Universität Rostock

Traditio et Innovatio

# Titel der Arbeit, bei Bedarf auch zweizeilig

Untertitel der Arbeit, auch mehrzeilig oder ganz weglassen.


| | |
|---|---|
| Name: | Vorname Nachname |
| Matrikelnummer: | 123 45678 |
| Abgabedatum: | 12.08.2021 |
| Betreuer und Gutachter: | Name des Betreuers und ersten Gutachters<br>Universität Rostock<br>Fakultät |
| Gutachter: | Name des zweiten Gutachters<br>Universität Musterstadt<br>Fakultät |


**Masterarbeit am Institut für Mathematik**
**Mathematisch-Naturwissenschaftliche Fakultät**

## Zusammenfassung

Platz für eine kurze Zusammenfassung.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Algorithmenverzeichnis

# Verzeichnis der Listings

# Abkürzungsverzeichnis

# Symbolverzeichnis

# 1. Einleitung

Es mag euch wundern, dass die Einleitung in einem separaten File abgelegt ist. Dies muss natürlich nicht so sein. Es könnte aber bei einer langen Abschlussarbeit durchaus die Übersichtlichkeit erhöhen, wenn ihr für verschiedene Kapitel einzelne Dateien anlegt und diese mittels

```
\input{<DateiName>}
```

oder

```
\include{<DateiName>}
```

einfügt.

Verwendet keine Umlaute oder Leerzeichen in Dateinamen.
`input` fügt den Text direkt an die Stelle des `input`-Befehls ein.
`include` fügt den Text auf einer neuen Seite ein.

# 2. Grundlagen

## Mathe/ ML Learning

Problemstellung(Einleitung)
  Training, Aufgabe Leistung
  supervisies, unsupervised erklären
  Klassifikationsproblem
  Merkmalsextraktion( 1FFT 2FFT, IFFT NFFT)
  (Faltung)
  (FFT Regeln insb convolution/coprrelation theorem mit FFT)
  Trennbarkeit linear/nichtlinear Entscheidungsgrenzen Hyperebene
  Perzeptron Theorem
  numerische Minimierung, kurz Abstiegsverfahren in einfacher Version
  falls nötig adaptive Verfahren
  warum NN?
  warum später CNN?

## SQL

Relationen, Tensoren
  Marizen/Vektoren als Relationen
  Basisoperationen

# 3. Grundlagen neuronaler Netze

In diesem Abschnitt werden Künstliche Neuronale Netze[**dayhoff1990neural**], kurz KNN, als Forschungsgegenstand der Informatik eingeführt und deren mathematischen Grundlagen präzisiert. Sie stellen informationsverarbeitende Systeme nach dem Vorbild von tierischen beziehungsweise menschlichen Gehirnen dar und bestehen aus Neuronen in gewissen Zuständen und Schichten, die über gewichtete Verbindungen miteinander gekoppelt sind. Jene Gewichte sind als freie Parameter des neuronalen Netzes zu verstehen und können während des Trainingsprozesses so angepasst werden, um eine entsprechende Aufgabe zu lösen. Gelingt dies, so können neuronale Netze genutzt werden, um bestimmte Muster in Daten, typischerweise in Bildern, Audio oder Stromdaten, effizient zu erkennen[**pandya1995pattern**, **pao1989adaptive**, **urbaniak2021quality**]. Sie eignen sich daher für viele typischen Aufgaben des maschinellen Lernens, beispielsweise für die Klassifikation digitalisierter Objekte.

Im ersten Abschnitt wird das Perzeptron[12] als Grundeinheit eines neuronalen Netztes eingeführt. Im folgendem Abschnitt wird das Konzept der Multi-Layer-Perzeptronen[17] durch die Kopplung mehrerer Perzeptronen mit bestimmten Übertragungs- und Aktivierungsfunktion in einem Netz erläutert. Diese Repräsentierung eines KNN wird im weiteren Verlauf dieser Arbeit genutzt.

## 3.1. Das Perzeptron

Zunächst wird das *Perzeptron* ähnlich wie in Minsky [10] als fundamentaler Baustein eines neuronalen Netzes eingeführt. Das Perzeptron stellt im Allgemeinen ein erstes trainierbares Modell dar und wird oft als Basis moderner KNN angeführt.

**Definition 1** (Perzeptron)**.** *Für eine gegebene Funktion $\psi : \mathbb{R} \to \mathbb{R}$, einen Vektor $w \in \mathbb{R}^n$ und ein Skalar $\theta \in \mathbb{R}$ wird die Funktion*

$$\Psi : \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto \psi(w^T x + \theta) =: y,$$

*Perzeptron genannt. Mit $x \in \mathbb{R}^n$ wird die vektorwertige Eingabe und mit $y \in \mathbb{R}$ die skalare Ausgabe des Perzeptrons bezeichnet. Dabei ist mit $w^T x = \sum_{i=1}^n w_i x_i$ das Standardskalarprodukt im euklidischen Vektorraum $\mathbb{R}^n$ gemeint. Die Komponenten von $w$ werden Gewichte und der Skalar $\theta$ Schwellwert oder auch Bias genannt.*

Die Funktionsweise eines Perzeptrons ist in Abbildung 3.1 dargestellt.

Bei der Wahl der Funktion $\psi$ gibt es mehrere Möglichkeiten. Wird wie in Minsky[10]

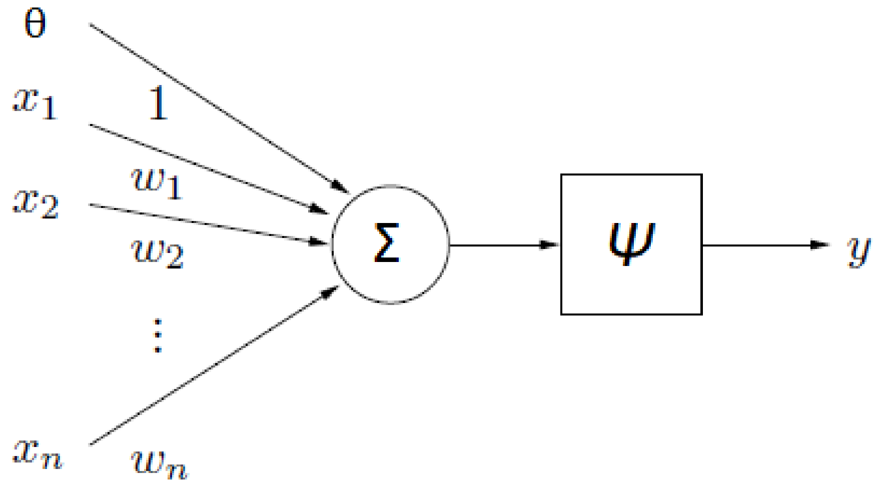



Abbildung 3.1.: Arbeitsweise eines Perzeptrons mit entsprechender Notation aus Definition 1.

die Heavyside-Funktion

$$\psi : \mathbb{R} \to \mathbb{R}, \quad \psi(x) = \begin{cases} 1 & , x \geq 0 \\ 0 & , \text{sonst} \end{cases}$$

genutzt, kann das Perzeptron als binärer Klassifikator wie in **??** interpretiert werden. Dabei dient $w^T x + \theta = 0$ als trennende Hyperebene. Ist $w^T x + \theta < 0$, so ist $\psi(x) = 0$ und $x$ wird der Klasse $K_{-1}$ zugeordnet. Gilt jedoch $w^T x + \theta \geq 0$ und damit $\psi(x) = 1$, so ist der Vektor $x$ der Klasse $K_1$ zugehörig.

