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 Universität Rostock Fakultät
Gutachter:	Name des zweiten Gutachters Universität Musterstadt Fakultät

Masterarbeit am Institut für Mathematik
Mathematisch-Naturwissenschaftliche Fakultät

Zusammenfassung

Platz für eine kurze Zusammenfassung.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

List of Algorithms

Verzeichnis der Listings

Abkürzungsverzeichnis

RNN Rekurrentes Neuronales Netz 20

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)

Definition 1. *Eine Matrix $X \in [0,1]^{h \times b}$ heißt (Grauwert)-Bild mit der Höhe h und Breite b. Mit $X_{i,j}$ wird der Grauwert des Pixels $p = (i, j)$ bezeichnet.*

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 Kapitel werden Künstliche Neuronale Netze[8], kurz KNN, als Forschungsgegenstand der Informatik eingeführt und deren mathematische 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, zu erkennen[24, 25, 36]. Sie eignen sich daher für viele typische Aufgaben des maschinellen Lernens, beispielsweise für die Klassifikation digitalisierter Objekte.

Im ersten Abschnitt wird das Perzeptron[28] als Grundeinheit eines neuronalen Netzes eingeführt. Im folgenden Abschnitt wird das Konzept der Multi-Layer-Perzeptronen[37] 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. Weiter wird das Training neuronaler Netze hinsichtlich der Klassifikationsaufgabe im Abschnitt 3.3 erläutert und schließlich eine kurze Zusammenfassung 3.4 gegeben.

3.1. Das Perzeptron

Zunächst wird das *Perzeptron* ähnlich wie in Minsky [22] als fundamentaler Baustein eines neuronalen Netzes eingeführt. Das Perzeptron wird oft als Basis moderner KNN angeführt und kann mithilfe des Perzeptron-Lernalgorithmus[28] trainiert werden, um das Problem der linearen Trennbarkeit **??**von Punktmengen zu lösen.

Definition 2 (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 θ Schwellwert oder auch Bias genannt.

Die Funktionsweise eines Perzeptrons ist in Abbildung 3.1 dargestellt. Bei der Wahl

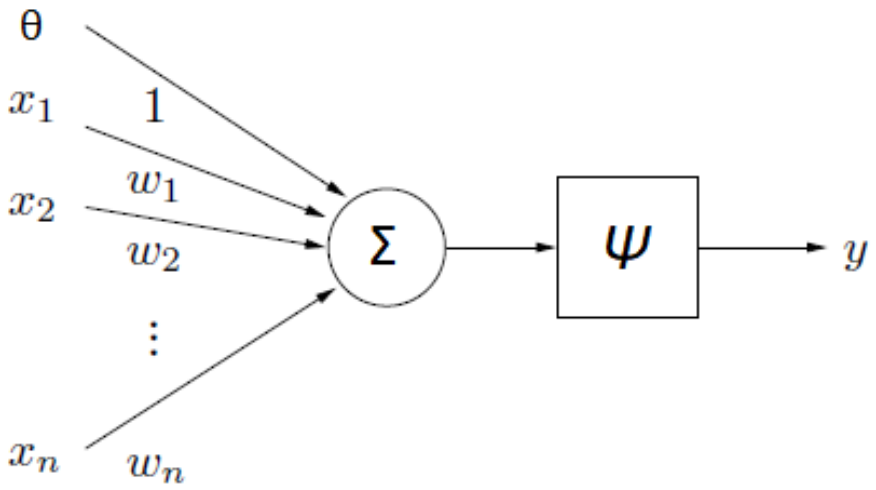


Abbildung 3.1.: Arbeitsweise eines Perzeptrons mit entsprechender Notation aus Definition 2.

der Funktion ψ gibt es mehrere Möglichkeiten. Wird wie in Minsky[22] 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-Jahren Gegenstand der Forschung[5, 4, 30]. Zunächst sind einige Definition notwendig, um eine lesbare Notation des MLPs zu geben.

Definition 3 (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 4 (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[26], welche jedoch in dieser Arbeit nicht weiter von Interesse sind. Typische Aktivierungsfunktionen, weclhe heutzutage 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 ψ eine Aktivierungsfunktion, so wird für $x \in \mathbb{R}^n$ mit

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

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

Bei Klassifikationsproblemen wird oft die *Softmax*-Funktion[9] genutzt, welche die gesamte Eingabe berücksichtigt. Im Abschnitt **??** wird erläutert, warum sich in diesem Fall die Softmax-Funktion eignet.

Definition 5 (Softmax-Funktion)**.** *Für* $x \in \mathbb{R}^n$ *wird die Funktion* $\psi : \mathbb{R}^n \to (0,1]^n$ *mit*

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

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 existiert, 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 $\psi(x) = \frac{1}{1+e^{-x}}$ 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 $\tanh(x) = \frac{2}{e^{2x}+1}$ genutzt und das Differenzieren mittels Quotientenregel liefert die Aussage. □

Ähnlich der Definition des Perzeptrons 2 wird nun eine Schicht als Verknüpfung von Übertragungsfunktion und Aktivierungsfunktion definiert.

Definition 6 (Neuronenschicht)**.** *Ist $\Psi^{W,b}$ eine Übertragungsfunktion mit den Parametern $W \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^m$ und ψ 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 2. 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 7 (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. Das MLP wird auch Feed-Forward-Netz (FFN) genannt und die Funktionsauswertung $\Lambda_l(x)$ für eine Eingabe x wird mit Vorwärtsrechnung, engl. forward propagation, bezeichnet.

