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

Symbolverzeichnis

1. Einleitung

Assistenzsysteme stellen Informationen und Hilfestellungen bei bestimmten Produkten bereit, um deren Bedienung zu erleichtern. Im Allgemeinen dienen sie der Verbesserung beziehungsweise Erleichterung verschiedenster Situationen aus dem modernen Alltag. Heutzutage spielen Assistenzsysteme in fast allen Bereichen der Forschung und Industrie eine große Rolle und unterstützen Menschen bei ihren Tätigkeiten [62, 30, 45]. Um sinnvolle Hilfestellungen zu geben, ist die korrekte Erfassung der aktuellen Situation und der wahrscheinlichen Intentionen der zu assistierenden Benutzer notwendig. In diesem Zuge werden solche Systeme meist intelligent genannt, denn sie sind in der Lage, durch die Verarbeitung von Sensordaten bestimmte Szenarien und Aktivitäten zu erkennen, um so die optimalen Assisenzfunktionen zur Verfügung zu stellen. Darüber hinaus können diese Modelle genutzt werden, um Vorhersagen über zukünftige Situationen zu treffen. Sensoren, z.B. in Smartphones, Autos oder allgemeinen Multimediasystemen, liefern im Kontext des *Internet of Things*[65, 64]meist große Datenmengen, die geschickt verarbeitet werden müssen. Um Muster in diesen immer größer werdenden Datenmegen zu erkennen, können Verfahren und Algorithmen des *Maschinellen Lernens*[20], kurz ML, genutzt werden. Die Analyse solcher Assistenzsysteme ist daher Forschungsgegenstand in den Bereichen der Big Data Analytics[13] und Künstlichen Intelligenz.

Das PArADISE-Projekt[38] des Lerhstuhls für Datenbank- und Informationsysteme der Universität Rostock beschäftigt sich mit dem Designprozess von Assisenzsystemen mit dem Ziel, die Ersteller von assistiven Systemen zu unterstützen. In der Entwicklungsphase versuchen Datenwissenschaftler, Benutzeraktivitäten zu erkennen und vorherzusagen, indem sie Sensordaten von einer kleinen Anzahl von Testpersonen über einen kurzen Zeitraum sammeln. Diese Daten werden dann durch Expertenwissen mit Aktivitätsinformationen versehen, um anschließend Aktivitätsmodelle mit Hilfe von Maschinellen-Lern-Algorithmen zu lernen. Hier werden Verfahren des überwachten maschinellen Lernens genutzt, welche sich typischerweise in zwei Phasen einteilen lassen. In einer Trainingsphase werden mithilfe von annotierten Trainingsdaten Modelle zur Erkennung von Mustern abgeleitet. In der späteren Erkennungsphase wird dann das trainierte Modell genutzt, um erkannte Muster in einer Zieldatenmenge effizient ableiten zu können.

Motivation

Eines der schwierigsten Probleme stellt das Verwalten der riesigen Datenmengen, besonders in der Trainingsphase maschineller Lernverfahren, dar. Um in der Implementierung Perfomance-Probleme zu lösen, kann die verwendete Hardware verbessert, der Code optimiert oder Parallelisierungstechniken wie das berühmte *MapReduce*-Verfahren[8]

benutzt werden. Ein andere Möglichkeit besteht in der von Verwendung einer transparenten Datenbankunterstützung[39] für Big Data Analytics. In diesem Kontext werden Techniken aus relationalen Datenbanksystemen eingebunden, um ML-Tools sinnvoll zu unterstützen. Der Schlüssel liegt dabei in der Transformation von Maschinellen-Lern-Algorithmen in SQL-Datenbanksysteme. Gelingt diese Übersetzung, so können verschiedenste Resultate der relationalen Datenbankforschung vorteilhaft genutzt werden:

- Techniken der parallelen Datenbanksysteme ermöglichen es, SQL-Anfragen auf Rechnercluster zu verteilen, um so die Perfomance von ML-Algorithmen zu verbessern. Besonders interressant ist hier die Umsetzung von Operationen der linearen Algebra, unter anderem motiviert durch Test-of Time-Award-Winner Dan Suciu[63]. Die Transformation von Matrixopeartionen in parallele relationale Datenbanksysteme bei dichtbestzten bzw. dünnbesetzten Problemen wird beispielsweise in Marten et.al.[41] beleuchtet.
- Konzepte wie *Query Decomposition*[11] und *Answering Queries using Views*[1, 35] werden genutzt, um bestimmte Auswertungen von Daten näher an den Sensoren durchzuführen und damit *Privacy*[2] Aspekte der Benutzer zu berücksichtigen.
- *Data Provenance* [22, 9] kann genutzt werden, um herauszufinden, welche Daten für die Detektierung bzw. Vorhersage von Aktivitäten gebraucht werden. So wird unter anderem festgestellt, welche der vielen eingesetzten Sensoren für das Modell essentiell beziehungsweise uninteressant sind.

In der Arbeit von Marten et.al.[40] wird am Beispiel eines Meeting Szenarios ein Maschinelles Lernverfahren namens Hidden-Markov-Modelle untersucht. Es wird erläutert, wie die Erkennungsphase eines zuvor trainierten Modells datenbankgestützt in parallelen SQL-Datenbanksystemen realisiert werden kann. Als Ergebnis wird festgehalten, dass die datenbankgestützte Umsetzung von ML-Algorithmen, hier speziell bei Hidden-Markov-Modellen, in bestimmten Szenarien gute Skalierungseigenschaften hinsichtlich der Datenmenge besitzt und verschiedenste Resultate der relationalen Datenbankforschung vorteilhaft genutzt werden können. Dies motiviert die Analyse weiterer ML-Verfahren und deren Transformation in SQL.

Problemstellung

Diese Arbeit beschäftigt sich mit *Convolutional Neural Networks*, kurz CNN, als weitverbreitete ML-Methode und deren Realisierung in SQL-Datenbanksysteme unter Zuhilfenahme von Werkzeugen der linearen Algebra. Die fundamentale mathematische Operation innerhalb von CNN stellt die Faltung $f * g$ zweier Funktionen bzw. Signale f und g dar. Es liegt daher nahe, die Faltungsoperation von zeitdiskreten Signalen, in dieser Arbeit einfach Matrizen, datenbankgestützt umzusetzen.

Problem 1. *Sind zwei Matrizen $X \in \mathbb{R}^{m \times n}$ und $K \in \mathbb{R}^{k \times k}$ gegeben, so ist die Matrixfaltung, siehe Definition 28, in eine Folge von SQL-Anweisungen zu übersetzen. Der Speicher- und Zeitaufwand dieser Anweisungen ist in Abhängigkeit von den Dimensionen der beteiligten Matrizen zu untersuchen.*

CNN werden zur Lösung von Aufgaben der Computergrafik[29, 32, 12] erfolgreich genutzt. Im Kontext der Mustererkennung soll die Klassifikation von digitalisierten Bildern als Anwendungsbeispiel durch ein vorher traniertes CNN-Modell dienen. In dieser Arbeit werden dazu Grauwertbilder aus dem MNIST-Datensatz[33] genutzt.

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. In dieser Arbeit werden Matrizen als diskrete zweidimensionale Signale interpretiert.*

Der MNIST-Datensatz bietet 60.000 Trainingsbilder handgeschriebener Ziffern und 10.000 Testbilder, welche jeweils durch menschliches Wissen annotiert sind. Dieser Datensatz gilt als typischer Benchmark zur Klassifikation von Ziffern. Um dabei erfolgreich zu sein, muss ein CNN vorher mithilfe von Traininsdaten angelernt werden. Hier wird Backpropagation, etabliert von Rumelhart et. al.[51], als Lernalgorithmus basierend auf dem Gradientenverfahren genutzt.

Problem 2. *Die Trainingsphase neuronaler Netze und speziell gefalteter neuronale Netze gilt es konzeptionell darzustellen sowie am Beispiel des MNIST-Datensatzes für ein konkret gewähltes Modell möglichst perfomant umzusetzen.*

Schließlich soll die datenbankgestützte Erkennungsphase in einem trainierten gefalteten neuronalen Netz implementiert werden.

Problem 3. *Angenommen es wird ein traniertes CNN als Modell zur Klassifikation von Grauwertbildern aus dem MNIST-Datensatz genutzt. Dann ist die Vorwärtsrechnung, vgl. Definition 33, in eine Folge von SQL-Anfragen zu transformieren.*

Aufbau der Arbeit

Im Kapitel 2 wird zunächst das Maschinelle Lernen als mathematisches Problem eingeordnet und die konkrete Klassifikationsaufgabe im Kontext des überwachten Lernens definiert. Im Abschnitt 2.1 werden wichtige Begriffe zur linearen und nichtlinearen Trennbarkeit erläutert und ein erster Lernalgorithmus vorgestellt. Im Folgenden Abschnitt 2.2 werden wichtige Grundbegriffe relationaler Datenbanksysteme erläutert und erklärt, wie Daten in Relationen repräsentiert und verarbeitet werden können. Hier gelingt die Darstellung von Objekten der linearen Algebra als Relationen und die damit verbundenen Basisoperationen, beispielsweise die Matrixvektormultiplikation, welche datenbankgestützt umgesetzt werden.

Im Kapitel 3 werden Künstliche Neuronale Netze, kurz KNN, als Forschungsgegenstand der Informatik eingeführt und deren mathematische Grundlagen präzisiert. Im Hinblick auf CNN werden vorwärtgerichtete neuronale Netze, kurz FFN, im Abschnitt 3.2 definiert und deren Training hinsichtlich der Klassifikationsaufgabe im Abschnitt 3.3 erläutert.

Darauf aufbauend werden im Kapitel 4 gefaltete neuronale Netze als spezielle neuronale Netze eingeführt und deren Vorteile gegenüber FFN hinsichtlich der Bildklassifikation kurz erläutert. In den Abschnitten 4.1 und 4.2 wird die Faltungsoperation und

die CNN-Archietektur vorgestellt. Die Problemstellung 2 hinsichtlich des Trainings von gefalteten neuronalen Netzen wird im Abschnitt 4.3 behandelt.

Das Kapitel 5 beschäftigt sich mit der Transformation von gefalteten neuronalen Netzen und deren Operationen in SQL-Anweisungen. Im Abschnitt 5.1 werden Antworten zu Problem 1 geliefert und numerische Resultate vorgestellt. Zusammen mit der datenbankgestützten Implementierung von vorwärtgerichteten neuronalen Netzen, beleuchtet im Abschnitt 5.1.3, gelingt hinsichtlich Problem 3 die Transformation der Vorwärtsrechnung für ein aus Kapitel 4 trainiertes CNN im folgenden Abschnitt 5.2. Schließlich werden in Kapitel 6 die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf fortführende Forschungsthemen gegeben.

2. Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Definition erläutert, die im weiteren Verlauf dieser Arbeit genutzt werden. Zunächst wird im Abschnitt 2.1 ein kurzer Überklick über das Maschinelle Lernen und dessen mathematische Grundlagen gegeben. Weiter wird die Klassifikationsaufgabe definiert und ein erster Lernalgorithmus vorgestellt, welcher im weiteren Verlauf der Arbeit verfeinert wird. Im Hinblick auf die Transformation von ML-Algorithmen in SQL-Anweisungen werden im Abschnitt 2.2 relationale Datenbanksysteme und die Anfragesprache SQL eingeführt.

2.1. Maschinelles Lernen

Maschinelle Lernverfahren können genutzt werden, um Muster in digitalisierten Objekte zu erkennen. Hier kommen Algorithmen zum Einsatz, welche hinsichtlich einer bestimmten Aufgabe, engl. *task T*, und einem Leistungsmaß, engl. *perfomance P* an der Erfahrung, engl. *experience E* lernen, vgl.[20]. Dabei ist mit Lernen gemeint, dass das Computerprogramm bezüglich der Aufgabe T sein Leistungsmaß P mit wachsener Erfahrung E schrittweise steigert. Diese Begriffe werden im Kontext des überwachten Lernens im Folgenden eingeführt.

2.1.1. Die Klassifikationsaufgabe

Seien gewisse Objekte $O_1, \ldots, O_m$ durch Vektoren repräsentiert, welche in der Menge $\mathcal{M} := \{x_i \ : \ 1 \leq i \leq m\}$ gesammelt werden. Jedes der Objekte lässt sich durch die Funktion $f : \mathcal{M} \to \{K_1, K_2, \ldots, K_s\}$ zu einer der s Klassen zuordnen. Die Funktion f wird Klassifikationsfunktion genannt. Im Allgemeinen ist diese Funktion jedoch unbekannt und es steht nur eine endliche Menge von Tupeln der Form $(x_i, f(x_i))_{i=1}^m$ zur Verfügung.

Definition 2 (Trainingsmenge)**.** *Seien die Mengen $\mathcal{M}$ und $\mathcal{C} := \{K_1, \ldots, K_s\}$ gegeben. Weiter sei $f : \mathcal{M} \to \mathcal{C}$ eine (unbekannte)Klassifikationsfunktion. Dann wird die Menge*

$$\mathcal{T} := \{(x_i, f(x_i)) \ : \ 1 \leq i \leq m\} \tag{2.1}$$

Traininsmenge genannt.

Nun kann die Klassifikation als *task T* definiert werden, welche im weiteren Verlauf dieser Arbeit im Fokus steht.

Definition 3 (Klassifikationsaufgabe)**.** *Seien $\mathcal{M}$ und $\mathcal{C}$ wie in Definition 2 gegeben. Die Klassifikationsaufgabe besteht darin, ein Modell $\tilde{f}_{\mathcal{W}} : \mathcal{M} \to \mathcal{C}$, abhängig von Parametern $\mathcal{W}$, zu entwickeln, welche die Funktion f hinreichend gut approximiert. Zur Vereinfachung der Notation wird das Modell in Zukunft mit $\tilde{f}$ bezeichnet.*

Im Zuge des überwachten Lernens wird eine Trainingsphase durchgeführt, bei welcher die Modellparameter $\mathcal{W}$ mithilfe der Traininsmenge $\mathcal{T}$ (2.1) in einem iterativen Prozess schrittweise angepasst werden. Die Tupel in $\mathcal{T}$, welche die gewünschten Ausgaben beinhalten, können als *experience E* interpretiert werden. Die Güte der Approximation $\tilde{f}$ wird durch Fehlerfunktionen gemessen. Ziel ist es, die *perfomance P* des Modells durch die Minimierung solcher Fehlerfunktionen zu steigern.

Definition 4. *Seien eine Klassifikationsfunktion f, eine Traininsmenge $\mathcal{T}$ und Modellparameter $\mathcal{W}$ gegeben. Für das Modell $\tilde{f}$ soll das Minimierungsproblem*

$$\sum_{i=1}^{m} \mathcal{E}(\tilde{f}(x_i), f(x_i)) \to \min \tag{2.2}$$

durch die Anpassung der Modellparameter $\mathcal{W}$ gelöst werden. Dabei wird $\mathcal{E}$ Fehlerfunktion genannt.

Bei neuronalen Netzen wird das Optimierungsproblem (2.2) mithilfe des Backpropagationsalgorithmus bearbeitet. Jener wird im Hinblick auf Problem 2 in den Kapiteln 3 und 4 für die dort verwendeten Modelle ausführlich vorgestellt.

In der Praxis werden Testdaten, für die die Klassenzugehörigkeiten bekannt sind, genutzt, um die Generalisierungsfähigkeit des Modells zu messen. Diese Daten werden während des Trainingsprozesses zur Anpassung der Modellparameter $\mathcal{W}$ nicht benutzt.

Definition 5 (Testmenge)**.** *Seien die Mengen $\mathcal{M}' := \{x_i \ : \ 1 \leq i \leq m'\}$ und $\mathcal{C} := \{K_1, \ldots, K_s\}$ gegeben. Weiter sei $f : \mathcal{M}' \to \mathcal{C}$ eine Klassifikationsfunktion. Dann wird mit*

$$\mathcal{T}' := \{(x_i, f(x_i)) \ : \ 1 \leq i \leq m'\} \tag{2.3}$$

eine Testmenge bezeichnet.

Auf Training- bzw. Testmengen lassenn sich Fehlerraten als Gütemaße definieren.

