressionModel(nn.Module):
  def __init__(self):
    super(LinearRegressionModel, self).__init__()
```

```
50      self.layer_1 = nn.Linear(in_features=1, out_features=1, bias=
            True, dtype=torch.float32)
51      #self.layer_2 = nn.Linear(in_features=5, out_features=1, dtype=
            torch.float32)
52    def forward(self, x):
53      #return self.layer_2(self.layer_1(x))
54      return self.layer_1(x)
55
56  #Test Modell erstellen
57  test_model = LinearRegressionModel()
58
59  #Kontrolle obs geht
60  '''with torch.inference_mode():
61    a = test_model(X_test.to(torch.float32))
62  a'''
63
64  #loss function
65  loss_fn = nn.L1Loss()
66  #loss_fn = nn.MSELoss()
67
68  #optimizer Function
69  optimizer = torch.optim.SGD(test_model.parameters(),
70                              lr=0.00001)
71
72  #Loop vorbereiten
73  #wie oft iterieren
74  epochs = 1000
75  #Zaehlen in Liste
76  train_loss_values = []
77  test_loss_values = []
78  epoch_count = []
79
80  #der eigentliche Loop
81  for epoch in range(epochs):
82    ###Training
83    test_model.train()
84    y_pred = test_model(X_train.type(torch.float))
85    #print(y_pred.shape, y_train.shape)
86    loss = loss_fn(y_pred, y_train)
87    optimizer.zero_grad()
88    loss.backward()
89    optimizer.step()
90
91    ###Testing
92    test_model.eval()
93    with torch.inference_mode():
94      test_pred = test_model(X_test.type(torch.float))
95      test_loss = loss_fn(test_pred, y_test.type(torch.float))
96      #test_pred = test_pred.cpu
97    if epoch % 10 == 0:
98      epoch_count.append(epoch)
99      train_loss_values.append(loss.detach().numpy())
100     test_loss_values.append(test_loss.detach().numpy())
101     print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss
            : {test_loss} ")
102
103 print(test_model.state_dict())
104 print(list(test_model.parameters()))
105
```

```
106 with torch.inference_mode():
107   #f_input = torch.tensor(input("Welche Last wirkt auf Stuetze?"))
108   f_input = torch.tensor([5])
109   result = test_model(f_input.type(torch.float32))
110   result_print = round(float(result[0]), 2)
111   print(f"Erforderlicher Stuetzenquerschnitt: {result_print} cm2")
112
113 #Erforderlicher Stuetzenquerschnitt: 2.83 cm2
114 #Kontrolle:  5 * 10 / 24 = 2,083 cm2
```

B01 - Grasshopper: EFT-Bemessung mittels Lunchbox

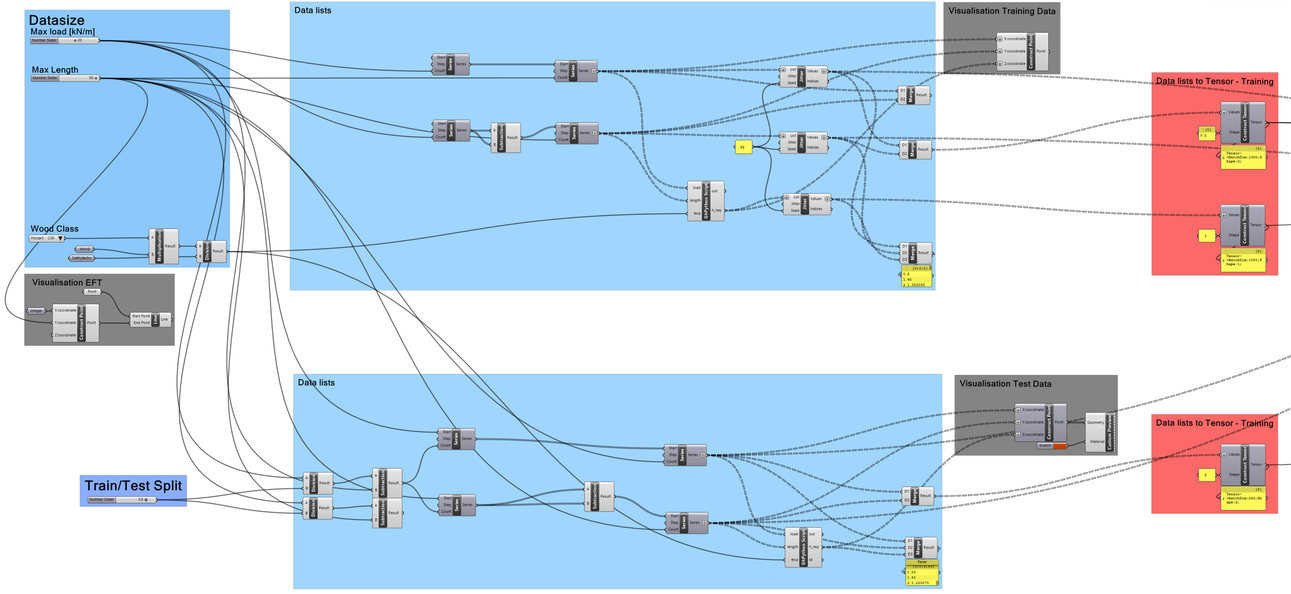
siehe Kap. [4]


DATASET
AI - Model
Datasize
Max load [kN/m]
Max Length
Wood Class
Visualisation EFT
Data lists
Visualisation Training Data
Train/Test Split
Data lists
Visualisation Test Data
Data to ML.NET Dataset - Training
Data to ML.NET Dataset - Testing
AI - Training
AI - Testing known data
AI - Testing unknown data
IST
SOLL
Visualisation Result

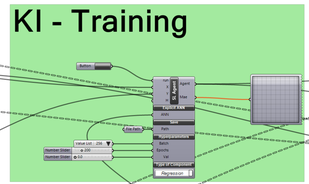
B02 - Grasshopper: EFT-Bemessung mittels Pug

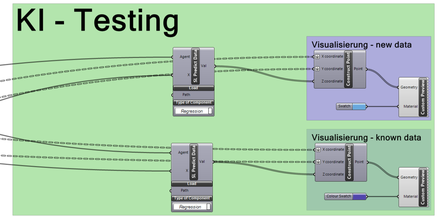
siehe Kap. [4]

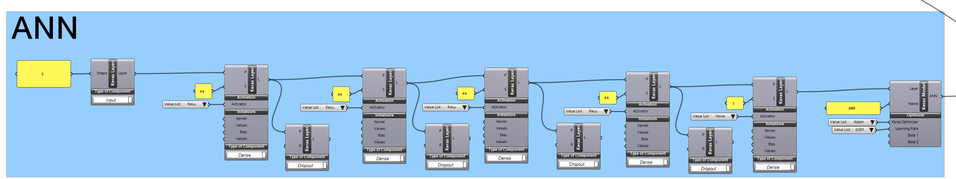
DATASET



AI - Model







C01 - Programmcode: Datensatzerstellung Fachwerkträger

siehe Kap. [4.2.2]

Listing 2: TrussCalc.py