Algorithm 1 Vorwärtsrechnung

Require: MLP Λ_l, Eingabe $x_0 \in \mathbb{R}^n$
Ensure: $y = \Lambda_l(x) \in \mathbb{R}^m$
  $x = x_0$
  for $i = 1, \ldots l$ **do**
    $u = W^{(i)\,T} x + b^{(i)}$
    $x = \psi_i(u)$
  end for
  $y = x$

Das MLP-Modell wird im weiteren Verlauf dieser Arbeit repräsentativ als Künstliches Neuronales Netz bezeichnet. Die Begriffe MLP und FFN sind austauschbar. Die Funktionsauswertung eines FNN wird im Algorithmus Vorwärtsrechnung 1 festgehalten. Das zuvor angesprochene XOR-Problem kann nun beispielsweise mithilfe eines KNN bestehend aus zwei Schichten gelöst werden[13]. Es lassen sich zwischen Modell- und Hyperparameter von KNN unterscheiden.

Definition 8 (Hyper- und Modellparameter)**.** *Sei für $l \in \mathbb{N}$ ein KNN Λ_l gegeben. Dann werden die Eingabe- und Ausgabedimension s_0, s_l, die Anzahl l der (verdeckten) Schichten sowie die verwendeten Aktivierungsfunktion ψ_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 dies geschieht, wird im folgenden Abschnitt **??** 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[13]. 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 als *task* T im Mittelpunkt stehen. Weiter werden KNNs als Modellschätzer aus der Wahrscheinlichkeitstheorie interpretiert und fundamentale Aussagen wie das *Universal-Approximation-Theroem*[17] 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 9. *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.*

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 a 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 $x \in D$ 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 ψ_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[16] 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[32, 21].

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.[33]. □

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* ψ_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 9 digitalisierte Objekte als Vektoren $x \in \mathbb{R}^n$ 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, also die *perfomance* P, bezüglich des Klassifikationsproblems messbar zu machen, werden Fehlerfunktionen eingeführt. Mit Trainingsdaten als *experience* E und dem Gradientenverfahren[23] sollen optimale Parameter $\mathcal{W}$ gefunden werden, sodass die gewählte Fehlerfunktion minimiert wird. Im folgenden steht ein MLP $\Lambda(\,\cdot\,; \mathcal{W}) : D \to [0,1]^m$ mit der Softmax-Funktion als Aktivierungsfunktion im Mittlepunkt, welches als parametrisiertes Modell f_{Modell} wie in 3.3 genutzt wird. Die Trainingsdaten werden in Trainingsmengen und Testmengen aufgeteilt.

Definition 10 (Trainingsmenge, Testmenge)**.** *Sei p_{daten} eine Datenverteilung auf der Ergebnismenge $\Omega = D \times \mathcal{C}$. Dann heißen für $n_{train}, n_{test} \in \mathbb{N}$ die Mengen*

$$\mathcal{T} := \left\{(x^{(i)}, c^{(i)}) \mid i \in [n_{train}]\right\} \subset \Omega$$
$$\mathcal{T}' := \left\{(x^{(i)}, c^{(i)}) \mid i \in [n_{test}]\right\} \subset \Omega$$

Traingsmenge $\mathcal{T}$ und Testmenge $\mathcal{T}'$, jeweils bestehend aus Datenpaaren, welche unabhängig durch p_{Daten} generiert wurde. Oft werden die Mengen disjunkt gewählt. Die Menge $\mathcal{T}$ wird zum Trainieren und die Menge $\mathcal{T}'$ zur Validierung des Modells P_{Modell} bezüglich P_{Daten} wie in 3.1 genutzt.

Die Approximationsgüte des Modells f_{Modell} wird als Likelihood gegeben einer Trainingsmenge $\mathcal{T}$ gemessen und lässt sich als

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

wie in Bishop[2] berechnen. Für eine Trainingsmenge $\mathcal{T}$ soll das Produkt über alle Wahrscheinlichkeiten der korrekten Klassenzugehörigkeiten c gegeben der Eingaben x maximiert werden. Dieser Ansatz wird *Maximum Likelihood-Methode*[31] genannt und eine Parameterwahl ist durch eine Lösung des Optimierungsproblems

$$\prod_{(x,c)\in\mathcal{T}} p_{Modell}(c \mid x; \mathcal{W}) \to \max \tag{3.6}$$

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

Für ein Trainingspaar $(x, c) \in \mathcal{T}$ bezeichne $t(x, c) \in \mathbb{R}^m$ den Zielvektor der Klasse c mit sogenannter (1 aus m)-Kodierung. Die Komponenten des Zielvektors sind

$$t_k(x, c) := \begin{cases} 1 & \text{, wenn } k = c \\ 0 & \text{, sonst} \end{cases}, \quad \forall k \in [m].$$

Mit dieser Bezeichnung lässt sich das Optimierungsproblem 3.6 als Minimierungproblem mithilfe der *negative log likelihood* schreiben.