Für ein Klassifikationsproblem, bei dem die Klassen nicht linear trennbar sind, scheitern diese einfachen Perzeptronen. Hier wird oft das zweidimensionale XOR-Problem angeführt, bei denen die Punktmengen $P_{-1} = \{(0,0),(1,1)\}$ und $P_1 = \{(1,0),(0,1)\}$ getrennt werden sollen. Um solche Aufgaben zu lösen, ist es notwendig, mehrere Perzeptronen geschickt zu verknüpfen, um komplexe Entscheidungsgrenzen zu erhalten.

## 3.2. Multi-Layer-Perzeptron

In dieser Arbeit wird ein Künstliches Neuronales Netz als eine Menge von Perzeptronen, die in gewissen Schichten partitioniert und miteinander verbunden sind, notiert. Diese sogenannten *Multi-Layer-Perzeptronen*, kurz MLP, gelten als erste tiefe neuronale Netze und sind seit den späten 1980er Gegenstand der Forschung[3, 2, 13]. Zunächst sind einige Definition notwendig, um eine lesbare Notation des MLPs zu geben.

**Definition 2** (Übertragungsfunktion). *Für eine gegebene Matrix $W \in \mathbb{R}^{n \times m}$ und einen Vektor $b \in \mathbb{R}^m$ ist*

$$\Psi^{W,b} : \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto W^T x + b$$

*als Übertragungsfunktion definiert. Der Vektor* $y = \Psi^{W,b}(x) \in \mathbb{R}^m$ *wird als Netzeingabe bezeichnet.*

Hierbei ist $W$ eine Gewichtsmatrix und $b$ ein Biasvektor, welche als freie Parameter fungieren und die Netzeingabe eines Eingabevektors $x \in \mathbb{R}^n$ auf lineare Art und Weise beeinflussen. Um auch nichtlineare Zusammenhänge darzustellen, werden Aktivierungsfunktionen benutzt.

**Definition 3** (Aktivierungsfunktion)**.** *Eine stetige, monoton steigende und nicht notwendigerweise lineare Funktion* $\psi : \mathbb{R} \to \mathbb{R}$ *wird als Aktivierungsfunktion bezeichnet.*

Es sei erwähnt, dass auch nicht monotone Aktivierungsfunktionen genutzt werden können, beispielsweise radiale Basisfunktionen[11], welche jedoch in dieser Arbeit nicht weiter von Interesse sind. Typische Aktivierungsfunktionen, die oft verwendet werden, sind die:

$$\begin{aligned}
\text{Identität}: \quad & \psi(x) = x, \\
\text{Logistische Funktion}: \quad & \psi(x) = \frac{1}{1+\mathrm{e}^{-x}}, \\
\text{Tangens Hyperbolicus}: \quad & \psi(x) = \tanh(x), \\
\text{ReLU (rectified linear unit)}: \quad & \psi(x) = \max\{0, x\}.
\end{aligned}$$

*Bemerkung* 1. Ist $\psi$ eine Aktivierungsfunktion, so wird für $x \in \mathbb{R}^n$ mit

$$\psi(x) := (\psi(x_1), \dots, \psi(x_n))^T \in \mathbb{R}^n$$

der Vektor bezeichnet, welcher sich durch die elementweise Auswertung der Aktivierungsfunktion $\psi$ an den Stellen $x_1, \dots, x_n$ ergibt.

Bei Klassifikationsproblemen wird oft die *Softmax*-Funktion[4] genutzt, welche die gesamte Eingabe berücksichtigt. Im Abschnitt 3.3 wird erläutert, warum sich in diesem Fall die Softmax-Funktion eignet.

**Definition 4** (Softmax)**.** *Für* $x \in \mathbb{R}^n$ *wird die Funktion* $\psi : \mathbb{R}^n \to (0,1]^n$ *mit*

$$\psi(x) := \left( \frac{\mathrm{e}^{x_1}}{\sum_{i=1}^n \mathrm{e}^{x_i}}, \dots, \frac{\mathrm{e}^{x_n}}{\sum_{i=1}^n \mathrm{e}^{x_i}} \right)^T$$

*als Softmax-Funktion definiert. Die Einträge des Vektors* $\psi(x)$ *summieren sich zu Eins.*

Für den späteren Trainingsprozess ist es nützlich, die Ableitung der verwendeten Aktivierungsfunktion, sofern sie exisitiert, zur Verfügung zu haben. Zudem ist es möglich, für bestimmte Aktivierungsfunktionen die Ableitung nur mithilfe der verwendeten Funktion zu berechnen.

**Lemma 1.** *(i) Für die ReLU* $\psi(x) = \max\{0, x\}$ *gilt*

$$\psi'(x) = \begin{cases} 0 & , x < 0 \\ 1 & , x > 0 \end{cases}.$$

*An der Stelle 0 ist die Ableitung nicht definiert und wird oft mit $\psi'(0) = \frac{1}{2}$ festgelegt.*

*(ii) Für die logistische Funktion $\psi(x) = \frac{1}{1+\mathrm{e}^{-x}}$ gilt*

$$\psi'(x) = \psi(x)(1 - \psi(x))$$

*für alle $x \in \mathbb{R}$.*

*(iii) Für den Tangens Hyperbolicus $\psi(x) = \tanh(x)$ gilt*

$$\psi'(x) = 1 - \psi^2(x)$$

*für alle $x \in \mathbb{R}$.*

*Beweis.* Einfaches Differenzieren liefert für (*i*) und (*ii*) die Resultate. Bei (*iii*) wird die Darstellung $\tanh(x) = \frac{2}{\mathrm{e}^{2x}+1}$ genutzt und das Differenzieren mittels Quotientenregel liefert die Aussage. □

Ähnlich der Definition des Perzeptron 1 wird nun eine Schicht als Verknüpfung von Übertragungsfunktion und Aktivierungsfunktion definiert.

**Definition 5** (Neuronenschicht)**.** *Ist $\Psi^{W,b}$ eine Übertragungsfunktion mit den Parametern $W \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^m$ und $\psi$ eine Aktivierungsfunktion, so wird das Paar $(\Psi^{W,b}, \psi)$ als Schicht $\mathcal{S}$ bezeichnet. Für eine Eingabe $x \in \mathbb{R}^n$ ist die Ausgabe $y \in \mathbb{R}^m$ der Schicht $\mathcal{S}$ durch*

$$y = \psi \circ \Psi^{W,b}(x) = \psi\left(\Psi^{W,b}(x)\right)$$

*gegeben. Die Komponenten $y_i$ werden für $1 \leq i \leq m$ Neuronen der Schicht $\mathcal{S}$ genannt und gleichen jeweils der Ausgabe eines einfachen Perzeptrons wie in Definition 1. Eine Schicht besteht aus $m$ Perzeptronen $\tilde{\Psi}_i$ mit $y_i = \tilde{\Psi}(x_i) = \psi(W_{i,:}^T x + b_i)$ für $1 \leq i \leq m$.*

Im Hinblick auf MLPs werden nun mehrere Schichten so verbunden, dass die Ausgabe einer Schicht $\mathcal{S}_k$ als Eingabe einer darüberliegenden Schicht $\mathcal{S}_{k+1}$ für ein $k \in \mathbb{N}$ dient. Die Anzahl der Neuronen kann dabei von Schicht zu Schicht variieren. Dementsprechend werden die Dimensionen der beteiligten Gewichtsmatrizen $W^{(k)}$ und Biasvektoren $b^{(k)}$ passend gewählt. Um die Notation übersichtlich zu halten, bezeichne $\Psi^{W^{(k)},b^{(k)},\psi_k}$ die Schicht $\mathcal{S}_k$ mit $\Psi^{W^{(k)},b^{(k)},\psi_k}(x) := \psi_k\left(\Psi^{W^{(k)},b^{(k)}}(x)\right)$.