```
###Knickwert rechner
import math
pi = math.pi

def Knickbeiwert(width, h, height, lf, E005, fck):
    if width>h:
        width, h = h, width
    beta_c = 0.1
    height = height * 1000
    lf = lf * 1000
    lcr = (height**2 + lf**2)**0.5
    if lcr > 3000:  #Ab 3m wird stab gehalten, sonst werden Ergebnisse unplausibel, lcr wird durch Satz des Pythagoras ueberschaetzt
        lcr = 3000
    I = (h*width**3)/12
    A_F = width*h
    i = (I/A_F)**0.5
    lamda_weak = lcr/i
    lamda_rel = lamda_weak/pi * (fck/E005)**0.5
    k_y = 0.5 * (1+beta_c*(lamda_rel-0.3)+lamda_rel**2)
    k_cy = (k_y+((k_y**2)-(lamda_rel**2))**0.5)**(-1)
    return(k_cy)


###Beam-by-Beam-Analysis#########
#Trusssolver for field in Truss
def TrussSolverSHbbb(load, length, fields, height, fcd, fck, ftd, field, width, E005):
    #Systemberechnungen
    A = load*length/2 #reaction force
    lf = length/fields #length of field
    F = load * length / (fields + 1) #Load on each node
    Dv = height / (((lf**2)+(height**2))**0.5) #Diagonalforce transformed in partional vertical Force
    Dh = lf / (((lf**2)+(height**2))**0.5)   #Diagonalforce transformed in Partional horizontal Force
    Dsv = Dv ** (-1)    #partional vertical Force transformed to full Diagonalforce
    #Dsh = Dh ** (-1)    #partional horizontal Force transformed to full Diagonalforce

    k_cy_upper, k_cy_post, k_cy_diag, k_cy_lower = 0,0,0,0

    #Sectionheight Beams
    MS_upper0 = 0
    # print("nf1: " + str(nf1))
    for nf2 in range(0, field):  # loop for sumM with F on every joint
        MS_upper1 = F * (nf2) * lf
        MS_upper0 = MS_upper0 + MS_upper1
        # print("nf2: "+str(nf2) + "; S_upper: "+str(S_upper1))
```