Definition 11 (negative log likelihood)**.** *Seien die Mengen D und $\mathcal{C} = \{c_1, \ldots, c_m\}$ mit einer dazugehörigen Trainingsmenge $\mathcal{T}$ sowie entsprechende Zielvektoren gegeben. Weiter seien die a posterior Wahrscheinlichkeiten $p_{Modell}(c \mid x; \mathcal{W})$ wie in Gleichung 3.2 gegeben. Die negative log likelihood ist als Funktion*

$$L_{NNL}(\mathcal{T}, \mathcal{W}) := -\sum_{(x,c)\in\mathcal{T}} \sum_{i=1}^{m} t_i(x, c) \log\left(p_{Modell}(c_i \mid x; \mathcal{W})\right) \tag{3.7}$$

definiert.

Das Minimieren der negative log likelihood ist äquivalent zur Maximierung der Like-

lihood aus 3.5, denn es gilt

$$\log\left(\prod_{(x,c)\in\mathcal{T}} p_{Modell}(c \mid x;\mathcal{W})\right) = \sum_{(x,c)\in\mathcal{T}} \log\left(p_{Modell}(c \mid x;\mathcal{W})\right)$$

und der natürliche Logarithmus ist monoton steigend. Wird zusätzlich angenommen, dass die a posterior Verteilung $p_{Daten}(c \mid x)$ einer Normalverteilung mit konstanter Varianz entspricht, so ist das Maximieren von 3.5 äquivalent zur Minimierung der mittleren quadratischen Abweichung

$$L_{MSE}(\mathcal{T},\mathcal{W}) := \frac{1}{2}\sum_{(x,c)\in\mathcal{T}} ||\hat{c} - t(x,c)||_2^2,$$

wobei $\hat{c} = f_{Modell}(x)$ und $t(x,c)$ der Zielvektor des Datenpaars (x,c) ist, siehe Goodfellow[13]. Das Problem 3.6 wird nun allgmemein mit Fehlerfunktionen definiert.

Definition 12 (Fehlerfunktion)**.** *Seien $\mathcal{T}$ eine Trainingsmenge und $\mathcal{W}$ Modellparameter eines KNN. Mithilfe des Gradientenverfahrens soll das Problem*

$$\mathcal{E}(\mathcal{T},\mathcal{W}) \to \min \tag{3.8}$$

gelöst werden. Dabei wird $\mathcal{E}$ Fehlerfunktion gennant.

In dieser Arbeit wird $\mathcal{E}$ immer als stückweise stetig differenzierbare Funktion gewählt, damit das Gradientenverfahren angewendet werden kann. Sowohl die negative log likelihood L_{NNL} als auch die mittlere quadratische Abweichung L_{MSE} sind als Fehlerfunktion geeignet. Die Optimierung der Parameter geschieht iterativ und besteht jeweils aus zwei Schritten. Zuerst wird eine Abstiegsrichtung

$$\Delta_n := \nabla_{\mathcal{W}}\mathcal{E}(\mathcal{T},\mathcal{W}) \tag{3.9}$$

berechnet und dann die Parameter

$$\mathcal{W}_{n+1} := \mathcal{W}_n - \lambda\Delta_n \tag{3.10}$$

aktualisiert. Es werden also Gradienten der Fehlerfunktion bezüglich der Gewichtsmatrizen und Biasvektoren ermittelt und anschließend werden jene Parameter mit einer wählbaren Lernrate $\lambda \in \mathbb{R}$ angepasst. In 3.9 wird der Gradient über alle Trainingspaare berechnet. Diese Variante nennt sich *Offline-Version* des Gradientenverfahrens und ist besonders für große Trainingsmengen ineffizient. Die *Online-Version* berechnet den Gradienten lediglich für ein Trainingspaar und passt die Parameter direkt an. In dieser Arbeit wird ein Kompromiss aus beiden Verfahren verwendet und zwar das *Mini-Batch-Verfahren*, siehe Algorithmus 2, bei dem die Gradienten über kleine Teilmengen $\mathbb{T} \subset \mathcal{T}$ der Trainingsmenge berechnet werden.

Als Abbruchbedingung im Mini-Batch-Verfahren kann eine fest gewählte Anzahl von Updateoperationen der Parameter gewählt werden. Andere Abbruchbedingungen können mit der Norm der Abstiegsrichtungen[2] oder abhängig vom Traingsfehler formuliert

Algorithm 2 Mini-Batch-Verfahren, vgl. [**gruening**]

Require: Trainingsmenge $\mathcal{T}$, Modellparameter $\mathcal{W}_0$, Fehlerfunktion $\mathcal{E}$, Batch-Größe n
Ensure: optimierte Modellparamter $\mathcal{W}$
  $\mathcal{W} = \mathcal{W}_0$
  $\mathbb{T} = \mathcal{T}$
  Setze Lernrate λ ▷ Wahlmöglichkeiten, siehe Text
  while Abbruchbedingung nicht erfüllt **do** ▷ Abbruchbedingung, siehe Text
    if $|\mathbb{T}| < n$ **then**
      $\mathbb{T} = \mathcal{T}$
    end if
    $\mathbb{T}_n$ = zufällig gewählte n Datenpaare von $\mathbb{T}$
    $\mathcal{W} = \mathcal{W} - \lambda \nabla_{\mathcal{W}} \mathcal{E}(\mathbb{T}_n, \mathcal{W})$
    $\mathbb{T} = \mathbb{T} \setminus \mathbb{T}_n$
  end while

werden. Bei der Wahl der Lernrate werden heutzutage werden oft adaptive Verfahren genutzt, welche vorangegangene Gradienten berücksichtigen und die Lernrate so anpassen. Bekannte Verfahren sind *Nesterov accelerated gradient*[34], *AdaGrad*[10], *RMSProp*[35] sowie *Adam*[18]. So sollen Probleme wie des *vanishing gradients* oder des *exploding gradients* vermieden werden[15]. Für eine tiefere Analyse des Gradientenverfahrens und dessen Varianten sei auf die jeweilgen Arbeiten beziehungsweise Ruder[29] verwiesen. Darüber hinaus gibt es andere Techniken wie die Regularisierung, um Probleme wie Überanpassung, engl.*overfitting*, zu entgegnen. Für die Problemstellung in dieser Arbeit genügt es, das Mini-Batch-Verfahren mit konstanter Lernrate zu nutzen.