**Definition 6** (Multi-Layer-Perzeptron, vgl. gruening)**.** *Für eine gegebene Anzahl $l \in \mathbb{N}$, $l > 1$ von Schichten $\Psi^{W^{(1)},b^{(1)},\psi_1}, \ldots, \Psi^{W^{(l)},b^{(l)},\psi_l}$ bezeichne $s_l \in \mathbb{N}$ die Anzahl der Neuronen in Schicht $l$. Für eine Eingabe $x \in \mathbb{R}^{s_0}$ lässt sich die Ausgabe $y \in \mathbb{R}^{s_l}$ eines Multi-Layer-Perzeptron $\Lambda_l : \mathbb{R}^{s_0} \to \mathbb{R}^{s_l},\ x \mapsto y$ mit $l$ Schichten durch*

$$y = \Psi^{W^{(l)},b^{(l)},\psi_l} \circ \ldots \circ \Psi^{W^{(1)},b^{(1)},\psi_1}(x)$$

*berechnen. Dabei gelten für die Gewichtsmatrizen die Dimensionsbedingungen*

$${}_1W^{(1)} = s_0,\ \ {}_2W^{(l)} = s_l,\ \ \forall i \in [l-1] :\ {}_2W^{(i)} = {}_1W^{(i+1)}.$$

*Die Eingabeschicht $\mathcal{S}_0$ besitzt keine Parameter $W$ und $b$ und besteht nur aus dem Eingabevektor $x \in \mathbb{R}^{s_0}$. Die letzte Schicht $\Psi^{W^{(l)},b^{(l)},\psi_l}$ wird als Ausgabeschicht bezeichnet. Weiter werden die Schichten $\mathcal{S}_1, \ldots, \mathcal{S}_{l-1}$ als verdeckte Schichten definiert. Die Funktionsauswertung $\Lambda_l(x)$ für eine Eingabe $x$ wird Vorwärtsrechnung, engl. forward propagation, genannt.*

Das MLP-Modell wird im weiteren Verlauf dieser Arbeit repräsentativ als Künstliches Neuronales Netz bezeichnet. Das zuvor angesprochene XOR-Problem kann nun beispielsweise mithilfe eines KNN bestehend aus zwei Schichten gelöst werden[6]. Es lassen sich zwischen Modell- und Hyperparameter von KNN unterscheiden.

**Definition 7** (Hyper- und Modellparameter)**.** *Sei für $l \in \mathbb{N}$ ein KNN $\Lambda_l$ gegeben. Dann werden die Eingabe- und Ausgabedimension $s_0, s_l$, die Anzahl $l$ der (verdeckten) Schichten sowie die verwendeten Aktivierungsfunktion $\psi_l$ Hyperparameter des neuronalen Netzes genannt. Die Gewichtsmatrizen und Biasvektoren mit den entsprechend passenden Abmessungen stellen die Modellparameter $\mathcal{W} := \{(W^{(i)}, b^{(i)}) : \ i = 1, \ldots, l\}$ des neuronales Netzes dar.*

Die Hyperparameter werden oft anwendungsspezifisch für das jeweilige Problem gewählt, während die Modellparameter dynamisch in einem Trainingsprozes angepasst werden, sodass die gegebene Aufgabe zufriedenstellend gelöst wird. Wie sies geschieht, wird im folgenden Abschnitt 3.3 erläutert.

## 3.3. Training neuronaler Netze

Künstliche Neuronale Netze gehören zu den typischen Vertretern von maschinellen Lernalgorithmen, welche hinsichtlich einer bestimmten Aufgabe, engl. *task* $T$, und einem Leistungsmaß, engl. *perfomance* $P$ an der Erfahrung, engl. *experience* $E$ lernen[6]. Dabei ist mit Lernen gemeint, dass das Computerprogramm bezüglich der Aufgabe $T$ sein Leistungsmaß $P$ mit wachsener Erfahrung $E$ schrittweise steigert. Wie in Kapitel **??** erläutert, gibt es viele verschiedene Aufgaben, wie die Regression, Klassifikation oder Clusterung bestimmter Objekte.

In den folgenden Abschnitten wird das Klassifikationsproblem im Mittelpunkt stehen. Weiter werden KNNs als Modellschätzer aus der Wahrscheinlichkeitstheorie interpretiert und fundamentale Aussagen wie das *Universal-Approximation-Theroem*[8] gegeben. Schließlich wird bezüglich der Klassifikationsaufgabe das Training neuronaler Netze erläutert.

### 3.3.1. Neuronale Netze als universelle Schätzer

Beim Klassifikationsproblem müssen bestimmte bedingte Wahrscheinlichkeiten, die in diesem Abschnitt erklärt werden, ermittelt werden. Oft wird dazu die Ausgabeschicht eines KNN als Wahrscheinlichkeit interpretiert und daher KNN als Schätzer der bedingten Wahrscheinlichkeiten eingesetzt. Zunächst werden Klassifikationsfunktion und -problem definiert.

**Definition 8.** *Seien die Mengen $D \subset \mathbb{R}^n$ und $\mathcal{C} = \{c_1, \dots, c_m\}$ gegeben. Eine Funktion $f : D \to \mathcal{C}$, welche ein Element aus $D$ einer Klasse $c_i \in \mathcal{C}$ zuordnet, wird Klassifikationsfunktion genannt. Hier gibt es $m \in \mathbb{N}$ verschiedene Klassenlabels. Die Menge $D$ wird abstrakter Datensatz genannt.*

Das Ziel beim Klassifikationsproblem ist die Approximation einer nicht bekannten Klassifikationsfunktion $f : D \to \mathcal{C}$ durch ein Modell $\tilde{f} : D \to \mathcal{C}$. In dieser Arbeit werden dafür KNNs genutzt, welche als probabilistische Modelle auf folgende Weise genutzt werden. Auf der Ergebnismenge $\Omega = D \times \mathcal{C}$ sei die nicht bekannte gemeinsame (Wahrscheinlichkeits-) Verteilung $p_{Daten}(x, c)$, genannt Datenverteilung, gegeben. Ein Modell soll nun konstruiert werden, welches die posterior-Verteilung $p_{Daten}(\cdot \mid x)$ der Klassen schätzt.

In dieser Arbeit werden KNN so benutzt, dass die Klassenzugehörigkeit direkt anhand der Eingabe $x \in D$ geschätzt wird. Die Funktion $P_{Daten} : D \to [0,1]^m$ mit

$$P_{Daten}(x) := (p_{Daten}(c_1 \mid x), \dots, p_{Daten}(c_m \mid x))^T \in \mathbb{R}^m \tag{3.1}$$

soll für alle $x \in D$ approximiert werden. Dazu wird die Funktion $P_{Modell} : D \to [0,1]^m$ mit

$$P_{Modell}(x; \mathcal{W}) := (p_{Modell}(c_1 \mid x; \mathcal{W}), \dots, p_{Modell}(c_m \mid x; \mathcal{W}))^T \in \mathbb{R}^m \tag{3.2}$$

für alle $x \in D$ als Approximation genutzt, welche von den Modellparametern $\mathcal{W}$ abhängig ist. Die Klassifikationsfunktion des Modells ergibt sich als

$$f_{Modell}(x) := \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, p_{Modell}(c \mid x). \tag{3.3}$$

Es stellt sich die Frage, inwiefern das MLP als Modell genutzt werden kann, um beliebige Datenverteilungen $P_{Daten}$ zu approximieren. Folgende Resultate liefern die Antwort.