```
45      f_upper = (-A*(field-1)*lf + MS_upper0) / height
46      if f_upper < 0:
47          f_upper_pos = ((f_upper)**2)**0.5
48          h = 10
49          eta = 1.1
50          while eta > 1:
51              k_cy_upper = Knickbeiwert(width, h, height, lf, E005,
                    fck)
52              A_QS = h * width
53              eta = f_upper_pos*1000/(A_QS*k_cy_upper*fcd)
54              h = h+10
55          SH_upper = h-10
56      else:
57          #SH_upper = f_upper / (ftd * width) * 1000   - math.ceil(b
                / 10) * 10
58          SH_upper = math.ceil((f_upper / (ftd * width) * 1000) / 10)
                * 10
59
60      #force and SH in diagonals
61      f_diag = (-A + (field)*F) * Dsv
62      if f_diag < 0:
63          f_diag_pos = ((f_diag) ** 2) ** 0.5
64          h = 10
65          eta = 1.1
66          while eta > 1:
67              k_cy_diag = Knickbeiwert(width, h, height, lf, E005,
                    fck)
68              A_QS = h * width
69              eta = f_diag_pos*1000/(A_QS*k_cy_diag*fcd)
70              h = h+10
71          SH_diag = h-10
72      else:
73          #SH_diag = f_diag / (ftd  * width) * 1000   math.ceil(b /
                10) * 10
74          SH_diag = math.ceil((f_diag / (ftd * width) * 1000) / 10)
                *10
75
76      #force and SH in lower beams
77      f_lower = -f_diag * Dh - f_upper
78      if f_lower < 0:
79          f_lower_pos = ((f_lower) ** 2) ** 0.5
80          h = 10
81          eta = 1.1
82          while eta > 1:
83              k_cy_lower = Knickbeiwert(width, h, height, lf, E005,
                    fck)
84              A_QS = h * width
85              eta = f_lower_pos*1000/(A_QS*k_cy_lower*fcd)
86              h = h+10
87          SH_lower = h-10
88      else:
89          #SH_lower = f_lower / (ftd * width) * 1000   math.ceil(b /
                10) * 10
90          SH_lower = math.ceil((f_lower / (ftd * width) * 1000) / 10)
                *10
91
92      #force in Posts
93      if field == 1:
94              f_post = -F
```

```
95      else:
96          f_post = -f_diag * Dv
97      if f_post < 0:
98          f_post_pos = ((f_post) ** 2) ** 0.5
99          h = 10
100         eta = 1.1
101         while eta > 1:
102             k_cy_post = Knickbeiwert(width, h, height, lf, E005,
                    fck)
103             A_QS = h * width
104             eta = f_post_pos*1000/(A_QS)/(k_cy_post*fcd)
105             h = h+10
106         SH_post = h-10
107     else:
108         #SH_post = f_post / (ftd * width) * 1000          math.ceil(
                b / 10) * 10
109         SH_post = math.ceil((f_post / (ftd * width) * 1000) / 10)
                *10
110
111     if field == fields:
112         h = 10
113         eta = 1.1
114         A_pos = ((A) ** 2) ** 0.5
115         while eta > 1:
116             k_cy_lastpost = Knickbeiwert(width, h, height, lf, E005
                    , fck)
117             A_QS = h * width
118             eta = A_pos * 1000 / (A_QS * k_cy_lastpost * fcd)
119             h = h + 10
120         SH_last_post = h - 10
121         #SH_last_post = -A/(fcd * width) * 1000
122     else:
123         SH_last_post = 0
124
125     SH_upper = (SH_upper**2)**0.5
126     SH_diag = (SH_diag**2)**0.5
127     SH_post = (SH_post**2)**0.5
128     SH_lower = (SH_lower**2)**0.5
129     SH_last_post = (SH_last_post**2)**0.5
130
131     print(f'{load} {length} {fields} {field} fupper{f_upper,
            k_cy_upper} fpfosten{f_post, k_cy_post} fdiag {f_diag,
            k_cy_diag} flower{f_lower, k_cy_lower}')
132
133     return (SH_upper, SH_diag, SH_post, SH_lower, SH_last_post)
```

Listing 3: TrussDATA.py

```
1  #Datasetgeneration for Trusssystem
2  #bbb = beam by beam
3
4  from TrussCalc import TrussSolverSHbbb as tsh
5
6  #Clearing all DATA - File has to exist!
7  with open("DATAbbb.txt", "w") as f_empty:
8      f_empty.write("")
9
10 #input values
11 load = 20
12 length = 50
```

```
fields = 10
height = 1
fck = 24
ftk = 16.5
width = 300
E005=9400

gamma = 1.3
kmod = 0.9

#script
fcd = kmod/gamma*fck
ftd = kmod/gamma*ftk

load_list = []
length_list = []
sum_list = []
pos_list = []
field_list = []

#List with all force variations
for i in range(0,load):
    for i_j in range(0,length):
        load_list.append(i)
#print(load_list)

#list with all length variations
for j in range(0,load):
    for j_j in  range(0,length):
        length_list.append(j_j)
#print(length_list)

#lists combined
for load_count in range(0,load):
    for length_count in range(0,length):
        sum_list.append(
            (load_count, length_count)
        )

#print(sum_list)

#getting positions of every beam in the Truss


#return(SH_upper, SH_diag, SH_post, SH_lower, SH_last_post)

#MAIN LOOP
#Systemvariation
for load_count in range(0,load):
    for length_count in range(0,length):
        for field_count_system in range(1,fields+1):
            '''for pos_count in range(0, field_count_system):
                pos_list.append("Obergurt")
                pos_list.append("Pfosten")
                pos_list.append("Diagonale")
                pos_list.append("Untergurt")
                field_list.append(pos_count + 1)
                field_list.append(pos_count + 1)
                field_list.append(pos_count + 1)
```