Die Abstiegsrichtungen werden werden mithilfe der mehrdimensionalen Kettenregel in einer Backpropagationsphase, etabliert von Rumelhart et. al. [30], berechnet.

Satz 2 (Mehrdimensionale Kettenregel)**.** *Ist* $f = f(x_1(y_1, \dots, y_m), \dots, x_n(y_1, \dots, y_m))$ *und sind alle beteiligten Funktionen stetig differenzierbar, so ergeben sich die partielle Ableitungen mittels Kettenregel zu*

$$\frac{\partial f}{\partial y_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial x_i} \frac{\partial x_j}{\partial y_i}.$$

Beweis. siehe [12] □

Im Bereich des tiefen Lernens zählt die effiziente Berechnung dieser Gradienten zu den schwersten Aufgaben. Eine ausführliche Erklärung der Backpropagationsphase für klassische FFN ist beispielsweise in Rojas[27] zu finden.

3.4. Zusammenfassung

In diesem Kapitel wurden KNN als Feed-Forward-Netze eingeführt. Es stellt sich heraus, dass diese Netze als probabilistische Modelle genutzt werden können, um Klassifikationsaufgaben hinreichend gut zu lösen. Dabei ist es wichtig die Hyper- und Mo-

dellparameter je nach Anwendung und Leistungsmaß optimal zu wählen. Dazu werden Trainingsdaten genutzt, um während eines Lernprozesses eine Fehlerfunktion zu Minimieren. Eine Lösung von 3.8 kann wegen der Komplexität der Fehlerfuntion $\mathcal{E}$ bzw. der großen Menge von Parametern $\mathcal{W}$ selten direkt angegeben werden[3]. Daher wird das Mini-Batch-Verfahren als iterativer Ansatz zur numerischen Mnimierung der Fehlerfunktion genutzt. Dabei sind partielle Ableitungen der Fehlerfunktion bezüglich der Parameter nötig, welche in der Backpropagationsphase mithilfe der mehrdimensionalen Kettenregel berechnet werden können.

Bei der Analyse von Zeitreihen oder Bildern eignen sich abgewandelte Architekturen wie gefaltete neuronale Netze (CNN), engl. *convolutional neural networks*, welche im folgenden Kapitel 4 näher erläutert werden. Diese Art neuronaler Netze wird im weiteren Verlauf dieser Arbeit im Fokus stehen.

4. Gefaltete neuronale Netze

Feed-Forward-Netze gelten als leistungsstarke maschinelle Lernmethoden, da sie so trainiert werden können, um beliebige komplexe Funktionen abhängig von einer vektorwertigen Eingabe zu approximieren. Ist die Dimension der Eingabeschicht jedoch zu groß, treten bei klassischen FFN Probleme hinsichtlich der Paramteranzahl auf. Im weiteren Verlauf dieser Arbeit sollen digitalisierte Bilder klassifiziert werden. Wird ein MLP mit 100 Ausgabeneuronen genutzt und jeder Pixel eines Bildes mit den Abmessungen 1000×1000 als Merkmal genutzt, so ergeben sich bereits $10^8 + 100$ freie Parameter. Stehen nur relativ wenige Traingsdaten zur Verfügung, ist die Struktur des FFN zu komplex und dies kann zur Überanpassung führen[6, 1]. Die Parameteranzahl muss also deutlich reduziert werden. Konzepte wie *Parameter Sharing* und spärliche Konnektivität, engl. *sparse connectivity* erlauben diese Reduktion, vgl. Goodfellow[13] und werden in den folgenden Abschnitten erläutert.

Ein weiterer Nachteil des FFN ergibt sich dadurch, dass Korrelationen von benachbarten Eingabeneuronen, z.B. Bilsegmente wie Kanten oder Ecken, nicht miteinbezogen werden. Es muss also eine Modell entwickelt werden, welches diese lokalen Muster extrahiert und sie miteinander verknüpft. Das Modell sollte zudem äquivariant gegenüber Translationen sein.

In diesem Kapitel wird erläutert, wie gefaltete neuronale Netze die erwähnten Nachteile von FFN umgehen. CNN sind in der Lage, lokale Muster zu erkennen, sind äquivariant gegenüber Translationen und realisieren Konzepte wie Parameter Sharing, um die Anzahl der freien Parameter drastisch zu reduzieren. So gelingt es, besonders bei Aufgaben der Computergrafik[19, 20, 7] die Generalisierungsrate gegenüber klassichen FFN zu erhöhen.

Gefaltete neuronale Netze unterscheiden sich von FFN bei der Berechnung der Übertragungsfunktion. Dazu wird die gefaltete Übertragungsfunktion definiert, welche das Konzept der diskreten Faltung nutzt. Im folgenden Abschnitt 4.1wird zunächst die Faltung als mathematische Operation eingeführt und deren Zusammenhang zur Fourier-Transformation[38] erläutert. Anschließend wird im Abschnitt **??** das CNN-Modell definiert.