Definition 6 (Erkennungsrate, Generalisierungsrate)**.** *Sei die Approximation $\tilde{f}$ gegeben. Die Erkennungsrate von $\tilde{f}$ ergibt sich durch den Anteil der richtig klassifizierten Lerndaten in der Traininsmenge. Die Generalisierungsrate von $\tilde{f}$ ist durch den Anteil der richtig klassifizierten Lerndaten in der Testmenge gegeben.*

Schließlich ist noch das Problem der Überanpassung, engl. *overfitting*, zu erwähnen. Dieses Phänomen tritt auf, wenn die Erkennungsrate eines verwendeten Modells $\tilde{f}$ während der Trainingsphase weiter steigt, gleichzeitig aber die Generalisierungsrate sinkt. In diesem Fall passt sich das Modell an zufälliges Rauschen in den Trainingsdaten an und ist für die Klassifikation allgemeiner Daten meist unbrauchbar. Für eine tiefere Analyse des Overfittings und anderer Probleme sowie deren Behandlung sei auf Goodfellow[20] verwiesen. Fraglich bleibt zudem, unter welchen Vorraussetzungen überhaupt eine sinnvolle Approximation möglich ist. Dazu wird im Folgenden das Problem der linearen und nichtlinearen Trennbarkeit von Punktmengen behandelt.

2.1.2. Trennbarkeit und ein erster Lernalgorithmus

Seien wieder Vektoren $x_1, \ldots, x_m \in \mathbb{R}^n$ von Objekten $O_1, \ldots, O_m$ gegeben. Zunächst wird der einfache Fall betrachtet, bei dem die Punkte x_i jeweils zu genau einer der zwei Klassen K_0 und K_1 zugeordnet sind. Seien dazu die Punktmengen

$$\begin{aligned} \mathcal{M} &:= \{x_i \ : \ 1 \leq i \leq m\} \subseteq \mathbb{R}^n, \\ \mathcal{M}_i &:= \{x_l \ : \ O_l \text{ gehört zur Klasse } K_i\}, \ \ i = 0, 1 \end{aligned}$$

definiert.

Definition 7. *Zwei Punktmengen $P_0, P_1 \subseteq \mathbb{R}^n$ heißen linear trennbar, genau dann wenn Parameter $w \in \mathbb{R}^n$ und $\theta \in \mathbb{R}^n$ existieren, sodass*

$$w^T x - \theta \begin{cases} < 0, \ x \in \mathcal{M}_0 \\ \geq 0, \ x \in \mathcal{M}_1 \end{cases}$$

gilt. Dabei wird $w^T x - \theta = 0$ als trennende Hyperebene bezeichnet. Oft wird auch von einer linearen Entscheidungsgrenze gesprochen.

Sind w und θ bekannt, so kann die Zugehörigkeit eines Objektes O_i leicht bestimmt werden. Dazu wird $d := w^T x - \theta$ berechnet. Ist $d < 0$, so gehört das Objekt O_i zur Klasse K_0, andernfalls zur Klasse K_1. Seien

$$\chi(x) := \begin{cases} -1, & x \in \mathcal{M}_0 \\ 1, & x \in \mathcal{M}_1 \end{cases}$$

und

$$w := \begin{pmatrix} w \\ \theta \end{pmatrix}, \ \ x := \begin{pmatrix} x \\ -1 \end{pmatrix}$$

definiert. Der sogenannte Perzeptron-Lernalgorithmus[49], kurz PLA, kann genutzt werden, um den Parameter w schrittweise zu bestimmten. Verändert sich in einem Zyklus, also das einmalige Durchlaufen aller Punkte in $\mathcal{M}$, der Parameter w nicht, so wird der PLA, beschrieben in Algorithmus 1, abgebrochen.

Rosenblatt[49] bewies, dass für linear trennbare Punktmengen der PLA terminiert.

Satz 1. *Sind $\mathcal{M}_0$ und $\mathcal{M}_1$ linear trennbar, so bricht der Perzeptron-Lernalgorithmus in endlich vielen Schritten ab.*

Beweis. Ein Beweis ist von Rosenblatt[49] gegeben. □

Es lassen sich jedoch einfache Beispiele anegeben, bei der die Punktmengen nicht auf lineare Art und Weise getrennt werden können. Ein Beispiel ist das zweidimensionale XOR-Problem, bei dem die Punktmengen $P_0 = \{(0,0),(1,1)\}$ und $P_1 = \{(0,1),(1,0)\}$ getrennt werden sollen. Hier scheitert der Perzeptron-Lernalgorithmus, da diese Punktmengen nicht linear trennbar sind.

Definition 8. *Zwei Punktmengen $\mathcal{M}_0, \mathcal{M}_1 \subseteq \mathbb{R}^n$ heißen nichtlinear trennbar, genau dann wenn eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ existiert, sodass*

$$f(x) \begin{cases} < 0, & x \in \mathcal{M}_0 \\ \geq 0, & x \in \mathcal{M}_1 \end{cases}$$

gilt. Oft wird hier f als komplexe Entscheidungsgrenze bezeichnet.

Neuronale Netze, welche im Kapitel 3 vorgestellt werden, sind in der Lage, solch komplexe Entscheidungsgrenzen auch bei Multiklassenproblemen, also $s \geq 2$, widerzuspiegeln. Für weitere Ausführungen über Verfahren des Maschinellen Lernens sei auf das umfangreiche Buch von Goodfellow[20] verwiesen.

Algorithm 1 Der Perzeptron-Lernalgorithmus

Require: Punktmenge $\mathcal{M} \subseteq \mathbb{R}^n$
Ensure: w, θ zur Trennung der Punktmengen $\mathcal{M}_0$ und $\mathcal{M}_1$
  Wähle $w = w_0$ beliebig
  while Abbruchbedingung nicht erfüllt **do** ▷ Abbruchbedingung, siehe Text
    for $i = 1, \ldots m$ **do**
      $d = \chi(x) w^T x$
      if $d \leq 0$ **then**
        $w = w + \chi(x) x$
      end if
    end for
  end while

2.2. Relationale Datenbanksysteme

Relationale Datenbanksysteme gehören zu den erfolgreichsten und verbreitetsten Datenbanken, welche zur elektronischen Datenverwaltung in Computersystemen eingesetzt werden. In diesem Abschnitt werden wichtige Grundbegriffe relationaler Datenbanksysteme erläutert und erklärt, wie Daten in Relationen repräsentiert und verarbeitet werden können. Die Notation und Bezeichnungen basieren auf Heuer et al.[23]. Zum Abfragen von Datenbeständen wird die Datenbanksparche SQL im Abschnitt 2.2.3 eingeführt und deren theoretische Grundlage im Abschnitt 2.2.2 beleuchtet. Schließlich werden im Abschnitt 2.2.4 Methoden vorgestellt, um Objekte der linearen Algebra als Relationen darzustellen und damit verbundene Operationen, beispielsweise die Matrixvektormultiplikation, datenbankgestützt umzusetzen.

2.2.1. Das Relationenmodell

Der Grundbaustein relationaler Datenbanksysteme bildet die Relation. Sie stellt eine mathematische Beschreibung einer Tabelle, welche aus Attributen und zugehörigen Domänen besteht, dar.

Definition 9 (Universum, Attribut, Domäne)**.** *Bezeichne die endliche Menge $\mathcal{U} \neq \emptyset$ das Universum. Ein Element $A \in \mathcal{U}$ heißt Attribut. Für $m \in \mathbb{N}$ sei $\mathcal{D} = \{D_1, \ldots, D_m\}$ eine Menge nichtleere Mengen. Ein Element $D_i \in \mathcal{D}$ wird Domäne genannt. Für eine Funktion* $\mathrm{dom} : \mathcal{U} \to \mathcal{D}$ *bezeichne* $\mathrm{dom}(A)$ *den Wertebereich von A und $w \in \mathrm{dom}(A)$ ein Attributwert.*

Nun können das Relationenschema und zugehörige Begriffe wie Relation und Tupel definiert werden.

Definition 10 (Relationenschema, Relation, Tupel)**.** *Eine Menge $R \subseteq \mathcal{U}$ heißt Relationenschema über dem Universum $\mathcal{U}$. Für $R = \{A_1, \ldots, A_n\}$ ist eine Relation r über R, kurz $r(R)$, als eine endliche Menge von Abbildungen*

$$t : R \to \bigcup_{i=1}^{m} D_i$$

definiert. Dabei gilt $t(A) \in \mathrm{dom}(A)$. Die Abbildungen t werden Tupel genannt und mit $t(A)$ ist die Restriktion der Abbildung t auf $A \in R$ gemeint.

Vereinfacht gesagt, setzt sich eine Datenbank als Menge von Relationen und ein Datenbankschema als Menge der zugehörigen Relationenschemata zusammen.

Definition 11 (Datenbank, Datenbankschema, vgl.[23])**.** *Für $p \in \mathbb{N}$ ist eine Menge von Relationenschemata $S = \{R_1, \ldots, R_p\}$ als Datenbankschema definiert. Eine Datenbank d über dem Schema S, kurz $d(S)$, ist eine Menge von Relationen*

$$d = \{r_1, \ldots, r_p\}$$

mit $r_i(R_i)$ für $1 \leq i \leq p$. Eine Relation $r \in d$ wird Basisrelation genannt.

Weiter können Beziehungen zwischen Attributen und Relationen definiert werden. Für diese Arbeit ist der Begriff des Schlüsselattributs wesentlich.

Definition 12 (Schlüssel)**.** *Sei R ein Relationenschema und $K = \{B_1, \ldots, B_k\} \subseteq R$. Gilt für jede Relation $r(R)$ die Beziehung*

$$\forall t_1, t_2 \in r : \ [t_1 \neq t_2 \Rightarrow \exists B \in K : \ t_1(B) \neq t_2(B)],$$

so wird K identifizierende Attributmenge genannt. Ein Schlüssel ist eine bezüglich der Mengeninklusion $\subset$ minimal identifizierende Attributmenge. Die Attribute eines Schlüssels werden Primattribute genannt.

Mit diesen Begriffen lässt sich das relationale Datenbanksystem definieren.

Definition 13. *Ein relationales Datenbanksystem ist eine Kombination aus Datenbank und Datenbankmanagementsystem, wobei letzteres zur Verwaltung der Daten verwendet wird.*

Das Managementsystem ist als abgekapseltes Softwaremodul zu interpretieren, welches bestimme Funktionen zur Verwaltung der Datenbank unter gewissen Anforderungen liefert. Typischerwiese sind die von Edgar F. Codd etablierten Anforderungspunkte, siehe [23], von einem Datenbankmanagementsystem umzusetzen. Eine der geforderten Funktion bildet die Anfrage an eine Datenbank, um Daten auslesen zu können. Dabei sei bemerkt, dass je nach Datenbanksystem die Anfragebearbeitung unterschiedlich abläuft. Für eine vertiefende Analyse von Architekturen, Funktionalitäten und Implementierungsmöglichkeiten von relationalen Datenbanksystemen sei auf Heuer et. al.[24, 23] verwiesen.

Im Folgenden wird die Relationenalgebra als Anfragesprache vorgestellt. Sie bildet die theoretische Grundlage der weitverbreiteten Anfragesprache SQL, welche im Folgeabschnitt 2.2.3 im Fokus steht. Zusammen mit der erweiterten Relationalgebra gelingt im Abschnitt 2.2.4 die Umsetzung von Basisoperationen der linearen Algebra in SQL.

2.2.2. Die Relationenalgebra

In der Relationenalgebra werden Relationen als abstrakte Datentypen mit darauf definierten Operationen definiert. Eine Anfrage ist eine Komposition von Operatoren aus einem gewissen Operatorensystem. Ein geeignetes System ist $\omega = \{\pi, \sigma, \bowtie, \cup, \setminus, \beta\}$, welches im Folgenden definiert wird.

- Für eine relation $r(R)$ mit Tupeln t wird die Projektion $\pi_X(r)$ auf das Attribut $X \subseteq R$ durch
$$\pi_X(r) := \{t(X) \mid t \in r\}$$
definiert.

- Die Konstantenselektion $\sigma_{X\theta c}$ ist als
$$\sigma_{X\theta c}(r) := \{t \mid t \in r \wedge t(X)\ \theta\ c\}$$
definiert. Hierbei ist $\theta \in \{=, \neq\}$ möglich und bei Wertebereichen, welche mit einer Halbordnung ausgestattet sind, ist $\theta \in \{\leq, <, \geq, >, =, \neq\}$ möglich. Bei Attributen mit demselben Wertebereich ist die Attributselektion $\sigma_{X\theta Y}$ für $X, Y \subseteq R$ als
$$\sigma_{X\theta Y}(r) := \{t \mid t \in r \wedge t(X)\ \theta\ t(Y)\}$$
definiert. Zudem können mehrere Selektionsbedingungen beliebig logisch mit $\wedge, \vee$ und $\neg$ in F verknüpft werden. Die Selektion σ ist eine Konstanten- oder Attributselektion.

- Sind $r_1(R_1)$ und $r_2(R_2)$ Relationen, so verbindet der natürliche Verbund $\bowtie$ Tupel der beiden Relationen mit gleichen Attributwerten von gleichnamigen Attributen, also
$$r_1 \bowtie r_2 := \{t \mid t(R_1 \cup R_2) \wedge \exists t_1 \in r_1 : t_1 = t(R_1) \wedge \exists t_2 \in r_2 : t_2 = t(R_2)\}.$$

Ist $R_1 = R_2$ so wird $\bowtie$ zum mengentheoretischen Durchschnitt und für $R_1 \cap R_2 = \emptyset$ ergibt sich das kartesische Produkt von r_1 und r_2.

- Die Vereinigung zweier Relationen $r_1(R)$ und $r_2(R)$ über dem Relationenschema R ist durch
$$r_1 \cup r_2 := \{t \mid t \in r_1 \wedge t \in r_2\}$$
definiert.

- Die Differenz zweier Relationen $r_1(R)$ und $r_2(R)$ über dem Relationenschema R ist als
$$r_1 \setminus r_2 := \{t \mid t \in r_1 \wedge t \notin r_2\}$$
definiert.

- Die Umbenennung β wird für die obigen Operationen benötigt, da diese von der Attributbenennung abhängen. Für $A \in R, B \notin (R \setminus \{A\})$ sei $R' := (R \setminus \{A\}) \cup B$. Die Umbenennung β von A zu B in $r(R)$ ist für $\mathrm{dom}(A) = \mathrm{dom}(B)$ als
$$\beta_{B \leftarrow A} := \{t' \mid \exists t \in r : t'(R \setminus \{A\}) = t(R \setminus \{A\}) \wedge t'(B) = t(A)\}$$
erklärt.

Es sei darauf hingewiesen, dass weitere Operatorensysteme existieren, welche zu ω äquivalent sind[23].

Erweiterung der Relationenalgebra

Mit dem Operatorensystem ω von oben sind noch keine arithmetischen Berechnungen über Attribute möglich, welche jedoch zwingend für die Umsetzung linearer Algebra benötigt werden. Daher wird im Folgenden die Relationenalgebra aus Abschnitt 2.2.2 erweitert. In dieser Arbeit wird oft die Gruppierung von Tupeln bezüglich gleicher Attributwertkombinationen benötigt und daher ein Gruppierungsoperator γ eingeführt. Darüber hinaus wird die Projektion π erweitert, um tupelweise Funktion, z.B. die Summierung **SUM**, aus Attributen verwenden zu können. Dazu werden die bisher eingeführten Operatoren leicht modifiziert, indem sie auf Multimengen definiert werden. Dafür sei an dieser Stelle auf [19] verwiesen, um die angepasste Definition des Operatorensystems ω und die zugrunde liegende Theorie nachzuvollziehen.

Der Gruppierungsopeartor γ

Der Operator $\gamma_L(r)$ dient zur Aggregation von Attributwerten in Gruppen für eine gegebene Relation r. Die Liste L kann aus Attributen der Relation r oder aus bestimmten Aggregatfunktionen bestehen, welche auf den jeweiligen Attributen berechnet werden. Die Ergebnisrelation besteht aus Tupeln von r, welche in Gruppen partitioniert ist. Dabei ergeben sich die jeweiligen Gruppen durch gleiche Attributwertkombinationen der Gruppierungsattribute in L. Es wird für jede Gruppe ein Tupel berechnet, welches

dann aus den Attributwerten der Attribute aus L oder den Ergebnissen der Aggregatfunktionen aus L besteht. Folgendes Beispiel illustriert die Funktionsweise des Gruppierungsoperator γ. Dazu sei die Relation A in Tabellenform 2.1 gegeben. Die Relation

A		
i	j	v
1	1	3
1	2	1
2	1	4
2	2	1

Tabelle 2.1.: Zu sehen ist die Beispielrelation bestehend aus den Attributen i, j und v.

A kann als Matrix

$$A = \begin{pmatrix} 3 & 1 \\ 4 & 1 \end{pmatrix}$$

interpretiert werden. Weitere Ausführungen dazu werden im späteren Abschnitt 2.2.4 vorgestellt. Die Spaltensumme von A kann mit der Gruppierungsoperationen

$$\gamma_{j,\,\mathbf{SUM}(v)\to v}(\mathbf{A}) \Rightarrow$$

j	v
1	7
2	2

berechnet werden. Mit dem Pfeil ist die Umbenennung β gemeint.