**Satz 1** (Universal-Approximation-Theroem[**gruen**]). *Sei $\psi_1$ eine nichtkonstante, beschränkkte Aktivierungsfunktion und $id : \mathbb{R} \to \mathbb{R}$ die Identität sowie $D \subset \mathbb{R}^n$ kompakt. Dann existieren für alle $\varepsilon > 0$ und stetigen Funktionen $f : D \to \mathbb{R}$ Parameter $N \in \mathbb{N}, W^{(1)} \in \mathbb{R}^{n \times N}, b^{(1)} \in \mathbb{R}^N$ sowie $W^{(2)} \in \mathbb{R}^{N \times 1}$, sodass*

$$\left| f(x) - \Psi^{W^{(2)},0,id} \circ \Psi^{W^{(1)},b^{(1)},\psi_1}(x) \right| < \varepsilon, \quad \forall x \in D \tag{3.4}$$

*gilt.*

*Beweis.* Ein Beweis kann in Hornik[7] nachgelesen werden. □

Das Universal-Approximation-Theroem kann ebenfalls auf unbeschränkte und nichtkonstante Funktion $f : D \to \mathbb{R}^m$ erweitert werden. Heutzutage wird oft die ReLU-Funktion als Aktivierungsfunktion verwendet[15, 9].

**Korollar 1.** *Mit den gleichen Vorraussetzungen wie in Satz 1 gilt die Abschätzung 3.4 für $\psi_1(x) = \max\{0, x\}$.*

*Beweis.* Siehe Sonoda et. al.[16]. □

Hinsichtlich der Approximation von beliebigen Funktionen $P_{Daten}$ mithilfe eines neuronalen Netzes mit der Softmax-Funktion als Aktivierungsfunktion liefert Strauß[**strauss**] folgendes Resultat.

**Korollar 2.** *Ein MLP mit zwei Schichten, wobei $\psi_2$ die Softmax-Funktion ist, kann genutzt werden, um stetige Funktionen $f : K \to [0,1]^m$, welche von einem Kompaktum $K \subset \mathbb{R}^n$ in eine (Wahrscheinlichkeits)-Verteilung über die Klassen $\mathcal{C}$ abbilden, beliebig genau zu approximieren.*

*Beweis.* Siehe [**strauss**]. □

Die Aussage kann auf das MLP mit beliebig vielen Schichten erweitert werden. In dieser Arbeit umfasst die Menge $D$ aus Definition 8 digitalisierte Objekte und ist endlich und damit kompakt. Daher kann wegen Korollar 2 das MLP als Modell genutzt werden, um stetige Funktionen $P_{Daten}$ sinnvoll zu approximieren.

### 3.3.2. Optimale Parameterwahl bei neuronalen Netzen

Wird ein künstliches neuronales Netz als probabilistisches Modell genutzt und sind die Hyperparameter festgelegt, müssen die Modellparameter $\mathcal{W}$ gewählt werden. Um die Approximationsgüte messbar zu machen, werden Fehlerfunktionen genutzt, welche mithilfe des Gradientenverfahrens aus Kapitel **??** durch die Anpassung der Parameter $\mathcal{W}$ in einem Trainingsprozess minimiert werden.

**Definition 9** (Trainingsmenge)**.** *Seien $p_{daten}$ eine Datenverteilung und $\Omega = D \times \mathcal{C}$. Dann heißt für ein $k \in \mathbb{N}$ die Menge*

$$\mathcal{T} := \left\{ (x^{(i)}, c^{(i)}) \mid i \in [k] \right\} \subset \Omega$$

*Trainingsmenge bestehend aus $k$ Trainingsdaten, welche unabhängig durch $p_{Daten}$ generiert wurde.*

Die Güte eines Modells $f_{Modell}$ wird als Likelihood gegeben einer Trainingsmenge $\mathcal{T}$ gemessen und ist durch

$$L(\mathcal{T}, \mathcal{W}) := \prod_{(x,c)\in\mathcal{T}} p_{Modell}(c \mid x; \mathcal{W})$$

gegeben[1]. Ziel ist es, dass $P_{Modell}$ und $P_{Daten}$ übereinstimmen und daher wird versucht, die Likelihood zu maximieren. Für einen gegebene Trainingsmenge $\mathcal{T}$ soll das Produkt über alle Wahrscheinlichkeiten der korrekten Klassenzugehörigkeiten maximiert werden. Dieser Ansatz wird *Maximum Likelihood-Methode*[14] genannt und die Parameterwahl des Modells wird durch eine Lösung des Optimierungsproblems

$$\prod_{(x,c)\in\mathcal{T}} p_{Modell}(c \mid x; \mathcal{W}) \to \max \tag{3.5}$$

gegeben. Dabei sei bemerkt, dass die Optimierung unabhängig von den Hyperparametern vorgenommen wird.

Für ein Trainingspaar $(x, c) \in \mathcal{T}$ bezeichne $t \in \mathbb{R}^m$ den Zielvektor mit den Komponenten

$$t_i := \begin{cases} 1 & \text{, wenn } c = c_i \\ 0 & \text{, sonst} \end{cases}, \quad \forall i \in [m].$$

# 4. Die Faltung

In diesem Kapitel wird erläutert, wie die Faltung (engl. *convolution*) bei CNNs zu verstehen ist und wie diese Operation bei diesen neuronalen Netzen motiviert wird. In diesem Zusammenhang werden Begriffe wie Merkmalskarten (engl. *feature maps*) und Filter (engl. *kernels*) eingeführt. Des Weiteren wird die Arithmetik der Faltungsoperation für zweidimensionale Eingaben, repräsentiert durch Matrizen, erklärt und das Verfahren *padding* beziehungsweise das Nutzen von *strides* erläutert. Das gesamte Kapitel wird mit konkreten Beispielen begleitet, um die verschiedenen Effekte der Faltungsoperation zu beleuchten.

## 4.1. Die Faltungsoperation

In der Analysis ist die Faltung ein mathematischer Operator und liefert für zwei Funktionen $f$ und $g$ die Funktion $f * g$, wobei mit dem Sternchen die Faltungsoperation gemeint ist.

**Definition 10** (Faltung). *Für zwei Funktionen $f, g : \mathbb{R}^n \to \mathbb{C}$ ist die Faltung als*

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x - \tau)\mathrm{d}\tau$$

*definiert, wobei gefordert wird, dass das Integral für fast alle $x$ wohldefiniert ist.*

Bei klassischen neuronalen Netzen, siehe Kapitel **??** werden Eingabedaten durch eine Verkettung von affinen Transformationen verarbeitet. Typischerweise wird die Eingabe als Vektor dargestellt und mit einer Matrix multipliziert, gegebenenfalls mit einem Biasvektor manipuliert und schließlich so die Ausgabe generiert. Bilder-, Audio- oder Videoaufnahmen besitzen jedoch mehrere Merkmale in unterschiedlichen Achsen. Oft sind solche Eingabedaten im Bereich des Machine-Learnings als mehrdimensionale Arrays abgelegt, welche eine oder mehrere Achsen repräsentieren, wobei die Ordnung dieser eine Rolle spielt. Bei digitalisierten Bildern sind das bespielsweise die Höhe und Breite des Bildes, bei Audioaufnahmen gibt es nur eine Achse, und zwar die Zeitachse. Hinzu kommen Kanalachsen als weitere Verfeinerung der Daten, zum Beispiel besitzen RGB-Farbbilder drei Kanäle der Farben rot, grün und blau.