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 13 (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. Für $f, g \in L^1(\mathbb{R}^n)$ ist dies der Fall.

Für die Faltung gelten einige Rechenregeln.

Lemma 2. *Seien $f, g, h \in L^1(\mathbb{R}^n)$ und $a \in \mathbb{C}$. Dann gelten*

(i)	$f * g = g * f$	*(Kommutativität)*
(ii)	$f * (g * h) = (f * g) * h$	*(Assoziativität)*
(iii)	$f * (g + h) = (f + g) * h$	*(Distributivität)*
(iv)	$a(f * g) = (af) * g = f * (ag)$	*(Assoziativität mit skalarer Multiplikation)*

Beweis. Eine Beweis dieser Rechenregeln kann in Werner[38] nachgelsen werden. □

In der digitalen Signal- und Bildverarbeitung werden meist diskrete Funktionen analysiert und daher die diskrete Faltung genutzt, bei der statt der Intgration eine Summation auftaucht. Die Regeln aus Lemma 2 gelten analog.

Definition 14 (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.

Ist für $f, g : D \to \mathbb{C}$ der Definitionsbereich D endlich, so können die Funktionen als zeitdiskrete Signale $f = (f_0, \ldots, f_{n-1})^T \in \mathbb{C}^n$ und $g = (g_0, \ldots, g_{n-1}) \in \mathbb{C}^n$ aufgefasst werden. Durch das Fortsetzen mit Nullen besitzen die Vektoren f und g die gleiche Länge. In diesem Fall kann die Faltung als Matrix-Vektor-Produkt mit einer zyklischen Matrix ausgedrückt werden.

Definition 15 (Zyklische Matrix, vgl. Gray[14])**.** *Eine quadratische Matrix heißt zyklisch im Vektor $a = (a_0, \ldots, a_{n-1})^T \in \mathbb{R}^n$, wenn sie die Gesatlt*

$$\mathrm{zyk}(a) := \begin{pmatrix} a_0 & a_{n-1} & a_{n-2} & \ldots & a_1 \\ a_1 & a_0 & a_{n-1} & \ldots & a_2 \\ a_2 & a_1 & a_0 & \ldots & a_3 \\ & \ddots & \ddots & \ddots & \\ a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0 \end{pmatrix}$$

besitzt.

Bemerkung 2. Für ein zeitdiskrete Signal $f = (f_0, \ldots, f_{n-1})^T \in \mathbb{C}^n$ sei $F = \text{zyk}(f)$ die zyklische Matrix im Vektor f. Sei weiter $g = (g_0, \ldots, g_{n-1}) \in \mathbb{C}^n$. Dann lässt sich mit

$$(Fg)_k = \sum_{j=0}^{n-1} f_{k-j} g_j, \quad k = 0, \ldots, n-1$$

die diskrete Faltung von f und g darstellen. Dabei werden Indizies außerhalb von $0, \ldots, n-1$ zyklisch durch Modulo-Rechnung (mod n) in den gültigen Indexbereich abgebildet.

In Hinblick auf die Klassifikation von digitalisierten Bildern, dargestellt als zweidimensionale Signale, wird die zweidimensionale Faltung mit sogenannten quadratischen Kernen $K \in \mathbb{R}^{k \times k}$ mit ungeradem $k \in \mathbb{N}$ definiert.

Definition 16 (Zweidimensionale Faltung, vgl. [**gruening**])**.** *Für gegebene Matrizen $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ sei $h = \lfloor k/2 \rfloor$. Die zweidimensionale Faltung $Y = X * K \in \mathbb{R}^{h \times w}$ ist als*

$$(Y)_{i,j} := \sum_{l=-h}^{h} \sum_{m=-h}^{h} X_{i+l,j+m} K_{l+h_l+1,m+w_l+1} \quad \forall i \in [h], j \in [b] \tag{4.1}$$

mit $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [b]$ definiert. In der Literatur wird das Aüffüllen mit Nullen am Rand von X mit zero padding bezeichnet. In dieser Definition besitzt das Ergebnis Y der Faltung die gleichen Abmessungen wie X.

Bei gefalteten neuronalen Netzen wird oft eine Reduktion der Dimensionen angestrebt. Dafür werden natürliche Zahlen als Schrittweiten, engl. *strides*, genutzt.

Bemerkung 3. Für Schrittweiten $s_h, s_b \in \mathbb{N}$ ergibt sich die reduzierte zweidimensionale Faltung $Y = X * K$ zu

$$(Y)_{i,j} := \sum_{l=-h}^{h} \sum_{m=-h}^{h} X_{i \cdot s_h + l, j \cdot s_b + m} K_{l+h_l+1,m+w_l+1} \quad \forall i \in [\lceil h/s_h \rceil], j \in [\lceil b/s_b \rceil].$$

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

4.2. CNN Architektur

Beim maschinellen Lernen sind Eingabedaten oft 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, welche als Raumachsen bezeichnet werden. Hinzu kommen Kanalachsen als weitere Verfeinerung der Daten, zum Beispiel besitzen Grauwert-Bilder einen Farbkanal, während RGB-Farbbilder drei Kanäle der Farben rot, grün und blau besitzen. Dementsprechend werden Grauwert-Bilder wie in Definition 1 nun als dreidimensionale Arrays $X \in [0,1]^{h \times b \times 1}$ dargestellt. Dies erlaubt die Definition der gefalteten Übertragungsfunktion, wie in Gruening[**gruening**]