```
72                  field_list.append(pos_count + 1)
73                  if pos_count == field_count_system - 1:
74                      pos_list.append("Letzer Pfosten")
75                      field_list.append(pos_count + 1)'''
76
77
78 #Calcualtion of SH in every field (Ritterschnitt)
79              for field_count_calc in range(1, field_count_system+1):
80                  SH_upper, SH_diag, SH_post, SH_lower, SH_last_post
                        = tsh(load_count, length_count,
                        field_count_system, height, fcd, fck, ftd,
                        field_count_calc, width, E005)
81
82              #text file creation with data stored:
83              #load, length, fields, field, posname, posnumber, SH
84                  if field_count_calc == field_count_system:
85                      with open("DATAbbb.txt", "a") as f:
86                          f.write(str(load_count) + "|" +
87                                  str(length_count) + "|" +
88                                  str(field_count_system) + "|" +
89                                  str(field_count_calc) + "|" +
90                                  "Obergurt" + "|" +
91                                  str(1) + "|" +
92                                  #str("b/h=")+ str(width) + "/" +
93                                  str(SH_upper) + "|" +
94                                  "\n" +
95                                  str(load_count) + "|" +
96                                  str(length_count) + "|" +
97                                  str(field_count_system) + "|" +
98                                  str(field_count_calc) + "|" +
99                                  "Pfosten" + "|" +
100                                 str(2) + "|" +
101                                 #str("b/h=") + str(width) + "/" +
102                                 str(SH_post) + "|" +
103                                 "\n" +
104                                 str(load_count) + "|" +
105                                 str(length_count) + "|" +
106                                 str(field_count_system) + "|" +
107                                 str(field_count_calc) + "|" +
108                                 "Diagonale" + "|" +
109                                 str(3) + "|" +
110                                 #str("b/h=") + str(width) + "/" +
111                                 str(SH_diag) + "|" +
112                                 "\n" +
113                                 str(load_count) + "|" +
114                                 str(length_count) + "|" +
115                                 str(field_count_system) + "|" +
116                                 str(field_count_calc) + "|" +
117                                 "Untergurt" + "|" +
118                                 str(4) + "|" +
119                                 #str("b/h=") + str(width) + "/" +
120                                 str(SH_lower) + "|" +
121                                 "\n" +
122                                 str(load_count) + "|" +
123                                 str(length_count) + "|" +
124                                 str(field_count_system) + "|" +
125                                 str(field_count_calc) + "|" +
126                                 "letzer Pfosten" + "|" +
127                                 str(5) + "|" +
```

```
                                    #str("b/h=") + str(width) + "/" +
                                    str(SH_last_post) + "|" +
                                    "\n"
                                    )

                else:
                    with open("DATAbbb.txt", "a") as f:
                        f.write(str(load_count) + "|" +
                                    str(length_count) + "|" +
                                    str(field_count_system) + "|" +
                                    str(field_count_calc) + "|" +
                                    "Obergurt" + "|" +
                                    str(1) + "|" +
                                    #str("b/h=") + str(width) + "/" +
                                    str(SH_upper) + "|" +
                                    "\n" +
                                    str(load_count) + "|" +
                                    str(length_count) + "|" +
                                    str(field_count_system) + "|" +
                                    str(field_count_calc) + "|" +
                                    "Pfosten" + "|" +
                                    str(2) + "|" +
                                    #str("b/h=") + str(width) + "/" +
                                    str(SH_post) + "|" +
                                    "\n" +
                                    str(load_count) + "|" +
                                    str(length_count) + "|" +
                                    str(field_count_system) + "|" +
                                    str(field_count_calc) + "|" +
                                    "Diagonale" + "|" +
                                    str(3) + "|" +
                                    #str("b/h=") + str(width) + "/" +
                                    str(SH_diag) + "|" +
                                    "\n" +
                                    str(load_count) + "|" +
                                    str(length_count) + "|" +
                                    str(field_count_system) + "|" +
                                    str(field_count_calc) + "|" +
                                    "Untergurt" + "|" +
                                    str(4) + "|" +
                                    #str("b/h=") + str(width) + "/" +
                                    str(SH_lower) + "|" +
                                    "\n"
                                    )
```

Listing 4: TrussDataShuffle.py

```
#Zeilen von DATASET shufflen

import random

def randomize_lines(filename):
    # Datei lesen
    filename_txt = filename +".txt"
    with open(filename_txt, 'r') as f:
        lines = f.readlines()

    # Zeilen randomisieren
    random.shuffle(lines)

```

```
14      # Textdatei umbenennen
15      filename_shuffled = filename + "_shuffled.txt"
16
17      # Randomisierte Zeilen in die Datei zurueckschreiben
18      with open(filename_shuffled, 'w') as f:
19          f.writelines(lines)
20
21 #Dateiname ohne .txt
22 randomize_lines('DATAbbb')
```

C02 - Programmcode: GHPython Komponenten Fachwerk-KI

siehe Kap. [4.2.3]

Listing 5: Trainingsdaten einlesen

```
import rhinoscriptsyntax as rs

# Pfad zur Datei
filepath = 'D:\\...\\...\\...\\DATAbbb_shuffled.txt'

# Leere Listen fuer die Variablen
load_values = []
length_values = []
field_max_values = []
field_count_values = []
pos_values = []
SH_values = []

#Listensumme
with open(filepath, 'r') as file:
    total_lines = sum(1 for line in file)