Der erweiterte Projektionsoperator

Desweiteren soll der Projektionsoperator $\pi_F(r)$ auf tupelweise arithmetische Berechnungen bezüglich einer oder mehrer Attribute einer Relation **r** erweitert werden. Dazu wird wieder eine Liste F genutzt, welche

- Attribute der Relation r,
- Umbenennungen der Form $x \to y$, wobei x eine Attribut der Relation r ist, oder
- Ausdrücke der Form $E \to z$, wobei E aus Attributen der Relation r, Konstanten oder arithmetischen Operationen besteht,

beinhalten kann. Die Funktionsweise des erweiterten Projektionsoperators wird an der Beispielrelation 2.1 beleuchtet. Die Matrixaddition

$$2 \cdot A + \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

lässt sich durch

$$\pi_{i,\,j,\,2*v+1\to v}(\mathbf{A}) \Rightarrow$$

i	j	v
1	1	7
1	2	3
2	1	9
2	2	3

berechnen. Für diese Arbeit genügt die erweiterte Relationenalgebra mit der Gruppierung γ_L und der erweiterten Projektion π_F, um fundamentale Operationen der linearen Algebra in der Datenbanksprache SQL darzustellen. Diese wird im Folgenden Abschnitt vorgestellt.

2.2.3. Die Anfragesprache SQL

SQL als Abkürzung für *Structured Query Language* ist eine Datenbanksparche zur Definition von Datenstrukturen in relationalen Datenbanksystemen. Sie wird genutzt, um Datenbestände zu bearbeiten und abzufragen und basiert dabei auf der im vorherigen Abschnitt eingeführten Relationenalgebra. SQL wird als standardisierte Sprache in allen kommerziellen und frei zugänglichen Datenbankmanagementsystemen unterstützt. Da die Sprache unter Mitwirkung von Normungsgremien wie des *American National Standard Institute* (ANSI) und der *International Organization for Standardization* (ISO) seit 1986 standardisiert ist, ist es möglich, über die Grenzen spezieller Systeme hinaus SQL als einheitliche Datenbanksprache zu benutzen. Für eine detailliertere Darstellung der einzelen Standards sei auf die entsprechende Literatur verwiesen. Ein guter Überblick ist in Heuer et. al.[23] zu finden.

SQL ist aus mehreren Teilsprachen aufgebaut, welche jeweils unterschiedliche Aufgaben des Datenbankmanagementsystems hinsichtlich der Datendefinition, Dateiorganisation und Anfragebearbeitung übernehmen. Um Methoden der linearen Algebra in Sequenzen von SQL-Anfragen darzustellen, wird der Anfrageteil *Interactive Query Language*, kurz IQL, als deklarative Programmiersprache im Folgenden beleuchtet. Diese bildet das Fundament der Datenselektion und wird zur Analyse von Daten in relationalen Systemen genutzt. Eine Anfrage hat die Form

SELECT Projektionsliste
FROM Relationenliste (2.4)
[**WHERE** Bedingung].

Das Resultat einer Anfrage ist wieder eine Relation. Daher ist es möglich, verschachtelte Anfragen zu formulieren. Die **SELECT**-Anweisung gibt durch die Attribute in der Projektionsliste das Schema der Ergebnistabelle vor. Die **FROM**-Klausel beinhaltet die Relationenliste, in welcher die Relationen aufgeführt sind, aus denen Attribute selektiert werden sollen. Die Liste kann aus einer oder aus mehreren Unteranfragen bestehen, welche optional durch Operationen wie dem natürlichen Verbund oder dem kartesischem Produkt kombiniert werden können. Ebenfalls optional ist die **WHERE**-Klausel, welche Selektionsbedingungen bezüglich Konstanten oder Attributen enthält.

Angestellte				
ID	Name	Spezialisierung	Projektnummer	Gehalt
1	Martin	Elektrotechnik	3	2300
2	Lennardt	Informatik	1	1500
3	Johann	Informatik	3	1800
4	Anna	Buchhaltung	3	2000
5	Antonia	Buchhaltung	2	2000

Tabelle 2.2.: Abgebildet ist die Beispielrelation Angestellte mit den Attributen ID, Name, Spezialisierung, Projektnummer und Gehalt.

Projekt				
Projektnummer	Projektname	Budget	Ort	Status
1	Datenbank 2.0	50000	Rostock	abgeschlossen
2	Verwaltung	25000	Rostock	offen
3	Forschungabteilung	40000	Schwerin	offen

Tabelle 2.3.: Abgebildet ist die Beispielrelation Projekt mit den Attributen Projektnummer, Projektname, Budget, Ort und Status.

Um dden Anfragemechanismus nachzuvollziehen, seien im Folgenden zwei Relationen *Angestellte* und *Projekt* wie in den Tabellen 2.2 und 2.3 gegeben. Die Anfrage

SELECT A.Name, P.Projektname
FROM Angestellte A **JOIN** Projekt P **ON** A.Projektnummer = P.Projektnummer
WHERE P.Status = 'offen'

(2.5)

liefert die Namen der Angestellten von offenden Projekten. Dementsprechend ist die Ergebnisrelation in Tabelle 2.4 dargestellt. Die Aliasnamen „A“ und „P“ dienen hier

Ergebnis	
A.Name	P.Projektname
Martin	Forschungabteilung
Johann	Forschungabteilung
Anna	Forschungabteilung
Antonia	Verwaltung

Tabelle 2.4.: Zu sehen ist die Ergebnisrelation der zuvor beschriebenen Anfrage (2.5).

zur Übersicht und sind bei geschalteten SQL-Anfragen zwingend notwendig. Die klassiche Anfrage (2.4) kann mit arithmetischen Operationen und Aggregatfunktionen sowie Gruppierungsfunktionen erweitert werden. Die Funktion **SUM** kann zur Summierung numerischer Attributwerte eines Attributs über mehrere Tupel verwendet werden. Die **GROUP BY**-Klausel dient zur Gruppierung von Tupeln bezüglich gleicher Attribut-

wertkombinationen. Eine Kombination aus der Aggregatfunktion **SUM** und der Gruppierung wird beispielsweise in der Anfrage

SELECT A.Spezialisierung, **SUM**(A.Gehalt) **AS** Angestelltenkosten
FROM Angestellte A (2.6)
GROUP BY A.Spezialisierung

verwendet. Diese berechnet die Angestelltenkosten und gruppiert sie nach den jeweiligen Spezialisierungen. Die Ergebnistabelle ist in Tabelle 2.5 dargestellt. Ein Beispiel einer

Ergebnis	
A.Spezialisierung	Angestelltenkosten
Elektrotechnik	2300
Informatik	3300
Buchhaltung	4000

Tabelle 2.5.: Es ist die Ergebnisrelation der zuvor beschriebenen Anfrage (2.6) abgebildet.

geschalteten Anfrage zur Bestimmung des Angestellten mit dem höchstem Gehalt ist durch

SELECT A.Name **AS** Topverdiener, A.Gehalt
FROM Angestellte A (2.7)
WHERE A.Gehalt = (**SELECT MAX**(Gehalt) **FROM** Angestellte)

gegeben. Dabei wird die Aggregatfunktion **MAX** verwendet. Die Ergebnisrelation beinhaltet in diesem Beispiel ein Tupel mit Martin als Topverdiener mit einem Gehalt von 2300 Euro.

Weitere Operationen der Relationenalgebra wie die Vereinigung (**UNION**) und die Mengendifferenz (**EXCEPT**) können ebenfalls in SQL-Anfragen eingebunden werden. Es sei bemerkt, dass Ergebnisse von SQL-Anfrage immer als Multimengen fungieren. Um eine Duplikateleminierung zur Verfügung zu haben, kann die **SELECT**-Klausel durch den **DISTINCT**-Operator erweitert werden. Die Repräsentierung der Operatoren der Relationenalgebra durch SQL-Anfragen wird in Tabelle 2.6 dargestellt. In vielen Datenbankmanagementsystemen gibt es weitere Funktionalitäten und Operatoren, welche in dieser Arbeit nicht weiter beleuchtet werden. Für die datenbankgestützte Umsetzung linearer Algebra genügt der SQL-Anfragekern wie in Tabelle 2.6.

2.2.4. Lineare Algebra in SQL

In diesem Abschnitt werden Darstellungsformen für Vektoren und Matrizen als Relationen vorgestellt. Es werden zwei Strategien erläutert, welche jeweils zur Repräsentierung von dicht und dünn besetzten Matrizen genutzt werden. Weiter werden Ideen zur Umsetzung wichtiger Basisoperationen mit Vektoren und Matrizen in SQL beleuchtet, da diese mathematischen Objekte bei zahlreichen statistischen Analysen eingesetzt werden.

Relationenalgebra	SQL
Projektion π	**SELECT DISTINCT** [**GROUP BY**]
Selektion σ	**WHERE** ohne Schachtelung
Verbund $\bowtie$	**FROM**, **WHERE** **FROM** mit **JOIN**
Umbenennung β	**FROM** mit Tupelvariable **AS**
Differenz $\setminus$	**WHERE** mit Schachtelung **EXCEPT**
Vereinigung $\cup$	**UNION**

Tabelle 2.6.: Vergleich des Operatorensystems aus Abschnitt 2.2.2 mit dem SQL-Anfragekern, vgl.[23].

Dicht besetzte Matrizen

Im Folgenden wird das *Coordinate-Schema*[37] als Schema für die Darstellung dichtbesetzter Matrizen genutzt. Dieses Schema gestaltet sich als einfach und ist daher weit verbreitet [53]. Für eine weiterführende Diskussion anderer Darstellungsmöglichkeiten und deren Vor- und Nachteile, sei auf Marten[37] verwiesen. Das Coordinate-Schema beinhaltet drei Arrays, welche den Zeilenindex, Spaltenindex und den Matrixeintrag angeben. Für $A \in \mathbb{R}^{m \times n}$ kann das Tupel $(i, j, A_{i,j})$ als Schlüssel-Wert-Paar interpretiert werden. So ergibt sich auf natürliche Weise die Überführung von Matrizen und Vektoren in relationale Schemata.

Definition 14 (Coordinate-Schema)**.** *Für $x \in \mathbb{R}^n$ und $A \in \mathbb{R}^{m \times n}$ ergeben sich die Relationen*

$$\begin{aligned}\mathbf{x}(&\underline{i} \text{ int},\\ &v \text{ double})\end{aligned}$$

für den Vektor x und

$$\begin{aligned}\mathbf{A}(&\underline{i} \text{ int},\\ &\underline{j} \text{ int},\\ &v \text{ double})\end{aligned}$$

für die Matrix A. Die Indizies i und j sind jeweils als ganzzahlige Datentypen, engl. integer, dargestellt. Die Matrixeinträge werden in double precision[27] abgespeichert. Die unterstrichenen Attribute stellen die Schlüsselattribute, vgl. Definition 12 dar. Diese Darstellungsform wird Coordinate-Schema genannt und für dichtbesetzte Matrizen im weiteren Verlauf dieser Arbeit genutzt.

A		
i	j	v
1	1	1
1	2	2
2	1	-5
2	2	2
3	1	0
3	2	7

Tabelle 2.7.: Das Coordinate Schema zur Matrix A aus Beispiel 1.

Beispiel 1. *Ist*

$$A = \begin{pmatrix} 1 & 2 \\ -5 & 2 \\ 0 & 7 \end{pmatrix} \in \mathbb{R}^{3\times 2}$$

gegeben, so ergibt sich Coordinate Schema wie in Tabelle 2.7.

Basisoperationen

In diesem Abschnitt werden typische Operationen der linearen Algebra beschrieben. Einfache Funktionen wie die Summation und Multiplikation für reelle Zahlen sind bereits im SQL-Standard[31] enthalten. Seien nun Vektoren $x, y \in \mathbb{R}^n$ sowie Matrizen $A, B \in \mathbb{R}^{m\times n}$ und Skalare $r, s \in \mathbb{R}$ gegeben. Die jeweiligen Relationen werden gemäß dem Coordinate-Schema erstellt. Die SQL-Anweisung für die Vektoraddition $rx + sy$ lautet

SELECT $x.i$ **AS** $i,\ r * x.v + (s * y.v)$ **AS** v
FROM x **JOIN** y **ON** $x.i = y.i$

Ähnlich ergibt sich die Matrixaddition $rA + sB$ zu

SELECT $A.i$ **AS** $i,\ A.j$ **AS** $j, (r * A.v) + (s * B.v)$ **AS** v
FROM A **JOIN** B **ON** $A.i = B.i$ **AND** $A.j = B.j$

Mit der Aggregation **SUM** können zudem Skalarprodukte und damit Längenbegriffe wie Normen formuliert werden. Die entsprechenden SQL-Anfragen sind im Anhang A.1 zu finden.

Weitere wichtige Oprationenen sind die Matrixvektor- und Matrixmatrixmultiplikation. Durch Kombination vorheriger Basisoperationen ergeben sich die entsprechende Transformation für die Matrixvektormultiplikation $Ax \in \mathbb{R}^m$ einer Matrix $A \in \mathbb{R}^{m\times n}$ und Vektors $x \in \mathbb{R}^n$ zu

SELECT $A.i$ **AS** i, **SUM**$(A.v * x.v)$ **AS** v
FROM A **JOIN** x **ON** $A.j = x.i$
GROUP BY $A.i$.

Die Matrix $C = AB \in \mathbb{R}^{m \times n}$ als Produkt zweier Matrizen $A \in \mathbb{R}^{m \times k}$ und $B \in \mathbb{R}^{k \times n}$ lässt sich durch

SELECT $A.i$ **AS** i, $B.j$ **AS** j, **SUM**$(A.v * B.v)$ **AS** v
FROM A **JOIN** B **ON** $A.j = B.i$
GROUP BY $A.i$, $B.j$

berechnen. Schließlich kann auch die Transponierte A^T einer Matrix A einfach berechnet werden, siehe dazu Anhang A.1.

Dünn besetzte Matrizen

Bei bestimmten Anwendungen werden dünnbesetzte Matrizen benötigt, deren Zeilen- und Spaltenanzahl erheblich größer als bei dicht besetzten Problemen sind. Oft enthalten diese Matrizen dann eine Menge von Nicht-Null-Elementen, welche im Vergleich zu den Null-Elementen verschwindend gering ist. Daher lohnt es sich, nur die Nicht-Null-Elemente sinnvoll zu speichern. In dieser Arbeit wird das *Compressed-Sparse-Column-Schema*[16] verwendet, um dünn besetzte Matrizen zu repräsentieren. Es besteht ähnlich des Coordinate-Schemas wieder aus drei Arrays. Das erste Array gibt das Spaltenmuster **colPtr**, das zweite den Zeilenindex **rowInd** und das dritte den Matrixeintrag **val** an. Der formale Zusammenhang zwischen einer Matrix $A \in \mathbb{R}^{m \times n}$ und der Darstellung im Compressed-Sparse-Column-Schema ist durch

$$A_{i,j} = \mathbf{val}[k] \Leftrightarrow (\mathbf{rowInd}[k] = i) \wedge (\mathbf{colPtr}[j] \leq k < \mathbf{colPtr}[j+1])$$

gegeben. Folgendes Beispiel illustriert diese Beziehung.

Beispiel 2. *Für die Matrix*

$$A = \begin{pmatrix} a_{11} & 0 & 0 & a_{14} \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & a_{42} & 0 & a_{44} \end{pmatrix}$$

ergeben sich die Arrays des Compressed-Sparse-Column-Schemas zu

- ***val***$=[a_{11}, a_{21}, a_{22}, a_{42}, a_{33}, a_{14}, a_{44}]$,
- ***rowInd***$=[1, 2, 2, 4, 3, 1, 4]$,

- ***colPtr***$=[1, 3, 5, 6, 8]$.

Die indirekte Indizierung der Matrixelemente sorgt zwar für eine Verminderung des Speicherbedarfs, erschwert jedoch die rein relationale Umsetzung von Operationen wie die Matrixvektormultiplikation. Hier kann ein objekt-relationaler Ansatz Abhilfe schaffen, bei dem Array-Datentypen genutzt werden, welche jedoch nur in bestimmten relationalen Datenbankmanagementsystemen unterstützt werden.

Definition 15 (Spaltenkompression, vgl.[37])**.** *Eine dünn besetzte Matrix* $A \in \mathbb{R}^{m \times n}$ *wird als Relation*

$$\begin{aligned}\mathbf{A}(&i \text{ int ARRAY},\\ &\underline{j} \text{ int},\\ &v \text{ double ARRAY})\end{aligned}$$

gespeichert. Diese Repräsentierung wird Spaltenkompression genannt und ist insbesondere für die Matrixvektormultiplikation nützlich.