Definition 17 (Gefaltete Übertragungsfunktion)**.** *Sei ein vierdimensionales Array* $K \in \mathbb{R}^{k \times k \times z_{in} \times z_{out}}$ *und ein Biasvektor* $b \in \mathbb{R}^{z_{out}}$ *gegeben. Die Funktion*

$$\Psi_{conv}^{K,b} : \mathbb{R}^{\cdot \times \cdot \times z_{in}} \to \mathbb{R}^{\cdot \times \cdot \times z_{out}}$$

mit

$$\Psi_{conv}^{K,b}(X)_{:,:,l} := \sum_{p=1}^{z_{in}} K_{:,:,p,l} * X_{:,:,p} + b_l \; \forall l \in [z_{out}]$$

wird gefaltete Übertragungsfunktion bezeichnet. Mit $*$ *ist die zweidimensionale Faltung wie in Definition 16 und mit* $\cdot$ *beliebige Raumachsen gemeint.*

Bemerkung 4. Ist $\psi : \mathbb{R} \to \mathbb{R}$ eine Aktivierungsfunktion wie in Definition 4, so wird für $X \in \mathbb{R}^{\cdot \times \cdot \times z}$ mit

$$\psi(X)_{i,j,:} := (\psi(X_{i,j,1}), \ldots, \psi(X_{i,j,z}))^T \in \mathbb{R}^z \;\; \forall i \in [{}_1X], j \in [{}_2X]$$

der Vektor bezeichnet, welcher sich durch die elementweise Auswertung der Aktivierungsfunktion ψ ergibt.

Ähnlich der Definition 6 wird nun eine Faltungsschicht als Verknüpfung von gefalteter Übertragungsfunktion und Aktivierungsfunktion definiert.

Definition 18 (Faltungsschicht)**.** *Ist* $\Psi_{conv}^{K,b}$ *eine gefaltete Übertragungsfunktion und* ψ *eine Aktivierungsfunktion, so wird das Paar* $(\Psi_{conv}^{K,b}, \psi)$ *als Faltungsschicht* $\mathcal{S}_{conv}$ *bezeichnet. Für eine Eingabe* $X \in \mathbb{R}^{\cdot \times \cdot \times z_{in}}$ *ist die Ausgabe* $Y \in \mathbb{R}^{\cdot \times \cdot \times z_{out}}$ *der Schicht* $\mathcal{S}_{conv}$ *durch*

$$Y = \psi \circ \Psi_{conv}^{K,b}(X) = \psi\left(\Psi_{conv}^{K,b}(X)\right)$$

gegeben. Die Matrizen $Y_{:,:,p}$ *werden für* $1 \leq p \leq z_{out}$ *Merkmalskarten genannt.*

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).

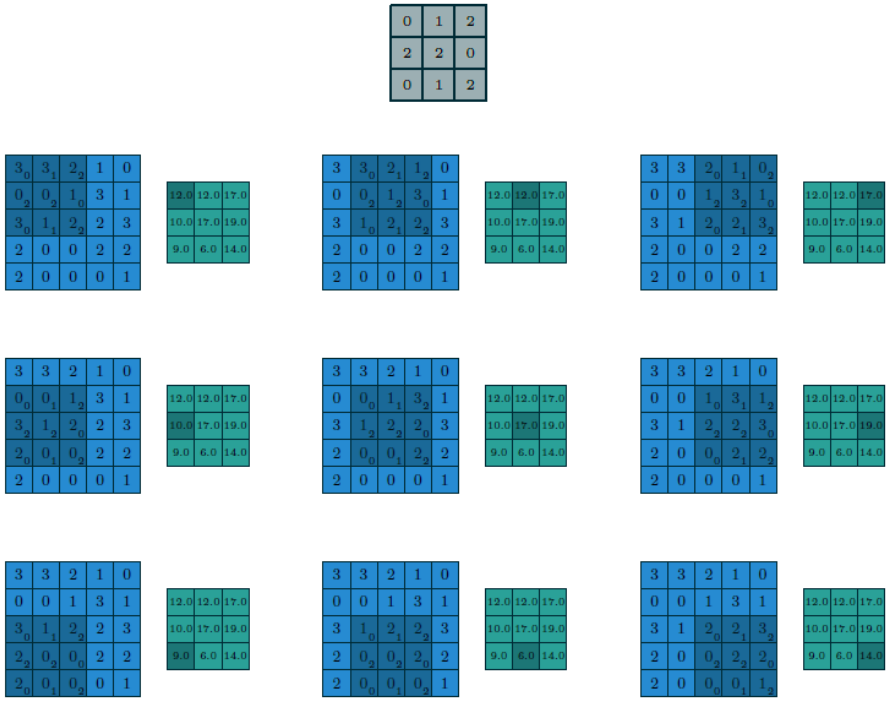


Abbildung 4.1.: Es wird die Merkmalskarte $S \in \mathbb{R}^{3\times3}$ mit den Parametern $h, w = 5, k_h = k_w = 3, s_h = s_w = 1$ und $p_h = p_w = 1$.