# Ersten 80% sind Trainingsdaten
# Berechne, bis zu welcher Zeile die ersten 80% gehen
end_line = int(total_lines * 0.8)

# oeffnen der Datei im Lese-Modus
with open(filepath, 'r') as file:
    # Gehe durch jede Zeile in der Datei
    for line_number, line in enumerate(file):
        # Verarbeite nur die ersten 80% der Zeilen
        if line_number < end_line:
            # Teile die Zeile an den | Zeichen
            parts = line.split('|')

            # Weise die Teile den entsprechenden Variablen zu
            load = int(parts[0])
            length = int(parts[1])
            field_max = int(parts[2])
            field_count = int(parts[3])
            pos = int(parts[5])
            SH = float(parts[6])

            # Fuege die Werte zu den Listen hinzu
            load_values.append(load)
            length_values.append(length)
            field_max_values.append(field_max)
            field_count_values.append(field_count)
            pos_values.append(pos)
            SH_values.append(SH)

# Zuweisung der Ausgangsvariablen
a = load_values
b = length_values
c = field_max_values
d = field_count_values
e = pos_values
```

```
f = SH_values
```

Listing 6: Testdaten einlesen

```
import rhinoscriptsyntax as rs

# Pfad zur Datei
filepath = 'D:\\...\\...\\...\\DATAbbb_shuffled.txt'

# Leere Listen fuer die Variablen
load_values = []
length_values = []
field_max_values = []
field_count_values = []
pos_values = []
SH_values = []

#Listensumme
with open(filepath, 'r') as file:
    total_lines = sum(1 for line in file)

#letzen 20% sind Testdaten
#Berechne, ab welcher Zeile die letzten 20% beginnen
start_line = int(total_lines * 0.8)


# oeffnen der Datei im Lese-Modus
with open(filepath, 'r') as file:
    # Gehe durch jede Zeile in der Datei
    for line_number, line in enumerate(file):
        # Verarbeite nur die letzten 20% der Zeilen
        if line_number >= start_line:
        # Teile die Zeile an den | Zeichen
            parts = line.split('|')

        # Weise die Teile den entsprechenden Variablen zu
            load = int(parts[0])
            length = int(parts[1])
            field_max = int(parts[2])
            field_count = int(parts[3])
            pos = int(parts[5])
            SH = float(parts[6])

        # Fuege die Werte zu den Listen hinzu
            load_values.append(load)
            length_values.append(length)
            field_max_values.append(field_max)
            field_count_values.append(field_count)
            pos_values.append(pos)
            SH_values.append(SH)

# Zuweisung der Ausgangsvariablen
a = load_values
b = length_values
c = field_max_values
d = field_count_values
e = pos_values
f = SH_values
```

Listing 7: Erstellung Inputdaten für KI Anwendung

```
import rhinoscriptsyntax as rs

load = float(load)
length = float(length)
fields = int(fields)


height, fcd, width = 1, 24, 200


###Beam-by-Beam-Analysis#########
#Trusssolver for field in Truss
def TrussSolverSHbbb(load, length, fields, height, fcd, field,
    width):
    #Systemberechnungen
    A = load*length/2 #reaction force
    lf = length/fields #length of field
    F = load * length / (fields + 1) #Load on each node
    Dv = height / (((lf**2)+(height**2))**0.5) #Diagonalforce
       transformed in partional vertical Force
    Dh = lf / (((lf**2)+(height**2))**0.5)   #Diagonalforce
       transformed in Partional horizontal Force
    Dsv = Dv ** (-1)    #partional vertical Force transformed to
       full Diagonalforce
    #Dsh = Dh ** (-1)    #partional horizontal Force transformed to
        full Diagonalforce

    #Sectionheight Beams
    MS_upper0 = 0
    # print("nf1: " + str(nf1))
    for nf2 in range(0, field):  # loop for sumM with F on every
       joint
        MS_upper1 = F * (nf2) * lf
        MS_upper0 = MS_upper0 + MS_upper1
        # print("nf2: "+str(nf2) + "; S_upper: "+str(S_upper1))
    f_upper = (-A*(field-1)*lf + MS_upper0) / height
    SH_upper = f_upper / (fcd * width) * 1000


    #force and SH in diagonals
    f_diag = (-A + (field)*F) * Dsv
    SH_diag = f_diag / (fcd  * width) * 1000


    #force and SH in lower beams
    f_lower = -f_diag * Dh - f_upper
    SH_lower = f_lower / (fcd  * width) * 1000


    #force in Posts
    if field == 1:
            f_post = -F
    else:
        f_post = -f_diag * Dv
    SH_post = f_post / (fcd  * width) * 1000


    if field == fields:
        SH_last_post = -A/(fcd * width) * 1000
```

```
    else:
        SH_last_post = 0

    SH_upper = (SH_upper**2)**0.5
    SH_diag = (SH_diag**2)**0.5
    SH_post = (SH_post**2)**0.5
    SH_lower = (SH_lower**2)**0.5
    SH_last_post = (SH_last_post**2)**0.5

    return (SH_upper, SH_diag, SH_post, SH_lower, SH_last_post)


# load is transformed to a pointload F=load*length/(field+1)
load_list = []
length_list = []
field_list = []
field_count_list = []
pos_list = []
fulltruss_list = []