In dem Open-Source-Datenbanksystem PostgreSQL[43], welches in dieser Arbeit genutzt wird, kann für die Matrixvektormultiplikation die **UNNEST**-Funktion genutzt werden. Diese Funktion wandelt ein Array in eine Menge von Tupeln um. Die Matrixvektormultiplikation $Ax \in \mathbb{R}^m$ mit dünnbesetzter Matrix $A \in \mathbb{R}^{m \times n}$ kann dann mithilfe der SQL-Anfage

SELECT $i, \mathbf{SUM}(v)$
FROM(
SELECT UNNEST$(A.i)$ **AS** $i, \mathbf{UNNEST}(A.v) * x.v$ **AS** v
FROM A **JOIN** x **ON** $A.j = x.i$) temp
GROUP BY i

berechnet werden.

Zusammenfassend stellt sich heraus, dass wesentliche Objekte der linearen Algebra und fundamentale Operationen im SQL-Kern umgesetzt werden können. Für dicht besetzte Matrizen wird das Coordinate-Schema 14 benutzt, während für dünn besetzte Probleme das Spaltenkompression-Schema 15 genutzt wird. Beide Darstellungsformen erlauben kompakte SQL-Anfragen für die Matrixmatrix- bzw. Matrixvektormultiplikation. Für eine tiefere Analyse dieser und anderer Darstellungsmöglichkeiten und deren Perfomance hinsichtlich verschiedener Basisoperationen sei auf Marten[37] verwiesen.

3. Grundlagen neuronaler Netze

In diesem Kapitel werden Künstliche Neuronale Netze[14], 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[46, 47, 59]. Sie eignen sich daher für viele typische Aufgaben des maschinellen Lernens, beispielsweise für die Klassifikation digitalisierter Objekte.

Im ersten Abschnitt wird das Perzeptron[49] als Grundeinheit eines neuronalen Netzes eingeführt. Im folgenden Abschnitt wird das Konzept der Multi-Layer-Perzeptronen[60] 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 im letzten Abschnitt 3.4 gegeben.

3.1. Das Perzeptron

Zunächst wird das *Perzeptron* ähnlich wie in Minsky[42] als fundamentaler Baustein eines neuronalen Netzes eingeführt. Das Perzeptron wird oft als Basis moderner KNN angeführt und kann mithilfe des Perzeptron-Lernalgorithmus 1 trainiert werden, um das Problem der Trennbarkeit von Punktmengen zu lösen.

Definition 16 (Perzeptron)**.** *Für eine gegebene Funktion $\phi : \mathbb{R} \to \mathbb{R}$, einen Vektor $w \in \mathbb{R}^n$ und ein Skalar $\theta \in \mathbb{R}$ wird die Funktion*

$$\Phi : \mathbb{R}^n \to \mathbb{R}, \quad x \mapsto \phi(w^T x - \theta) =: y,$$

Perzeptron genannt. Mit $x \in \mathbb{R}^n$ wird die vektorwertige Eingabe und mit $y \in \mathbb{R}$ die skalare Ausgabe, auch Aktivierung, des Perzeptrons bezeichnet. Dabei ist $w^T x = \sum_{i=1}^{n} w_i x_i$ das Standardskalarprodukt im euklidischen Vektorraum $\mathbb{R}^n$. 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. In dieser Arbeit wird das Perzeptron oft einfach Neuron genannt. Die Begriffe sind austauschbar und

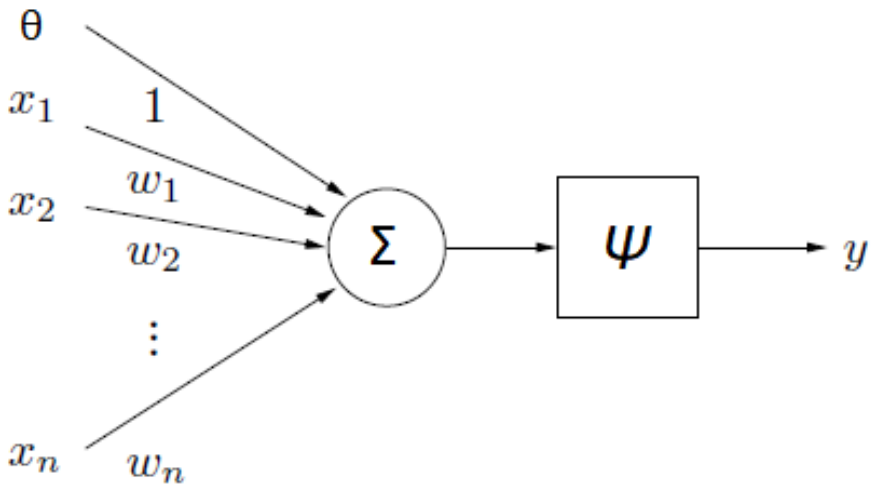

Abbildung 3.1.: Arbeitsweise eines Perzeptrons mit entsprechender Notation aus Definition 16.

meinen dasselbe. Bei der Wahl der Funktion ϕ gibt es mehrere Möglichkeiten. Wird wie in Minsky[42] 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 2.1.2 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_0 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. 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[7, 6, 51]. Zunächst sind einige Definition notwendig, um eine lesbare Notation des MLP zu geben.

Definition 17 (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 18 (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[48], welche jedoch in dieser Arbeit nicht weiter von Interesse sind. Typische Aktivierungsfunktionen, welche 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.*

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 $\psi(x) = \frac{1}{1+\mathrm{e}^{-x}}$ *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 $\tanh(x) = \frac{2}{e^{2x}+1}$ genutzt und das Differenzieren mittels Quotientenregel führt zur Aussage. □

Ähnlich der Definition des Perzeptrons 16 wird nun eine Schicht als Verknüpfung von Übertragungsfunktion und Aktivierungsfunktion definiert.

Definition 19 (Neuronenschicht)**.** *Ist* $\Psi^{W,b}$ *eine Übertragungsfunktion mit den Parametern* $W \in \mathbb{R}^{n\times m}$ *und* $b \in \mathbb{R}^m$ *sowie* ψ *eine Aktivierungsfunktion, so wird das Paar* $(\Psi^{W,b}, \psi)$ *als Neuronenschicht* $\mathcal{S}$ *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* $1 \leq i \leq m$ *Aktivierungen der Neuronen in der Schicht* $\mathcal{S}$ *genannt und gleichen jeweils der Ausgabe eines einfachen Perzeptrons wie in Definition 16. Eine Schicht besteht also aus* m *Perzeptronen* $\tilde{\Psi}_i$ *mit der Beziehung* $y_i = \Phi(x_i) = \phi(W^T_{i,:}x + b_i)$ *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 20 (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 2 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 $i = 1, \dots l$ **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 2 festgehalten. Das zuvor angesprochene XOR-Problem kann nun beispielsweise mithilfe eines KNN bestehend aus zwei Schichten gelöst werden[20]. Es lassen sich zwischen Modell- und Hyperparameter von KNN unterscheiden.

Definition 21 (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 $\mathcal{W} := \{(W^{(i)}, b^{(i)}) : \ i = 1, \dots, l\}$ 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 3.3 erläutert.

3.3. Training neuronaler Netze

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*[26] 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 22. *Seien die Mengen $D \subset \mathbb{R}^n$ und $\mathcal{C} = \{c_1, \ldots, c_m\}$ 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

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

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

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

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 2 (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[25] 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[54, 36].

Korollar 1. *Mit den gleichen Vorraussetzungen wie in Satz 2 gilt die Abschätzung 3.4 für $\psi_1(x) = \max\{0, x\}$.*

Beweis. Siehe Sonoda et. al.[56]. □

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 22 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[44] 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 23 (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[4] 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*[52] 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 Minimierungpproblem mithilfe der *negative log likelihood* schreiben.

Definition 24 (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 Likelihood 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[20]. Das Problem 3.6 wird nun allgmemein mit Fehlerfunktionen definiert.

Definition 25 (Fehlerfunktion)**.** *Seien $\mathcal{T}$ eine Trainingsmenge und $\mathcal{W}$ Modellparameter eines KNN. Es soll das Problem*

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

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

Backpropagationsalgorithmus

Bei FFN wird Backpropagation, etabliert von Rumelhart et. al. [51], als Lernalgorithmus genutzt, um eine gewählte Fehlerfunktion $\mathcal{E}$ zu minimieren. Dazu wird das Gradientenverfahren zur numerischen Minimierung von $\mathcal{E}$ im Raum der Modellparameter $\mathcal{W}$ genutzt. Dabei sei bemerkt, dass das Finden von globalen Minima durch den Backpropagationsalgorithmus nicht garantiert ist. 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. Weiter sollten auch die Aktivierungsfunktionen aus Definition 18 als stückweise stetig differenzierbare Funktion vorausgesetzt werden. Die Optimierung der Parameter geschieht iterativ und das Gradientenverfahren 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 werden 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 $\lambda \in \mathbb{R}$ in (3.10) 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. Ein Kompromiss aus beiden Verfahren ist das *Mini-Batch-Verfahren* bei dem die Gradienten über kleine Teilmengen $\mathbb{T} \subset \mathcal{T}$ der Trainingsmenge berechnet werden.

Die Abstiegsrichtungen werden mithilfe der mehrdimensionalen Kettenregel berechnet.

Satz 3 (Mehrdimensionale Kettenregel)**.** *Ist $f = f(x_1(y_1, \ldots, y_m), \ldots, x_n(y_1, \ldots, y_m))$ und sind alle beteiligten Funktionen stetig differenzierbar, so ergeben sich die partiellen 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. Ein Beweis kann in Forster[18] nachgelesen werden. □

Die effiziente Berechnung dieser Gradienten gehört zu den schwersten Aufgaben[34] im Bereich des Maschinellen Lernens. Im Folgenden wird der Online-Backpropagationsalgorithmus, im Englischen auch als *Stochastic Gradient Descent* bezeichnet, erläutert. Grundsätzlich lässt sich das Verfahren in zwei Schritte einteilen.

- Bei der Vorwärtsrechnung wird dem FFN Λ ein Trainingspaar $(x, c) \in \mathcal{T}$ präsentiert und die Aktivierungen der Schichten schrittweise von der Eingabeschicht über die verdeckten Schichten bis zur Ausgabeschicht berechnet.

- Bei der Rückwärtsrechnung wird für jedes Neuronen ein lokaler Gradient bezüglich E berechnet, beginnend in der Ausgabeschicht über die verdeckten Schichten bis zur Eingabeschicht.

Als Fehlerfunktion wird im weiteren Verlauf die mittlere quadratische Abweichung

$$E := \frac{1}{2} \sum_{(x,c)\in\mathcal{T}} ||y - t||_2^2, \tag{3.11}$$

genutzt, wobei wieder $y = \Lambda(x)$ der Ausgabevektor des FFN und $t = t(x, c)$ der Zielvektor des Datenpaars (x, c) ist. Beim Online-Verfahren wird der Fehler

$$E_x = \frac{1}{2}||y - t(x, c)||_2^2 = \frac{1}{2} \sum_{k=1}^{s_l} (y_k - t_k)^2 \tag{3.12}$$

für eine einzige Eingabe x berechnet. Die Ausgabeschicht von Λ besitze s_l Neuronen. Nun müssen die Abstiegsrichtungen

$$\Delta_{W^{(k)}} = \frac{\partial E_x}{\partial W^{(k)}}$$
$$\Delta_{b^{(k)}} = \frac{\partial E_x}{\partial b^{(k)}}$$

für $1 \leq k \leq s_l$ ermittelt werden. Dazu wird im Folgenden die Notation wie in [17] mit

$w^l_{k,j}$	Eintrag der Gewichtsmatrix $W^{(l)}$ der Ausgabeschicht,
$w^h_{j,i}$	Eintrag der Gewichtsmatrix $W^{(h)}$ einer verdeckten Schicht,
$V_j = \sum_i w^h_{j,i} x_i + b^h_j$	Netzeingabe des verdeckten Neurons j mit Bias $b^{(h)}$,
$z_j = \psi(V_j)$	Aktivierung des versteckten Neurons j,
$V_k = \sum_j w^l_{k,j} z_j + b^l_k$	Netzeingabe des Ausgabeneurons k mit Bias $b^{(l)}$,
$y_k = \psi(V_k)$	Aktivierung des Ausgabeneurons$k$,
$e_k = y_k - t_k$	Fehler des k-ten Ausgabeneurons

genutzt. Es wird nur eine verdeckte Schicht mit s_j Neuronen betrachtet. Die Verallgemeinerung für beliebig viele Schichten ist analog. Mit i ist ein Eingabeneuron, j eine verstecktes Neuron und k eine Ausgabeneuron gemeint. Außerdem werden die Gradienten komponentenweise berechnet und daher die Paramter komponentenweise aktualisiert.

Die mehrmalige Anwendung der Kettenregel 3 auf die Fehlerfunktion (3.12) liefert

$$\begin{aligned}\Delta w_{k,j}^{l} &= -\lambda \frac{\partial E_x}{\partial w_{k,j}^{l}} \\ &= -\lambda \frac{\partial E_x}{\partial e_k}\frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial V_k}\frac{\partial V_k}{\partial w_{k,j}^{l}} \\ &= -\lambda\, e_k \psi'(V_k) z_j \\ &= -\lambda\, \delta_k z_j\end{aligned}$$

mit

$$\delta_k := e_k \psi'(V_k) \tag{3.13}$$

als sogenannte lokale Fehler für $1 \leq k \leq s_l$. Für die Schwellwerte gilt analog

$$\Delta b_k^l = -\lambda \frac{\partial E_x}{\partial b_k^l} = -\lambda\, \delta_k.$$

Die Gradienten für die Parameter der verdeckten Schicht lassen sich mithilfe der lokalen Fehler δ_k der darüberliegenden Ausgabeschicht berechnen, also

$$\begin{aligned}\Delta w_{j,i}^{h} &= -\lambda \frac{\partial E_x}{\partial w_{j,i}^{h}} \\ &= -\lambda \frac{\partial E_x}{\partial z_j}\frac{\partial z_j}{\partial V_j}\frac{\partial V_j}{\partial w_{j,i}^{h}} \\ &= -\lambda \left(\sum_{k=1}^{s_l} e_k \frac{\partial e_k}{\partial z_j}\right) \psi'(V_j) x_i \\ &= -\lambda \left(\sum_{k=1}^{s_l} e_k \frac{\partial e_k}{\partial y_k}\frac{\partial y_k}{\partial V_k}\frac{\partial V_k}{\partial z_j}\right) \psi'(V_j) x_i \\ &= -\lambda \left(\sum_{k=1}^{s_l} e_k \psi'(V_k) w_{k,j}^{l}\right) \psi'(V_j) x_i \\ &= -\lambda\, \delta_j x_i\end{aligned}$$

mit x_i als Eingabeneuron und

$$\delta_j := \left(\sum_{k=1}^{s_l} \delta_k w_{k,j}^{l}\right) \psi'(V_j).$$

als lokale Fehler der verdeckten Neuronen für $1 \leq j \leq s_j$. Für die Schwellwerte gilt wieder

$$\Delta b_j^h = -\lambda \frac{\partial E_x}{\partial b_j^h} = -\lambda\, \delta_j.$$

Die Backpropagation wird als iterativer Algorithmus beschrieben. Daher tauchen im Algorithmus 3 Variablen in Abhängigkeit von n auf. Als Abbruchbedingung kann eine maximale Anzahl N von Iterationen vorgegeben werden. Andere Abbruchbedingun-

gen können mit der Norm der Abstiegsrichtungen oder abhängig von der Größe des Traingsfehlers mit einer Fehlertoleranz $\varepsilon > 0$ formuliert werden.

Algorithm 3 Online-Backpropagation für ein FFN, vgl. [17]

Require: Trainingsmenge $\mathcal{T}$, Modellparameter $\mathcal{W}_0$, Fehlerfunktion E, Lernrate λ
Ensure: optimierte Modellparameter $\mathcal{W}$
  Initialisiere zufällig alle Gewichte und Schwellwerte des FFN Λ
  $n = 0$
  while $E > \varepsilon$ **or** $n < N$ **do** ▷ Abbruchbedingung, siehe Text
    for $(x, c) \in \mathcal{T}$ **do**
      $y(n) = \Lambda(x(n))$
      $e_k(n) = y_k(n) - t_k(n)$
      Berechne die Gradienten bezüglich der Ausgabeschicht
      $\delta_k(n) = e_k(n)\psi'(V_k(n))$
      $\Delta w^l_{k,j}(n) = -\lambda\, \delta_k(n) z_j(n)$
      $\Delta b^l_k(n) = -\lambda\, \delta_k(n)$
      Berechne die Gradienten bezüglich der verdeckten Schicht
      $\delta_j(n) = \left(\sum_{k=1}^{s_l} \delta_k w^l_{k,j}\right) \psi'(V_j)$
      $\Delta w^h_{j,i}(n) = -\lambda\, \delta_j(n) x_i(n)$
      $\Delta b^h_j(n) = -\lambda\, \delta_j(n)$
      Aktualisiere Gewichte
      $\mathcal{W}_{n+1} = \mathcal{W}_n + \Delta\mathcal{W}_n$
      $n = n + 1$
    end for
    $E = \frac{1}{2} \sum_{(x,c)\in\mathcal{T}} ||y - t||_2^2$
  end while

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*[57], *AdaGrad*[15], *RMSProp*[58] sowie *Adam*[28]. So sollen Probleme wie des *vanishing gradients* oder des *exploding gradients* vermieden werden, siehe dazu[21]. Für eine tiefere Analyse des Gradientenverfahrens und dessen Varianten sei auf die jeweilgen Arbeiten beziehungsweise als Zusammenfassung auf Ruder[50] verwiesen. Darüber hinaus gibt es andere Techniken wie die Regularisierung, um die oben genannten Probleme zu entgehen. Für die Problemstellung in dieser Arbeit ist es ausreichend, das Online-Verfahren mit konstanter Lernrate zu nutzen.