4.3. 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**]. Spärliche Konnektivität bedeutet, dass Neuronen auf einer Schicht $\int_{l+1}$ nur durch wenige Neuronen der Schicht $\mathcal{S}_l$ 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. Im Zusammenhang von CNNs bedeutet Parameter Sharing während der Faltungsoperation, dass nur eine bestimmte Menge von Parametern erlernt werden müssen Noch mehr erklären + Abbildung

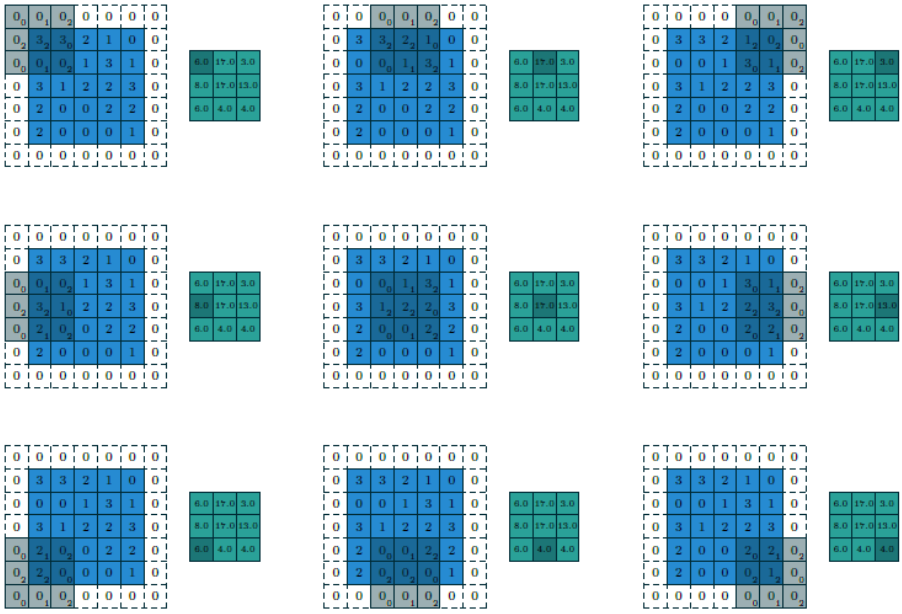


Abbildung 4.2.: Es wird die Merkmalskarte $S \in \mathbb{R}^{3\times3}$ mit den Parametern $h, w = 5, k_h = k_w = 3, s_h = s_w = 2$ und $p_h = p_w = 1$ berechnet.

5. Weiteres Kapitel

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

5.1. Umgebungen und Formeln

Definition 19. *... heißt Rekurrentes Neuronales Netz (RNN).*

Bemerkung 5. Bei jeder weiteren Verwendung der Abkürzung wird nur die Kurzform angezeigt: RNN.

Bemerkung 6. 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 3. *Ein kluger Satz ...*

Korollar 3. *Ein kluges Korollar ...*

Proposition 1. *Eine kluge Proposition ...*

Problem 1. *Ein schweres Problem ...*

Beispiel 1. Ein anschauliches Beispiel ...

Definition 20. *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.

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

[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

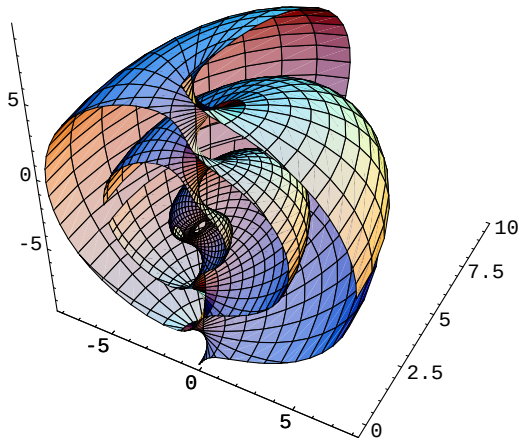
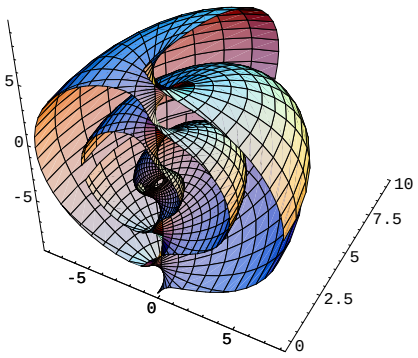
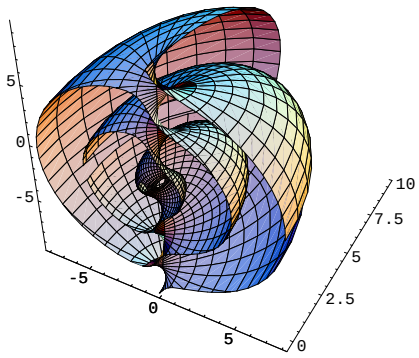


Abbildung 5.1.: Vektorgrafiken sind toll. Scrolle mal in mich rein!

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.

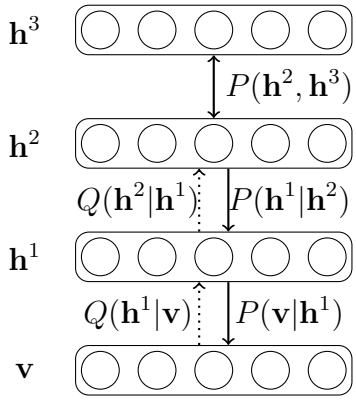


Abbildung 5.3.: Beispiel eines mit TikZ erzeugten Bildes

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!