#list creation - Deconstruction of Truss
for field_count in range(1, fields+1):
    fulltruss_list += [load, length, fields, field_count, 1]
    fulltruss_list += [load, length, fields, field_count, 2]
    fulltruss_list += [load, length, fields, field_count, 3]
    fulltruss_list += [load, length, fields, field_count, 4]
    if field_count == fields:
        fulltruss_list += [load, length, fields, field_count, 5]



SH_upper_list = []
SH_diag_list = []
SH_post_list = []
SH_lower_list = []
SH_last_post_list = []


#SH calculation
SH_sum_list = []
for field_calc in range(1, fields+1):
  SH_upper, SH_diag, SH_post, SH_lower, SH_last_post =
      TrussSolverSHbbb(load, length, fields, height, fcd, field_calc
      , width)
  SH_sum_list.append(SH_upper)
  SH_sum_list.append(SH_post)
  SH_sum_list.append(SH_diag)
  SH_sum_list.append(SH_lower)
  SH_last_post_list.append(SH_last_post)
  if field_calc == fields:
      SH_sum_list.append(SH_last_post_list[-1])
```

Listing 8: Auslesen der Kontrolldaten zu Inputdaten für KI Anwendung

```
import rhinoscriptsyntax as rs

load = float(load-1)
length = float(length-1)
fields = int(fields)

```

```
7
8  # Pfad zur nicht gemischelten Datei
9  filepath = 'D:\\...\\...\\...\\DATAbbb+Knicken.txt'
10
11 # Ausgabe Liste
12 SH_values = []
13
14 # oeffnen der Datei im Lese-Modus
15 with open(filepath, 'r') as file:
16     # Gehe durch jede Zeile in der Datei
17     for line in file.readlines():
18         # Teile die Zeile an den | Zeichen
19         parts = line.rstrip()
20         parts = line.split('|')
21         load_r = float(parts[0])
22         length_r = float(parts[1])
23         fields_r = int(parts[2])
24         SH_r = parts[6]
25         #print(load_r)
26
27         if load_r == load:
28             if length_r == length:
29                 if fields_r == fields:
30                     SH_values.append(SH_r)
31
32 # Drucke die Ausgabe
33 print(SH_values)
```

C03 - Programmcode (Google Colaboratory): Fachwerk-KI

siehe Kap. [4.2.3]

Listing 9: Verbindung zu Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Listing 10: Bibliotheken importieren

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

Listing 11: Datensatz einlesen

```
# Pfad zur Datei
filepath = '/content/drive/MyDrive/.../.../DATAbbb_shuffled.txt'

# Leere Listen fuer die Variablen
load_values = []
length_values = []
field_max_values = []
field_count_values = []
pos_values = []
SH_values = []

# oeffnen der Datei im Lese-Modus
with open(filepath, 'r') as file:
    # Gehe durch jede Zeile in der Datei
    for line in file:
        # Zeile an den | Zeichen teilen
        parts = line.split('|')

        # Teile den entsprechenden Variablen zuweisen
        load = int(parts[0])
        length = int(parts[1])
        field_max = int(parts[2])
        field_count = int(parts[3])
        pos = int(parts[5])
        SH = float(parts[6])

        # Werte zu Listen hinzufuegen
        load_values.append(load)
        length_values.append(length)
        field_max_values.append(field_max)
        field_count_values.append(field_count)
        pos_values.append(pos)
        SH_values.append(SH)

# Zuweisung der Ausgangsvariablen
a = load_values
b = length_values
c = field_max_values
d = field_count_values
e = pos_values
```

```
41 f = SH_values
```

Listing 12: Tensoren erstellen

```
1 # Erstellen 2D-Array fuer die Eingaben
2 input_data = np.array([a, b, c, d, e]).T  # ".T" ist fuer die
      Transposition, um die richtige Form zu bekommen
3
4 # Erstellen 1D-Array fuer die Ausgaben
5 output_data = np.array(f)
6
7 # Konvertieren der Arrays in Tensoren
8 input_tensor = tf.convert_to_tensor(input_data, dtype=tf.float32)
9 output_tensor = tf.convert_to_tensor(output_data, dtype=tf.float32)
10
11 # Groesse des neuen Tensors bestimmen
12 original_size = tf.shape(input_tensor)[0]
13 new_size = tf.cast(0.8 * tf.cast(original_size, tf.float32), tf.
      int32)
14
15 # Tensor in Trainings- und Testdaten aufteilen
16 input_train_tensor, input_test_tensor = tf.split(input_tensor, [
      new_size, original_size - new_size])
17
18 # input_train_tensor enthaelt 80% der Daten des originalen Tensors
19 # input_test_tensor enthaelt die restlichen 20% der Daten des
      originalen Tensors
20
21 #Groesse des neuen Tensors
22 original_size = tf.shape(output_tensor)[0]
23 new_size = tf.cast(0.8 * tf.cast(original_size, tf.float32), tf.
      int32)
24
25 # Tensor in Trainings- und Testdaten aufteilen
26 output_train_tensor, output_test_tensor = tf.split(output_tensor, [
      new_size, original_size - new_size])
27 output_train_tensor.shape, output_test_tensor.shape
```

Listing 13: KI-Modell erstellen