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 Modellparameter 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 Fehlerfunktion E bzw. der großen Menge von Parametern $\mathcal{W}$ selten direkt angegeben werden[5]. Daher wird das Gradientenverfahren als iterativer Ansatz zur numerischen Minimierung der Fehlerfunktion genutzt. Dabei sind partielle Ableitungen der Fehlerfunktion bezüglich der Parameter nötig, welche im Backpropagationsalgorithmus 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 Parameteranzahl auf. Die Problemstellung 3 dieser Arbeit besteht in der Klassifikation digitalisierter Bilder. Wird ein FFN mit 100 Ausgabeneuronen genutzt und jedes Pixel eines Bildes mit den Abmessungen 1000×1000 als Merkmal genutzt, so ergeben sich bereits $10^8 + 100$ freie Parameter. Stehen nur relativ wenige Trainingsdaten zur Verfügung, ist die Struktur des FFN zu komplex und dies kann zur Überanpassung führen[10, 3]. Die Parameteranzahl muss also deutlich reduziert werden. Konzepte wie *Parameter Sharing* und spärliche Konnektivität, engl. *sparse connectivity*, erlauben diese Reduktion, vgl. Goodfellow[20] und werden in den folgenden Abschnitten erläutert.

Ein weiterer Nachteil des FFN ergibt sich dadurch, dass Korrelationen von benachbarten Eingabeneuronen, z.B. Bildsegmente wie Kanten oder Ecken nicht mit einbezogen werden. Es muss also ein 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[29, 32, 12] 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 Abschnitt 4.1 wird zunächst die Faltung als mathematische Operation eingeführt. Anschließend wird im Abschnitt 4.2 das CNN-Modell definiert und schließlich im Abschnitt 4.3 der Backpropagationsalgorithmus 3 auf CNN verallgemeinert.

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 26 (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*

$$\begin{aligned}
&(i) && f * g = g * f && (\textit{Kommutativität})\\
&(ii) && f * (g * h) = (f * g) * h && (\textit{Assoziativität})\\
&(iii) && f * (g + h) = f * h + f * g && (\textit{Distributivität})\\
&(iv) && a(f * g) = (af) * g = f * (ag) && (\textit{Assoziativität mit skalarer Multiplikation})
\end{aligned}$$

Beweis. Eine Beweis dieser Rechenregeln kann in Werner[61] gelesen werden. □

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

Definition 27 (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 den gesamten Definitionsbereich D summiert. Ist D beschränkt, werden f beziehungsweise g durch Nullen fortgesetzt.

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

Definition 28 (Matrixfaltung, vgl. [**gruening**])**.** *Für gegebene Matrizen $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ sei $l = \lfloor k/2 \rfloor$. Die zweidimensionale Faltung $Y = X * K \in \mathbb{R}^{h \times w}$ ist als*

$$Y_{i,j} := \sum_{u=-l}^{l} \sum_{v=-l}^{l} X_{i+u,j+v} K_{u+l+1,v+l+1} \quad \forall i \in [h], j \in [b] \tag{4.1}$$

mit $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [b]$ definiert. In der Literatur wird dieses Auffüllen mit Nullen am Rand von X mit zero padding bezeichnet. In dieser Definition besitzt das Ergebnis Y der Faltung, genauer der Kreuzkorrelation, die gleichen Abmessungen wie X. In der Literatur werden die Begriffe Faltung und Kreuzkorrelation oft für dieselbe Operation genutzt. In dieser Arbeit wird (4.1) als Faltung bezeichnet

Bemerkung 2. *Oft wird eine kompakte Schreibweise der Faltung von $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ mit*

$$Y_{i,j} = \sum_{u=-l}^{l} \sum_{v=-l}^{l} X_{i+u,j+v} K_{u,v} \tag{4.2}$$

und $X_{i,j} = 0$ für $i \notin [h]$ und $j \notin [b]$ angegeben. Dabei wird der Kern speziell indiziert. Ist beispielsweise $k = 3$, so ist $K \in \mathbb{R}^3$ durch

$$K = \begin{pmatrix} K_{-1,-1} & K_{-1,0} & K_{-1,1} \\ K_{0,-1} & K_{0,0} & K_{0,1} \\ K_{1,-1} & K_{1,0} & K_{1,1} \end{pmatrix}$$

gegeben.

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*

$$Y_{i,j} := \sum_{u=-l}^{l} \sum_{v=-l}^{l} X_{i \cdot s_h + u, j \cdot s_b + v} K_{u+l+1, v+l+1} \quad \forall i \in [\lceil h/s_h \rceil], j \in [\lceil b/s_b \rceil].$$

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, zum Beispiel besitzen Grauwert-Bilder einen Farbkanal, während RGB-Farbbilder drei Kanäle der Farben Rot usw. 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 29 (Gefaltete Übertragungsfunktion)**.** *Sei ein vierdimensionales Array $K \in \mathbb{R}^{z_{out} \times z_{in} \times k \times k}$ 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

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

wird gefaltete Übertragungsfunktion bezeichnet. Mit $*$ ist die Matrixfaltung wie in Definition 28 gemeint und mit $\cdot$ werden beliebige Raumachsen bezeichnet. Die Skalare α_p

können als lernbare Parameter genutzt werden. In dieser Arbeit gelte $\alpha_p = 1$ *für alle* $p \in [z_{in}]$.

Bemerkung 4. *Ist* $\psi : \mathbb{R} \to \mathbb{R}$ *eine Aktivierungsfunktion wie in Definition 18, 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 19 wird nun eine Faltungsschicht als Verknüpfung von gefalteter Übertragungsfunktion und Aktivierungsfunktion definiert.

Definition 30 (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 sogenannte Eingabekarte* $X \in \mathbb{R}^{\cdot \times \cdot \times z_{in}}$ *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. Weiter bezeichne* $\Psi_{conv}^{K,b,\psi}$ *die Faltungsschicht* $\mathcal{S}_{conv}$ *mit* $\Psi_{conv}^{K,b,\psi}(X) := \psi\left(\Psi_{conv}^{K,b}(X)\right)$.

Bei CNN werden sogenannte *Pooling*-Schichten verwendet, um die Dimensionen der Raumachsen neben dem zero padding weiter zu verkleinern und das Modell robuster gegenüber Überanpassung zu machen. Dazu werden Pooling-Funktionen eingesetzt, welche unabhängig voneinander auf Merkmalskarten operieren und so die Rechenkomplexität des Modells reduzieren. Es ist sinnvoll, symmetrische Pooling-Funktionen T zu wählen, für die die Bedingung

$$\forall \pi \in S_n \ \forall x \in \mathbb{R}^n : T(x_1, \ldots, x_n) = T(x_{\pi(1)}, \ldots, x_{\pi(n)})$$

gilt. Mögliche Pooling-Funktionen sind

$$\begin{aligned} \text{Maximum}: \ T(x_1, \ldots x_n) &= \max\{x_1, \ldots x_n\}, \\ \text{Mittelwert}: \ T(x_1, \ldots, x_n) &= \frac{1}{n} \sum_{i=1}^{n} x_i. \end{aligned}$$

Für den späteren Trainingsprozess ist es nützlich, die Ableitung der verwendeten Pooling-Funktionen, sofern sie existiert, zur Verfügung zu haben. Für den Mittelwert T gilt $\nabla T = \frac{1}{n}\mathbf{1}$. Ist T die Maximum-Funktion so ergibt sich

$$\frac{\partial T}{\partial x_i} = \begin{cases} 1, \text{ falls } \forall j \neq i : \ x_i > x_j \\ 0, \text{ falls } \exists j \neq i : \ x_i < x_j \end{cases}$$

mit der Konvention, dass für $x_1 = x_2 = \ldots = x_n$ die Ableitung auf $\frac{1}{2}$ gesetzt wird. Pooling-Schichten werden wieder durch Schrittweiten parametrisiert.

Definition 31 (Pooling-Schicht, vgl. Grüning [**gruening**])**.** *Seien $p_h, p_b \in \mathbb{N}$ und T eine Pooling-Funktion. Die Funktion*

$$\Psi_{pool,T}^{p_h,p_b} : \mathbb{R}^{\cdot\times\cdot\times z} \to \mathbb{R}^{\cdot\times\cdot\times z}$$

mit

$$\Psi_{pool,T}^{p_h,p_b}(X)_{i,j,l} := \underset{\substack{(i-1)\cdot p_h < i' \leq \min\{i\cdot p_{h,1} X\} \\ (j-1)\cdot p_w < j' \leq \min\{j\cdot p_{w,2} X\}}}{T} (X_{i',j',l})$$

für alle $i \in [\lceil {}_1X/p_h \rceil], j \in [\lceil {}_2X/p_w \rceil]$ und $l \in [z]$ wird Pooling-Schicht genannt. Die Schrittweiten $p_h, p_b \in \mathbb{N}$ werden subsampling-Faktoren genannt.

Pooling-Schichten verdichten also Information von Eingabedaten, welche sich lokal in Fenstern der Größe $p_h \times p_b$ befinden und reduzieren so die Raumdimension. In dieser Arbeit wird das Maximum-Pooling oder Mittelwert-Pooling benutzt. Für andere Pooling-Funktion sei auf Yu et. al.[66] verwiesen. Die Idee bei CNN besteht nun darin, Faltungsschichten mit Pooling-Schichten zu kombinieren und schließlich ein FFN wie in Definition 20 anzuknüpfen. Dazu wird für allgemeine mehrdimensionale Arrays die Flatten-Schicht definiert.

Definition 32. *Sei $X \in \mathbb{R}^{n_1\times n_2\times n_3}$. Dann wird die Funktion $T_f : \mathbb{R}^{n_1\times n_2\times n_3} \to \mathbb{R}^{n_1\cdot n_2\cdot n_3}$ mit*

$$T_f(X)_{(i-1)\cdot(n_2\cdot n_3)+(j-1)\cdot n_3+k} := X_{i,j,k}, \quad \forall i \in [n_1],\ j \in [n_2],\ k \in [n_3]$$

Flatten-Funktion genannt. Die multidimensionale Eingabe wird also in einen Vektor umgewandelt.

Dies erlaubt die Definition des CNN-Modells als Kombination aus Faltungsschichten, Pooling-Schichten und Neuronenschichten des MLP aus Kapitel 3.

Definition 33. *(Gefaltetes Neuronales Netz) Seien $h, w, z_{in}, z_{out}, l, c \in \mathbb{N}$ und Schichten $\Psi^{W^{(1)},b^{(1)},\psi_1}, \ldots, \Psi^{W^{(l)},b^{(l)},\psi_l}$ sowie Faltungsschichten $\Psi_{conv}^{K^{(1)},b^{(1)},\psi_1}, \ldots, \Psi_{conv}^{K^{(l)},b^{(l)},\psi_l}$ gegeben. Die Vorwärtsrechnung eines CNN lässt sich als Komposition*

$$y = \Psi^{W^{(l)},b^{(l)},\psi_l} \circ \ldots \circ \Psi^{W^{(1)},b^{(1)},\psi_1} \circ T_f \circ \Psi_{conv}^{K^{(c)},b^{(c)},\psi_c} \circ \ldots \circ \Psi_{conv}^{K^{(1)},b^{(1)},\psi_l}(X) \tag{4.3}$$

darstellen. Dabei seien wieder die Dimensionen der Parameter passend gewählt, sodass die Komposition wohldefiniert ist. In (4.3) können zwischen den Faltungsschichten Pooling-Schichten $\Psi_{pool,T}^{p_h,p_b}(X)$ geschaltet werden.

Auch CNN bestehen aus Hyper- und Modellparametern.

Definition 34 (Hyper- und Modellparameter von CNN)**.** *Die Eingabe- und Ausgabedimensionen h, b, z_{in}, z_{out}, die Anzahl c der Faltungsschichten, die Anzahl l der Neuronenschichten, die Dimensionen der Kerne, die Schrittweiten bei der Faltung bzw. dem Pooling sowie die verwendeten Aktivierungs- und Poolingfunktionen sind Hyperparameter des CNN. Die Kerne, Gewichtsmatrizen und Biasvektoren mit den entsprechend passenden Abmessungen stellen die Modellparameter $\mathcal{W} := \{(W^{(i)}, b^{(i)}) :\ i = 1, \ldots, l\}$ und $\mathcal{W}_{conv} := \{(K^{(i)}, b^{(i)}) :\ i = 1, \ldots, c\}$ des CNN dar.*

Die Hyperparameter werden wieder anwendungsspezifisch für das jeweilige Problem gewählt und die Modellparameter während des Trainingsprozesses angepasst.

4.3. Backpropagation bei CNN

Der Backpropagationsalgorithmus 3 soll nun für das CNN-Modell verallgemeinert werden. Als Fehlerfunktion wird wieder die mittlere quadratische Abweichung (3.11) genutzt. Die Abstiegsrichtungen für die Schichten des FFN wurden bereits im vorherigen Kapitel 3 hergeleitet. Es müssen nun Gradienten innerhalb der Faltungsschichten beziehungsweise Pooling-Schichten berechnet werden. Die Einträge der Eingabe- und Merkmalskarten, in Zukunft einfach Karten genannt, können als Neuronenaktivierungen interpretiert werden. Weiter werden zur Vereinfachung die trainierbaren Gewichte der Kerne und Gewichtsmatrizen mit $w_{j,i}^{(l)}$ bezeichnet. Gemeint ist also das Gewicht zwischen Neuron i auf (Faltungs)-Schicht $l-1$ und Neuron j der (Faltungs)-Schicht l bezeichnet. Weiter sei $y_i^{(l-1)}$ die Aktivierung des Neurons i der (Faltungs)Schicht $l-1$ und $\delta_j^{(l)}$ sei der lokale Fehler des Neurons j der (Faltungs)Schicht l. Die Funktion ψ repräsentiere eine Aktivierungsfunktion, Pooling-Funktion oder Flatten-Funktion, je nachdem auf welcher Schicht sie benutzt wird.

Die Gradienten werden wieder komponentenweise berechnet. Angenommen, l ist eine Faltungsschicht. Die Aktivierung einer Merkmalskarte j an der Stelle (x, y) lässt sich mit der Matrixfaltung, siehe 28, 29, als

$$y_j^{(l)}(x,y) = \psi\left(\sum_{i=1}^{z_{in}} \sum_{(u,v)\in F} y_i^{(l)}(x+u, y+u)\, w_{j,i}^{(l)}(u,v) + b_j^{(l)}\right) \tag{4.4}$$

schreiben. Es ist zu beachten, dass (x, y) ein Pixel der Karte j meint. Mit k als Dimension des Kerns und $l = \lfloor k/2 \rfloor$ ist $F = \{(u, v) : \ -l \leq u, v \leq l\}$. Das Gewicht des Kerns von der Karte i zur Karte j an der Stelle (u, v) ist mit $w_{j,i}^{(l)}(u, v)$ bezeichnet. Der Gradient ergibt sich als Summe über alle beteiligten Pixel (x, y) der Merkmalskarte j, also

$$\Delta w_{j,i}^{(l)}(u,v) = \sum_{(x,y)} \left(y_i^{(l-1)}(x+u, y+v)\delta_j^{(l)}(x,y)\right) \tag{4.5}$$

Der Zusammenhang in Gleichung (4.5) ist in Abbildung **??** grafisch dargestellt. Analog ergibt sich für den Schwellwert

$$\Delta b_j^{(l)} = -\lambda \sum_{(x,y)} \delta_j^{(l)}(x,y).$$

Die Berechnung des lokalen Fehlers $\delta_j^{(l)}$ der Schicht l ist abhängig von der Schicht $l+1$. Auch hier wird wieder komponentenweise vorgegangen. Ist die Schicht $l+1$ eine Neuronenschicht mit s_k Neuronen, wird Gleichung 3.3.2 zu

$$\delta_j^{(l)}(x,y) = \sum_{k=1}^{s_k} \left(\sum_{(x,y)} \delta_k^{(l+1)} w_{k,j}^{(l+1)}(x,y)\right)$$

verallgemeinert. Ist die nachfolgende Schicht eine Faltungsschicht, so ergibt sich der lokale Fehler der Schicht l zu

$$\delta_j^{(l)}(x,y) = \sum_{k=1}^{z_{out}} \left(\sum_{(u,v)\in F} \delta_k^{(l+1)}(x,y) w_{k,j}^{(l+1)}(u,v) \right)$$

Ist $l+1$ eine Pooling-Schicht mit Schrittweiten p_h, p_b und einer Pooling-Funktion T, so gilt

$$\delta_j^{(l)}(x,y) = \delta_k^{(l+1)}(\lceil x/p_h \rceil, \lceil y/p_b \rceil) \nabla T(j)$$

Dieses Vorgehen wird auch Upsampling genannt.