Diese speziellen Eigenschaften können bei affinen Transformationen nicht berücksichtigt werden. Alle Merkmale sowie Achsen werden gewissermaßen gleich behandelt und die wesentliche topologische Struktur kann so nicht zum Vorteil ausgenutzt werden. Hier soll nun die sogenannte diskrete Faltung Abhilfe schaffen.

**Definition 11** (Diskrete Faltung)**.** *Für zwei Funktionen $f, g : D \to \mathbb{C}$ mit einem diskreten Definitionsbereich $D \subseteq \mathbb{Z}^n$ ist die diskrete Faltung als*

$$(f * g)(n) := \sum_{k \in D} f(k)g(x-k)$$

*definiert. Hier wird über dem gesamten Definitionsbereich $D$ summiert. Ist $D$ beschränkt, werden $f$ beziehungsweise $g$ durch Nullen fortgesetzt.*

Besonders bei der Bildverarbeitung wird oft die diskrete Faltung als lineare Operation verwendet.

**Definition 12** (Matrixfaltung, vgl. [**gruening**])**.** *Für gegebene Matrizen $X \in \mathbb{R}^{h \times w}$ und $K \in \mathbb{R}^{k_h \times k_w}$ seien*

$$h_l = \begin{cases} \lfloor k_h/2 \rfloor & , k_h \text{ ungerade} \\ k_h/2 - 1 & , \text{sonst} \end{cases}, \quad w_l = \begin{cases} \lfloor k_w/2 \rfloor & , k_w \text{ ungerade} \\ k_w/2 - 1 & , \text{sonst} \end{cases}.$$

*Die Matrixfaltung $K * X \in \mathbb{R}^{h \times w}$ ist als*

$$(K * X)_{i,j} := \sum_{l=-h_l}^{\lfloor k_h/2 \rfloor} \sum_{m=-w_l}^{\lfloor k_w/2 \rfloor} K_{l+h_l+1, m+w_l+1} X_{i+l, j+m} \quad \forall i \in [h], j \in [w] \tag{4.1}$$

*mit $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [w]$ definiert.*

*Bemerkung* 2. Die Matrixfaltung kann durch viele weitere Parameter genauer spezifiziert werden. Sogannte *strides* bestimmen die Reduktion bei der Faltung von $X$ in der Höhe $h$ beziehungsweise Breite $w$. Für strides $s_h, s_w \in \mathbb{N}$ ist die Matrixfaltung als

$$(K * X)_{i,j} := \sum_{l=-h_l}^{\lfloor k_h/2 \rfloor} \sum_{m=-w_l}^{\lfloor k_w/2 \rfloor} K_{l+h_l+1, m+w_l+1} X_{i \cdot s_h + l, j \cdot s_w + m} \quad \forall i \in [h], j \in [w].$$

Für $s_h = s_w = 1$ ergibt sich die Standardvariante wie in 4.1.

Im Folgenden werden konkrete Beispiele für verschiedene zweidimensionale Faltungen, welche in dieser Arbeit im Fokus stehen, gegeben. Dabei sind die Eingabe $X \in \mathbb{R}^{h \times w}$ und der Filter $K \in \mathbb{R}^{k_h \times k_w}$ immer als Matrizen zu verstehen. Das Ergebnis der Faltung $S = X * K$ wird als Merkmalskarte bezeichnet. Es sei angemerkt, dass oft $k_h = k_w$ sowie $k_h$ ungerade gewählt wird, z.B. $k_h = 3$ oder $k_h = 5$. Die Größe der Merkmalskarte wird durch die Parameter

- $h, w$: Die Höhe und Breite der Eingabe,
- $k_h, k_w$: Die Abmessungen des Filters,
- $s_h, s_w$: Die Wahl der strides,
- $p_h, p_w$: Die Größe des zero paddings

beeinflusst. Mit zero padding ist gemeint, dass künstliche Nullen um Randpixel der Eingabe $X$ eingefügt werden, damit die Berechnung mit dem Filter um jene Pixel gelingt. Ein Beispiel für das Verwenden von zero padding wird in Abbildung 4.2 gezeigt. In Abbildung 4.1 ist die Berechnung einer einfachen zweidimensionalen Matrixfaltung dargestellt. Ein vorher festgelegter Filter (grau) bewegt sich über die Eingabe (blau) und berechnet jeweils die Einträge der Ausgabe(grün).

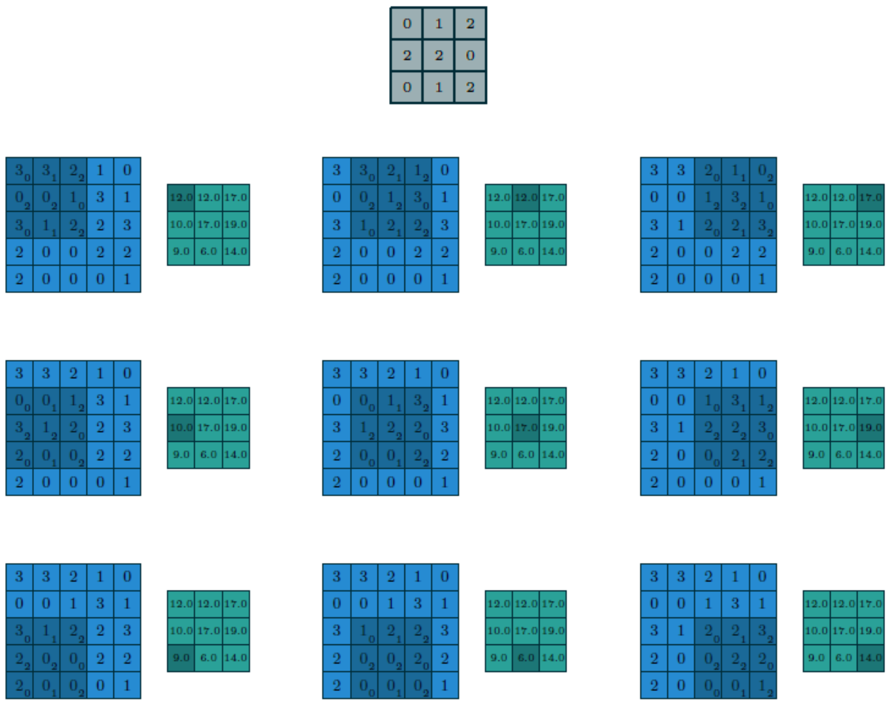


Abbildung 4.1.: Es wird die Merkmalskarte $S \in \mathbb{R}^{3\times 3}$ mit den Parametern $h, w = 5, k_h = k_w = 3, s_h = s_w = 1$ und $p_h = p_w = 1$.