```
1 #Modell definieren
2 model = keras.Sequential([
3     layers.Dense(64, activation='relu', input_shape=[5]),  # Die
          Eingabe hat 5 Features
4     layers.Dense(64, activation='relu'),
5     layers.Dense(64, activation='relu'),
6     layers.Dense(64, activation='relu'),
7     layers.Dense(64, activation='relu'),
8     layers.Dense(1),  # Die Ausgabe hat 1 Feature
9 ])
10
11 #Kompilieren
12 model.compile(
13     optimizer='adam',
14     loss='mae',  # Mean Absolute Error
15 )
16
17 #Training
18 history = model.fit(
19     input_train_tensor, output_train_tensor,
20     validation_data=(input_test_tensor, output_test_tensor),
```

```
    batch_size=256,
    epochs=100,
)

# Evaluation
loss = model.evaluate(input_test_tensor, output_test_tensor)

print(f'Loss: {loss}')
```

Listing 14: Auswertung darstellen

```
# Verlustverlauf plotten
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Trainingsverlust')
plt.plot(history.history['val_loss'], label='Validierungsverlust')
plt.title('Verlauf des Trainings- und Validierungsverlusts')
plt.xlabel('Epochen')
plt.ylabel('Verlust')
plt.legend()
plt.show()
```

Listing 15: KI Anwendung

```
load = 5
length = 6
fields = 3

# load is transformed to a pointload F=load*length/(field+1)
load_list = []
length_list = []
field_list = []
field_count_list = []
pos_list = []

for field_count in range(1, fields+1):
    load_list += [load, load, load, load]
    length_list += [length, length, length, length]
    field_list += [fields, fields, fields, fields]
    field_count_list += [field_count, field_count, field_count,
       field_count]
    pos_list += [1, 2, 3, 4]

    if field_count == fields:
        load_list.append(load)
        length_list.append(length)
        field_list.append(fields)
        field_count_list.append(field_count)
        pos_list.append(5)

# Erstellen 2D-Array fuer die neuen Eingaben
input_data_new = np.array([load_list, length_list, field_list,
    field_count_list, pos_list]).T