!!!!!!!

!!! noch als algorithmus aufschreiben und mit dem folgenden Beispiel vergleichen, ob Indizies/lokale Fehler usw. richtig sind, lokale Fehler einführen, Konsistenz zu Kapitel 3 überprüfen!!!

!!!!!!

4.4. Anwendung bei der Ziffernerkennung

In diesem Abschnitt wird ein CNN zur Klassifikation von Grauwert-Bildern aus einem konkreten Datensatz vorgestellt. Der MNIST-Datensatz[32] bietet 60.000 Trainingsbilder handgeschriebener Ziffern und 10.000 Testbilder, welche jeweils durch menschliches Wissen annotiert sind. Dieser Datensatz gilt als typischer Benchmark zur Klassifikation von Ziffern und wird im weiteren Verlauf dieser Arbeit genutzt. Jedes einzelne Bild besteht aus 28×28 Pixeln, welche jeweils einen Grauwert zwischen 0 und 1 annehmen, vgl. Abbildung 4.1. Ein Bild wird als $X \in [0,1]^{28\times 28\times 1}$ und ein Trainingspaar als (X, c) mit $c \in \{0, \ldots, 9\}$ bezeichnet.

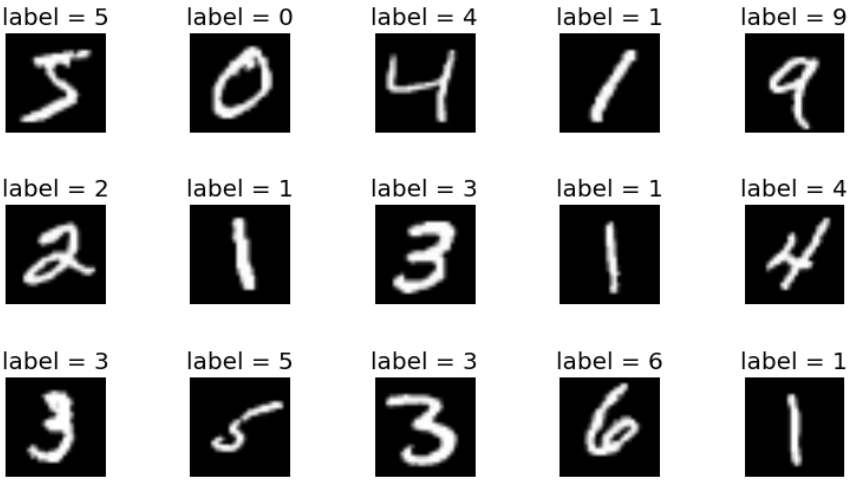

Abbildung 4.1.: Beispielbilder, vgl. [32] aus der öffentlichen MNIST-Datenbank. Zu sehen sind handgeschriebene Ziffern und zugehörige Annotationen.

Es wird ein CNN mit zwei Faltungsschichten C^1 und C^2 mit der logistischen Aktivierungsfunktion ψ genutzt. Die Ausgabe der Schicht C^1 umfasst sechs Merkmalskarten und die Ausgabe der Schicht C^2 zwölf Merkmalskarten. Zur Faltung werden quadratische 5×5 -Kerne verwendet. Außerdem werden zwei Pooling-Schichten P^1 und P^2 mit den Schrittweiten $p_h = p_b = 2$ und der Mittelwert-Funktion genutzt. Anschließend wird ein FFN mit 10 Ausgabeneuronen angekoppelt, sodass dessen Aktivierung zur Klassifikation der Eingabe genutzt werden kann. Die Architektur, ähnlich dem Le-Net-5-Modell[32], ist in Abbildung **??** dargestellt. Dabei sind

- X ein Grauwert-Bild der Größe $h = b = 28$,
- $K^1 \in \mathbb{R}^{6\times1\times5\times5}$ der Kern der Schicht C^1,
- $b^1 \in \mathbb{R}^6$ der Biasvektor der Schicht C^1,
- $K^2 \in \mathbb{R}^{12\times6\times5\times5}$ der Kern der Schicht C^2,
- $b^2 \in \mathbb{R}^{12}$ der Biasvektor der Schicht C^2,
- $W \in \mathbb{R}^{192\times10}$ die Gewichtsmatrix der Neuronenschicht,
- $b \in \mathbb{R}^{10}$ der Biasvektor der Neuronenschicht,
- $y \in \mathbb{R}^{10}$ die Ausgabe des CNN.

Im Folgenden wird der Online-Backpropagationsalgorithmus **??** verwendet, um das CNN zu trainieren. Dieser besteht zunächst aus der Initialisierung und der Vorwärtsrechnung. Zur Übersicht wird im Folgenden eine komponentenweise Schreibweise genutzt.

Vorwärtsrechnung

Alle Biasvektoren werden als Nullvektor initialisiert. Zur Initialisierung der Gewichte wird die Xavier-Initialisierung **??** genutzt, also

$$\begin{aligned}
K^1_{p,1}(u,v) &\sim \mathcal{N}\left(0, \frac{2}{5\cdot5\cdot(1+6)}\right), && \forall p \in [6],\\
K^2_{q,p}(u,v) &\sim \mathcal{N}\left(0, \frac{2}{5\cdot5\cdot(6+12)}\right), && \forall q \in [12], \forall p \in [6],\\
W(i,j) &\sim \mathcal{N}\left(0, \frac{2}{192+10}\right), && \forall i \in [192], \forall j \in [10].
\end{aligned}$$

Die Parameteranzahl ist damit

$$\underbrace{(5\cdot5+1)\cdot6}_{\text{Gewichte in } C^1} + \underbrace{(5\cdot5\cdot6+1)\cdot12}_{\text{Gewichte in } C^2} + \underbrace{(192+1\cdot10)}_{\text{Gewichte im FFN}} = 3898.$$

Sei $X \in \mathbb{R}^{28\times\times 1}$ das Eingabebild und ψ die logistische Funktion. In der Schicht C^1 werden sechs Merkmalskarten

$$\begin{aligned} C_p^1 &= \psi(X * K_{p,1}^1 + b_p^1 \mathbf{1}) \\ C_p^1(i,j) &= \psi\left(\sum_{u=-2}^{2}\sum_{v=-2}^{2} X(i+u, j+v)\cdot K_{p,1}^1(u,v) + b_p^1\right) \end{aligned} \tag{4.6}$$

für $1 \leq p \leq 6$ berechnet. Mit $\mathbf{1} \in \mathbb{R}^{24\times 24}$ ist die Matrix gemeint, deren Einträge nur Einsen sind. Die Matrixfaltung $*$ wird ohne zero padding genutzt und daher reduziert sich die Raumdimension. Es gilt $C_p^1 \in \mathbb{R}^{24\times 24\times 1}$. Anschließend folgen das Mittelwert-Pooling P^1 mit

$$P_p^1(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_p^1(2i-u, 2j-v),\ i,j \in [12] \tag{4.7}$$

für $1 \leq p \leq 6$ und die zweite Faltungsschicht C^2, welche zwölf Merkmalskarten

$$\begin{aligned} C_q^2 &= \psi\left(\sum_{p=1}^{6} P_p^1 * K_{q,p}^2 + \mathbf{1} b_q^2\right) \\ C_q^2(i,j) &= \psi\left(\sum_{p=1}^{6}\sum_{u=-2}^{2}\sum_{v=-2}^{2} P_p^1(i+u, j+v)\cdot K_{q,p}^2(u,v) + b_q^2\right) \end{aligned} \tag{4.8}$$

für $1 \leq q \leq 12$ berechnet. Die Matrixfaltung $*$ wird wieder ohne zero padding genutzt und daher reduziert sich die Raumdimension. Es gilt $C_q^2 \in \mathbb{R}^{8\times 8\times 1}$. Es folgt das Mittelwert-Pooling P^2 mit

$$P_q^2(i,j) = \frac{1}{4}\sum_{u=0}^{1}\sum_{v=0}^{1} C_p^1(2i-u, 2j-v),\ i,j \in [4] \tag{4.9}$$

für $1 \leq q \leq 12$. Es gilt $P^2 \in \mathbb{R}^{4\times 4\times 12}$. Diese zwölf 4×4- Matrizen werden durch die Flatten-Funktion T_f aus Definition 32 in einen Vektor $f \in \mathbb{R}^{4\cdot 4\cdot 12}$ umgewandelt. Dies sei durch

$$f = T_f(P^2) \tag{4.10}$$

beschrieben. Die Umkehrung wird mit

$$P^2 = T_f^{-1}(f). \tag{4.11}$$

bezeichnet. Die Ausgabe des FFN ergibt sich schließlich zu

$$y = \psi(W^T f + b). \tag{4.12}$$

und der mittlere quadratische Fehler lautet

$$E_X = \frac{1}{2} \sum_{k=1}^{10} \left(y(k) - t(k) \right)^2 , \tag{4.13}$$

wobei $t = t(X, c)$ der Zielvektor des Datenpaars (X, c) ist.

Backpropagation

Zuerst werden die Gradienten bezüglich der Gewichtsmatrix und Biasvektor der Neuronenschicht des FNN berechnet. Es gilt

$$\begin{aligned} \Delta W(j,k) &= -\lambda \frac{\partial E}{\partial W(j,k)} \\ &= -\lambda \frac{\partial L}{\partial y(k)} \cdot \frac{\partial y(k)}{\partial W(j,k)} \\ &= -\lambda \left(y(k) - t(k) \right) \cdot \frac{\partial}{\partial W(j,k)} \psi \left(\sum_{j=1}^{192} W(j,k) f(j) + b(k) \right) \\ &= -\lambda \left(y(k) - t(k) \right) \cdot y(k)(1 - y(k)) \cdot f(j). \end{aligned} \tag{4.14}$$

Mit $\delta^{\mathrm{FFN}}(k) = (y(k) - t(k)) \cdot y(k)(1 - y(k))$ für $1 \leq k \leq 10$ lässt sich ΔW als dyadisches Produkt

$$\begin{aligned} \Delta W(j,k) &= -\lambda \left(\delta^{\mathrm{FFN}}(k) \cdot f(j) \right) \\ \Rightarrow \Delta W &= -\lambda \left(f \otimes (\delta^{\mathrm{FFN}})^T \right) \end{aligned}$$

darstellen. Analog gilt

$$\Delta b = -\lambda \delta^{\mathrm{FFN}}.$$

Um $\Delta K^2_{q,p}$ zu bestimmen, ist zunächst der lokale Fehler der darüberliegenden Neuronenschicht zu ermitteln. Der lokale Fehler $\delta^f \in \mathbb{R}^{192}$ wird mithilfe des lokalen Fehlers

des FNN berechnet und lautet

$$\begin{aligned}
\delta^f(j) &= \frac{\partial E}{\partial f} \\
&= \sum_{k=1}^{10} \frac{\partial E}{\partial y(k)} \cdot \frac{\partial y(k)}{\partial f(j)} \\
&= (y(k) - t(k)) \cdot \frac{\partial}{\partial f(j)} \psi \left(\sum_{j=1}^{192} W(k,j) f(j) + b(k) \right) \\
&= -\sum_{k=1}^{10} (y(k) - t(k)) \cdot y(k)(1 - y(k)) \cdot W(k,j) \\
&= \sum_{k=1}^{10} \delta^{\mathrm{FFN}}(k) \cdot W(k,j) \\
\Rightarrow \delta^f &= W \delta^{\mathrm{FFN}}.
\end{aligned}$$

In der Pooling-Schicht P^2 sind keine Gradienten zu bestimmen, da diese nicht von den Modellparametern abhängt. Mit

$$\delta^{P^2} = T_f^{-1}(\delta^f) \in \mathbb{R}^{4 \times 4 \times 12}$$

folgt mit der Mittelwert-Funktion $\phi : \mathbb{R}^4 \to \mathbb{R}^4$ und $\nabla \phi = \frac{1}{4}\mathbf{1}$ das Upsampling

$$\delta_q^{C^2}(i,j) = \delta_q^{P^2}\left(\lceil i/2 \rceil, \lceil j/2 \rceil\right) \cdot \frac{1}{4} \quad \forall i,j \in [8], \forall q \in [12].$$

Es gilt $\delta^{C^2} \in \mathbb{R}^{8 \times 8 \times 12}$. Nun kann $\Delta K_{q,p}^2$ an einer Stelle (u,v) als

$$\begin{aligned}
\Delta K_{q,p}^2(u,v) &= -\lambda \frac{\partial E}{\partial K_{q,p}^2(u,v)} \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \frac{\partial E}{\partial C_q^2(i,j)} \cdot \frac{\partial C_q^2(i,j)}{\partial K_{q,p}^2(u,v)} \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \delta_q^{C^2}(i,j) \cdot \frac{\partial}{K_{q,p}^2(u,v)} \psi \left(\sum_{p=1}^{6} \sum_{u=0}^{4} \sum_{v=0}^{4} P_p^1(i+u, j+v) \cdot K_{q,p}^2(u,v) + b_q^2 \right) \\
&= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \delta_q^{C^2}(i,j) \cdot C_q^2(i,j) \left(1 - C_q^2(i,j)\right) \cdot P_p^1(i+u, j+v)
\end{aligned}$$

für $1 \leq q \leq 12, 1 \leq p \leq 6$ berechnet werden. Bezeichne

$$\begin{aligned}
\delta_{q,\psi}^{C^2}(i,j) :&= \delta_q^{C^2}(i,j) \cdot C_q^2(i,j) \left(1 - C_q^2(i,j)\right) \\
&= \delta_q^{C^2}(i,j) \cdot \psi'(C_q^2(i,j)), \quad \forall i,j \in [8], \forall q \in [12]
\end{aligned}$$

den lokalen Fehler $\delta^{C^2}_{q,\psi}$ der Faltungschicht C^2. Damit ergibt sich

$$C^2_{q,\psi}(i,j) = \sum_{p=1}^{6} \sum_{u=-2}^{2} \sum_{v=-2}^{2} P^1_p(i+u, j+v) \cdot K^2_{q,p}(u,v) + b^2_q.$$

Es gilt

$$\begin{aligned} \Delta K^2_{q,p}(u,v) &= -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} P^1_p(u+i, v+j) \cdot \delta^{C^2}_{q,\psi}(i,j) \\ \Rightarrow \Delta K^2_{q,p} &= -\lambda \, P^1_p * \delta^{C^2}_{q,\psi}, \quad \forall q \in [12], \forall p \in [6]. \end{aligned}$$

Dies ist besonders für die effiziente Implementierung der Rückwärtsrechnung nützlich, da Abstiegsrichtungen wieder mithilfe von Faltungsoperationen ausgedrückt werden können. Es muss also besonders die Faltung perfomant implementiert werden. Analog ergibt sich

$$\Delta b^2_q = -\lambda \sum_{i=1}^{8} \sum_{j=1}^{8} \delta^{C^2}_{q,\psi}(i,j), \quad \forall q \in [12].$$

Um $\Delta K^1_{p,1}$ zu bestimmen, ist zunächst der lokale Fehler der darüberliegenden Pooling-Schicht zu ermitteln. Dieser lässt sich wieder mit dem lokalen Fehler $\delta^{C^2}_{q,\psi}$ der Faltungsschicht C^2 ausdrücken. Es gilt

$$\begin{aligned} \delta^{P^1}_p(i,j) &= \frac{\partial E}{\partial P^1_p(i,j)} \\ &= \sum_{q=1}^{12} \sum_{u=-2}^{2} \sum_{v=-2}^{2} \frac{\partial E}{\partial C^2_{q,\psi}(i-u, j-v)} \cdot \frac{\partial C^2_{q,\psi}(i-u, j-v)}{\partial P^1_p(i,j)} \\ &= \sum_{q=1}^{12} \sum_{u=-2}^{2} \sum_{v=-2}^{2} \delta^{C^2}_{q,\psi}(i-u, j-v) \cdot \frac{\partial}{\partial P^1_p(i,j)} \left(\sum_{p=1}^{6} \sum_{u=-2}^{2} \sum_{v=-2}^{2} P^1_p(i,j) \cdot K^2_{q,p}(u,v) + b^2_q \right) \\ &= \sum_{q=1}^{12} \sum_{v=-2}^{2} \sum_{u=-2}^{2} \delta^{C^2}_{q,\psi}(i-u, j-v) \cdot K^2_{q,p}(u,v), \quad \forall i,j \in [12], \forall p \in [6] \end{aligned}$$