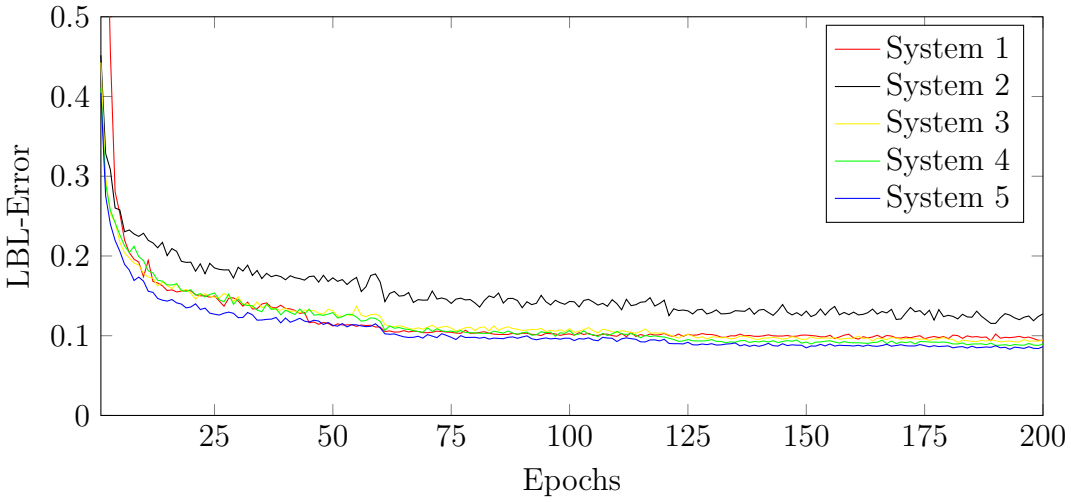


Abbildung 5.4.: Datenreihen mittels TikZ visualisiert

6. Ein letztes Kapitel

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

6.1. Weiteres Korollar

In diesem Abschnitt ...

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

Bemerkung 7. Eine vollständige ...

6.2. Pseudocode

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

Algorithm 3 An algorithm with caption

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow \frac{N}{2}$ ▷ This is a comment
    else if N is odd **then**
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

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 [11].
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] Imanol Bilbao und Javier Bilbao. „Overfitting problem and the over-training in the era of data: Particularly for Artificial Neural Networks“. In: *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*. 2017, S. 173–177. DOI: 10.1109/INTELCIS.2017.8260032.

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

[3] Avrim L Blum und Ronald L Rivest. „Training a 3-node neural network is NP-complete“. In: *Neural Networks* 5.1 (1992), S. 117–127.

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

[5] 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.

[6] Rich Caruana, Steve Lawrence und Lee Giles. „Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping“. In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. MIT Press, 2000, S. 381–387.

[7] Dan C. Ciresan, Ueli Meier und Jürgen Schmidhuber. „Multi-column deep neural networks for image classification“. In: *CVPR*. IEEE Computer Society, 2012, S. 3642–3649.

[8] Judith E Dayhoff. *Neural network architectures: an introduction*. Van Nostrand Reinhold Co., 1990.

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

[10] John Duchi, Elad Hazan und Yoram Singer. „Adaptive subgradient methods for online learning and stochastic optimization.“ In: *Journal of machine learning research* 12.7 (2011).

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

[12] Otto Forster. *Analysis 2: Differentialrechnung im IRn, gewöhnliche Differentialgleichungen*. Springer-Verlag, 2017.

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

[14] Robert M Gray u. a. „Toeplitz and circulant matrices: A review“. In: *Foundations and Trends® in Communications and Information Theory* 2.3 (2006), S. 155–239.

[15] Boris Hanin. „Which neural net architectures give rise to exploding and vanishing gradients?“ In: *Advances in neural information processing systems* 31 (2018).

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

[17] 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.

[18] Diederik P. Kingma und Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *CoRR* abs/1412.6980 (2015).

[19] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks“. In: *NIPS*. 2012, S. 1106–1114.

[20] Yann LeCun u. a. „Gradient-based learning applied to document recognition“. In: *Proc. IEEE* 86.11 (1998), S. 2278–2324.

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

[22] 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.

[23] Jorge Nocedal und Stephen J Wright. *Numerical optimization*. Springer, 1999.

[24] Abhijit S Pandya und Robert B Macy. *Pattern recognition with neural networks in C++*. CRC press, 1995.

[25] Yohhan Pao. „Adaptive pattern recognition and neural networks“. In: (1989).

[26] 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.

[27] Raul Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, 1996.

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

[29] Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *arXiv preprint arXiv:1609.04747* (2016).

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

[31] Ludger Rüschendorf. *Mathematische Statistik*. Bd. 62. Springer, 2014.

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

[33] 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.

[34] Ilya Sutskever u. a. „On the importance of initialization and momentum in deep learning“. In: *International conference on machine learning*. PMLR. 2013, S. 1139–1147.

[35] Tijmen Tieleman, Geoffrey Hinton u. a. „Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude“. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), S. 26–31.

[36] Ilona Urbaniak und Marcin Wolter. „Quality assessment of compressed and resized medical images based on pattern recognition using a convolutional neural network“. In: *Communications in Nonlinear Science and Numerical Simulation* 95 (2021), S. 105582.

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

[38] D. Werner. *Funktionalanalysis*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2011.

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}
}
```