# Konvertieren Array in einen Tensor
input_tensor_new = tf.convert_to_tensor(input_data_new, dtype=tf.
    float32)

# Vorhersagen fuer neue Daten
predictions = model.predict(input_tensor_new)
print(predictions)
```

C04 - Grasshopper: Skript Fachwerkträger

Dataset import

Sectionwidth = 0.3m
Trusssystem heigth = 1m

Tensor creation

KI - Training

KI - Testing/Validating

KI - Usage Example

TRAINING

TESTING

Features

Determine Trussesystem Geometry

load values

length values

max field - values

e.g. given Information for locating the beams position in the Truss

field count - values

Position - values

Also required when using the AI - a small python component will do it

Label

required Height - values

AI Regression-Trainer

ANN

ANN - SpeicherPFAD

"Select a Directory"

Ist

Soll

Betrag der Ergebnisdifferenz

ANN - DATEI
Weigths and Bias einlesen

"Select one existing file"

Truss Visualisation

erforderliche Querschnittshöhe

Literatur

[1] **.NET Foundation**. *IronPython - Offizielle Website*. 2022. URL: https://ironpython.net/.

[2] **Apellániz**, D. *PUG auf food4Rhino.com*. 2023. URL: https://www.food4rhino.com/en/app/pug.

[3] **Baduge**, SK u. a. „Artificial intelligence and smart vision for building and construction 4.0: Machine and deep learning methods and applications“. In: *Automation in Construction* 141 (2022), S. 104440.

[4] **Bogon**, T. *Agentenbasierte Schwarmintelligenz*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013.

[5] **Breymann**, W. *C++ für Anfänger: Eine Einführung*. München: Hanser, 2016.

[6] **Burnett**, M. „Visual Programming“. In: *Wiley encyclopedia of electrical and electronics engineering* 16 (1999), S. 768.

[7] **Caversan**, FL. *Fuzzy Computing*. 2009. URL: http://aforgenet.com/articles/fuzzy_computing_basics/.

[8] **Choo**, K u. a. *Machine Learning kompakt: Ein Einstieg für Studierende der Naturwissenschaften*. Wiesbaden: Springer Fachmedien Wiesbaden, 2020.

[9] **Cuvilliers**, P. *GHPython remote auf github.com*. 2022. URL: https://github.com/pilcru/ghpythonremote.

[10] **Davidson**, S. *Grasshopper 3D - Website*. 2023. URL: https://www.grasshopper3d.com/.

[11] **Davidson**, S. *Rhino Developer - An Overview of the GhPython Component*. 2019. URL: https://developer.rhino3d.com/guides/rhinopython/ghpython-component/.

[12] *Der Grasshopper Primer (DE) - Third Edition V3.3*. Mode Lab LLC für Bob McNeel and Scott Davidson and Entwicklungsteam von Robert McNeel and Associates. 2016.

[13] **Digital Structures**. *GHPython remote*. 2023. URL: http://digitalstructures.mit.edu/page/tools%5C#gh-python-remote.

[14] **Digital Structures**. *GHPython remote auf food4Rhino.com*. 2022. URL: https://www.food4rhino.com/en/app/gh-python-remote.

[15] **Dlubal Software**. *Prospekt Dlubal Software*. 2018.

[16] **Dlubal Software**. *Prospekt RFEM 5*. 2017.

[17] **Dlubal Software**. *Prospekt RSTAB 8*. 2018.

[18] **Dwivedi**, YK u. a. „Opinion Paper: “So what if ChatGPT wrote it?” Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy“. In: *International Journal of Information Management* 71 (2023), S. 102642.

[19] **Gerlach**, S. „Programmieren in C“. In: *Computerphysik: Einführung, Beispiele und Anwendungen*. Berlin: Springer-Verlag GmbH Deutschland, 2019, S. 31–54.

[20] **Google**. *Machine Learning*. 2023. URL: https://developers.google.com/machine-learning.

[21] **Google**. *Tensorflow - Dokumentation zu Module:tf*. 2023. URL: https://www.tensorflow.org/api_docs/python/tf.

[22] **Goyal**, M u. a. „Artificial intelligence-based image classification methods for diagnosis of skin cancer: Challenges and opportunities“. In: *Computers in Biology and Medicine* 127 (2020), S. 104065.

[23] **joachim für McNeel**. *McNeel Wiki - Die Geschichte von Rhino*. 2021. URL: https://wiki.mcneel.com/de/rhino/rhinohistory.

[24] **karamba**. *Karamba3D auf food4Rhino.com*. 2022. URL: https://www.food4rhino.com/en/app/karamba3d.

[25] **karamba**. *Official Guide*. 2023. URL: https://manual.karamba3d.com/.

[26] **Kuhail**, MA u. a. „Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review“. In: *IEEE Access* 9 (2021), S. 14181–14202.

[27] **Liberty**, J. *Programming C#*. Princeton, N.J.: Recording for the Blind & Dyslexic, 2006.

[28] **Microsoft**. *Excel-Hilfe und -Learning*. 2023. URL: https://support.microsoft.com/de-de/excel.

[29] **Mishra**, P. *Practical Explainable AI Using Python: Artificial Intelligence Model Explanations Using Python-based Libraries, Extensions, and Frameworks*. Apress, 2021.

[30] **Nahrstedt**, H. *Algorithmen für Ingenieure: Technische Realisierung mit Excel und VBA*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018.

[31] **Osterhage**, WW. „Was ist Intelligenz?“ In: *IT-Kompendium: Die effiziente Gestaltung von Anwendungsplattformen*. Berlin: Springer-Verlag GmbH Deutschland, 2017, S. 1–13.

[32] **Paaß**, G und **Hecker**, D. *Künstliche Intelligenz: Was steckt hinter der Technologie der Zukunft?* Erstauflage. Wiesbaden: Springer Fachmedien Wiesbaden GmbH, 2020.

[33] **Proving Ground**. *Lunchbox Documentation*. 2023. URL: https://apps.provingground.io/docs/lunchbox-documentation/.

[34] **PyTorch Foundation**. *Official Website - Tutorials*. 2023. URL: https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html.

[35] **Rahman**, MA. *GHCPython auf food4Rhino.com*. 2023. URL: https://www.food4rhino.com/en/app/ghcpython.

[36] **Rahman**, MA. *GHCPython auf github.com*. 2021. URL: https://github.com/MahmoudAbdelRahman/GH_CPython.

[37] **Robert McNeel and Associates**. *Rhino3D - Eigenschaften von Rhino*. 2023. URL: https://www.rhino3d.com/features/.

[38] **Salehi**, H und **Burgueño**, R. „Emerging artificial intelligence methods in structural engineering“. In: *Engineering Structures* 171 (2018), S. 170–189.

[39] **Seim**, W und **Hummel**, J. *Ingenieurholzbau: Basiswissen: Tragelemente und Verbindungen*. 1. Auflage. Bauingenieur-Praxis. Berlin: Ernst et Sohn, 2019.

[40] **Sofistik**. *Über Sofistik*. 2023. URL: https://www.sofistik.de/unternehmen/ueber-uns.

[41] **Steyer**, R. *Programmierung in Python: Ein kompakter Einstieg für die Praxis*. Springer Fachmedien Wiesbaden, 2018.

[42] **Walsh**, B. *PRODUCTIONIZING AI WITH PYTHON: How to deliver ai b2b solutions*. Ely, UK: Apress, 2023.

[43] **Ziegler**, A und **Heller**, K. „Intelligenz“. In: *Lexikon der Psychologie* (2000). URL: https://www.spektrum.de/lexikon/psychologie/intelligenz/7263.

[44] **Zimmermann**, H. „Prinzipien der Fuzzy Logic“. In: *Spektrum der Wissenschaft* 3 (1993). URL: https://www.spektrum.de/magazin/prinzipien-der-fuzzy-logic/820699 (besucht am 02. 05. 2023).

[45] **Zuckarelli**, JL. *Programmieren lernen mit Python und JavaScript: Eine praxisorientierte Einführung für Einsteiger*. Wiesbaden: Springer Fachmedien Wiesbaden, 2021.

[46] **Zwierzycki**, M. *OWL auf food4Rhino.com*. 2023. URL: https://www.food4rhino.com/en/app/owl.

[47] **Zwierzycki**, M. *Owl auf github.com*. 2023. URL: https://github.com/mateuszzwierzycki/Owl.