Mit $K^2_{q,p,rot180}(-u,-v) := K^2_{q,p}(u,v)$ ergibt sich

$$\begin{aligned} \delta^{P^1}_p(i,j) &= \sum_{q=1}^{12} \sum_{v=-2}^{2} \sum_{u=-2}^{2} \delta^{C^2}_{q,\psi}(i+(-u), j+(-v)) \cdot K^2_{q,p,rot180}(-u,-v) \\ \Rightarrow \delta^{P^1}_p &= \sum_{q=1}^{12} \delta^{C^2}_{q,\psi} * K^2_{q,p,rot180}, \quad \forall p \in [6]. \end{aligned}$$

Es gilt $\delta^{P^1} \in \mathbb{R}^{12\times12\times6}$. Mit dem Upsampling

$$\delta^{C^1}_p(i,j) = \delta^{P^1}_p(\lceil i/2 \rceil, \lceil j/2 \rceil) \cdot \frac{1}{4}, \quad \forall i,j \in [24], \forall p \in [6]$$

kann nun der Gradient $\Delta K^1_{p,1}$ an einer Stelle (u, v) als

$$\begin{aligned}
\Delta K^1_{p,1}(u,v) &= -\lambda \frac{\partial E}{\partial K^1_{p,1}(u,v)} \\
&= -\lambda \sum_{i=1}^{24}\sum_{j=1}^{24} \frac{\partial E}{\partial C^1_p(i,j)} \cdot \frac{\partial C^1_p(i,j)}{\partial K^1_{p,1}(u,v)} \\
&= -\lambda \sum_{i=1}^{24}\sum_{j=1}^{24} \delta^{C^1}_p(i,j) \cdot \frac{\partial}{K^1_{p,1}(u,v)} \psi\left(\sum_{u=-2}^{2}\sum_{v=-2}^{2} X(i+u, j+v) \cdot K^1_{p,1}(u,v) + b^1_p\right) \\
&= -\lambda \sum_{i=1}^{24}\sum_{j=1}^{24} \delta^{C^1}_p(i,j) \cdot C^1_p(i,j)\left(1 - C^1_p(i,j)\right) \cdot X(i+u, j+v).
\end{aligned}$$

für $1 \leq p \leq 6$ berechnet werden. Sei wieder

$$\begin{aligned}
\delta^{C^1}_{p,\psi}(i,j) :&= \delta^{C^1}_p(i,j) \cdot C^1_p(i,j)\left(1 - C^1_p(i,j)\right) \\
&= \delta^{C^1}_p(i,j) \cdot \psi'(C^1_p(i,j)), \quad \forall i,j \in [8], \forall p \in [6].
\end{aligned}$$

Dann gilt schließlich

$$\begin{aligned}
\Delta K^1_{p,1}(u,v) &= -\lambda \sum_{i=1}^{24}\sum_{j=1}^{24} X(u+i, v+j) \cdot \delta^{C^1}_{p,\psi}(i,j) \\
\Rightarrow \Delta K^1_{p,1} &= -\lambda\, X * \delta^{C^1}_{p,\psi}, \quad \forall p \in [6]
\end{aligned}$$

und

$$\Delta b^1_p = -\lambda \sum_{i=1}^{24}\sum_{j=1}^{24} \delta^{C^1}_{p,\psi}(i,j), \quad \forall p \in [6].$$

4.5. Motivation der Faltung

Weiter machen! optionaler Abschnitt falls noch Zeit ist(wenn nicht im Einleitungstext des kapitels entfernen!!) 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

5. Datenbankgestützte Implementierung von CNN

In diesem Kapitel werden Ideen zur Umsetzung der datenbankgestützten Mustererkennung durch tranierte gefaltete neuronale Netze vorgestellt. Der Schlüssel liegt dabei in der effektiven Implementierung der Faltungsoperation als SQL-Anfrage. Gelingt dies, so kann die Vorwärtsrechnung eines CNN durch Komposition von Faltungs-, Pooling- und Matrixvektoroperationen umgesetzt werden. Im Abschnitt 5.1 werden drei Implementierungsmöglichkeiten der Matrixfaltung, siehe 28, in SQL beleuchtet. Dabei stellt sich heraus, dass es sich lohnt, die diskrete Faltung als lineare Operationen mithilfe der in Kapitel 2 vorgestellten Basisoperationen umzusetzen. So gelingt es, die Problemstellung 1 zu beantworten und auch für relativ große Grauwertbilder akzeptable Laufzeiten zu erreichen.

Im Abschnitt 5.1.3 wird die Vorwärtsrechnung eines FFN als Komposition von affin linearen Transformationen mit dem SQL-Anfragekern dargestellt. Zusammen mit den Ergebnissen aus dem vorherigen Abschnitt wird hinsichtlich Problem 3im Abschnitt 5.2 eine (objekt-) relationale Umsetzung der Vorwärtsrechnung für das tranierte Modell **??** zur Ziffernerkennung vorgestellt. Dies spiegelt zugleich das Hauptresultat dieser Arbeit wieder, welches schließlich im Abschnitt 5.3 evaluiert wird.

5.1. Die Faltungsoperation in SQL

Die Faltung zweier zeitdiskrete Signale stellt die Kernoperation innerhalb von CNN dar. Sind die entsprechenden Signale zweidimensional wie in Problemstellung 1, so muss die Matrixfaltung $X * K$ für $X \in \mathbb{R}^{h \times b}$ und $K \in \mathbb{R}^{k \times k}$ mithilfe des im Abschnitt 2.2 vorgestellten SQL-Kerns umgesetzt werden. Dazu werden in diesem Abschnitt drei Varianten vorgestellt, welche das Coordinate-Schema bzw. das Spaltenkompression-Schema zur Darstellung der Matrizen X und K nutzen.

5.1.1. Faltung mit Nachbarschaften

Der erste Ansatz beruht auf den Nachbarschaftsbeziehungen der Pixel innerhalb der Matrix X, die durch die Faltung mit einem Kern K mit den Abmessungen $k \times k$ gegeben sind. Dazu werden einige Bezeichnungen im Folgenden eingeführt. Seien $l = \lfloor k/2 \rfloor$ und die Mengen

$$N := \{(i, j) \ : \ 1 \leq i \leq b, 1 \leq j \leq h\}$$

sowie

$$F := \{(i,j) \ : \ -l \leq i \leq l, -l \leq j \leq l\}$$

gegeben, welche die Positionen der Matrizen X und K widerspiegeln. Hier wird wieder die spezielle Indizierung des Kerns K wie in Bemerkung 2 genutzt. Nun kann für jeden Pixel (i,j) von X eine Umgebung $U(i,j)$ in Abhängigkeit von F definiert werden, und zwar

$$U(i,j) := \{(i',j') \ : \ (i'-i, j'-j) \in F\}. \tag{5.1}$$

Die Menge $U(i,j)$ wird auch als Nachbarschaft des Pixels (i,j) bezeichnet. Die Matrixfaltung $Y = X * K$ lässt sich mit den Nachbarschaften (5.1) durch

$$Y_{i,j} = \sum_{(i',j') \in U(i,j)} X_{i',j'} K_{i'-i,j'-j}, \quad \forall i \in [h], \forall j \in [b] \tag{5.2}$$

berechnen. Dabei wird X außerhalb des Definitionsbereiches mit Nullen aufgefüllt, sodass das Ergebnis Y wieder die gleichen Abmessungen wie X besitzt.

Zur Umsetzung der Faltungsoperation sind also zunächst die Nachbarschaften für jeden Pixel von X zu berechnen. In SQL kann dies mithilfe des kartesischem Produkts implementiert werden. Dazu bezeichnen **X** und **K** die Relation zur Darstellung der Matrizen X und K im Coordinate-Schema. Eine rein relationale Umsetzung von (5.2) ist durch die SQL-Anfrage 5.1 gegeben. Zur Übersicht sind die Attributbezeichnungen groß geschrieben und die SQL-Klausel blau gekennzeichnet.

Listing 5.1: SQL-Code zur Umsetzung der Faltung mit Nachbarschaften

```
1  SELECT KREUZ.I,
2    KREUZ.J,
3    SUM(KREUZ.VSTRICH * F.V) AS V
4  FROM
5    (SELECT X1.I AS I,
6        X1.J AS J,
7        X1.V AS V,
8        X2.I AS ISTRICH,
9        X2.J AS JSTRICH,
10       X2.V AS VSTRICH
11     FROM X X1
12     CROSS JOIN X X2
13     WHERE X2.I - X1.I BETWEEN -l AND l
14       AND X2.J - X1.J BETWEEN -l AND l ) KREUZ JOIN
15     K
16 ON K.I = KREUZ.ISTRICH - KREUZ.I + (l+1)
17    AND K.J = KREUZ.JSTRICH - KREUZ.J + (l+1)
18 GROUP BY KREUZ.I, KREUZ.J
```

In der Teilanfrage (Zeile 4-14) werden die Nachbarschaften aller Pixel in der temporären **KREUZ** Relation berechnet. Dabei wird die **BETWEEN**-Funktion zur kompakten Darstellung der Konstantenselektion bezüglich $l = \lfloor k/2 \rfloor$ genutzt. In den Zeilen 15-17 werden dann die Nachbarschaften über die entsprechenden Indizies der Filtermatrix verbunden und schließlich die **SUM**-Funktion genutzt, um das Faltungsergebnis zu

berechnen. Dieser naive Ansatz nutzt keine lineare Algebra in Form von Matrixvektor- oder Matrixmatrixmultiplikation und ist hinsichtlich des Problems 1 schon für verhältnismäßig kleine Grauwertbilder ineffizient. Zu Erkennen ist dies in der Abbildung **??**, bei der die Laufzeiten der SQL-Anfrage 5.1 in Abhängigkeit von der Dimension n für allgemeine Matrizen $X \in [0,1]^{n \times n}$ dargestellt ist.

TODO Grafik ergebnis erklären.

Eine Verbesserung der Laufzeiten kann mithilfe von Umsetzungstabellen, engl. *Lookup tables*, erreicht werden. Da die Dimensionen aller vorkommenden Merkmalskarten und Kerne eines CNN von Anfang an durch die Wahl der Hyperparameter festgelegt werden, müssen die Nachbarschaften für die Faltung und das Pooling nur einmalig berechnet werden. Dies kann zudem vor der eigentlichen Erkennungsphase durchgeführt werden. So wird zwar der Speicheraufwand erhöht, aber der Zeitaufwand hinsichtlich der Faltungs- und Poolingoperation deutlich vermindert. Dazu werden pro Faltungs- und Poolingschicht jeweils eine Umsetzungstabelle **U** in der Form

$$\begin{aligned}\mathbf{U}(&\underline{i} \ \text{int},\\ &\underline{j} \ \text{int},\\ &\underline{\text{id}} \ \text{int},\\ &\text{istrich} \ \text{int},\\ &\text{jstrich} \ \text{int})\end{aligned}$$

mit dem zusammengesetzten Schlüssel (i, j, id) benötigt. Hier werden für jeden Pixel (i, j) die Nachbarpixel $(i', j') \in U$ mit einer entsprechenden Nummer id hinterlegt. So wird die zeitintensive Berechnung der Nachbarschaften mit dem kartesischen Produkt in der Anfrage 5.1 umgangen. Die Laufzeiten dieser Verbesserung sind in Abbildung **??** dargestellt.

TODO Grafik ergebnis erklären.

5.1.2. Faltung als Matrixvektorprodukt

Zwei beliebige Funktionen $f, g : D \to \mathbb{R}$ mit endlichem Definitionsbereich D können als zeitdiskrete Signale $f = (f_0, \ldots, f_{n-1})^T \in \mathbb{R}^n$ und $g = (g_0, \ldots, g_{n-1})^T \in \mathbb{R}^n$ aufgefasst werden. Die diskrete Faltung dieser eindimensionalen Signale wird im Folgenden erklärt.

Definition 35. *Das Faltungsergebnis $y \in \mathbb{R}^n$ zweier zeitdiskreter Signale $f \in \mathbb{R}^n$ und $g \in \mathbb{R}^n$ ist durch*

$$y_k := \sum_{j=0}^{k} f_j g_{k-j}, \quad 0 \leq k \leq n-1 \tag{5.3}$$

definiert.

In diesem Fall kann die diskrete Faltung von f und g als Matrixvektorprodukt mit Toeplitz-Matrizen formuliert werden.

Definition 36 (Toeplitz-Matrix, Zyklische Matrix)**.** *Eine diagonalkonstante Matrix $A \in \mathbb{R}^{n\times n}$ der Gestalt*

$$A = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{pmatrix}$$

wird Toeplitz[1]-Matrix genannt. Hierbei gilt $A_{i,j} = A_{i+1,j+1} = a_{i-j}$ für alle Indizies $1 \leq i, j \leq n$. Eine quadratische Toeplitzmatrix ist damit durch ihre erste Zeile und Spalte eindeutig bestimmt. Für den Spezialfall $a_i = a_{-(n-i)} = a_{i-n}$ für alle $1 \leq i \leq n-1$ wird A zyklische Matrix genannt.

Für ein zeitdiskrete Signal $g \in \mathbb{R}^n$ wird die Toeplitz-Matrix

$$G_n = \begin{pmatrix} g_0 & 0 & 0 & \dots & \dots & 0 \\ g_1 & g_0 & 0 & \ddots & & \vdots \\ g_2 & g_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & & \ddots & g_1 & g_0 & 0 \\ g_{n-1} & \dots & \dots & g_2 & g_1 & g_0 \end{pmatrix}$$

konstruiert. Sei ein zeitdiskretes Signal $f \in \mathbb{R}^n$ gegeben und der Vektor $y \in \mathbb{R}^n$ durch

$$y := G_n f = \begin{pmatrix} g_0 & 0 & 0 & \dots & \dots & 0 \\ g_1 & g_0 & 0 & \ddots & & \vdots \\ g_2 & g_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & 0 \\ \vdots & & \ddots & g_1 & g_0 & 0 \\ g_{n-1} & \dots & \dots & g_2 & g_1 & g_0 \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ \vdots \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} g_0 f_0 \\ g_1 f_0 + g_0 f_1 \\ \sum_{j=0}^{2} f_j g_{2-j} \\ \vdots \\ \vdots \\ \sum_{j=0}^{n-1} f_j g_{n-1-j} \end{pmatrix}$$

mit den Einträgen

$$y_k = \sum_{j=0}^{k} f_j g_{k-j}, \quad 0 \leq k \leq n-1$$

gegeben. Dann spiegelt y das Ergebnis der eindimensionalen diskreten Faltung von f und g wider. Die Faltung zweidimensionaler Signale kann mithilfe von zyklischen Blockmatrizen dargestellt werden.

[1] Otto Toeplitz 1881-1940

Definition 37. *Eine Blockmatrix $A \in \mathbb{R}^{n^2 \times n^2}$ bestehend aus Blockmatrizen $B_{i,j} \in \mathbb{R}^{n \times n}$ heißt zyklische Blockmatrix, genau dann wenn die Matrizen $B_{i,j}$ für alle $1 \leq i, j \leq n$ zyklisch im Sinne von Definietion 36 sind.*

Die Konstruktion solcher zyklischen Blockmatrizen soll im Folgenden beleuchtet werden. Dazu seien für $h = b = n$ die Matrizen $X \in \mathbb{R}^{n \times n}$ und $K \in \mathbb{R}^{k \times k}$ einer Faltungschicht gegeben. Zunächst wird der Kern K mithilfe des zero paddings ebenfalls in eine $n \times n$-Matrix eingebettet. Dazu wird der Kern ja nach der gewünschten Größe des Faltungsergebnis von unten und von rechts mit Nullen aufgefüllt, siehe dazu Beispiel 3. Weiter bezeichne $\text{vec}(X)$ die Transformation der Matrix X in einen Vektor der Länge n^2, indem die Spalten von X untereinander geschrieben werden, ähnlich wie bei der Flatten-Funktion 32 aus Kapitel 4. Das folgende Lemma liefert die Darstellung der Faltungsoperation als Matrixvektorprodukt.