## 4.2. Motivation der Faltung

.. Sie nutzt wichtige Konzepte zur Optimierung von Machine-Learning-Verfahren wie spärliche Konnektivität (engl. *sparse connectivity*), *Parameter Sharing* und *äquivariante Repräsentation*, vgl. [**goodfellow**]. Spärlicher Konnektivität bedeutet, dass die Ausgabeeinheit auf einer bestimmten Schicht nur durch wenige Eingabeeinheiten beeinflusst wird. Dies ist bei CNNs typisch, da meist die verwendeten Filter viel kleiner als die Eingabe ist. Noch mehr erklären + Abbildung

Mit Parameter Sharing ist die Nutzung von gleichen Parametern für mehrere Funktionen im neuronalen Netz gemeint. In herkömmlichen Feed-Forward-Netzen wird jedes Element der Gewichtsmatrizen für die Berechnung der Aktivierungen der jeweiligen Schichten verwendet. Anschließend werden diese Gewichte dann nicht mehr gebraucht.

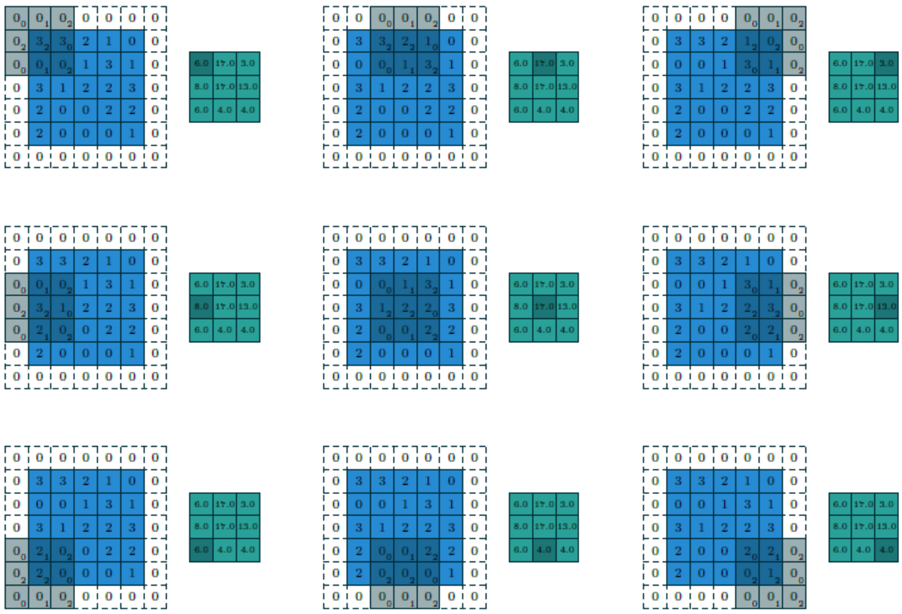


Abbildung 4.2.: Es wird die Merkmalskarte $S \in \mathbb{R}^{3\times 3}$ mit den Parametern $h, w = 5, k_h = k_w = 3, s_h = s_w = 2$ und $p_h = p_w = 1$ berechnet.

Im Zusammenhang von CNNs bedeutet Parameter Sharing während der Faltungsoperation, dass nur eine bestimmte Menge von Parametern erlernt werden müssen Noch mehr erklären + Abbildung

# 5. Weiteres Kapitel

Hier wird dies und das vorgestellt. Unter anderem Fußnoten.[1]

## 5.1. Umgebungen und Formeln

**Definition 13.** *… heißt Rekurrentes Neuronales Netz (RNN).*

*Bemerkung* 3. Bei jeder weiteren Verwendung der Abkürzung wird nur die Kurzform angezeigt: RNN.

*Bemerkung* 4. Die Verwendung des Symbolverzeichnisses ist analog der des Abkürzungsverzeichnisses, siehe Confidence Matrix ($\mathcal{C}$).

**Annahme 1.** *Eine kluge Annahme …*

**Hilfssatz 1.** *Ein kluger Hilfssatz …*

**Satz 2.** *Ein kluger Satz …*

**Korollar 3.** *Ein kluges Korollar …*

**Proposition 1.** *Eine kluge Proposition …*

**Problem 1.** *Ein schweres Problem …*

*Beispiel* 1. Ein anschauliches Beispiel …

**Definition 14.** *Seien* $a, b \in \mathbb{C}$ *definiere*

$$a + b \tag{5.1}$$

*als …*

Auf Formeln kann nun verwiesen werden (siehe (5.1)). Formeln können natürlich auch im normalen Text $a^2 + b^2 = c^2$ auftauchen.

$$\left.\begin{aligned} a^2 + b^2 &= c^2 \qquad (5.2) \\ f &= b - a \qquad (5.3) \end{aligned}\right\} \text{ohne Sinn}$$

[1]Dies ist eine Fußnote.

## 5.2. Aufzählung und Nummerierung

Für Literaturvereichnisse siehe Kapitel 6.3, eine einfache Aufzählung geht so:

- Eins
- Zwei
- Viele

## 5.3. Tabellen

... gibt es viele verschiedene, z. B. Tab. 5.1 und Tab. 5.2.

Tabelle 5.1.: Einfache Tabelle

| Column 1 | Column 2 | Column 3 | Column 4 |
|---|---|---|---|
| Nein | Softmax | 85.0 % | 87.0 % |
| Nein | Linear | 88.8 % | 85.9 % |
| Ja | Softmax | 80.0 % | 89.1 % |
| Ja | Linear | 84.6 % | 89.8 % |

Tabelle 5.2.: Nicht mehr ganz so einfache Tabelle

| | source prior | `abs` | | `prior` | | `da` | |
|---|---|---|---|---|---|---|---|
| | source posterior | `path` | `ctc` | `path` | `ctc` | `path` | `ctc` |
| gAP | `normed` | 94.81 | 94.89 | 95.36 | **95.42** | 94.99 | 95.04 |
| | `unnormed` | 94.77 | | 91.73 | 91.87 | 92.58 | |
| mAP | `normed` | 89.71 | **89.90** | 89.58 | 89.76 | 89.63 | 89.82 |
| | `unnormed` | 89.42 | | 88.59 | 88.89 | 89.13 | |
| gNDCG | `normed` | 96.72 | 96.78 | 96.78 | **96.83** | 96.73 | 96.77 |
| | `unnormed` | 96.69 | | 96.34 | 96.41 | 96.46 | |
| mNDCG | `normed` | 90.77 | **90.97** | 90.66 | 90.85 | 90.70 | 90.89 |
| | `unnormed` | 90.61 | | 89.96 | 90.25 | 90.36 | |

## 5.4. Bilder

### 5.4.1. Einzelnes Bild

Das ist Text. Das ist Text. Das ist Text über Abb. 5.1. Das ist Text.[2]

[2]Dieser Textteil ist von wesentlicher Bedeutung

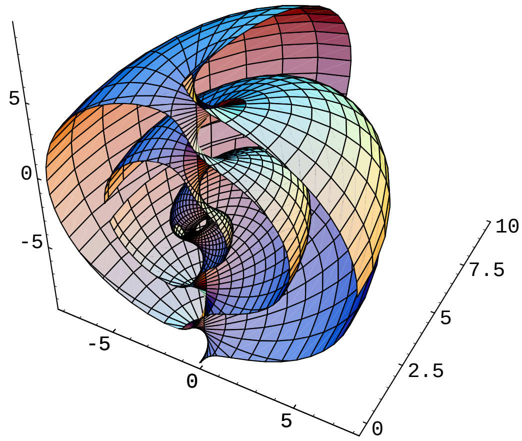
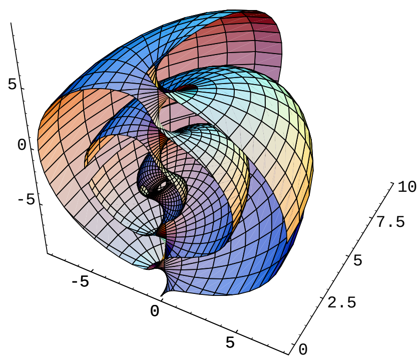
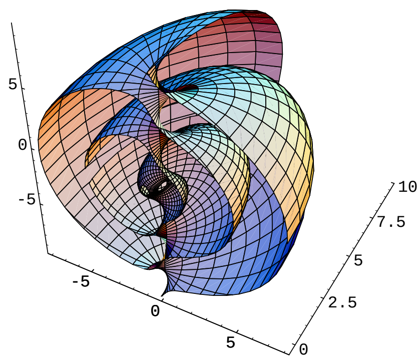



Abbildung 5.1.: Vektorgrafiken sind toll. Scrolle mal in mich rein!

### 5.4.2. Mehrere Bilder




(a) Schnecke 1




(b) Schnecke 2

Abbildung 5.2.: Vergleich verschiedener Schnecken

Die Schnecke aus Abb. 5.2a ist hübscher anzusehen als die aus Abb. 5.2b.

## 5.5. TikZ

TikZ bietet ein mächtiges Werkzeug Grafiken selber zu erzeugen.

### 5.5.1. Einfache Grafiken

Es gibt viele, viele Tutorials und Beispiele die leicht im Internet zu finden sind. Aber ein Beispiel sei an dieser Stelle trotzdem eingefügt, siehe Abb. 5.3.

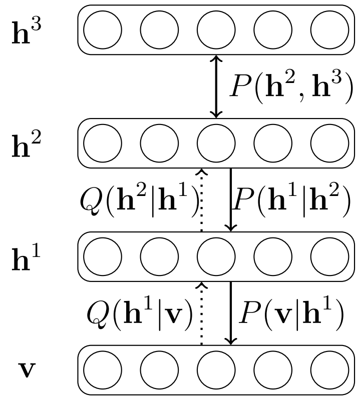



Abbildung 5.3.: Beispiel eines mit TikZ erzeugten Bildes

### 5.5.2. Graphen und ähnliches

Wer keine Lust hat z. B. Achsenbeschriftungen eines Matlab-Plots auf Font etc. des LaTeX-Dokuments anzupassen, kann Datenreihen auch einfach mittels TikZ darstellen, siehe dazu Abb. 5.4. Es ist natürlich auch möglich aus z. B. Matlab oder Gnuplot Tikz Grafiken zu exportieren!

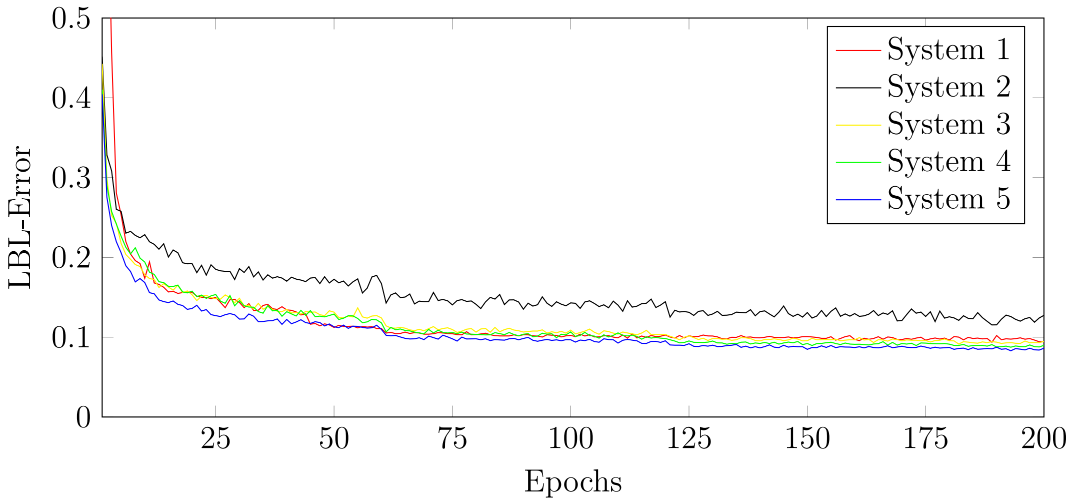



Abbildung 5.4.: Datenreihen mittels TikZ visualisiert

# 6. Ein letztes Kapitel

**Korollar 4.** *Wird für die Festlegung ...*

## 6.1. Weiteres Korollar

In diesem Abschnitt ...

**Korollar 5.** *Wird für die Festlegung ...*

*Bemerkung* 5. Eine vollständige ...

## 6.2. Pseudocode

In vielen Fällen ist es notwendig, Programmteile als Pseudocode darzustellen. Algorithmus 1 stellt ein einfaches Beispiel dar. Es gibt weitere Pakete zur Darstellung von Pseudocode, `algorithm` + `algpseudocode` sei an dieser Stelle erwähnt.

```
Data: this text
Result: how to write algorithm with LaTeX2e
initialization;
while not at end of this document do
    read current;
    if understand then
        go to next section;
        current section becomes this one;
    else
        go back to the beginning of current section;
    end
end
```

**Algorithmus 1:** How to write algorithms

## 6.3. Zitate

Umfangreichen Quellenangaben sollte man in einer Literaturdatenbank pflegen. Um diese in LaTeX zu verwenden bietet sich das Paket biblatex mit dem Sortierprogramm Biber an, da es gewisse Vorteile gegenüber dem klassischen BibTeX besitzt. Die Verweise liegen in einer separaten Datei (hier: `literatur.bib`) und werden mit

```
\addbibresource{<nameDerDatei>}
```

eingefügt. Zitiert wird dann mittels

```
\cite{key}
```

was in unserem Beispiel dann so aussieht [5].
ACHTUNG: Beim ändern der `.bib`-Datei und/oder der Zitate muss mehrfach compiliert werden, damit die änderungen auch wirksam werden. Sicher geht man, wenn man die folgende Reihenfolge beachtet:

1. LaTeX
2. Biber
3. LaTeX
4. LaTeX

Eine genauere Beschreibung findet Ihr im Anhang A.2.

# Literatur


[1] Christopher M Bishop und Nasser M Nasrabadi. *Pattern recognition and machine learning*. Bd. 4. 4. Springer, 2006.

[2] David G Bounds u. a. „A multilayer perceptron network for the diagnosis of low back pain.“ In: *ICNN*. Bd. 2. 1988, S. 481–489.

[3] Herve Bourlard und Christian J Wellekens. „Links between Markov models and multilayer perceptrons“. In: *IEEE Transactions on pattern analysis and machine intelligence* 12.12 (1990), S. 1167–1178.

[4] John Denker und Yann LeCun. „Transforming neural-net output levels to probability distributions“. In: *Advances in neural information processing systems* 3 (1990).

[5] Otto Forster. „Analysis 1“. In: *Vieweg, Braunschweig* (1983).

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

[7] Kurt Hornik. „Approximation capabilities of multilayer feedforward networks“. In: *Neural networks* 4.2 (1991), S. 251–257.