Lemma 3 (vgl. Jain[**jain**], Goodfellow[20])**.** *Für $K \in \mathbb{R}^{n \times n}$ wird die zyklische Blockmatrix $A \in \mathbb{R}^{n \times n}$ als*

$$A = \begin{bmatrix} \text{zyk}(K_{1,:}) & \text{zyk}(K_{2,:}) & \dots & \text{zyk}(K_{n,:}) \\ \text{zyk}(K_{n,:}) & \text{zyk}(K_{1,:}) & \dots & \text{zyk}(K_{n-1,:}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{zyk}(K_{2,:}) & \text{zyk}(K_{3,:}) & \dots & \text{zyk}(K_{1,:}) \end{bmatrix}$$

gegeben. Dann entspricht $Y = A\text{vec}(X)$ dem Ergebnis der Matrixfaltung $X * K$. Die Matrix A ist dünnbesetzt.

Beispiel 3. *TODO*

5.1.3. Diskrete Fourier-Transformation

Die Diskrete Fourier[2]-Transformation ist eine Methode aus dem Bereich der Fourier-Analysis. Dabei werden zeitdiskrete endliche Signale auf sogenannte diskrete, periodische Frequenzspektren abgebildet. In diesem Kontext wird zwischen Zeitbereich und Frequenzbereich unterschieden.

Definition 38 (Diskrete Fourier-Transformation)**.** *Im Zeitbereich sei ein diskretes Signal $x = (x_0, \dots, x_{n-1})^T \in \mathbb{R}^n$ gegeben. Dann wird mit $\hat{x} = (\hat{x}_0, \dots, \hat{x}_{n-1}) \in \mathbb{C}^n$ das Ergebnis der diskreten Fourier-Transformation, kurz $\hat{x} = \text{DFT}(x)$, im Frequenzbereich bezeichnet. Die sogenannten Fourier-Koeffizienten sind als*

$$\hat{x}_k := \sum_{j=0}^{n-1} \mathrm{e}^{-\frac{2\pi i}{n} jk} \cdot x_k$$

für $0 \leq k \leq n-1$ definiert.

Mit der inversen Fourier-Transformation kann aus dem Signal im Frequenzbereich das Signal im Zeitbereich rekonstruiert werden. Damit ist es möglich, Signale im Frequenzbereich zu manipulieren und zwischen Zeit- und Frequenzbereich beliebig zu wechseln.

[2]Jean Baptiste Joseph Fourier 1786-1830

Definition 39 (Inverse Diskrete Fourier-Transformation)**.** *Sei* $\hat{x} = (\hat{x}_0, \ldots, \hat{x}_{n-1}) \in \mathbb{C}^n$. *Mit den Koeffizienten*

$$x_k := \frac{1}{n} \sum_{j=0}^{n-1} \mathrm{e}^{\frac{2\pi i}{n} jk} \cdot \hat{x}_k, \quad 0 \leq k \leq n-1$$

lässt sich die inverse diskrete Fourier-Transformation, kurz $x = \mathrm{iDFT}(\hat{x})$, *angeben.*

Die diskrete Fourier-Transformation aus Definition 38 lässt sich in ein Matrixvektorprodukt $\hat{x} = Fx$ überführen, wobei $F \in \mathbb{C}^{n \times n}$ eine symmetrische Matrix der Gestalt

$$F = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{(n-2)} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{(n-2)} & \ldots & \omega_n \end{pmatrix} \tag{5.4}$$

mit

$$\omega_n^j := \mathrm{e}^{-\frac{2\pi i}{n} j}, \quad 0 \leq j \leq n-1$$

ist. Diese Matrix wird Fourier-Matrix genannt und deren Einträge ω_n^j als n-te Einheitswurzeln bezeichnet. Es gilt $\omega_n^n = 1$.

Lemma 4. *Es gilt* $F^H = \bar{F}$ *und die Matrix* $\frac{1}{\sqrt{n}} F$ *ist unitär. Für* $x \in \mathbb{R}^n$ *sei* $\hat{x} = Fx$. *Dann gilt* $F^{-1} = \frac{1}{n}\bar{F}$ *und* $x = \frac{1}{n}\bar{F}\hat{x}$.

Beweis. Wegen $F = F^T$ gilt

$$F^H = \bar{F}^T = \bar{F}.$$

Mit $W := F\bar{F}$ gilt $W_{k,j} = \sum_{l=0}^{n-1} \omega_n^{(j-k)l}$. Ist $k = j$ so ergeben sich die Einträge auf der Hauptdiagonalen von W zu n. Ist $k \neq j$, so ist $\omega_0 := \omega_n^{j-k} \neq 1$ eine n-te Einheitswurzel. Mit der geometrischen Summenformel gilt

$$W_{k,j} = \sum_{l=0}^{n-1} \omega_0^l = \frac{1 - \omega_0^n}{1 - \omega_0} = 0.$$

Also ist $\left(\frac{1}{\sqrt{n}}\right) F \left(\frac{1}{\sqrt{n}}\right) F^H = I$ und damit $\frac{1}{\sqrt{n}} F$ unitär. Schließlich gilt $\frac{1}{n}\bar{F}F = \frac{1}{n}F\bar{F} = I$ und damit $\hat{x} = Fx \Leftrightarrow \frac{1}{n}\bar{F}\hat{x} = x$. □

Wird ein zweidimensionales diskretes Signal in Form einer Matrix $X \in \mathbb{R}^{n \times n}$ betrachtet, lässt sich die zweidimensionale diskrete Fourier-Transformation definieren.

Definition 40. *Sei* $X \in \mathbb{R}^{n \times n}$ *gegeben. Die zweidimensionale diskrete Fourier-Transformation,*

kurz $\hat{X} = \mathrm{2DFT}(X)$*, ist als*

$$\begin{aligned}\hat{X}_{u+1,v+1} :&= \sum_{l=0}^{n-1}\sum_{j=0}^{n-1} X_{l+1,j+1} \cdot \mathrm{e}^{\frac{2\pi i}{n}-(lu+jv)} \\ &= \sum_{l=0}^{n-1} \mathrm{e}^{-\frac{2\pi i}{n}lu} \left(\sum_{j=0}^{n-1} X_{l+1,j+1} \cdot \mathrm{e}^{-\frac{2\pi i}{n}jv} \right), \quad 0 \leq u, v \leq n-1\end{aligned}$$

definiert.

Die zweidimensionale diskrete Fourier-Transformation ist als Hintereinanderausführung von zwei eindimensionalen Fourier-Transformationen, vgl. Definition 38, zu verstehen. Zuerst wird die DFT der Zeilen und anschließend die DFT der Spalten von X berechnet. So lässt sich $\hat{X} = FXF^T$ als Matrixmatrixprodukt mit der Matrix F aus (5.4) darstellen. Die inverse zweidimensionale Fourier-Transformation kann als wieder als Fourier-Transformation beschrieben werden.

Lemma 5. *Sei* $X \in \mathbb{R}^{n\times n}$ *und* $\hat{X} = \mathrm{2DFT}(X)$*. Dann gilt* $X = \frac{1}{n}(DFT(X)^H)^H$*.*

Beweis. todo □

Zwischen der zyklischen Faltung 35 und der diskreten Fourier-Transformation 38 besteht ein fundamentaler Zusammenhang, und zwar das Faltungstheorem. Eine Version davon wird im weiteren Verlauf dieser Arbeit genutzt, um die Matrixfaltung, vgl. Definition 28, mithilfe von Fourier-Transformationen zu berechnen.

Satz 4 (Zyklisches Faltungstheorem)**.** *Seien Vektoren* $f, g \in \mathbb{R}^n$ *gegeben und* $y = f * g$ *das Ergebnis der zyklischen Faltung. Dann gilt*

$$\mathrm{DFT}(y) = \mathrm{DFT}(f) \odot \mathrm{DFT}(g). \tag{5.5}$$

Dabei bezeichne $\odot$ *die elementweise Multiplikation der Einträge von den beteiligten Vektoren.*

Beweis. Ein Beweis ist von Smith[55] gegeben. □

Bemerkung 5. *Seien die Matrizen* $X \in \mathbb{R}^{n\times n}$ *und* $K \in \mathbb{R}^{k\times k}$ *mit ungeradem* k *gegeben. Weiter sei* $l = \lfloor k/2 \rfloor$*. Die Matrixfaltung* $Y = X * K$ *ist durch*

$$Y_{i,j} = \sum_{u=-l}^{l}\sum_{v=-l}^{l} X_{i+u,j+v}K_{u,v}, \quad 1 \leq i, j \leq n$$

erklärt, vgl. Bemerkung 2. Der Kern K *wird in eine* $n \times n$*-Matrix wie in* **??** *eingebettet und diese wird wieder mit* $K \in \mathbb{R}^{n\times n}$ *bezeichnet wird. Sind* $\hat{X} = \mathrm{2DFT}(X)$ *und* $\hat{K} = \mathrm{2DFT}(K)$ *die zweidimensionalen diskreten Fourier-Transformationen, so gilt mit dem Faltungstheorem 4 der Zusammenhang*

$$\mathrm{2DFT}(Y) = \mathrm{2DFT}(X) \odot \mathrm{2DFT}(K).$$

Die Matrixfaltung innerhalb einer Faltungsschicht eines CNN lässt sich mit den obigen Resultaten in drei Schritten berechnen.

1. Es sind jeweils die zweidimensionalen diskreten Fourier-Transformationen für die Eingabe $\hat{X}$ und den Kern $\hat{K}$ zu berechnen.
2. Die Matrix $\hat{Y} = \hat{X} \odot \hat{K}$, welche sich aus der elementweisen Multiplikation ergibt, ist zu bestimmen.
3. Schließlich stellt $Y = \text{i2DFT}(\hat{Y})$ die Matrixfaltung dar, welche mithilfe der inversen diskreten Fourier-Transformation ermittelt wird.

5.2. Datenbankgestützte Vorwärtsrechnung für CNN

5.3. Evalution

6. Zusammenfassung und Ausblick

Algorithm 4 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

A. Anhang

A.1. Weitere Basisoperation in SQL

Die euklidische Norm

$$||x||_2 := \left(\sum_{i=1}^{n} x_i^2\right)^{\frac{1}{2}}$$

eines Vektors $x \in \mathbb{R}^n$ und die Frobeniusnorm

$$||A||_F := \left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2\right)^{\frac{1}{2}}$$

einer Matrix $A \in \mathbb{R}^{m \times n}$ sind als SQL-Anfrage durch

SELECT sqrt(**SUM**$(v * v)$) **AS** $2Norm$
FROM x

beziehungsweise

SELECT sqrt(**SUM**$(v * v)$) **AS** $FNorm$
FROM A

gegeben. Im euklidischen Vektorraum $\mathbb{R}^n$ ist das Skalarprodukt $\langle x, y\rangle$ zweier Vektoren $x, y \in \mathbb{R}^n$ definiert als

$$\langle x, y\rangle := \sum_{i=1}^{n} x_i y_i.$$

Eine entsprechende Transformation in SQL mit dem Aggregationsoperator **SUM** lautet

SELECT SUM$(x.v * y.v)$ **AS** v
FROM x **JOIN** y **ON** $x.i = y.i$

Die Transponierte Matrix $A^T \in \mathbb{R}^{n \times m}$ einer Matrix $A \in \mathbb{R}^{m \times n}$ kann durch die Anfrage

SELECT $A.j$ **AS** i, $A.i$ **AS** j, $A.v$
FROM A

berechnet werden.

Literatur

[1] Foto Afrati und Rada Chirkova. „Answering queries using views“. In: *Synthesis Lectures on Data Management* 14.3 (2019), S. 1–275.

[2] Rakesh Agrawal und Ramakrishnan Srikant. „Privacy-preserving data mining“. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, S. 439–450.

[3] 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.

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

[5] 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.

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

[7] 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.

[8] Sergey Brin und Lawrence Page. „The Anatomy of a Large-Scale Hypertextual Web Search Engine“. In: *Comput. Networks* 30.1-7 (1998), S. 107–117.

[9] Ilvio Bruder u. a. „Konzepte für das Forschungsdatenmanagement an der Universität Rostock(Concepts for the Management of Research Data at the University of Rostock)“. In: *LWDA*. Bd. 1917. CEUR Workshop Proceedings. CEUR-WS.org, 2017, S. 165.

[10] 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.

[11] Rada Chirkova, Jun Yang u. a. „Materialized views“. In: *Foundations and Trends in Databases* 4.4 (2011), S. 295–405.

[12] 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.

[13] Sara D'Onofrio und Andreas Meier. *Big data analytics*. Springer, 2019.

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

[15] 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).

[16] Iain S Duff, Roger G Grimes und John G Lewis. „Sparse matrix test problems“. In: *ACM Transactions on Mathematical Software (TOMS)* 15.1 (1989), S. 1–14.

[17] Stefan Duffner. „Face image analysis with convolutional neural networks“. Diss. Freiburg (Breisgau), Univ., Diss., 2008, 2008.

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

[19] Hector Garcia-Molina, Jeffrey D. Ullman und Jennifer Widom. *Database systems - the complete book (2. ed.)* Pearson Education, 2009.

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

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

[22] Andreas Heuer. „METIS in PArADISE Provenance Management bei der Auswertung von Sensordatenmengen für die Entwicklung von Assistenzsystemen“. In: *BTW Workshops*. Bd. P-242. LNI. GI, 2015, S. 131–136.

[23] Andreas Heuer, Gunter Saake und Kai-Uwe Sattler. *Datenbanken - Konzepte und Sprachen, 6. Auflage*. MITP, 2018.

[24] Andreas Heuer, Gunter Saake und Kai-Uwe Sattler. *Datenbanken - Implementierungstechniken, 4. Auflage*. MITP, 2019.

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

[26] 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`.

[27] „IEEE Standard for Binary Floating-Point Arithmetic“. In: *ANSI/IEEE Std 754-1985* (1985), S. 1–20. DOI: `10.1109/IEEESTD.1985.82928`.

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

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

[30] Hiroyuki Kurihata u. a. „Rainy weather recognition from in-vehicle camera images for driver assistance“. In: *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* IEEE. 2005, S. 205–210.

[31] Rick F. van der Lans. *SQL standard - a complete reference*. Prentice Hall, 1989.

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

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

[34] Yann LeCun u. a. „Efficient BackProp“. In: *Neural Networks: Tricks of the Trade (2nd ed.)* Bd. 7700. Lecture Notes in Computer Science. Springer, 2012, S. 9–48.

[35] Alon Y Levy, Anand Rajaraman und Jeffrey D Ullman. „Answering queries using limited external query processors“. In: *Journal of Computer and System Sciences* 58.1 (1999), S. 69–82.

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

[37] Dennis Marten. „Big Data Analytics für die effiziente Aktivitätserkennung und -vorhersage in Assistenzsystemen“. Dissertation. Universität Rostock, 2021.

[38] Dennis Marten und Andreas Heuer. „A framework for self-managing database support and parallel computing for assistive systems“. In: *PETRA*. ACM, 2015, 25:1–25:4.

[39] Dennis Marten und Andreas Heuer. „Transparente Datenbankunterstützung für Analysen auf Big Data“. In: *GvD*. Bd. 1366. CEUR Workshop Proceedings. CEUR-WS.org, 2015, S. 36–41.

[40] Dennis Marten und Andreas Heuer. „Machine learning on large databases: transforming hidden Markov models to SQL statements“. In: *Open Journal of Databases (OJDB)* 4.1 (2017), S. 22–42.

[41] Dennis Marten u. a. „Sparse and Dense Linear Algebra for Machine Learning on Parallel-RDBMS Using SQL“. In: *Open J. Big Data* 5.1 (2019), S. 1–34.

[42] 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.

[43] Bruce Momjian. *PostgreSQL: introduction and concepts*. Bd. 192. Addison-Wesley New York, 2001.

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

[45] Edin Omerdic u. a. „Design & development of assistive tools for future applications in the field of renewable ocean energy“. In: *OCEANS 2011 IEEE-Spain*. IEEE. 2011, S. 1–6.

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

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

[48] 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.

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

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

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

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

[53] Youcef Saad. „SPARSKIT: A basic tool kit for sparse matrix computations“. In: (1990).

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

[55] Julius Orion Smith. *Mathematics of the discrete Fourier transform (DFT): with audio applications*. Julius Smith, 2007.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

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

[62] Hermann Winner u. a. *Handbook of driver assistance systems*. Springer International Publishing Amsterdam, The Netherlands, 2014.

[63] Marianne Winslett und Vanessa Braganholo. „Dan Suciu Speaks Out on Research, Shyness and Being a Scientist“. In: *SIGMOD Rec.* 46.4 (2017), S. 28–34.

[64] Felix Wortmann und Kristina Flüchter. „Internet of things“. In: *Business & Information Systems Engineering* 57.3 (2015), S. 221–224.

[65] Feng Xia u. a. „Internet of things“. In: *International journal of communication systems* 25.9 (2012), S. 1101.

[66] Dingjun Yu u. a. „Mixed pooling for convolutional neural networks“. In: *International conference on rough sets and knowledge technology*. Springer. 2014, S. 364–375.

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

B. Anhang

B.1. Listings

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

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

B.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}
}
```