[8] Kurt Hornik, Maxwell Stinchcombe und Halbert White. „Multilayer feedforward networks are universal approximators“. In: *Neural Networks* 2.5 (1989), S. 359–366. DOI: `https://doi.org/10.1016/0893-6080(89)90020-8`.

[9] Yuanzhi Li und Yang Yuan. „Convergence analysis of two-layer neural networks with relu activation“. In: *Advances in neural information processing systems* 30 (2017).

[10] Marvin Minsky und Seymour A Papert. *Perceptrons, Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou: An Introduction to Computational Geometry*. MIT press, 2017.

[11] J. Park und I. W. Sandberg. „Universal Approximation Using Radial-Basis-Function Networks“. In: *Neural Computation* 3.2 (1991), S. 246–257. DOI: `10.1162/neco.1991.3.2.246`.

[12] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), S. 386.

[13] Rumelhart u. a. *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations*. Jan. 1986.

[14] Ludger Rüschendorf. *Mathematische statistik*. Bd. 62. Springer, 2014.

[15] Johannes Schmidt-Hieber. „Nonparametric regression using deep neural networks with ReLU activation function“. In: *The Annals of Statistics* 48.4 (2020), S. 1875–1897.

[16] Sho Sonoda und Noboru Murata. „Neural network with unbounded activation functions is universal approximator“. In: *Applied and Computational Harmonic Analysis* 43.2 (2017), S. 233–268.

[17] Paul J Werbos. „Generalization of backpropagation with application to a recurrent gas market model“. In: *Neural networks* 1.4 (1988), S. 339–356.

# Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und ohne fremde Hilfe verfasst habe. Dazu habe ich keine außer den von mir angegebenen Hilfsmitteln und Quellen verwendet und die den benutzten Werken inhaltlich und wörtlich entnommenen Stellen habe ich als solche kenntlich gemacht.

Rostock, den 12.08.2021

Vorname Nachname

# A. Anhang

## A.1. Listings

Listing A.1: C Code - direkt eingefügt

```c
#include <stdio.h>
#define N 10
/* Block
 * comment */

int main()
{
    int i;

    // Line comment.
    puts("Hello world!");

    for (i = 0; i < N; i++)
    {
        puts("LaTeX is also great for programmers!");
    }

    return 0;
}
```

Listing A.2: Java Code - über externe Datei eingefügt

```java
public class HelloWorld {
  public String SayHello() {
    return "Hello World!";
  }
}
```

## A.2. Biber

## BibLaTeX mit Biber

### BibLaTeX

- Formatierungen von Zitaten und Literaturverzeichnis mit LaTeX-Befehlen
- biblatex unterstützt:
  - unterteilte Bibliographien (nach Kapitel, Überschrift, Typ, Schlüsselwort)
  - mehrere Bibliographien in einem Dokument
  - stellt mehrere Zitierstile zur Auswahl bereit
  - ersetzt folgende Einzelpakete: babelbib, bibtopic, bibunits, chapterbib, cite, inlinebib, mlbib, multibib, splitbib
- Kompatibilitätsmodus zu natbib und mcite/mciteplus
- FAQ zu `biblatex`
  `http://projekte.dante.de/DanteFAQ/LiteraturverzeichnisMitBiblatex`

### Biber

- `biber` ist ein backend bibliography processor für `biblatex`
- `biber` ist `bibtex`-Ersatz speziell für `biblatex`
- Vorteile:
  - löst alle `bibtex`-Probleme (richtige Sortierung da Unicodeunterstützung, Speicherbedarf, Kodierungen etc.)
  - `http://www.ctan.org/pkg/translation-biblatex-de`, S. 47

### Einbinden von Biber in Editoren

- **TexWorks** in aktueller Version bereits enthalten
- **TeXnicCenter**
  über `Ausgabe\Ausgabeprofile definieren...` im genutzten Profil, z. B. `Latex -> PDF`
  `C:\Program Files\MiKTeX 2.9\miktex\bin\x64\bibtex.exe` durch
  `C:\Program Files\MiKTeX 2.9\miktex\bin\x64\biber.exe` ersetzen

### Ablauf

1. pdflatex foo.tex
2. biber foo.bcf
3. pdflatex foo.tex
4. pdflatex foo.tex

In **TexWorks** nacheinander ausführen (evtl. Anzeige per Hand aktualisieren). In **TeXnicCenter** werden 1. und 2. zusammen ausgeführt. Dann sind noch zwei Durchläufe (3. und 4. erforderlich).

### Erläuterungen zum Ablauf

- in foo.tex muss `biblatex` mit `backend=biber` geladen sein, damit foo.bcf geschrieben wird
- `*.bcf` steht für `biber control file` und enthält Anweisungen
  (welche `bib`-Datei, welche Sortierung usw.)

### Literaturverwaltungsprogramm Citavi

Die Universität Rostock hat eine Campuslizenz Citavi erworben. Mit Citavi verwalten Sie Ihre Literatur, recherchieren in Fachdatenbanken und Bibliothekskatalogen, arbeiten Literatur inhaltlich auf, sammeln Zitate, organisieren Wissen, konzipieren Texte, planen Aufgaben und erstellen automatisch Literaturverzeichnisse in unterschiedlichen Zitationsstilen.

Durch die Campuslizenz haben alle Studierenden und Lehrenden unserer Hochschule die Möglichkeit, dieses leistungsfähige Programm kostenlos zu nutzen.

Weitere Details findet man unter:
https://www.itmz.uni-rostock.de/anwendungen/software/rahmenvertraege/citavi/

Erzeugung einer *.bib-Datei mit Citavi:

- Datei / Exportieren
- auswählen was exportiert werden soll
- beim ersten Mal Exportfilter hinzufügen: BibLatex (auswählen)
- Dateinamen angeben und Exportvorlage bei Bedarf speichern
- fertig

### Beispiel einer *.tex-Datei mit Nutzung der Literaturdatenbank test1.bib

```
\documentclass[parskip=half]{scrartcl}
\usepackage[utf8]{inputenc} %select encoding
\usepackage[T1]{fontenc} % T1 Schrift Encoding
\usepackage{lmodern} % Schriftfamilie lmodern
\usepackage[ngerman]{babel}% dt. Sprache
\usepackage[babel, german=quotes]{csquotes} % einfache Handhabung von quotations

\usepackage[backend=biber]{biblatex} %biblatex mit biber laden
\ExecuteBibliographyOptions{
        sorting=nyt, %Sortierung Autor, Titel, Jahr
        bibwarn=true, %Probleme mit den Daten, die Backend betreffen anzeigen
        isbn=false, %keine isbn anzeigen
        url=false  %keine url anzeigen
        }
\addbibresource{test1.bib} %Bibliographiedateien laden

\begin{document}
TEXT mit Beispielen, s.~\cite{Mittelbach.2013}

\printbibliography %hier Bibliographie ausgeben lassen
\end{document}
```

### Beispiel einer Literaturdatenbank test1.bib

```
% This file was created with Citavi 6.3.0.0

@book{Mittelbach.2013,
 author = {Mittelbach, Frank and Goossens, Michel and Braams, Johannes},
 year = {2013},
 title = {The LATEX companion},
 edition = {2. ed., 12. print},
 publisher = {Addison-Wesley},
 isbn = {978-0201362992},
 language = {eng},
 location = {Boston, Mass.},
 series = {Addison-Wesley series on tools and techniques for computer typesetting},
 abstract = {},
 pagetotal = {1090}
}
```